THE COMPUKIT© UK101 MANUAL

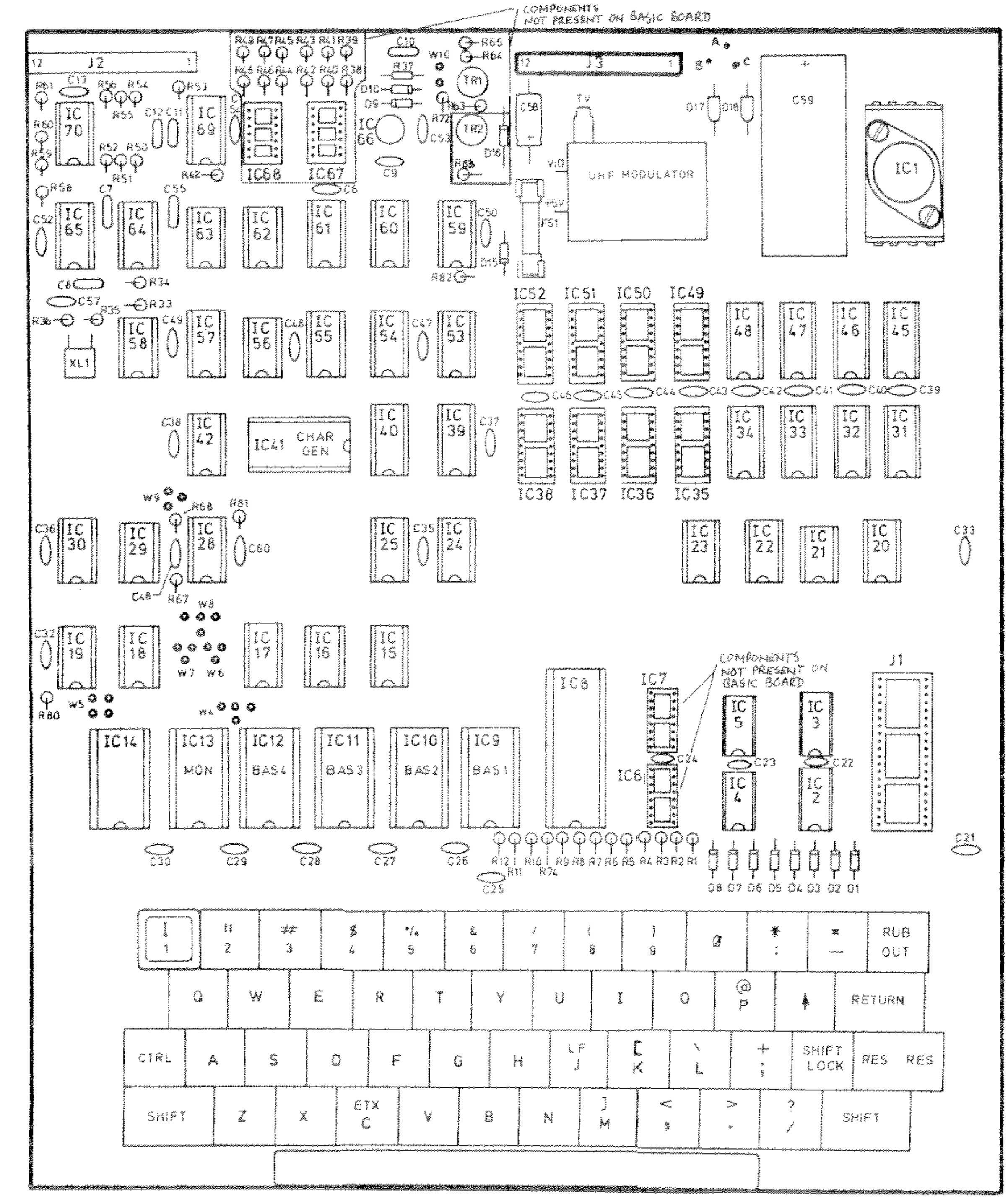
A.A. BERK B.Sc. Ph.D.

- * 8K Microsoft, full feature, BASIC
- * Fully expandable via on board sockets
- * Up to 8K RAM on board
- * Cassette interface (Cuts)
- * VDU-with its own dedicated RAM (1K)
- * Full ASCII keyboard
- * UHF modulator on board
- * PSU on board-transformer included in kit
- * Full machine code monitor and I/O utilities in ROM
- * Upper/lower case-graphics and gaming characters

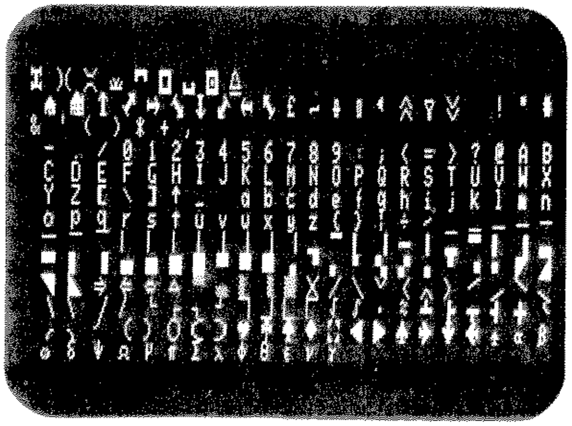
Suppliers

COMPUKIT LTD.
14 Station Road, New Barnet, Herts. EN5 1QW

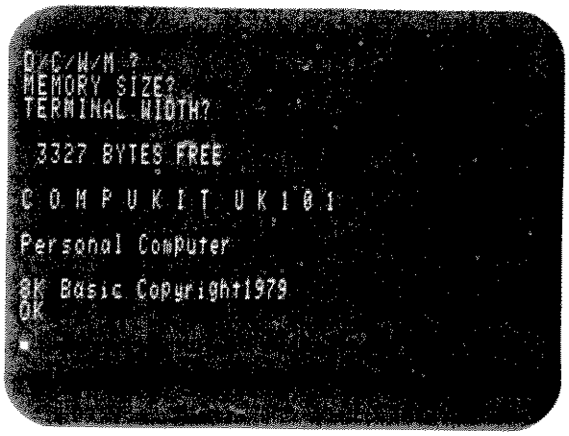
Component Positioning Diagram



KEYBOARD



CHARACTER SET AVAILABLE



```
D/C/W/M ?
MEMORY SIZE?
TERMINAL WIDTH?

 3327 BYTES FREE

C O M P U K I T  U K 1 0 1

Personal Computer

8K Basic Copyright1979
OK
■
```

INITIAL SYSTEM RESET

Foreword

This manual has been prepared for both the experienced user and the first time buyer.

It is intended to act as an initial introduction to the machine's usage as well as a reference manual for the realisation of its full potentialities.

The machine itself has been designed and produced in its present form to fulfil the need for an inexpensive 8K BASIC system capable of use in two forms. Firstly as a sophisticated one-board computer capable of solving many problems with no further addition; and secondly, as the centre of a much larger system with all the expansions of disc and extra memory, as well as almost any "control" type peripheral it is possible to conceive. Printers, floppy discs all plug in directly, as will any hardware expansions which the owner may feel able to interface and run for him or herself.

The possibilities for this unique machine are endless, and it is hoped that it will play some part in bringing true home computing within the range of every family and household.

Special thanks are due to A. Fisher for his extensive work on the Software side of the **COMPUKIT,** and to Practical Electronics for permission to use their excellent hardware & software photographic artwork.

Dr. A.A. BERK

Contents

Introduction

The **COMPUKIT UK101** has been adapted to satisfy a major need in the computer market. This is for an inexpensive and powerful system supporting a full BASIC package and with all the expansion, such as disk and hard printer etc., which the microcomputer industry is able to support.

The board consists of the following:

a. Full upper and lower-case ASCII keyboard — software scanned for speed and flexibility.

b. 8K MICROSOFT BASIC.

c. 2K monitor including machine code and floppy disk bootstrap.

d. Memory mapped VDU with its own 1K of dedicated RAM plus graphics. Line width selectable from 48 to 16 characters by 16 lines.

e. Up to 8K RAM on board making a total of 19K of addressable memory on the PCB.

f. Cassette interface (Kansas city) plus printer interface.

g. Full power supply — even the transformer is included.

h. Astec UHF modulator.

i. Supports all Ohio Scientific expansion as simple plug-in options including Floppy Disks.

The PCB is of full professional standard, plated-through holes — silk screened component positioning mask and sufficient mechanical strength to support the keyboard rigidly.

The entire system runs on a single +5v Supply and with 8K RAM, uses 2 - 3 Amps at this voltage from the on-board power supply.

This manual gives constructional details along with a technical description of the system — full circuit diagrams are included and are explicit enough for any level of personal hardware modification. A section on usage of the machine is followed by a reference description of the BASIC language available on the **COMPUKIT** — this is in no sense meant to be a primer on the subject and, for the beginner, it must be read in conjunction with a book on BASIC. A 6502 machine code instruction listing and some information on the processor is included at the end of this manual and is intended as a reference work only.

The **COMPUKIT UK101** is described in a set of articles, by the Author of this Manual, published in Practical Electronics starting in the August 1979 edition.

Hardware Description

BINARY COUNTING CHAINS

The clocking requirements for the system are supplied by the crystal oscillator and binary counting chains. Two gate of IC58 plus X1 form an 8 MHz oscillator buffered by a further gate in IC58 and divided by 8 by IC29. (IC29 thus has a spare ÷ 8, The (CLK) line feed the "Dot" clock of the VDU at 8 MHz. This governs the length of time available for displaying one of the dots of a character of the TV screen. Given the speed with which the electron beam strobes across the screen and the "dot" time, the width of a dot may be calculated. 8 MHz gives a dot size sufficiently small to fit about 48 characters across the screen (each 8 dots wide), while of low enough frequency to pass easily through the UHF modulator and IF stages of a TV set.

The D output of IC29 (at 1 MHz) then feeds the ∅0in line of the MPU, CO line of the VDU and the counting chain of 74163's (or 74161's) IC59-61 and IC30. The constraints on the counting chain are that it must produce ripple-count outputs for C1-C6 in between line-sync pulses separated by 64 us — note: 2^6 = maximum of 64 characters per line. There must be three outputs (C8 - C10) for the row inputs to the character generator, and a further four outputs (C11 - C14) for the 16 horizontal lines of characters. The entire picture must then be repeated at 50 times a second with a suitable frame-sync pulse. The final count output from the bottom of the chain is then inverted and fed to load the chain elements.

C3 is used to set the BAUD rate for the Cassette and serial interface via a further counter, IC57, and some decoding logic IC63 and IC58.
NB: R82 is a Common pull-up resistor for several devices.

VDU

The block diagram, fig 1, shows the basic parts of the VDU and the circuit diagram gives the details referred to below.

The VDURAM holds a screen full of characters (1024 in all). Through IC53-IC55, the RAM address lines, VA0 - VA9, are either fed from the counter chain or the MPU Address Bus — depending upon the state of VA (VDU Access). When VA is at a '1', C1 - C6 and C11-C14 are connected through to VA0 - VA9, and when VA is at a '0', the MPU busses have direct Read/Write access to the VDURAM. Reading or writing of data is controlled by the bidirectional buffers IC24 and IC25 which also disconnect the VDURAM from the MPU Data Bus when the counter chain is supplying addresses to VA0 - VA9. Thus when VA is in the Zero state, the VDURAM acts just like any other block of Read/Write memory, here based at location address D000 (Hex). This allows the screen to be read or written to during a program. With VA at a '1', the 10 VDU RAM addresses are derived from the counters sequentially. The RAM is in the READ condition when not selected by the MPU, and the contents of the RAM locations are sent to the character generator for interpretation into bit patterns forming characters on the screen.

Each character in the Character Generator, IC41, is stored as an 8x8 matrix of white and black dots. White is stored as a 'One', The characters appear on the outputs of IC41 (D0 - D7) one row at a time, see fig 2. Here an 'E' is being displayed on one of the 16 lines of text on the TV screen. C8, C9 and C10 from the counter chain determine which row (R0 - R7) is being output at any time. The sequence of events is as follows. C1 - C6, C11 - C14 contain an address of a location in VDURAM and hence of some character on the screen. The contents of this location (8 bits in parallel) are fed to IC41 which then outputs (in parallel) the 1's and 0's (white and black dots) of one row of the character along D7 - D0. Here, five 1's and three 0's are output to form the top row of the 'E'. IC42 serialises this parallel information at 8 MHz, and sends it out in a stream to IC70 to be mixed with TV sync. information etc. and displayed along a TV line as the electron beam strobes across the TV screen.

This takes 1us and each successive 1us sees IC42 loaded with another character-row for the same treatment — (LD) is fed from CO at 1 MHz via a monostable (half of IC71) to give a short negative going pulse, and (CLK) is at 8 MHz. This is the "Dot" clock — so named as each cycle displays one of the 8 dots of a character on the screen. After the top row of the "E" has been displayed, the top row of the next character on that line must be fetched. Again, C8, C9, C10 will not change but C1 - C6 will, hence selecting the next VDURAM location, and so on until C1 - C6 have displayed one row of 64 characters. Some of these are lost at the ends of the line as the Dot clock is only at 8 MHz. When C1 -C6 have finished rippling through, C7 changes and the whole is repeated. C6 synchronises the TV line (at 64us intervals) and thus starts a new line via IC65 on its downward edge. C7 is not used in the process and thus C1 - C6 must count through twice before C8, 9, 10 increment to a new row of the character, this causes each row of dots to occupy two TV lines as shown in Fig 2.

As C8, 9, 10 increment, the complete set of 16 TV lines builds up a row of text. The next step is to increment C11 - C14 to address the next row of characters stored in the VDURAM. The complete frame of 256 TV lines is built up as C1 - C14 count through. Normally, in TV transmissions, another frame slightly different from this, is interlaced in the space between the lines of the first frame. Also, each half frame is composed of more lines. Here, C15, via IC71, provides a frame-sync. pulse to the TV and the above process repeats exactly — each line occupying its previous position. The resolution thus obtained is not as high as a normal TV picture, but is more than adequate for 16 lines of VDU information.

The frame-sync. is delayed by half of IC65 to allow the TV picture to be moved up the screen and hence prevent the bottom left character from being lost. This is the most important slot on the screen and must be displayed clearly. The value of the components R33 and C8 may be adjusted to ensure its readability on any TV.

About 48 characters are able to be displayed on a normal TV, and hence some characters are lost from the edges of the screen. A few are missing from the start of the line and the rest from the end. The software of the **COMPUKIT** uses just the 48 slots to prevent loss of information — the others are available to the user, however, and may be forced into display by adjusting a TV or monitor to 'underscan'. The RAM locations are still perfectly valid and may be used as normal.

A note about Graphics should be made at this point. Since an 8x8 matrix of dots is used for characters in general and only a 7x5 matrix is used for the ASCII characters, spaces of varying sizes are left between text characters both horizontally and vertically. However, the **COMPUKIT's** character generator is very rich in blocks, lines and special patterns which use the full 8x8 array of dots. By this means, adjacent graphic characters may be chosen to run into each other, and graphs, large patterns, block diagrams etc may all be constructed from basic components. Also, some extra characters are included such as £, π, etc. for a very full variety of uses.

ADDRESS DECODING AND MEMORY

Address decoding is performed via 74138's and 74139's with some extra gating. The address map defines the operation of this block and it will not be described in full electrical detail — a TTL data book will provide all the information necessary to understand how this block works. RS0 - RS7 are selects for the RAM (8 blocks of 1K, each comprising two 2114's). BS0 - BS3 select the BASIC ROMs and MCS selects the monitor ROM. ACS selects the ACIA for the cassette. RKB and WKB are Read and Write selects for the keyboard and WVE and RVE for the VDU.

The RAMs are addressed so that IC31 and IC45 are at the lowest addresses and hence form the "first" 1K block of RAM (based at 0000). Addresses increase from right to left in pairs — the 2114s being arranged as 1K by four bits—IC32 and IC46 are next and so on. The ROMs are arranged to allow other options. When the 64K bit ROM is available, the four BASIC ROMs may occupy one package — A11 & A12 will be needed and an address decoded line to select it. This already exists on the **COMPUKIT** — BS supplying the necessary address decoding. W1, W2 and W4 are pads next to the ROMs bringing these lines in. When this option is available, there will be three spaces free for ROMs or EPROMs of the user's choice. The **COMPUKIT** even allows for active high or low BS line via U18.

The Monitor ROM also has some flexibility in packaging and this is catered for as shown.

PROCESSOR AND EXPANSION SOCKET J1

The processor is shown feeding all the Buses and control lines internally as well as externally via J1, whose data lines are fully buffered by IC6 and IC7. External devices decide the direction of data flow through these buffers by DD. A lower DD allows to READ from the external bus. A High allows it to write. This socket allows any external logic to overtake the MPU system via interrupts and can easily be extended to control anything. External memory may be added via the socket, disc storage, S100 Bus expansions etc. etc. may all be plugged in directly.

SERIAL AND CASSETTE INTERFACE

The serial interface is controlled by IC14 - the ACIA. This is primarily to drive a cassette interface. However, components are provided on-board to allow the ACIA to drive a RS232 interface if required. This will not be described here but is shown in the diagram.

The ACIA receives its clock from C3 of the counting chain via IC57,IC63 and IC58. Options exist,as shown, to separate the Tx and Rx clocks. In addition, driving the clock from C2, C1 or C0 will increase the BAUD rate from 300 by a factor of 2, 4 or 8 respectively.

The ACIA's Tx and Rx data lines are fed to the cassette interface as shown. The transmitter uses a 7476 (IC64) to present a high or low tone to the recorder as a "1" or "0" is to be recorded — this follows the usual Kansas City recording format.

Receiving depends upon the time-constant of a monostable. IC66 and IC62 are used to convert the sine-wave input, from cassette, to square-wave suitable for the monostable IC69 and the clock input of a D-type flip-flop, IC63. While the tone is high, the 74123's time-constant is set such that the Q-output has no time to reset to zero before the next positive edge at B forces it high again. D and CLR of IC63 thus remain high, as Q does, and Rx DATA presents a constant "1". When a low tone arrives, the cycles arriving at B are long enough to allow Q to reset, after its positive-going timing pulse, before B suffers a further positive-going edge forcing Q high again. This gives the timing diagram shown in fig 3 for IC63.

The leading edge of D is slowed by R62 and C55. The zero on CLR now sets Q to zero and, because D's rising edge is slowed down, IC63 sees a zero on D when the clock goes high thus preserving the zero on Q and hence the circuit decodes a constant zero for as long as the low tone continues.

This sort of circuit is quite reliable at 300 BAUD and any instability will be due either to a large variation in tape speed, or to the value of R53 and C11 having been incorrectly chosen thus allowing the negative-going edge on D to arrive too soon.

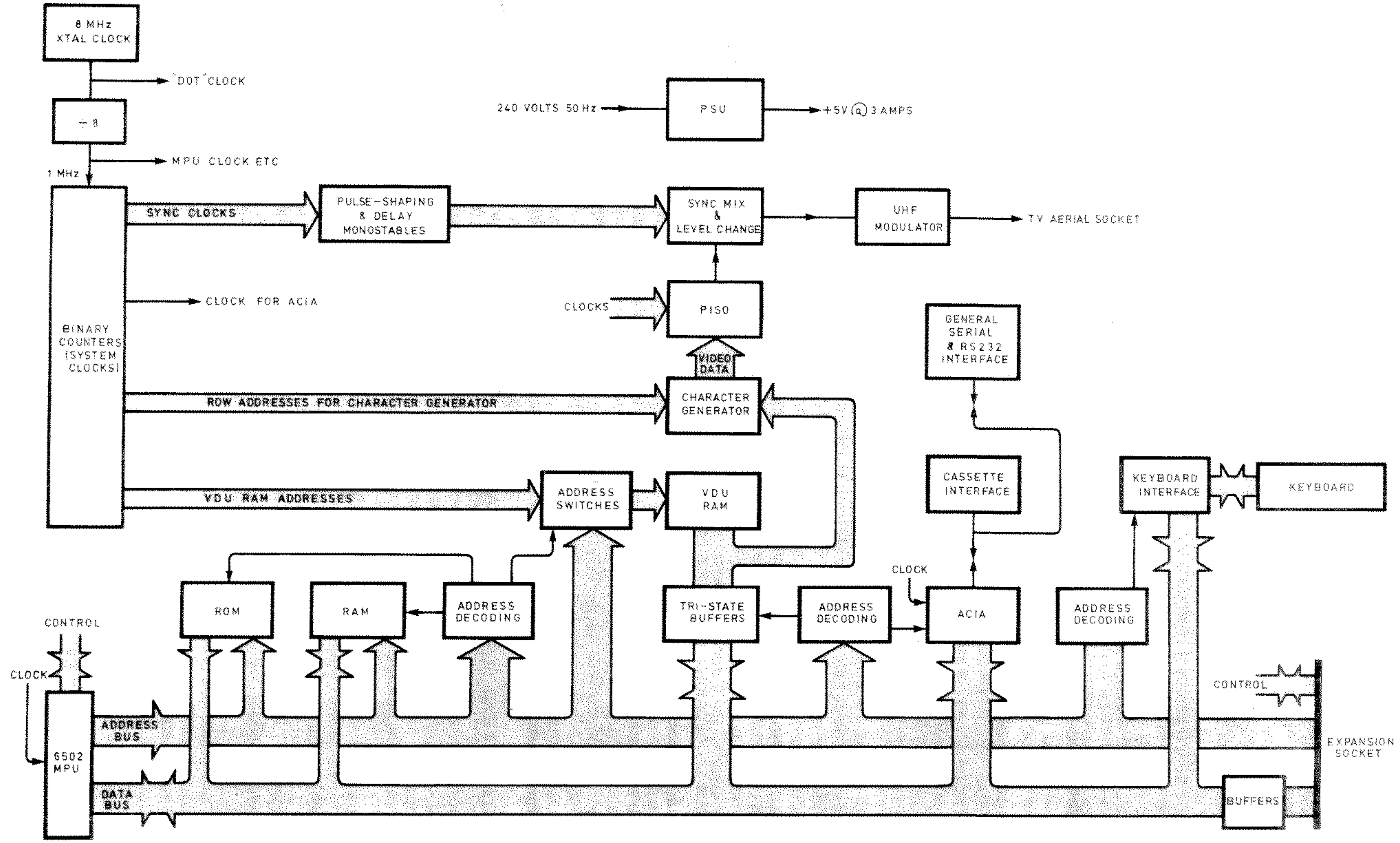
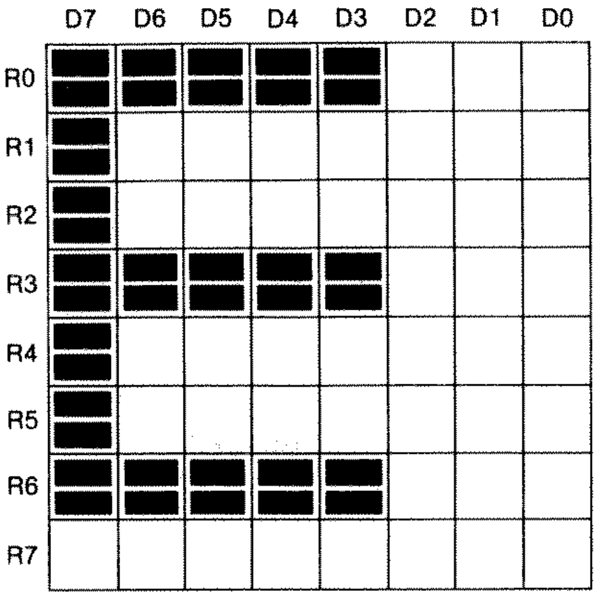


Fig. 1

BLOCK DIAGRAM



	D7	D6	D5	D4	D3	D2	D1	D0
R0	■	■	■	■	■			
R1	■							
R2	■							
R3	■	■	■	■	■			
R4	■							
R5	■							
R6	■	■	■	■	■			
R7								

Fig.2

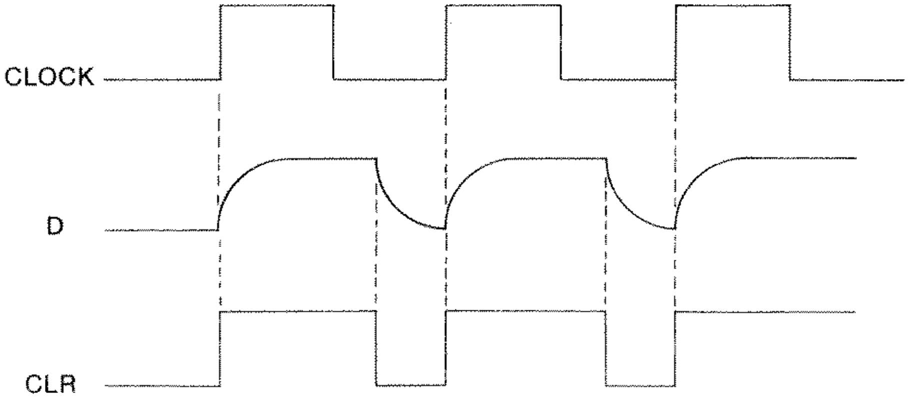


Fig.3

Component List COMPUKIT UK101

IC's

(U numbers and IC numbers are identical).

IC2,3	74LS75
IC4,5	74LS125
IC6,7	8T28 (Only used for expansion)
IC8	6502
IC9 - IC12	BASIC ROMs (nos 1 - 4 respectively)
IC13	MONITOR ROM
IC14	6850
IC15	7402
IC16,18,21,62	7404
IC17	74LS139
IC19, 56	74LS20
IC20,22,23	74LS138
IC24,25	8T28
IC28,65,69	74LS123
IC29	7493
IC30,59-61	74163 (or 74161)
IC31-40,IC45-52	2114
IC41	Character Generator
IC42	74165
IC53-55	74LS157
IC57	74163
IC58	7400
IC63	7474
IC64	7476
IC66	CA3130
IC70	7403
REG (ICI)	3 Amps +5 Volts

N.B.: LS or non-LS may be supplied for the above, except for IC's: IC17, 20, 22, 23, 2, 3.

SOCKETS (All DIL)

2 off	40-pin
7 off	24-pin
18 off	18-pin
21 off	16-pin
13 off	14-pin
1 off	8-pin

RESISTORS
(All 5% ¼ Watt carbon film)

R1-R8,10-12,68, 80,81	4K7
R33,37,55,63,64	10K
R34,50	15K
R36,35,60,61,65	470R
R51, 9	270R
R32,54,59,72,82	1K
R53	22K
R56	100K
R58	560R
R62	100R
R67	27K
R74	390R

CAPACITORS

C6	150pF (Ceramic)
C7,C12,C55	1nF (1000pF) (Mylar)
C8, C11	100nF (Mylar)
C9	68pF (Ceramic)
C10,C13	10nF (Mylar)
C48	220nF (Ceramic or Mylar)
C57	27pF (Ceramic)
C58	47uF (Electrolytic)
C59	3300uF (Electrolytic)
C60	22pF (Ceramic)

All others (31 in all) are 0.1uF Ceramic Capacitors for bypassing.

DIODES

D1 - D10	IN914 (or equivalent)
D15	IN4001 (or similar)
D17, 18	3Amp Rectifier diodes.

TRANSISTORS

Q1	any small signal PNP transistor.

MISCELLANEOUS

F1	3 Amp fuse plus holder
XL1	8 MHz crystal

UHF Modulator — 8MHz bandwidth version of Astec modulator.

Double-sided plated through PCB with solder mask and component legend.

Set of keyboard switches and key-tops

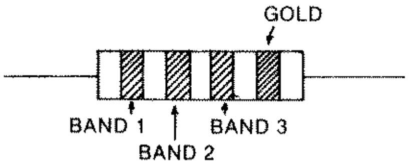
Regulator heat-sink plus nuts and bolts for fixing.

Transformer: 240v 50Hz primary, 8-0-8 volt 3 Amp secondary.

Construction

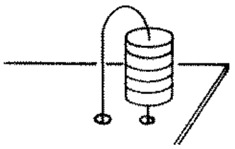
The first part of this section is for reference by beginners. The later part describes the construction itself.

Resistor colour codes:



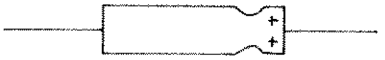
(5% ¼ Watt carbon film)

Most resistors are mounted on end:



RESISTOR	VALUE	BAND 1	BAND 2	BAND 3
R1-R8,R10-R12, R68,R80, R81	4K7	YELLOW	PURPLE	RED
R34, 50	15K	BROWN	GREEN	ORANGE
R58	560R	GREEN	BLUE	BROWN
R36,R60,R61, R65,35	470R	YELLOW	PURPLE	BROWN
R37,R55,R63, R64,R33	10K	BROWN	BLACK	ORANGE
R51,R9	270R	RED	VIOLET	BROWN
R52,R54,R59, R72,R82	1K	BROWN	BLACK	RED
R53	22K	RED	RED	ORANGE
R56	100K	BROWN	BLACK	YELLOW
R62	100R	BROWN	BLACK	BROWN
R67	27K	RED	PURPLE	ORANGE
R74	390	ORANGE	WHITE	BROWN

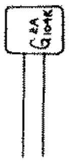
Capacitors:
Electrolytic capacitors are polarised by a + sign at one end



(C58 & C59)

Mylar capacitors are green with the following markings:

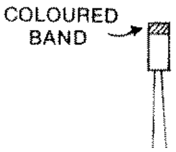
CAPACITOR	VALUE	MARKING
C7, C12, C55	1nF	102K
C10, C13	10nF	103K
C8, C11	100nF	104K
C48	220nF	224K



100nF

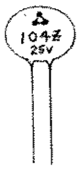
N.B. C48 may be supplied in black ceramic marked: 220nS

Two types of ceramic capacitor are used with markings as follows:
Low Values:



CAPACITOR	VALUE	MARKING
C6	150pF	BLACK BAND GREY BODY
C9	68pF	PURPLE BAND 68P MARKED ON BODY
C57	27pF	PURPLE BAND 27P MARKED ON BODY
C60	22pF	ORANGE: 22K

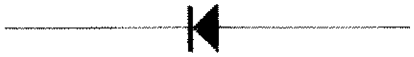
All Others:



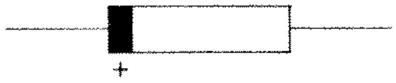
These are 100nF bypassing capacitors to limit the transmission of unwanted high frequency spikes along the power lines. These should be assembled into position last, and simply allowed to occupy any unused capacitor positions marked on the component overlay. They are not critical and one or two missed will not affect the system too greatly.

Diodes:

The component positioning diagram has every diode marked by the symbol:



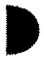
The component itself has a bar around its body as shown below



This bar corresponds to that in the diode symbol and the wire at that end of the diode must be soldered into the correct hole. The + sign on the PCB, on some diodes, also corresponds to the "bar" end of the diode.

Transistor:

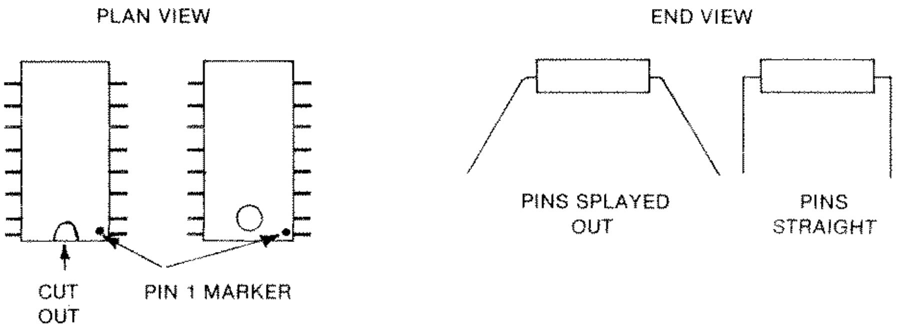
The transistor looks like:



from the top, and there is a corresponding pattern on the PCB to indicate which way round the transistor must be fitted.

Integrated circuits:

All integrated circuits have indications on their upper surface to ensure that they are fitted correctly.



Integrated circuits must be fitted with Pin One nearest to the keyboard — except for IC 41, the Character Generator, which is fitted with its Pin One towards IC 39 and IC 40 (i.e. to the right).

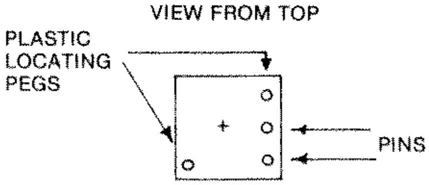
The pins of the IC's will be splayed out in most cases, these should be carefully straightened as shown above before any attempt is made to insert them into a socket.

IC numbers appear as U numbers on some diagrams.

The numbers printed on the surface of the devices may vary considerably. The numbers given on the component position diagram will be found somewhere on the upper surface of the device perhaps with a set of prefix or subscript letters. In some cases where an LS device has been specified the non-LS equivalent may be supplied. For instance, IC 19 is specified as a 74LS20 — the 7420 may be present in the kit instead.

The only critical IC's found are IC2, IC3 which will be supplied in the LS version.

Keyboard switches:

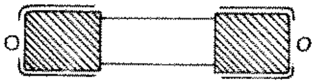


Note that the pins are offset to the right when the switch is viewed from the top. These components should be inserted and soldered with great care.

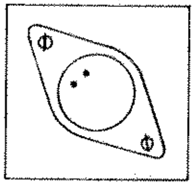
N.B. The SHIFT-LOCK position is occupied by a switch which remains down when pressed once and returns when pressed again.

IC sockets:
14-pin and 16-pin sockets look very similar — make sure the right sized socket is used for each IC, by counting the number of pins on the pad before selecting a socket.

Fuse Holder:
This comes in two parts — each should be soldered in as shown and the fuse pushed into place.



Regulator and heat sink:
Assemble regulator to heat sink and bolt onto PCB. Solder pins of regulator beneath board.



A certain amount of the following will be considered unnecessary by the experienced, while some points are very important — the owner is advised to read through this section at least once!

You will need a good pair of wire cutters, a small screw-driver and a soldering iron of around 15 - 20 Watts with a narrow bit. The bit should ideally be new — make sure you coat the end with solder as it very first warms up or a patina of corrosion will immediately form making soldering impossible. Also, iron-clad bits must not be filed to clean them or the iron protection is lost and they corrode very fast. The thinnest resin-cored solder should be used. Do not run too much solder into the joints as the board is plated through and thickly tinned to provide some of its own solder for pins etc. pushed through the holes.

Never try to drill any of the PCB holes out as this will destroy the plating-through and prevent tracks on the top of the board from connecting with those underneath. All solder connections are made to the bottom of the board and no IC pins must remain unsoldered even if they appear to go nowhere — the connections to these pins may well run on the top surface of the board and down through the pin hole to the solder pad below. The board should be protected at all times from excessive abrasion and flexion and should be handled as little as possible — and then only with clean hands.

Following the component legend very carefully, the most efficient sequence of construction is to start with the IC sockets. Locate and push their pins carefully through the holes, taking extreme care to prevent the pins from being bent under the socket — if this happens, the pins will usually break as they are quite brittle. The socket must be pressed very firmly against the PCB while two pins are soldered down to keep it in place.

Sockets are not supplied for the following positions:
IC67, IC68. Be careful not to use any in these places until the others are soldered in.

All IC's are fitted with pin 1 towards the keyboard except for IC41 (Character Generator) whose pin 1 is towards the RAM block. Sockets are normally polarised in some way and even though IC's will fit in any way round, it is a good plan to put the sockets in correctly as a reminder for the future. Do not insert the IC's yet.

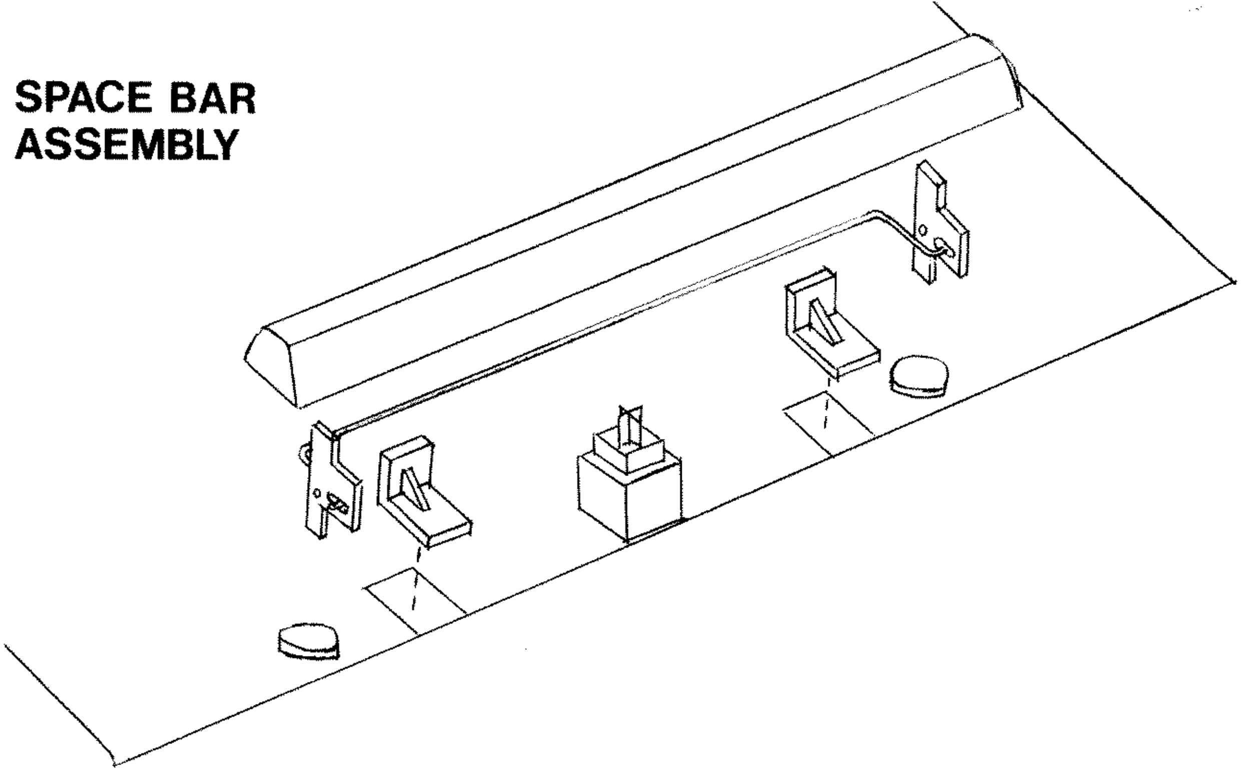
The best operation to perform next is the insertion of the discrete components except for the voltage regulator, UHF modulator and large capacitor which make the board unwieldy. The 100nF bypass capacitors should be soldered in last of all to prevent their being mixed up with the more important components. Each component must be carefully inserted, wires clipped and soldered in place — most of the resistors stand on end. None of the components will tolerate overheating — especially the crystal — and great care must be exercised here. It should be remembered that once a device is soldered in place, its removal is made very difficult by the plating-through in the holes. A solder sucker is very useful for this eventuality. Sockets are particularly troublesome to remove and are usually destroyed by this operation.

Ensure electrolytics and diodes are inserted the right way round — the bar on the diode symbol normally appears on the component as a circle around the cylindrical body of the device.

The next operation is to insert and solder the keyboard switches, preferably from top right to bottom left. Each switch is labelled on the PCB, and the switch, complete with correct key-top may be inserted carefully in place. Do not use undue force as the switch is quite delicate until held in place and correct operation will certainly be impaired if the switch pins are pushed into the thermoplastic body by too great a strain or temperature.

The pins must be soldered with the switch pressed firmly against the PCB as any leaning to one side of the device will certainly cause adjacent keys to foul against each other. All switches except the one in the SHIFT LOCK position are sprung to return after being pressed — do not make the mistake of fitting the SHIFT LOCK switch elsewhere. This switch will stay down when pressed once and return on the second pressing. The SPACE bar is fitted last — after its switch has been soldered in place. The bar should be pushed in to the switch and the white plastic base located into its holes. The soldering iron should then be used to carefully heat the projections beneath the board and force them into mushrooms to hold the bar in place. The spring may need bending slightly if the action is too soft.

SPACE BAR ASSEMBLY



Before continuing, it is an excellent idea to check for shorts across the key-switch terminals and between Data Bus and Address Bus lines at IC8. This could save much time later on.

The regulator — with heat-sink in place — UHF modulator and large electrolytic, may now be soldered in. Solder flux should then be removed with meths using an old tooth-brush and some rag and the board fully inspected for solder bridges or broken tracks — a watch-maker's glass is invaluable for this task.

The power supply can be checked at this point to ensure it delivers five volts to each of the IC sockets.

Insertion of the IC's is a delicate process and pins are very easily bent between the chip and the socket — this is often quite undetectable and causes hours of fruitless searching for the source of failures. Pins should be bent straight from their normal splayed out condition and pushed bit by bit, inspecting carefully, into their sockets. If a pin does become bent, it may be straightened several times before being permanently damaged as IC pins are very maleable.

A final check of IC orientation should be made. If you are not using the full complement of memory, the two right most RAM sockets (IC31 & IC45) must be populated first and so on from right to left in "vertical" pairs.

Once all seems correct, connect up and switch on. Tune the TV to the computer somewhere around channel 36 and press both RESET keys simultaneously. D/C/W/M? should appear. Check that SHIFT LOCK is in the down position and press C.

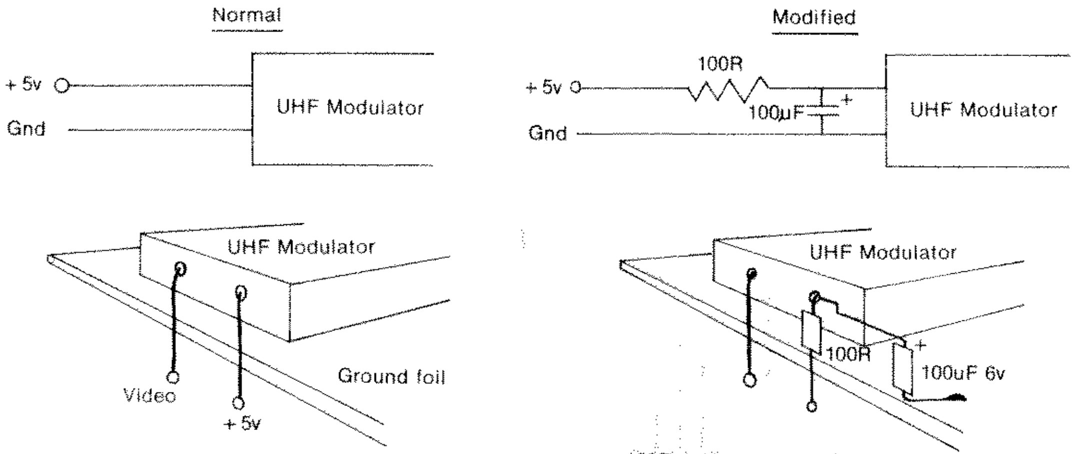
If this causes MEMORY SIZE? to appear, and pressing RETURN a couple of times gives the start up message on the screen, then you probably own one of the most advanced computers for its price ever made — congratulations.

NOTE

U numbers are the same as IC numbers on PCB component legend.

The kit of parts may not conform exactly with the stated values — LS devices may be replaced with ordinary TTL — IC2 and IC3, however, will be supplied as LS. Capacitor values and resistor values are normally exact. D1 - D10 are not critical as to type, but the correct type must be used for D15 (a 1Amp rectifying diode), and the PSU diodes which are large (3Amp) components.

1. On some TV's "patterning" of the screen may be noticed when the brilliance is too high. This may be removed by decoupling the UHF modulator in the following way:



100R colour code: BROWN BLACK BROWN

2. C 60 should be a 22pF capacitor but if the characters on the screen show a tendency to flicker and change then it should be increased slightly to 30 or 40pF.

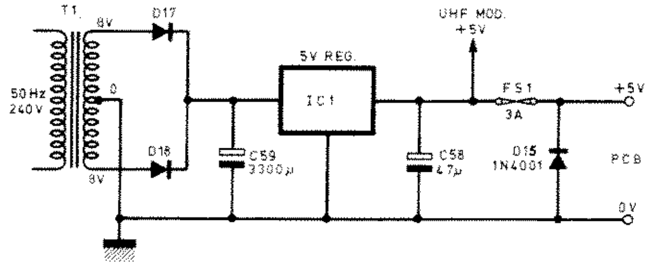
3. We are sure that you will have no trouble in constructing your **COMPUKIT**, but if you find that the machine does not work, even after considering the troubleshooting section, do not run the risk of destroying your board. For £25, **COMP COMPONENTS** will repair the system and send it back in perfect working order (postage inc.).

Address
COMP COMPONENTS,
14 STATION ROAD,
NEW BARNET,
HERTS.

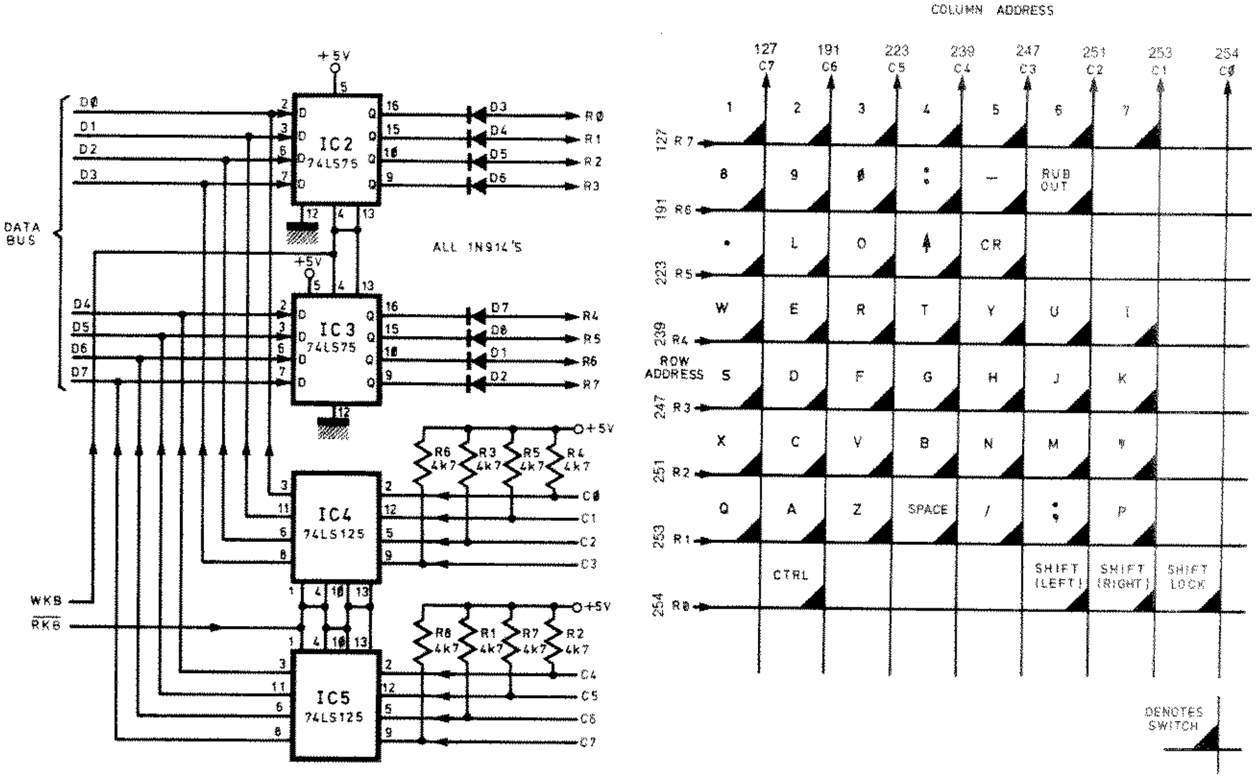
If you require advice of a technical nature:
ring MR. DAVID FINE, MILTON KEYNES (0908) 315 335.

4. The 8,0,8 of the transformer connect to B,A,C — labelled at the top of the board.

5. Make sure you insert the BASIC ROMS in their correct positions. BASIC 1 is nearest to the 6502 MPU, then numbers 2,3,4 from right to left followed by the MONITOR ROM and the ACIA respectively. (Refer to the component positioning diagram on page 5.)

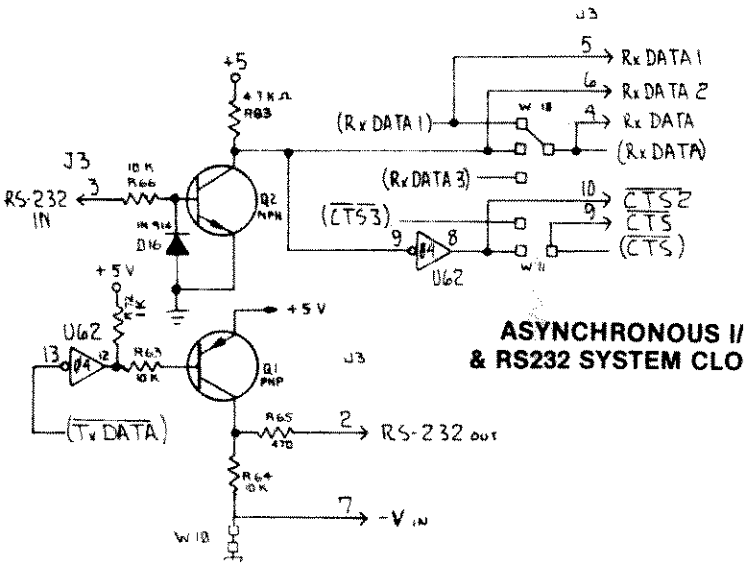


PSU



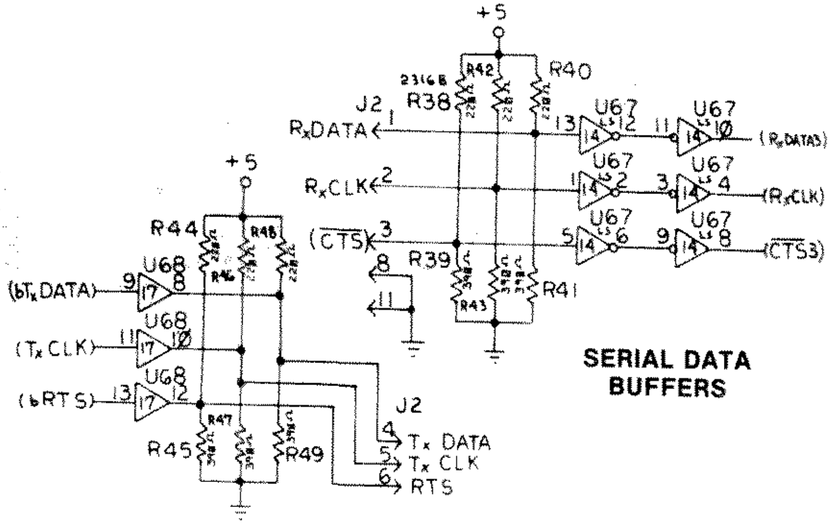
	127 C7	191 C6	223 C5	239 C4	247 C3	251 C2	253 C1	254 CØ
127 R7	1	2	3	4	5	6	7	
191 R6	8	9	Ø	:	—	RUB OUT		
223 R5	•	L	O	↑	CR			
239 R4	W	E	R	T	Y	U	I	
247 R3	S	D	F	G	H	J	K	
251 R2	X	C	V	B	N	M	,	
253 R1	Q	A	Z	SPACE	/	;	P	
254 RØ		CTRL				SHIFT (LEFT)	SHIFT (RIGHT)	SHIFT LOCK

KEYBOARD



ASYNCHRONOUS I/O & RS232 SYSTEM CLOCKS

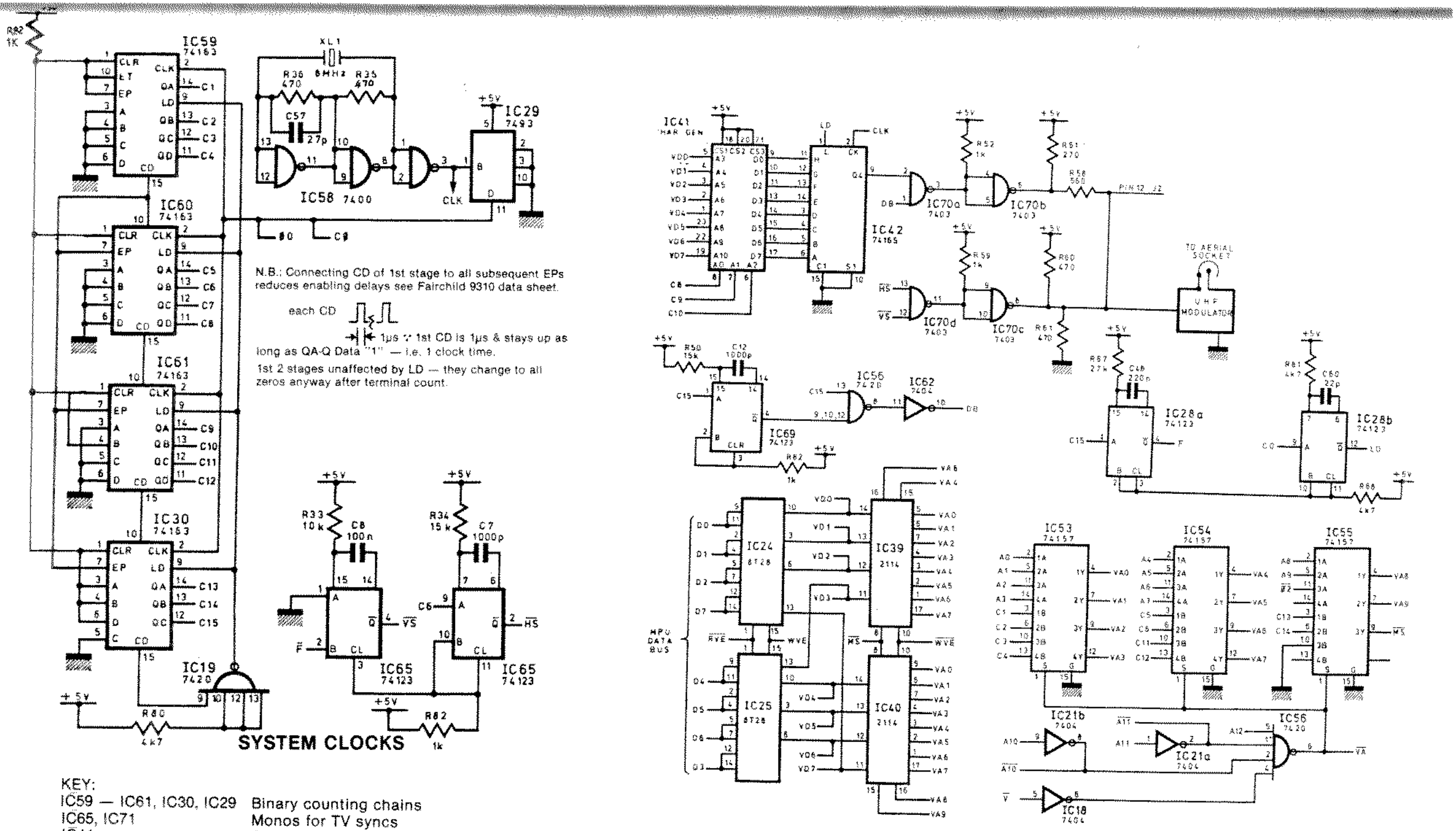
Components not present on basic board
except for Q1, R72, R63 — 65



SERIAL DATA BUFFERS

Components not present on basic board.

GENERAL SERIAL INTERFACE



N.B.: Connecting CD of 1st stage to all subsequent EPs reduces enabling delays see Fairchild 9310 data sheet.

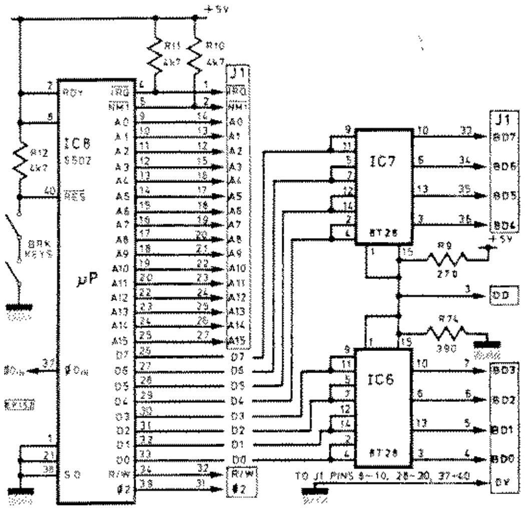
each CD

1μs ∵ 1st CD is 1μs & stays up as long as QA-Q Data "1" — i.e. 1 clock time.

1st 2 stages unaffected by LD — they change to all zeros anyway after terminal count.

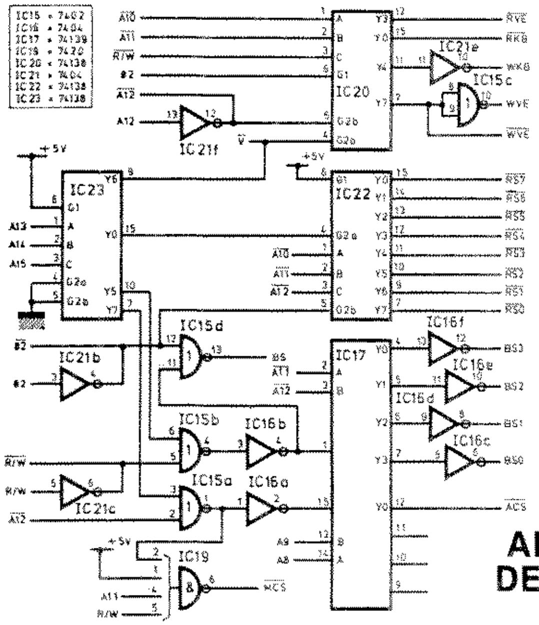
KEY:	
IC59 — IC61, IC30, IC29	Binary counting chains
IC65, IC71	Monos for TV syncs
IC41	Character Generator
IC42	PISO
IC39, 40	VDU RAM
IC24, 25	VDU Buffers
IC53 - 55	Address switches

BINARY COUNTERS AND VDU



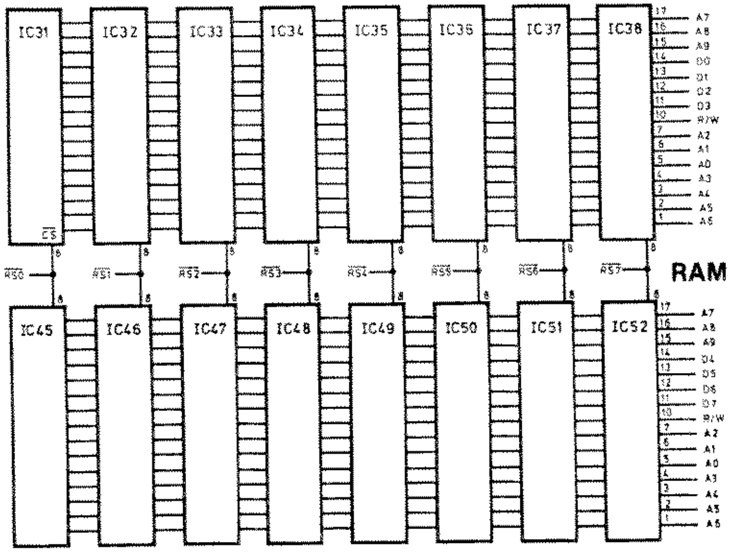
PROCESSOR & EXPANSION SOCKET J1

IC23 _ 8k blocks

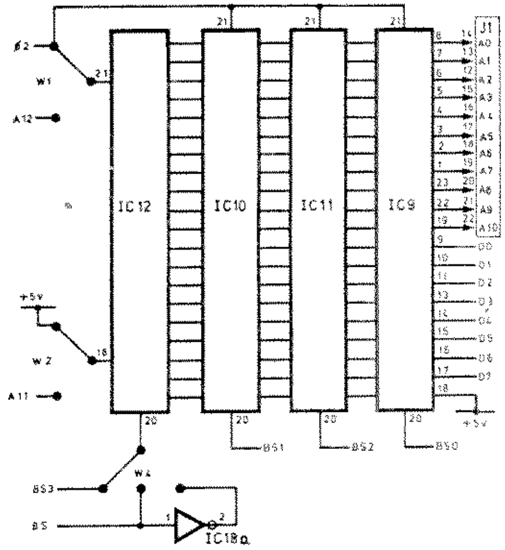


IC15	= 7402
IC16	= 7404
IC17	= 74139
IC18	= 7420
IC20	= 74138
IC21	= 7404
IC22	= 74138
IC23	= 74138

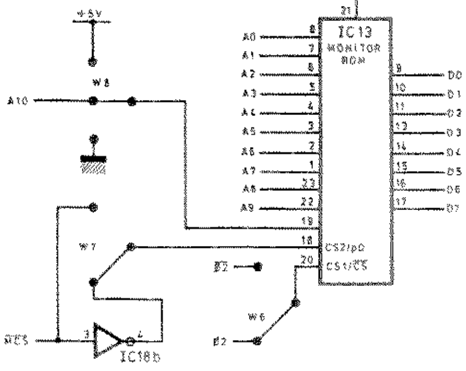
ADDRESS DECODING



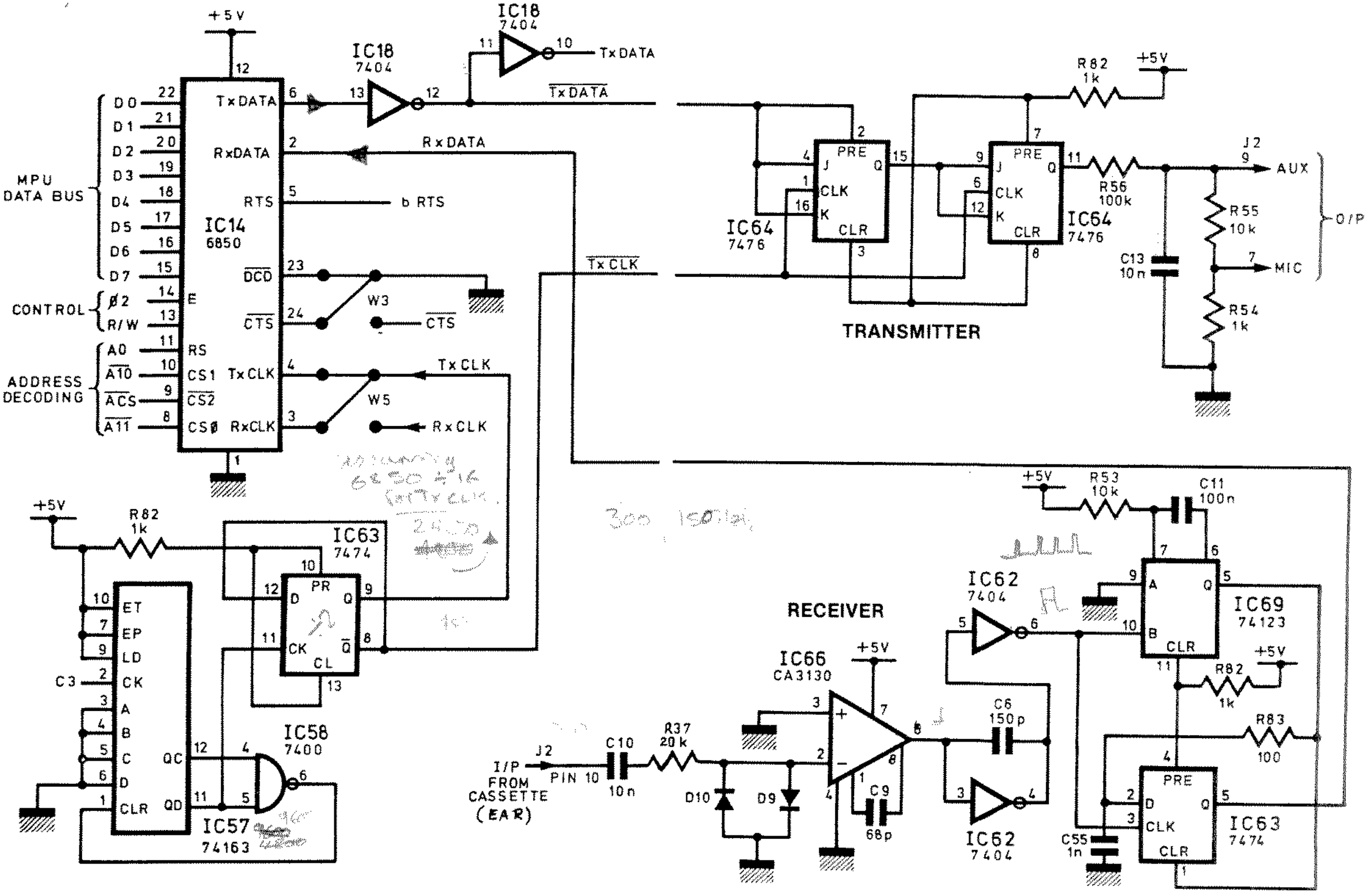
RAM



MONITOR ROM



ROM's & RAM's



CASSETTE INTERFACE

Troubleshooting

The process is best assimilated while the reader's mind is fresh from the hardware description. There are several categories of malfunction which may arise, and only one or two of a very definite nature are able to be mentioned here. The tools necessary for troubleshooting are an oscilloscope and a continuity tester. The latter may be all that is needed but a 'scope considerably speeds the process.

Other useful items include a bottle of methalated spirits and a watch-maker's glass to clean away flux and inspect the PCB for obvious solder bridges etc. The continuity tester is also generally used to detect shorts across tracks as well as to check the correct connection of components through the system.

The following assumes that you have checked the five volt supply and all the external connections and that the SHIFT LOCK key is in the down (or locked) position.

If the following procedures are ineffective, the unit should be returned to the suppliers (COMP COMPONENTS, 14 Station Road, New Barnet, Herts.) who have a standard charge for repair.

(a) UHF Modulator

This is detected by switching on and tuning the TV through the complete range, particularly near to channel 36, and finding no change throughout the band. A short band of blank screen should be detected near to channel 36. Check supply at modulator and connections, including ground connection to metal case. Scope the video input to the modulator — the waveform should be negative-going pulses 64 us apart with some fast spikes (positive-going) in between. If this is present, then the extremely rare occurrence of a faulty UHF modulator must be suspected.

(b) No Video Information At Modulator

Scoping through starts at the output of U58 (pin3) to detect the 8MHz clock. Work through the counter chain including U29 to check on oscillation of counters. If this is absent, the sequential nature of the chain will allow you to narrow down the point of failure quite closely. The most common fault is a solder bridge or bent IC pin.

If this cannot be detected by eye, the continuity tester must be used to check that all pins go to the right place and nowhere else! A chip must be suspected of failure only as the very last resort. Even then, try a chip from lower down the chain or elsewhere on the board in its place, if possible.

Once the chain is oscillating, failure may then be due to the area of the 74123 monostables U65, U28, U69 — again check through from the counting chains. HS should be negative-going pulses at 64 us separation (Horizontal sync), and VS at 20mS (Vertical sync). Pin 9 of U42 should be pure video information in short closely packed spikes.

(c) VDU ok — No Reset

If the two break keys, pressed simultaneously, do not produce D/C/W/M? on the screen then it is possible that the 6502, U8, is not receiving its clock (pin 37 at 1MHz) or its RESET (pin 40) — check both with scope.

The most likely cause, however, is almost always a simple bridge connecting a couple of Data Bus lines or Address Bus lines together. Check for any shorts between the pins of U8.

All the Data and Address lines should be oscillating and should all be affected by pressing the two RESET keys. If not, check the relevant lines through from start to finish for shorts and lack of continuity.

If D/C/W/M? appears but pressing C has no effect, you have almost certainly forgotten to lock the SHIFT LOCK in the "down" position — this must be checked every time a fault condition arises. If this is not the answer, check that none of the keyboard switches are permanently shorted and check that the C key is working electrically. R0 - R7 on the keyboard should be receiving a square-wave signal.

(d) Cassette Interface Not Receiving

The scope may be used to ensure that a sine-wave is present at the capacitor C10 and a square wave at pin 10 of U69. The waveforms described for the cassette interface may then be checked through. The ACIA should be checked for clock information.

(e) Transmitting

Checking this side is confined to looking for a signal at the MIC and AUX outputs and then working back through the system.

(f) Adjustments to the VDU

A certain amount of adjustment of picture density is possible on R58 if required for contrast. Adjustment of the time-constant of U71 by the capacitor C48 and resistor R67 will move the picture up or down.

Initial use of the machine

NOTE — Check that the **SHIFT LOCK** key is in the "down" position. This should always form the first check if the computer seems inoperative at any time.

When the machine is built and thoroughly checked, the power should be applied, shift-lock locked down and the two RESET keys pressed simultaneously — if all is well, the following will appear in the lower left hand corner of the screen:

```
D/C/W/M?
```

This is a question requiring the user to reply via the keyboard with one of the four letters requested.

D is for disc operation and is not covered in this manual. Now press M — this is for the machine code monitor — six characters will appear near the middle of the screen — four for address and two for data (both in HEX).

This is explained in a later section and the user should now press the two RESET keys again to restore D/C/W/M?

C & W are for COLD START and WARM START respectively and have the following meanings. If a program has been written and stored and is, say, in the operation of being executed, the user may RESET at any time. D/C/W/M? appears and pressing W (warm start) will revert the machine to its BASIC function without clearing memory of the current BASIC program. C, (cold start) on the other hand restarts the computer "from the top" and should now be pressed by the reader.

The words

```
MEMORY SIZE?
```

should have appeared — if not, check shift lock — if there is no success, switch off and check the PCB very thoroughly — especially around the ROMS. Typing any number after MEMORY SIZE? defines the number of bytes which may be used by BASIC from the start of RAM. The rest of the RAM is thus protected from being overwritten, and may be used to store data and machine-code blocks — accessable by PEEK, POKE, and the USR function defined later. Pressing RETURN, "defaults" to the full memory for BASIC — this is jargon for saying that the computer automatically assumes you would have typed a number of bytes equal to the total memory available. From now on, all entries from the keyboard must end with a RETURN — the computer will not look at any information until you press RETURN — this gives you time to change your mind about things and delete unwanted entries before the computer acts on them.

The words

```
TERMINAL WIDTH?
```

should have appeared now, and you are being asked to supply a number which will decide how far across the screen information is to be printed before a new line is started.

Pressing RETURN defaults to 48, but not all of these will appear on a normal T.V. screen. Try typing 46 followed by return, this will fit comfortably on most T.V.'s. At this point, the **COMPUKIT** does a complete scan of its RANDOM ACCESS MEMORY to determine how many bytes are free for writing in BASIC — this memory test can be used to determine whether the memory chips are working correctly — 3327bytes should be free in the 4K system and 7423 in the 8K system. The latter is given by the message

```
7423 BYTES FREE
```

followed by

```
COMPUKIT UK101
Personal Computer
8K Basic Copyright 1979
OK
■
```

Well done — you now own a powerful and versatile personal computer. With a little study of BASIC you will be able to pursuade it to perform almost any activity for which you are able to write down a logical set of steps.

The "OK" is to tell you that the computer is in BASIC and expects you to start programming. The "■" character is a CURSOR which tells you where on the screen your next keyboard entry will appear — try it!

Technical Specifications of Basic

VARIABLES

(a) **Numeric and string variable names:**
any number of alphanumeric characters starting with a letter and containing no BASIC words. Only the first two characters are relevant. String variables end in a $ e.g. AI$. 10's standard form is denoted by "E"
e.g. 3.2 x 10^3 denoted by 3.2 E3 and 6.8 x 10^{-4} by 6.8E – 4

(b) **Range and accuracy**
6½ digits accuracy for numerics between 10^{-38} and 10^{+38} automatically using 10's standard form when required. Strings are from 0 - 255 characters in length.

(c) **Arrays**
both string and numeric of any dimension and subscript range not causing an overflow.

COMMANDS

CONT, LIST, NEW, NULL, RUN, SAVE, LOAD

STATEMENTS

CLEAR, DATA, DEF FN, DIM, END, FOR...NEXT, GOTO, GOSUB...RETURN,
IF...GOTO, IF...THEN, INPUT, LET, ON...GOTO,
ON...GOSUB, PRINT, READ, REM, RESTORE, STOP, WAIT, POKE I, J.

RELATIONS, OPERATIONS and FUNCTIONS

–, +, *, /, 1, NOT, AND, OR, < , > , < = , > = , =, ABS(X),
ATN(X), COS(X), EXP(X), FRE(X), INT(X), LOG(X), PEEK(I),
POS(I), RND(X), SGN(X), SIN(X), SPC(I), SQR(X), TAB(I), TAN(X), USR(I).

STRING FUNCTIONS

ASC(X$), CHR$(I), FRE(X$), LEFT$(X$,I), LEN(X$), MID$(X$,I,J),
RIGHT$ (X$, I), STR$(X), VAL(X$).

ABBREVIATIONS AND SPECIAL CHARACTERS

(a) LET and END are optional,

(b) ? may replace "PRINT",

(c) : may be used to separate BASIC statements on the same program line,

(d) SHIFT P (before return) erases current line being typed,

(e) RUB OUTs erase previous character(s),

(f) CONTROL C interrupts program execution and returns to command mode. CONT returns to program execution,

(g) NEXT may be used without mentioning the stepped variable — even in nesting if the number of NEXT's equals the number of FOR's,

(h) No spaces are necessary.

General commands and use of BASIC

After the words:

OK

■

appear, the machine is said to be in the COMMAND MODE. At this point, two types of data may be entered, **always terminated by pressing RETURN** :

(i) COMMANDS

(ii) BASIC Statements

These are described below:

N.B.
Spaces are always ignored in Commands and BASIC Statements except in literals and string arguments.

(i) COMMANDS

Clear This causes all variables (numeric or string) to be set to zero (or null)

List This can be used in several forms as detailed below:

List Causes the whole stored BASIC program to be listed line by line until either the listing is complete or CONTROL C is pressed.

List n (For any whole number n) will list that line only

List n- will list all lines from n to the end of the program

List -n Will list all lines from the beginning of the program to line n

List n-m Will list from line n to line m

This allows any part of a program to be viewed at will.

NULL n Inserts n null before sending data to serial I/O devices

RUN Starts program execution from the first line with all variables cleared.

RUN n As above but starts program at line n.

NEW Wipes out current program.

CONT Continues execution of program after Control C, or after a STOP statement encountered within the program.

LOAD
SAVE Cassette commands dealt with elsewhere.

CONTROL C
This is effected by pressing the "CTRL" key and (with CTRL pressed) typing a "C". It suspends Computer activity and prints a message to give the line-number at which the break occurs.

The Computer then returns to COMMAND MODE. Many BASIC Statements may also be used as commands if unaccompanied by a line number — for instance:

GOTO n

would cause the Computer to begin executing from line number n without clearing all the variables. Similarly, many of the above may be used in programs — thereby causing a program to command the machine.

(ii) BASIC Statements
There are two modes of use of the BASIC language when using an interpreter, such as that employed on the **COMPUKIT**. These will be called:

(a) Immediate Mode
(b) File Mode.

(a) IMMEDIATE MODE
If a BASIC Statement is typed while in the Command mode, it is executed immediately a RETURN is encountered. It is not filed for later use but is lost after execution. This can be very useful. In this mode the BASIC language available on the **COMPUKIT** with its fast powerful floating-point calculation ability is able to act as a super calculator. For instance, answers to such calculations as:

SIN (0.781)

$$X = \frac{15.7 \times 13}{87 \times 10^4}$$

are found and will be displayed on the screen immediately. In this case, the user should type:

```
PRINT    15.7*13↑SIN (0.781)/87E4
```

after pressing RETURN the answer.

```
1.09796E — 04
```

will be displayed considerably more quickly than by the majority of electronic calculators on the market! In addition, of course, the Computer may be programmed to perform this or any other calculation many times with different values each time.

The immediate-mode use of the machine allows, for instance, instantaneous indication of remaining program space by typing:

```
PRINT FRE (N)
```

The answer (after RESET on an 8K RAM machine) will be 7420. (The FRE function is described later).

An important use of this mode is for program debugging. The final states of all the variables are retained when a program ends or is stopped. These states may be viewed by typing:

```
PRINT A, B, C etc.
```

where A, B, C are the variables whose values are required. Quite complex immediate-mode programming may be written by employing colons to separate the various statements. In order to write and retain a BASIC program, the File Mode must be employed and this is described next.

(b) FILE MODE

To signify to the Computer that it is to retain a program line for later execution, a line number must be typed before the line itself. This line number identifies the program line uniquely to the user and to the program. The line numbers may be looked upon as labels.

The concept of a program line must not be confused with a display line. A VDU display line may only contain up to 48 characters — a program line may occupy several VDU lines. The Computer accepts a maximum of 71 characters on a program line and depending upon the Terminal width set up after a system reset may occupy up to around four and a half VDU lines (if terminal width is 16).

For instance, the program

```
10 PRINT "HELLO"
20 X=3.6*4.8
30 PRINT "X";X
```

contains three program lines and three program statements — the first (labelled 10) Commands the VDU to display the word "Hello". The second to calculate a value for X and the third to print it.

The program may be run by pressing RUN (followed by RETURN as always) Try it!

The central point about File Mode is that the program is retained after execution — as are all the variable values — try typing:

```
PRINT X (RETURN)
```

in immediate mode, and then RUN again.

ERROR CODES

If during the execution of a program, the computer encounters a word it does not understand, or if it is asked to perform an impossible calculation, it may detect an error of a type which it is able to recognise. If this is so, it can inform the user of the type of error encountered. Some errors are undetectable and simply produce answers which are wrong or even more bizarre behaviour of an apparently random nature. If it does recognise a standard error, however, it will print up one of the standard error codes listed in the table at the end of this manual. Each error code consists of a letter which reminds the user of the type of error found plus a graphic character which makes that error coding unique.

This type of self checking activity makes computer programs considerably easier to debug as some of the work is done by the machine itself. There are pitfalls, nevertheless, and sometimes, though an error of a particular type has been flagged, it may be the consequence of a much subtler error elsewhere in the program — only experience can help under these circumstances.

EDITING

The program may be edited by writing further lines or rewriting existing ones.

For instance, typing the following to add to the last program:

```
15 PRINT "BYE"
```

will insert the new statement in its ordered position by line-no (i.e. between lines 10 and 20). Try typing LIST.

Similarly, typing,

```
10 PRINT "THIS";
```

followed by RETURN will simply wipe line 10 out altogether, now LIST.

If a mistake is made in typing a character, it may be deleted by pressing the RUB OUT key. If the RUB OUT key is pressed several times, that number of previous characters is deleted.

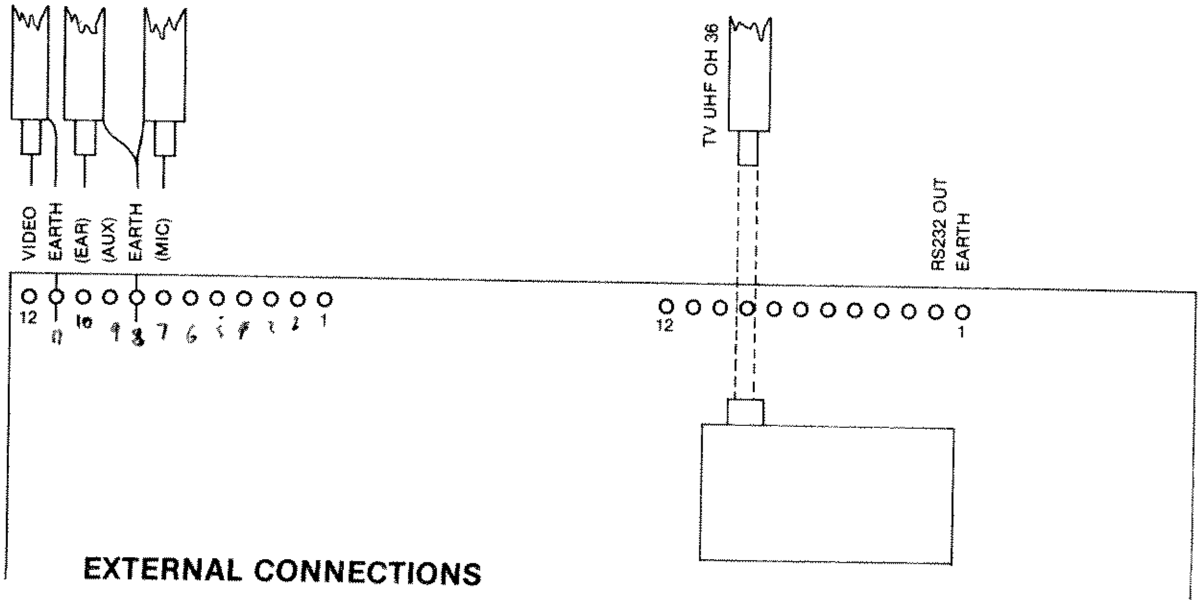
Try typing:

```
PRINT "HELLL (RUB OUT) 0"
```

The third L will be deleted and the VDU will show:

```
HELLO
```

The entire current line being typed may be deleted (before RETURN is pressed only) by pressing (SHIFT) P which displays an @ sign and places the cursor on the next line to await further instructions.



EXTERNAL CONNECTIONS

Cassette use

Check that pin 10 of J2 is connected to the earphone output and pin 9 or 7 to auxiliary input or microphone respectively, and pin 8 and/or 11 to the Earth of the cassette machine. Any ordinary cassette recorder should be suitable, but some care must be taken in selecting cassette tapes — the very cheapest are prone to giving continual errors. The best volume control is found by trial and error for playback, and one should start a little above the middle position and experiment.

Recording may be done by automatic level or manual (if available) — some experimentation will be necessary for the latter. A machine having a tape counter is of considerable assistance in accessing cassette-stored information, but not essential.

The following gives a set of steps to be followed for play-back and recording respectively.

PLAYING BACK A PROGRAM

(a) Rewind tape to "leader" or blank area of tape before program starts.

(b) Place computer in (BASIC) Command mode and type NEW (and RETURN).

(c) Type LOAD but do not press RETURN. When RETURN is pressed, control is passed over to the cassette and any spurious "noise" encountered will be interpreted as data and loaded to the computer.

(d) Turn on the recorder to PLAY the tape.

(e) Wait for a second or two — or for the "leader" to pass through — and press RETURN.

Some noise characters may be printed on the screen — if one of them is a number it may be interpreted as a line-number and appear in the final program. After a short time the program begins to be printed on the screen as if LIST had been typed — but more slowly. This allows its progress to be watched closely. In particular, if the program begins with REM statements, it can be positively identified from the screen. When play-back is complete, pressing SPACE and then RETURN returns to BASIC and the newly loaded program is resident for listing (type LIST and press RETURN) or running (type RUN and press RETURN) etc.

RECORDING A PROGRAM

This assumes the BASIC program is stored in the Computer ready for storage on cassette.

(a) Rewind tape to "Leader" or a blank noise-free portion of tape,

(b) Type SAVE (and press RETURN),

(c) Type LIST but do not press RETURN.

As soon as RETURN is pressed, the LIST function simply lists the program onto cassette — it is essential that the process begins only after the cassette machine has been turned on and allowed to settle down to a constant speed.

(d) Turn cassette on to RECORD and allow "leader" to pass plus a further 5 seconds — otherwise wait 10 or 12 seconds then press RETURN.

The program will list on the screen as it is being recorded

(e) When recording is complete, wait a few seconds, turn off tape recorder and type LOAD (and press RETURN),

Then press SPACE and RETURN.

N.B. Setting TERMINAL WIDTH manually after RESET (instead of pressing RETURN) will upset cassette recording. To prevent errors, execute the statement:

```
POKE 15, 72
```

in immediate mode before saving your program.

Address "15" is also worth remembering for resetting terminal width at any time without having to RESET.

BASIC reference and definitions

Introduction

Any Computer language is used to formalize a logical set of steps into a form suitable for execution on a machine. The machine's understanding is normally limited to a Grammar composed of a few statements and some variables. It is at present inappropriate, for instance, to expect a Computer to understand a Command to perform a calculation of the Mean of a set of a numbers unless it has been given a program (a logical set of steps) to explain, in its own language, the minute details of how to attain the end result.

The calculation must be broken down into "INPUT" steps "CALCULATION" steps and "OUTPUT" steps.

The following describes the Grammar and structure of the fast and powerful form of BASIC available to the user of the **COMPUKIT UK101**. The beginner should read this in conjunction with a suitable BASIC primer.

VARIABLES AND TYPES

NUMERIC variables may be one or two alphanumeric characters in length — the first must be alphabetic. Longer variable names will be identified by the first two characters only e.g. HELLO, HE, HE123XY are all indistuiguishable to the machine. Basic words (such as NEW,SIN etc.) may not be used in variables nor may non-alphanumeric characters e.g.

LEGAL	**ILLEGAL**
A	1B
B1	B*
B 175	TOP
TQ	COSQR3
EGG	18Z
TUESDAY 3	AND 2
	TELETYPE 3

check you understand why the right hand column members are all illegal. Clue: look for embedded BASIC words.

Spaces are irrelevant and the second and third members of the "LEGAL" column are indistinguishable. If you are worried about the validity of a variable, try giving it a value in immediate mode, for instance type:

```
BI = 3
```

(this type of statement is technically termed an ASSIGNMENT statement).

This will be accepted whereas:

```
BI*=3
```

will not.

If a variable is accepted, try printing it out again. For instance:

```
1B = 7
```

appears to be accepted but follow it with PRINT 1B and you will see that the answer is far from 7, then type LIST to see why.

The above applies to STRING variables too, except that each such variable must end with a $ sign. e.g. A1 is a numeric variable and has a floating-point value. A1$ is a STRING variable and its "value" is a string of characters of any type including graphic characters.

Some very powerful string manipulation functions are included in the **COMPUKIT** BASIC and these are described later.

RANGE AND ACCURACY

Numeric variables are allowed to take on values between 10^{-38} and 10^{+38} (approximately) and have 6½ figures of accuracy (i.e. 6 figures displayed, and one extra "guarding").

Strings may be from 0 to 255 characters in length.

ARRAYS

Arrays are available for both numeric and String variables to any dimension which does not cause an overflow to be registered — this depends upon the range of each dimension's subscript. A little experimentation is worthwhile if Arrays are to be used extensively.

STATEMENTS

The BASIC language is written as a set of statements — several of these may appear on the same program line (up to a maximum of 71 characters on any such line).

Statements on a program line are separated by ":" and spaces may be omitted.

e.g.

```
10 X=13*14.6
20 PRINT X
```

may be written as:

10X=13*14.6:PRINTX or even 10X = 13*14.6 : ?X

This format has the advantage of saving memory, space and time while producing program code which is harder to modify and edit.

BASIC OPERATORS

(a) – this is the usual "minus" sign and may be used for subtraction or negation e.g. A=B–C or D= –E

(b) + Addition

(c) * Multiplication sign

(d) / Division sign

(e) ↑ Raise to a power (exponentiation)
e.g. X^3 is written as X ↑ 3 and $\sqrt[3]{X}$ is written as X ↑ (1/3)

(f) = may be used in assignments : A=3 (or optionally, LET A=3)
B= K+I etc. The equals sign is also used in Boolean relations with values true or false. Its use is illustrated as follows:

A=3 is a statement whose truth, or otherwise, may cause an action to take place. e.g. IF A=3 THEN GOTO 30.

This last use of "=" is similar to the use of the next five relations:

(g) > Greater than
(h) < less than
(i) <> or >< not equal to
(j) < = or = < less than or equal
(k) > = or = > greater than or equal
(l) AND This Boolean operator combines logical statements, and with the next two may be used to form complex logic expressions with the value True or False
(m) OR
(n) NOT

BOOLEAN EXPRESSIONS

Boolean (or logical) expressions using the above are given a numerical value by the **COMPUKIT's** BASIC. A true statement is given the value "–1", a false statement has value Zero.

Thus : K=(A=3 AND A=4) gives K a zero value since the expression (in brackets) set equal to K is false. Similarly:

K= (A= (A+A)/2) will give the value –1 or "TRUE",
this is a numerical value and may be used as such. For instance:

PRINT (A= (A+A)/2) *6
will print the number –6 on the VDU.

In addition :AND,OR,NOT may be used in BIT manipulation mode for Boolean operations of 16 BIT 2's complement numbers from –32768 to 32767.

e.g.

```
63 AND 16 =16
–1 AND 8 =8
4 OR 2 =6
10 OR 10 =10
NOT 0 = –1
NOT 1 = –2 etc.
```

OPERATOR EVALUATION ORDER

Expressions are evaluated with the following Precedence order BRACKETS first, then (in order):

(1) ↑
(2) negation
(3) */ from left to right
(4) + − from left to right
(5) =, <> ,<,>,<=,>= from left to right
(6) NOT
(7) AND
(8) OR

Two separate numbers or variables may not stand next to each other similarly two operators unless the **second** is +or−

e.g. (i) A+ −6 is equivalent to A−6 and A− +6
(ii) A* −5= −5*A but A− *5 is illegal
(iii) 3↑2 *7+5/10*2 will be calculated as follows: 3↑2=9 first, then 9*7=63,5/10=½,½*2=1 in that order; then, finally, 63+1 giving 64 as the result. To change this order, brackets must be used.

DEFINITIONS OF BASIC STATEMENTS

In the following:
V and W are numeric variables,
X, Y and Z are numeric expressions which may contain numeric and Boolean operators or functions.
B is a Boolean expression.
I and J are truncated integers,
$ denotes a string variable.

READ ... DATA. DATA statements contain lists of data for READ statements in strict order of use — strings are included. e.g.:

```
100 READ V, W$
200 DATA 1, "HELLO", 2, "BYE"
```

Each time the READ statement is executed, a pair of data is read into the variables V and W$, in order, until the data is exhausted. The data types must match up with the READ variables.

RESTORE restores the data pointed to the start of the data list for reuse by a READ statement.

DEF FN This is a user-defined function of one argument used as follows:
DEFFNA(V)=3*V↑2 defines a function FNA(V)
e.g. W=FNA(3) gives W the value 27

The argument may also be a numeric-valued expression.

DIM is used to allocate space for arrays and set all array variables to zero.

e.g. DIM V (12,12,2) allocates a 3 dimensional numeric array with first two subscripts from 0 to 12 and third from 0 to 2; similarly, DIM V$ (12,12,2) allocates a string array of the same size.

Not dimensioning cause a default to 10 for one and two dimensional arrays.

The same array name may not be used for arrays of different dimensions.

END Terminates program (optional) useful in statements such as IF A=3 THEN END

FOR V=X TO Y STEP Z ... NEXT V

This "FOR-loop" executes all program statements down to NEXT V for all the values of V from V=X to V=Y in steps of V's value equal to that of Z. The program statements may include further "nested" FOR-loops. NEXT V may be abbreviated to NEXT. If two FOR-loops are nested and each terminates at the same NEXT, this may be written NEXT V,W.

e.g.

```
10 FOR I=1 TO 10 STEP 2
20 FOR J=2 to −3 STEP −0.1
30 PRINT I*J
40 NEXT J, I
```

Note that the NEXT statement names the variables in the order J, I, i.e. "inner" variable first. Line 40 may also be written : 40 NEXT:NEXT

Note also that omitting "STEP Z" in a FOR-loop defaults the step value to 1

The FOR statement uses those values of the expressions X, Y, and Z which are encountered upon first entering the FOR loop. Thus X, Y, and Z may be used and changed within the FOR loop without affecting its operation.

GOTO I Forces execution to jump to line I. I may only be a positive number — non integers are truncated towards zero.

GOSUB I...RETURN
This causes execution of a subroutine starting at line I (a positive number as for GOTO) and terminating in a RETURN statement which forces execution back to the line following GOSUB I. Subroutines may be nested.

IF B THEN P
P is a statement or set of program statements separated by colons which will be executed upon the expression B's having a "TRUE" value. Strictly speaking B is a Boolean expression such as
A = 3 AND C = 5.8 OR T > = Q ↑ 2

However as has been pointed out, this is assigned a numeric value as follows:

If "TRUE" then − 1
If "FALSE" then 0.

This B may be any numeric expression and if its value is 0 it will be taken to have the value "FALSE". − 1 is normally taken to be "TRUE", but here any non zero value for B will have this affect.

e.g. If A ↑ 2 THEN PRINT "NON ZERO" will print "NON ZERO" whenever A ↑ 2 is non zero.
Similarly for:
IF B GOTO (line number) and IF B THEN (line number).

ON I GOTO L,M,N etc.
The technical term for this statement is the "Computed GOTO". The line no. L, M or N etc., chosen by the GOTO statement, depends upon the value of the (truncated integer) expression I.

If I = 1 (after truncation) GOTO L is executed, if I = 2 then M is chosen etc.

Negative values of I give an error message, whereas larger values of I than the number of members in the line-number list, will cause the next line after the computed GOTO statement to be executed.

REM — all characters after REM are disregarded by BASIC and this space is available for comments (REMARKS)

STOP causes execution to cease at that line and print out the line-number. The program may be restarted by CONT, after variable values have been printed if desired.

PRINT (list)
This causes output to the VDU depending upon the members of the print list as follows:
PRINT 3 causes the number 3 to appear as with any other number in the list.
PRINT X will cause X's value or contents to be printed where X is any numeric, Boolean or string variable or expression. e.g.

(a) PRINT A = (A + A)/2
will cause − 1 to appear

(b) PRINT 3   2 + 2
will cause 11 to appear

(c) PRINT X$
will cause the contents of the string variable X$ to be printed

(d) PRINT XY will cause the combined (concatenated) contents of X$ and Y$ to be printed.
e.g. Try:
X$ = "WE":Y$ = "LL": PRINT XY
in immediate mode.

Messages may be printed literally by using "(termed LITERALS),

e.g.
PRINT "HELLO" will cause HELLO to appear.

Any combination of print list members may be included in a list separated by COMMAS or SEMI-COLONS. COMMAS cause the members to be printed in columns beginning fourteen spaces apart. SEMI-COLONS cause printing in adjacent positions.

e.g.
PRINT 3,4;7 will give:

```
3             47
```

as output.

If a PRINT list is terminated with a comma or semi-colon, the next print statement will continue where the last terminated. The cursor (▪) always indicates the next PRINT position.

e.g.

```
10 PRINT 4,6,
20 GOTO 30
30 GOTO 20
```

Causes

```
4        6        ■
```

to be output before the infinite loop is entered.

PRINT with an empty list causes the Cursor to move to the start of a new line.

e.g.

```
PRINT: PRINT:PRINT
```

causes three "New Lines".

The cursor position is called the "Print Head", and it is that screen position at which the next PRINT statement will begin its output. SPC (I) and TAB (I) may also be included in a print list where I is a positive truncated integer expression. SPC (I) prints I spaces and places the print head I places ahead of its former position. TAB (I) merely moves the print head I places without overwriting existing material. POS (I) gives the current position, on the line, of the Print Head.

INPUT list This statement allows the user to input data to a program during its execution. The INPUT list may be started with a message in literals followed by a semi-colon then a list of variables whose values are to be input separated by commas. The list may include numeric or string variables. When an INPUT statement is executed, the display shows any initial message first, followed by a "?" symbol. This asks for the first piece of data. If further data is required, these may be typed in separated by commas on the same line, or by RETURN's on subsequent lines.

e.g.

```
10 INPUT "INPUT A,B,C$,D";A,B,C$,D will cause:
INPUT A,B,C$,D?
```

to be output.

The reply from the user may be, for instance,

```
10,15.6,HELLO, -6E51
```

or

```
10
?? 15.6
?? HELLO
?? -6E51
```

(the computer prints ?? to ask for further data)

A,B,C$ and D are all assigned values as indicated. Literals may be omitted when presenting string data in this way unless the data contains commas.

Care should be taken to ensure that the data presented is of the correct type for each of the input list members.

If too much data is presented, a message saying "EXTRA IGNORED" will appear.

If RETURN is pressed on an empty piece of data, the program is aborted and return is made to the command mode (this is a useful way of leaving a program whenever the INPUT stage has been reached).

If the wrong type of data is presented, the machine will ask the user to "RE DO" the INPUT from the start.

NUMERIC FUNCTIONS

(X is any numeric or Boolean expression)

ABS(X) For X> = 0 ABS (X) =X
For X<0 ABS (X) = -X

INT (X) ROUNDS down to nearest integer

e.g.

```
INT (8.1) = 8
INT (-3.3) = -4
```

RND (X)

gives a random number between 0 and 1.

Each time RND is executed with a non-zero argument, the random number generator advances to the next number. RND (0) will give the same number each time unless interspersed with a RND execution having a non-zero argument.

The expression (B-A)* RND (1)+A gives a random number between A and B.

SGN (X)
If X>0 SGN (X) = 1
If X< =0 SGN(X) = 0
SIN (X), COS(X), TAN(X), ATN(X)
are the usual trig. functions with all angles in radians.

SQR(X) = square root of X.

EXP(X) = e ↑ X where e = 2.71828

LOG(X) = log of X to base e

FRE(X) for any X this gives the number of remaining bytes of user work-space for BASIC programming.

(PRINT FRE (X) is very useful for use in immediate mode to determine the remaining memory space).

TAB (I), SPC (I) and POS (I) are described in the section on PRINT

PEEK (I) Returns a decimal number equal to the value of the contents of the memory location I, which is also decimal.

POKE I,J loads memory location
I (decimal) with value J (decimal).
In both of the above two functions,
0< = I< = 65535
0< = J< = 255
otherwise an error is indicated.

STRING FUNCTIONS:

X$ is any STRING EXPRESSION or VARIABLE

ASC (X$) This returns the ASCII value (in decimal) of the first character in the string
e.g. ASC ("AB")=65

CHR $ (I) equals the string character having ASCII value I.

e.g. PRINT CHR $ (65)
would give the character A.

LEFT$(X$,I) and RIGHT$(X$,I)
give a string composed of the left most and right most I characters of string X$ respectively.

MID$(X$,I,J) gives the string-subset of J characters of X$, starting at the Ith character. If J is omitted, all characters from Ith to end of string are given.

LEN(X$) Gives length of string in characters.

STR$(X) Converts a numeric expression into the string of characters representing its value.

e.g.
STR $ (−6.8) = "−6.8"
and
STR $ (1.3E29) = "1.3E+29"

VAL(X$) gives the numeric value corresponding to a string of digits. (This is the inverse of STR$)

STRING EXPRESSIONS AND OPERATIONS

Any of the above functions may act on a X$ composed of those functions and the operator "+".

e.g.
X$ = "HE" + "LLO"
gives X$ the value "HELLO".
+ is the operation of CONCATENATION.
Thus LEFT $ ("HE" + "LLO", 3) = "HEL"
etc.

Strings may be compared to produce Boolean functions — the ASCII values of their characters are used from left to right for the comparison.

e.g.
"HELLO" is "greater" than "ABC" because ASC ("H")> ASC("A"). In this way, a file of string records can be sorted alphabetically. Strings may also contain Graphic characters, and in this way complicated screens of varied patterns can be generated. Finally, by using the VAL(X$) and STR$(X) functions, numeric strings can be converted into numbers, acted upon by the normal rules of algebra and converted back into strings.

INPUT/OUTPUT (WAIT Statement)

WAIT I,J,K. This is used to send the computer into a wait state until the memory location I (decimal) takes on a certain value dependent upon J and K. WAIT takes the contents of location I, exclusive OR's it with K AND's with J and waits until the result is non-zero (omitting K defaults it to zero). Thus any bit of location I can be considered as providing a flag. This could be used, for instance, with a medium speed printer and allows fast servicing from BASIC of I/O devices connected into the system at specific memory locations. Other examples would be for the control of industrial equipment directly via BASIC.

CALLING MACHINE CODE ROUTINES

USR(I) This function is used to call machine code routines which may be useful purely due to their greater speed, or for their ability to service directly (and speedily) I/O devices occupying specific memory locations.

The USR function is called in BASIC by a statement such as:
X = USR(X)

This causes a jump to a machine code routine either in ROM or RAM. To access USR, the start of the routine must be poked into the (decimal) addresses 11 (low part in decimal) and 12 (high part in decimal). Executing X = USR (X) will automatically cause the machine code routine, which must be terminated with an RTS, to be executed. If the machine code program is to be started in RAM, a block must be protected against overwriting by BASIC. This is done by pressing the BREAK keys and answering MEMORY SIZE? with a number less than the total RAM available. This restricts BASIC to that number of bytes and leaves the remainder, at the "top" of memory, protected. Note that 770 is the minimum number allowed for memory size, but this does not allow space for any BASIC programs.

Even though only one USR function is provided, use of POKE on addresses 11 and 12, before each USR call, enables any number of routines to be executed, one at a time, during the running of a BASIC program. In addition, values stored in RAM locations may be passed back and forth between BASIC and the machine code programs by using PEEK and POKE.

The following provides an example of the application of USR to clear the screen and print up a message. Reference must be made to the Machine Code Monitor section later in the manual, as the routines are stored by the user. The example will work on the basic 4K RAM machine.

Break should be pressed and the answer 1024 given to the question MEMORY SIZE? This restricts the RAM space as follows (see memory map of machine) (all addresses below are in HEX).

0000 to 03FF (1023 Decimal)	BASIC WORKSPACE ETC.
0400 (1024 Decimal) to End of RAM	PROTECTED FOR MACHINE CODE ETC.

This quantity of protected RAM is not necessary for the following example, but it illustrates the point that the user is able to control this aspect as he or she wishes.

The machine Code Monitor may now be used to load the following three blocks of HEXADECIMAL number pairs starting at the address shown.

STARTING ADDRESS	DATA	COMMENTS
Hex: 0500 (Decimal equivalent is 1280)	A2 00 BD 00 06 C9 5F F0 07 9D E5 D1 E8 18 90 F2 60	This program stores a message in the VDU RAM (resident at D000-D3FF). The message is stored from 0600 onwards and terminated by 5F
Hex: 0600 (Decimal 1536)	43 4F 4D 50 55 4B 49 54 5F	Any set of ASCII character or graphic character codes may be placed here by the user for display: ending in 5F
Hex: 0700 (Decimal 1792)	A9 00 85 E1 A8 A9 D0 85 E2 A9 20 91 E1 C8 C0 00 D0 F9 A6 E2 E0 D3 F0 06 E8 86 E2 18 90 ED 60	This routine clears the VDU screen

To return to BASIC, RESET must be pressed. The message D/C/W/M? should be answered with W to conserve the above program. The following program gives an example of the use of the above.

```
10      PRINT" TO CLEAR SCREEN TYPE C"
20      PRINT
30      PRINT "TO DISPLAY MESSAGE TYPE M"
40      INPUT A$
50      IF A$= "C" THEN 100
60      IF A$= "M" THEN 200
70      GOTO 40
100     POKE 11,0: POKE 12,7:X=USR(X)
110     GOTO 40
200     POKE 11,0:POKE 12,5:X = USR(X)
210     GOTO 40
```

To leave the program press RETURN without C or M.

Note that in POKEing the address of the machine code routine into 11 and 12 the Hex address is split into low and high bytes and then separately converted into decimal and loaded into 11 and 12 respectively.

e.g. if the routine were to start at EA32 (Hex) the following holds.
low part: 32 (Hex) = 50 (decimal)
high part: EA (Hex) = 234 (decimal)
thus POKE 11,50 and POKE 12,234 are used.

To write messages other than that shown above, stored at 0600, the user may either use the machine code monitor to write in the Hex codes of the symbols to be displayed, ending in 5F; or a BASIC program may be written to POKE the ASCII values of any characters typed on the keyboard into that area of memory using the ASC function. Data blocks or machine code programs may also be written directly into the protected RAM space using the POKE, READ and DATA statements. Remember that to POKE a machine code routine into RAM from BASIC, the 6502 operation codes must be converted to decimal notation, unless you include a routine in your program to perform the conversion automatically.

MACHINE CODE MONITOR

The machine code monitor program provides a simple but adequate method of loading and running machine code routines — including loading from cassette. To prevent their being overwritten by BASIC, MEMORY SIZE? (After RESET) must be answered with a number restricting the BASIC's use of RAM. The number, n, thus typed restricts BASIC according to the following map.

ADDRESS IN DECIMAL	USE
0 255	Page Zero
256 768	Scratch-pad RAM used by BASIC and system monitor
769 n - I	BASIC workspace
n End of RAM	protected against use by BASIC

It is clear from the above that n must be at least greater than 769. In a 4K machine, the end of RAM occurs at memory location 4095, 8K finishes at 8191.

After RESET, the machine code monitor is entered by pressing M. The display:

```
0000 4C
```

then appears.

The first four characters form the address field, the second two data — all in HEXADECIMAL notation. Typing any HEX characters at this point will load the address field — the data field is kept constantly updated as the address changes. Mistakes may be corrected by typing further characters — these will continue to be loaded into the right hand position and then rotated left as further entries are made.

The following commands are available:

/ changes to data mode to allow data to be loaded — RETURN then opens the next location while still in data mode etc.

. changes back to address mode.

G (used after setting up an address with .) This jumps to the address showing on the screen and begins execution.

L transfers control to cassette — loading 00FB with 00 transfers control back to the keyboard.

After L, the monitor is in data mode and simply accepts all its commands from cassette instead of the keyboard. Thus the cassette tape must have a series of commands, stored as ASCII codes, to control the Monitor. To load a program from cassette, it must be stored byte by byte separated by RETURNS and ending with:

```
.00FB/00
```

This loads 00FB with 00 which is the flag to switch the monitor back to accepting commands from the keyboard. The program can be run from cassette, if desired, by ending with G after setting up the start of the routine in the address field.

There is no command that enables you to save a machine code program on tape. Shown on the next page is a small routine that will enable you to do just that:

SOME NOTES ABOUT THE MONITOR

The monitor was written with versatility in mind and if you consult the UK101 memory map you will see that great use is made of vectors.

A vector is an address that the monitor jumps to when performing various tasks. This address is held in RAM which means the user may alter these addresses and write his own input/output routines.

The following are the most important vectors:—
Address
0218-0219 — Contains the address to which the monitor jumps when inputting a character — usually FFBA
021A-021B — Contains the address to which the monitor jumps when outputting a character — usually FF69.

COMPUKIT CASSETTE SAVE/HEX MEMORY DUMP

Andy Fisher June 1979

To use, place the start address of code to be saved in 00F7, 00F8 and then the end address in 00F9, 00FA. Turn on the tape recorder and execute.

```
0222                    ORG     $0222

0222 A9 OD      START   LDAIM   $OD      CARRIAGE RETURN
0224 20 2D BF           JSR     $BF2D    CRT
0227 20 7A FF           JSR     $FF7A    10 NULLS TO CASSETTE
022A A9 2E              LDAIM   $2E      "." ADDRESS MODE
022C 20 75 02           JSR     CC
022F A5 F8              LDA     $00F8    FROM LOCATION (HIGH)
0231 20 63 02           JSR     AOUT
0234 A5 F7              LDA     $00F7    FROM LOCATION (LOW)
0236 20 63 02           JSR     AOUT
0239 A9 2F              LDAIM   $2F      "/" DATA MODE
023B 20 75 02           JSR     CC

023E A2 00      LOOP    LDXIM   $00
0240 A1 F7              LDAIX   $00F7    GET BYTE
0242 20 63 02           JSR     AOUT     OUTPUT
0245 A9 0D              LDAIM   $0D      CARRIAGE RETURN
0247 20 B1 FC           JSR     $FCB1    CASSETTE OUTPUT
024A A9 20              LDAIM   $20      SPACE
024C 20 2D BF           JSR     $BF2D    CRT
024F E6 F7              INC     $00F7    INCREMENT FROM ADDRESS
0251 D0 02              BNE     BUMP
0253 E6 F8              INC     $00F8
0255 38         BUMP    SEC              CHECK IF DONE
0256 A5 F9              LDA     $00F9    TO
0258 E5 F7              SBCZ    $00F7    FROM
025A A5 FA              LDA     S00FA    TO + 1
025C E5 F8              SBCZ    $00F8    FROM + 1
025E 10 DE              BPL     LOOP
0260 4C 43 FE           JMP     $FE43    YES, RETURN TO MONITOR

0263 85 FC      AOUT    STA     $00FC    USE MONITOR DISPLAY
0265 20 AC FE           JSR     $FEAC    TO UNPACK
0268 AD 64 D1           LDA     $D164    HI
026B 20 75 02           JSR     CC
026E AD 65 D1           LDA     $D165    LO
0271 20 75 02           JSR     CC
0274 60         RTS
0275 20 BI FC   CC      JSR     $FCB1    OUTPUT TO CASSETTE
0278 20 2D BF           JSR     $BF2D    AND CRT
027B 60                 RTS
```

Various flags are used to control the operation of the UK101:
The Control-C flag at location 0212 activates the Control-C or break in keys. If this value is non-zero then the break-in facility in BASIC is inhibited.
The Load flag at location 0203 tells the monitor whether its input comes from the cassette or the keyboard. A zero value signifies the keyboard.
Experiment with these vectors and flags, once you understand how to use them they will assist you when programming in machine code.

USE OF THE CASSETTE PORTS

The cassette interface uses memory-mapped ports located at F000 and F001.

The basic way the software works is to scan the status port at F000. If the ACIA is ready to transmit the second bit from the right will be set. If the ACIA is ready to receive the first bit from the right will be set. Using this information you can write your own cassette I/O routines.

e.g. To transmit a character in the accumulator.

```
        PHA             ; save the character on the stack
LOOP:   LDA $F000       ; controls of status port in A.
        LSR A           ; rotate right 2 places
        LSR A           ; to place 2nd bit in carry.
        BCC LOOP        ; if carry clear loop round.
        PLA             ; if ready to transmit return char.
        STA $F001       ; output char. to cassette.
```

USING THE POLLED KEYBOARD

*, the row and column addresses are shown in the circuit diagram of the keyboard.

The polled keyboard contains a firmware-scanned switch matrix which is outwardly similar to a standard ASCII keyboard (see diagram in circuit diagrams). The I/O port for the polled keyboard resides at memory location DF00 (hex) or 57088 (dec).

In operation, the polling routine successively addresses each row of key switches R0 - R7. Between these row scans, the routine checks the columns C0 - C7 for closed key switches. If a key closure is detected, the polling routine supplies the CPU with the ASCII code corresponding to the face of the key pressed. Each of the rows is addressed in turn, thus all key switches are scanned rapidly.

The BASIC statements used for programming special keyboard functions are POKE 57088, (row address) and IF PEEK (57088) = (column address). * After RUN is entered, these statements assume control of the key board since the normal polling routine is disabled (except where INPUT statements are encountered). In essence, the POKE statement turns on a row of keys, and the PEEK statement monitors the columns for a key closure. Upon detection of a closure, the PEEK statement can then transfer control to subroutines, GOTO statements, etc. This permits the function of each key to be software-defined for implementation of passwords, gaming controls, etc.

Please note: The Control-C function must be disabled to poll the keyboard by using POKE 530, 1.

To test the machine-code monitor, the message program used to illustrate the use of USR may be adapted as follows.

Place the monitor in address mode either by pressing RESET followed by M, or by pressing full-stop if already in the monitor. Enter the characters 0500 followed by / to access data. Type in the following pairs of digits — each pair separated by pressing RETURN.

```
A2  00  BD  00  06  C9  5F  F0
07  9D  E5  D1  E8  18  90  F2
4C  43  FE
```

This ends with a jump to location FE43 which places the monitor in address mode after the message has been displayed, thus preventing the clear screen routine in the monitor from erasing the message immediately after its appearance.

The following pairs of HEX digits are ASCII codes for the characters of the message— the list may be of any length but must start at 0600 and end with the pair 5F.

Press . and then type 0600 followed by / and the following pairs separated by RETURNs :

```
43  4F  4D  50  55  4B  49  54  5F
```

To run the program, type

```
.0500G
```

This will display the message for which the ASCII codes are given above and leave the machine code monitor in address mode for further use. Memory size need not be specified for the above unless BASIC is to be entered and the above protected against being overwritten.

Graphics

Character-slot graphics are used by the **COMPUKIT** whereby 255 different graphic characters are available to fill any given character slot.

To view the available characters, the BASIC function CHR$ may be used as follows.

Typing:

PRINT CHR$ (24)

followed by pressing RETURN causes a £ sign to be printed. Each number between 1 and 255 inclusive, corresponds to a character as 24 does to £. (0 corresponds to a null character).

Two of these numbers correspond to (non-printing) commands for the "Print Head" whose position is continuously shown by the cursor. Thus:

PRINT CHR$ (10).

causes a line-feed — i.e. the cursor jumps to the next line and the screen scrolls upwards.

PRINT CHR$ (13)

causes a carriage return.

The rest of the numbers correspond to ASCII characters, special characters and graphic characters.

The ASCII characters start at 32 (SPACE) and finish at 127. These are all accessable from the keyboard — the upper-case set with SHIFT-LOCK down, lower case otherwise.

Some of the characters are inaccessible from the keyboard directly — they must be printed using CHR$(I). The general graphic characters are best seen by writing a program to print them on the screen — this will be given later. Try pressing SHIFT-LOCK into the "up" position; with the CTRL key pressed, some of the keys will give graphic characters.

The following is a list of **special,** as distinct from **graphic** characters, with their corresponding numbers:

Number	Character
0	null
10	line-feed
13	carriage-return
24	£
32	Space
179	⇛
180	⇚
211	√
212	∫
241	∝
242	β
243	ω
244	δ
245	ψ
246	Ω
247	μ
248	π
249	≤
250	λ
251	φ
252	β
253	ε
254	ν
255	γ

In order to select a particular graphic character, a list of those available may be displayed on the screen with corresponding number next to each one. The following program achieves this by allowing the user to specify which block of characters is to be displayed — there are too many to appear at once. The instructions for the program are as follows.

The program is loaded and run. The words:

WHICH BLOCK?

appear. Answer with a number between 1 and 4 inclusive followed by a RETURN. The first two numbers display the graphic characters available, a 3 shows the special characters given above and 4 displays the ASCII set.

To exit the program, just press RETURN instead of a number.

The line numbers chosen for the program put it well above any other program you may be working on. If the program under development ends with an END then the following program will never be entered by the command RUN. RUN 10000 will be necessary. This allows the graphic program to remain in memory as a reference for use as necessary. It will be lost if NEW is typed or if RESET is pressed followed by C.

THE PROGRAM LISTING

(N.B. Spaces may be omitted and PRINT may be replaced by ?, for speed.)

```
10000      INPUT "WHICH BLOCK"; B : FL = 0
10010      IF B = 1 THEN S = 1 : F = 31 : GOTO 10060
10020      IF B = 2 THEN S = 128 : F = 219 : GOTO 10070
10030      IF B = 3 THEN S = 220 : F = 225 : GOTO 10060
10040      IF B = 4 THEN S = 32 : F = 127 : GOTO 10070
10050      GOTO 10000
10060      FL = -1
10070      FOR I = S TO F
10080      IF I = IO OR I = 13 THEN 10110
10090      PRINT I ; CHR$(I) ; : H = I + 3
10100      IF INT(H/7) = H/7 THEN PRINT : IF FL THEN PRINT
10110      NEXT
10120      PRINT
10130      GOTO 10000
```

When Block 2 is requested, some of the vertically adjacent symbols run into each other — use CHR$ in the immediate mode to inspect individual characters, e.g.:

PRINT CHR$ (161)

reveals that this character fills the entire character slot.

The fact that characters run into each other in this manner allows the user to build up quite complex graphic patterns as well as graphs and bar-charts etc.

The user may find it useful to store the above program on cassette tape for future reference.

Short BASIC Programs

The following short BASIC programs are provided here to allow you to gain some experience with your computer through fully debugged programs which are known to be working. These programs in no way depict the total capability of your computer. They are simple programs which are very short to facilitate manual entry. Each of the programs can be entered in your computer as listed. Remember to type NEW before entering each program. This clears out the computer's workspace. You can substitute a ? for the word PRINT. The **COMPUKIT's** 8K BASIC allows you this particular shorthand notation wherever the word PRINT occurs. Before you try to write lengthy programs of your own in BASIC, try modifying or customizing any of these programs to get a good feel for how BASIC works.

PROGRAM 1: Number Guess

In this program the computer generates random numbers, and you try to guess what the number is. When you guess the correct number, the computer tells you how many attempts you took to arrive at the correct number.

```
10     PRINT "I WILL THINK OF A"
15     PRINT "NUMBER BETWEEN 1 AND 100"
20     PRINT "TRY TO GUESS WHAT IT IS"
25     N = 0
30     X=INT (RND(56)*99+1)
35     PRINT
40     PRINT "WHATS YOUR GUESS";
50     INPUT G
52     N=N+1
55     PRINT
60     IF G=X THEN GOTO 110
70     IF G>X THEN GOTO 90
80     PRINT "TOO SMALL, TRY AGAIN";
85     GOTO 50
90     PRINT "TOO LARGE, TRY AGAIN";
100    GOTO 50
110    PRINT "YOU GOT IT IN ";N;" TRIES"
113    IF N>6 THEN GOTO 120
117    PRINT "VERY GOOD"
120    PRINT
130    PRINT
140    GOTO 10
150    END
```

PROGRAM 2: Heads-Tails Flipping

This program exercises the RND function of the computer by producing heads and tails. The long-term average out of many runs of this program should be approximately fifty percent heads, fifty percent tails.

```
5      REM HEADS/TAILS FLIPPING
10     Y=1
20     C=0
30     X=1
40     F=INT (RND(45)*2)
50     IF F=1 GOTO 80
60     PRINT "T";
70     GOTO 100
80     C=C+1
90     PRINT "H";
100    X=X+1
110    IF X < 51 GOTO 40
120    PRINT
130    PRINT C; "HEADS OUT OF 50 FLIPS"
132    PRINT
133    PRINT
135    Y=Y+1
140    IF Y< 11 GOTO 20
150    END
```

PROGRAM 3: ESP Test

This is another number-guess program where you are simply guessing heads or tails as the computer flips a coin. The computer keeps constant tabs on how many right and wrong answers you have given.

```
10      REMESP TESTER
15      REMTYPE E TO END
20      H=1
25      W=0
30      T=0
35      C=0
37      E=10
40      F=INT (RND(12)*2)
42      IF F=0 THEN A$ = "H"
43      IF F=1 THEN A$ = "T"
50      PRINT "H OR T";
60      INPUT X$
70      PRINT
80      IF X$=A$ THEN GOTO 100
83      IF X$="E" THEN GOTO 150
85      W=W+1
87      PRINT "WRONG"
90      GOTO 120
100     C=C+1
110     PRINT "RIGHT"
120     PRINT "W=";W;" R =" ;C
130     PRINT
140     GOTO 40
150     PRINT "BYE"
160     END
```

PROGRAM 4: Power Generation

This program generates powers of two up to the mathematical limit of the computer. It demonstrates the fact that BASIC automatically reverts back to scientific notation (E-format) when numbers are more than about six digits long up to a maximum of 10 to the 32d power. BASIC can also handle fractions as small as 10 to the – 32d power. Note the "delay" statements in line 77 to slow the VDU output.

```
5       PRINT
7       PRINT
10      PRINT "POWERS OF TWO"
20      PRINT
30      PRINT "POWER", "VALUE"
40      X=0
50      Y=1
60      PRINT X,Y
70      Y=Y*2
75      X=X+1
77      FOR I = 1 TO 300 : NEXT
80      IF X = 25 THEN 100
90      GOTO 60
100     END
```

PROGRAM 5: Decimal-Binary Conversion

The following program displays the binary equivalent of any decimal number typed in, up to the machine's maximum numerical capability.

```
50     PRINT
60     PRINT
70     PRINT "DECIMAL TO BINARY CONVERTER"
90     PRINT
93     PRINT
95     PRINT
100    INPUT X
101    IF X < 0 THEN 330
102    IF X > 32767 THEN 330
104    PRINT
105    PRINT "X = ";
110    Y = 16384
120    A = INT (X/Y)
130    IF A = 0 THEN 200
140    PRINT "1";
150    X = X - Y
160    GOTO 300
200    PRINT "0";
300    Y = Y/2
310    IF INT(Y)  = 0 THEN 320
315    GOTO 120
320    GOTO 90
330    PRINT "NUMBER TOO LARGE"
340    GOTO 90
```

PROGRAM 6: Prime Number Generation

Try to figure out how this program works.

```
10     PRINT "PRIME NUMBER GENERATOR"
13     Y = 2
15     A = 1
17     GOTO 80
18     X = 1
20     X = X + 1
50     Z = INT(Y/X)
60     IF INT (Z*X) = Y GOTO 85
70     IF X*X > Y GOTO 80
75     GOTO 20
80     PRINT A,Y
82     A = A + 1
85     Y = Y + 1
90     GOTO 18
100    END
```

PROGRAM 7: Acey-Deucy

This is a longer program that should be fun to play. Once you get this program in and running, it would be wise to store it on audio cassette for future use.

```
10     PRINT "ACEY-DUCEY"
12     PRINT "YOU WILL GET 25 HANDS"
13     H=1
15     PRINT
17     T=100
19     PRINT "YOU HAVE $"; T
20     X=INT (7*RND(67)+6)
21     IF X>12 THEN GOTO 20
30     Y=INT (X*RND(23)+1)
31     IF Y> =X THEN GOTO 30
32     IF Y=1 THEN Y=2
40     A=X
50     GOSUB 500
60     A=Y
70     GOSUB 500
80     PRINT
100    PRINT "YOUR BET";
110    INPUT B
111    IF B< =T THEN GOTO 120
112    PRINT "YOU DONT HAVE THAT MUCH"
113    GOTO 100
120    Z=INT (13*RND(99)+2)
121    IF Z>14 THEN GOTO 120
130    A=Z
140    GOSUB 500
150    PRINT
160    IF Z< =Y GOTO 200
170    IF Z> =X GOTO 200
180    PRINT "YOU WIN"
181    PRINT
182    PRINT
190    T=B+T
195    GOTO 300
200    PRINT "YOU LOSE"
201    PRINT
202    PRINT
210    T=T-B
220    IF T< =0 GOTO 380
300    H=H+1
310    IF H>25 GOTO 400
320    GOTO 19
380    PRINT "YOUR OUT!"
390    STOP
400    PRINT "THAT'S 25 HANDS"
410    STOP
500    IF A<11 THEN GOTO 505
501    IF A>14 THEN PRINT "ERROR": STOP
502    ON A-10 GOTO 522, 524, 526, 528
505    PRINT A;
510    RETURN
522    PRINT "JACK";
523    RETURN
524    PRINT "QUEEN";
525    RETURN
526    PRINT "KING";
527    RETURN
528    PRINT "ACE";
529    RETURN
```

PROGRAM 8: Multiplication Quiz

This demonstrates the use of the computer as a teaching aid.

```
10     PRINT "MULTIPLICATION QUIZ"
13     N = 0
15     C = 0
16     I = 0
20     X = INT(RND(56)*13)
30     Y = INT(RND(54)*13)
40     Z = X*Y
50     PRINT
60     PRINT X; "*"; Y; " = ";
70     INPUT W
75     PRINT
80     IF W = Z GOTO 120
90     PRINT "STUPID!"
91     PRINT "THE ANSWER IS"; Z
100    I = I + 1
110    GOTO 140
120    PRINT "YOU ARE RIGHT!"
130    C  =  C  +  1
140    PRINT C; "       RIGHT"
150    PRINT I; "       WRONG"
160    N = N + 1
170    IF N < = 9 GOTO 20
180    IF C > = 6 GOTO 190
183    PRINT "YOU FLUNKED!"
184    PRINT "PRACTICE!"
185    GOTO 13
190    IF C > = 9 GOTO 200
195    PRINT "YOU DID OK"
198    GOTO 210
200    PRINT "NICE JOB!"
210    PRINT "TRY AGAIN?"
220    INPUT T$
230    IF T$  =  "Y" GOTO 13
240    END
```

PROGRAM 9: Fahrenheit-Celsius and Celsius-Fahrenheit Conversions

```
10     PRINT "THIS PROGRAM CONVERTS"
20     PRINT "FAHRENHEIT TO CENTIGRADE"
30     PRINT "AND VICE-VERSA"
40     PRINT
41     PRINT "TYPE THE TEMPERATURE TO BE CONVERTED",
42     PRINT "FOLLOWED BY A COMMA AND F OR C"
43     PRINT "FOR FAHRENHEIT OR CENTIGRADE"
44     PRINT "RESPECTIVELY".
50     C = 0
60     F = 1
70     INPUT X,Y$
75     IF Y > 1 GOTO 250
80     IF Y$  =  "F" THEN 200
90     A = (9*X)/5 + 32
100    PRINT "     = ";A;"F"
110    PRINT
120    GOTO 70
200    A = (5*(X - 32))/9
210    PRINT "     = ";A;"C"
220    PRINT
230    GOTO 70
250    END
```

COMPUKIT UK101 Memory Map

Page 0 Usage

0000	JMP to warm start in BASIC
00FB	cassette/keyboard flag for monitor
00FC	data temporary hold for monitor
00FE-00FF	address temporary hold for monitor

Page 1

0100-0140	stack
0130	NMI vector—NMI interrupt causes a jump to this point
01C0	IRQ vector

Page 2

0200	cursor position
0203	load flag
0205	save flag
0206	CRT simulator baud rate—varies from 0 = fast to FF = slow
0212	Control-C flag
0218	input vector = FFBA
021A	output vector = FF69
021C	Control C check vector = FF9B
021E	load vector = FF8B
0220	save vector = FF96
0222-02FA	unused

Page 3 and up to end of RAM is BASIC workspace

A000-BFFF	BASIC in ROM
D000-D3FF	Video refresh memory
DF00	Polled keyboard
F000-F001	Cassette port 6850
F800-FFFF	Monitor EPROM
FC00	Floppy bootstrap
FD00	Keyboard input routine
FE00	Monitor
FF00	BASIC I/O support
FFFA	NMI vector
FFFC	RESET vector
FFFE	IRQ vector

Useful Subroutine entry points

A274	warm start for BASIC
BD11	cold start for BASIC
BF2D	CRT simulator — prints char in A register
FD00	input char from keyboard, result in A
FCB1	output 1 byte from A to cassette
FE00	entry to monitor, clears screen, resets ACIA
FE0C	entry to monitor, bypasses stack initialization
FE43	entry to address mode of monitor
FE80	input ASCII char from cassette, result in A, 7 bit cleared
FE93	convert ASCII hex to binary, result in A, =80 if bad
FF69	BASIC output to cassette routine, outputs one char to cassette, displays on screen, outputs 10 nulls if carriage return character
FF00	Reset entry point
FF8B	Load flag routine
FF96	Save flag routine
FF9B	Control-C routine
FFBA	BASIC input routine

BASIC Error Codes

CODE	DEFINITION
D /	Double dimension: variable dimensioned twice. Remember subscripted variables default to Dim. 10.
F /	Function call error: parameter passed to function out of range.
I /	Illegal direct: INPUT cannot be used in immediate mode.
N \	NEXT without a FOR.
O /	Out of data: more READs than DATA.
O ┐	Out of memory: program too big or too many nested GOSUBs, FOR NEXT loops or variables.
O ϟ	Overflow: result of calculation too large for BASIC.
S ┘	Syntax error: typing mistakes etc.
R \	RETURN without GOSUB.
U √	Undefined statement: attempt to jump to non-existent line-number.
/ ◢	Division by zero.
C ┘	CONTINUE errors: inappropriate attempt to CONT after BREAK or STOP.
L √	Long string: string longer than 255 characters.
O √	Out of string space: same as O ┐
S ∫	String temporaries: string expression too complex.
T ┘	Type mismatch: string variable mismatched to numeric variable
U \	Undefined function.
B √	Bounds error. Trying to reference a non-defined array element.

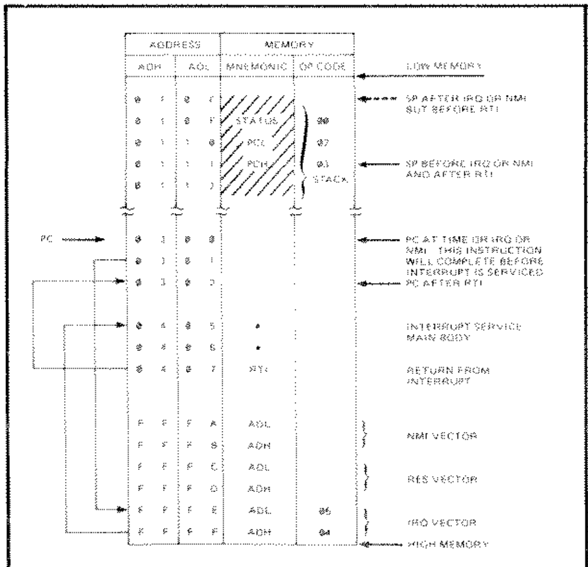
6502 Machine Code and Architecture

		IMPLIED			ACCUM			ABSOLUTE			ZERO PAGE			IMMEDIATE			ABS X		
MNEMONIC		OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#
ADC	(1)							6D	4	3	65	3	2	69	2	2	7D	4	3
AND	(1)							2D	4	3	25	3	2	29	2	2	3D	4	3
ASL					0A	2	1	0E	6	3	06	5	2				1E	7	3
BCC	(2)																		
BCS	(2)																		
BEQ	(2)																		
BIT								2C	4	3	24	3	2						
BMI	(2)																		
BNE	(2)																		
BPL	(2)																		
BRK		00	7	1															
BVC	(2)																		
BVS	(2)																		
CLC		18	2	1															
CLD		D8	2	1															
CLI		58	2	1															
CLV		B8	2	1															
CMP								CD	4	3	C5	3	2	C9	2	2	DD	4	3
CPX								EC	4	3	E4	3	2	E0	2	2			
CPY								CC	4	3	C4	3	2	C0	2	2			
DEC								CE	6	3	C6	5	2				DE	7	3
DEX		CA	2	1															
DEY		88	2	1															
EOR	(1)							4D	4	3	45	3	2	49	2	2	5D	4	3
INC								EE	6	3	E6	5	2				FE	7	3
INX		E8	2	1															
INY		C8	2	1															
JMP								4C	3	3									
JSR								20	6	3									
LDA	(1)							AD	4	3	A5	3	2	A9	2	2	BD	4	3
LDX	(1)							AE	4	3	A6	3	2	A2	2	2			
LDY	(1)							AC	4	3	A4	3	2	A0	2	2	BC	4	3
LSR					4A	2	1	4E	6	3	46	5	2				5E	7	3
NOP		EA	2	1															
ORA								0D	4	3	05	3	2	09	2	2	1D	4	3
PHA		48	3	1															
PHP		08	3	1															
PLA		68	4	1															
PLP		28	4	1															
ROL					2A	2	1	2E	6	3	26	5	2				3E	7	3
ROR					6A	2	1	6E	6	3	66	5	2				7E	7	3
RTI		40	6	1															
RTS		60	6	1															
SBC	(1)							ED	4	3	E5	3	2	E9	2	2	FD	4	3
SEC		38	2	1															
SED		F8	2	1															
SEI		78	2	1															
STA								8D	4	3	85	2					9D	5	3
STX								8E	4	3	86	2							
STY								8C	4	3	84	2							
TAX		AA	2	1															
TAY		A8	2	1															
TSX		BA	2	1															
TXA		8A	2	1															
TXS		9A	2	1															
TYA		98	2	1															

	ABS Y			(IND. X)			(IND)Y			Z PAGE, X			RELATIVE			INDIRECT			Z PAGE, Y			PROCESSOR STATUS CODES						
MNEMONIC	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	N	V	B	D	I	Z	C
ADC	79	4	3	61	6	2	71	5	2	75	4	2										●	●				●	●
AND	39	4	3	21	6	2	31	5	2	35	4	2										●					●	
ASL										16	6	2										●					●	●
BCC													90	2	2													
BCS													B0	2	2													
BEQ													F0	2	2													
BIT																						M_7	M_6				●	
BMI													30	2	2													
BNE													D0	2	2													
BPL													10	2	2													
BRK																								1		1		
BVC													50	2	2													
BVS													70	2	2													
CLC																												0
CLD																									0			
CLI																										0		
CLV																							0					
CMP	D9	4	3	C1	6	2	D1	5	2	D5	4	2										●					●	●
CPX																						●					●	●
CPY																						●					●	●
DEC										D6	6	2										●					●	
DEX																						●					●	
DEY																						●					●	
EOR	59	4	3	41	6	2	51	5	2	55	4	2										●					●	
INC										F6	6	2										●					●	
INX																						●					●	
INY																						●					●	
JMP																6C	5	3										
JSR																												
LDA	B9	4	3	A1	6	2	B1	5	2	B5	4	2										●					●	
LDX	BE	4	3																B6	4	2	●					●	
LDY										B4	4	2										●					●	
LSR										56	6	2										0					●	●
NOP																												
ORA	19	4	3	01	6	2	11	5	2	15	4	2										●					●	
PHA																												
PHP																												
PLA																						●					●	
PLP																						●	●	●	●	●	●	●
ROL										36	6	2										●					●	●
ROR										76	6	2										●					●	●
RTI																						●	●	●	●	●	●	●
RTS																												
SBC	F9	4	3	E1	6	2	F1	5	2	F5	4	2										●	●				●	●
SEC																												1
SED																									1			
SEI																										1		
STA	99	5	3	81	6	2	91	6	2	95	4	2																
STX																			96	4	2							
STY										94	4	2																
TAX																						●					●	
TAY																						●					●	
TSX																						●					●	
TXA																						●					●	
TXS																												
TYA																						●					●	

(1) *Add 1 to n if crossing page boundary*

(2) Add 2 to n if branch within page
Add 3 to n if branch to another page



ADDRESS
MEMORY
ADH
ADL
MNEMONIC
OP CODE
LOW MEMORY
SP AFTER IRQ OR NMI BUT BEFORE RTI
STATUS
PCL
PCH
STACK
SP BEFORE IRQ OR NMI AND AFTER RTI
PC
PC AT TIME OR IRQ OR NMI · THIS INSTRUCTION WILL COMPLETE BEFORE INTERRUPT IS SERVICED
PC AFTER RTI
INTERRUPT SERVICE MAIN BODY
RTI
RETURN FROM INTERRUPT
ADL
ADH
NMI VECTOR
RES VECTOR
IRQ VECTOR
HIGH MEMORY

FIG. 1 IRQ, NMI, RTI, BRK OPERATION

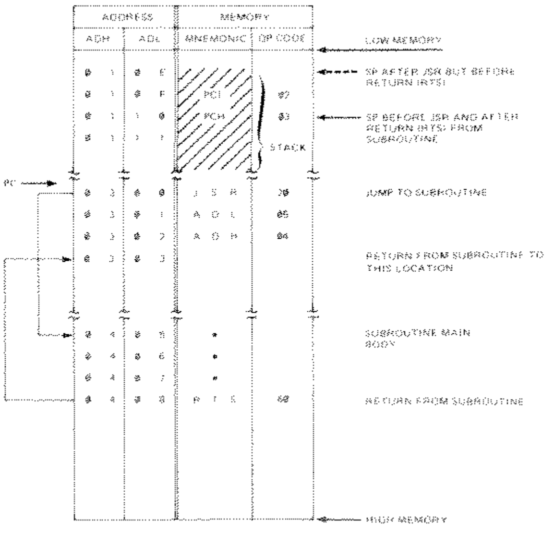
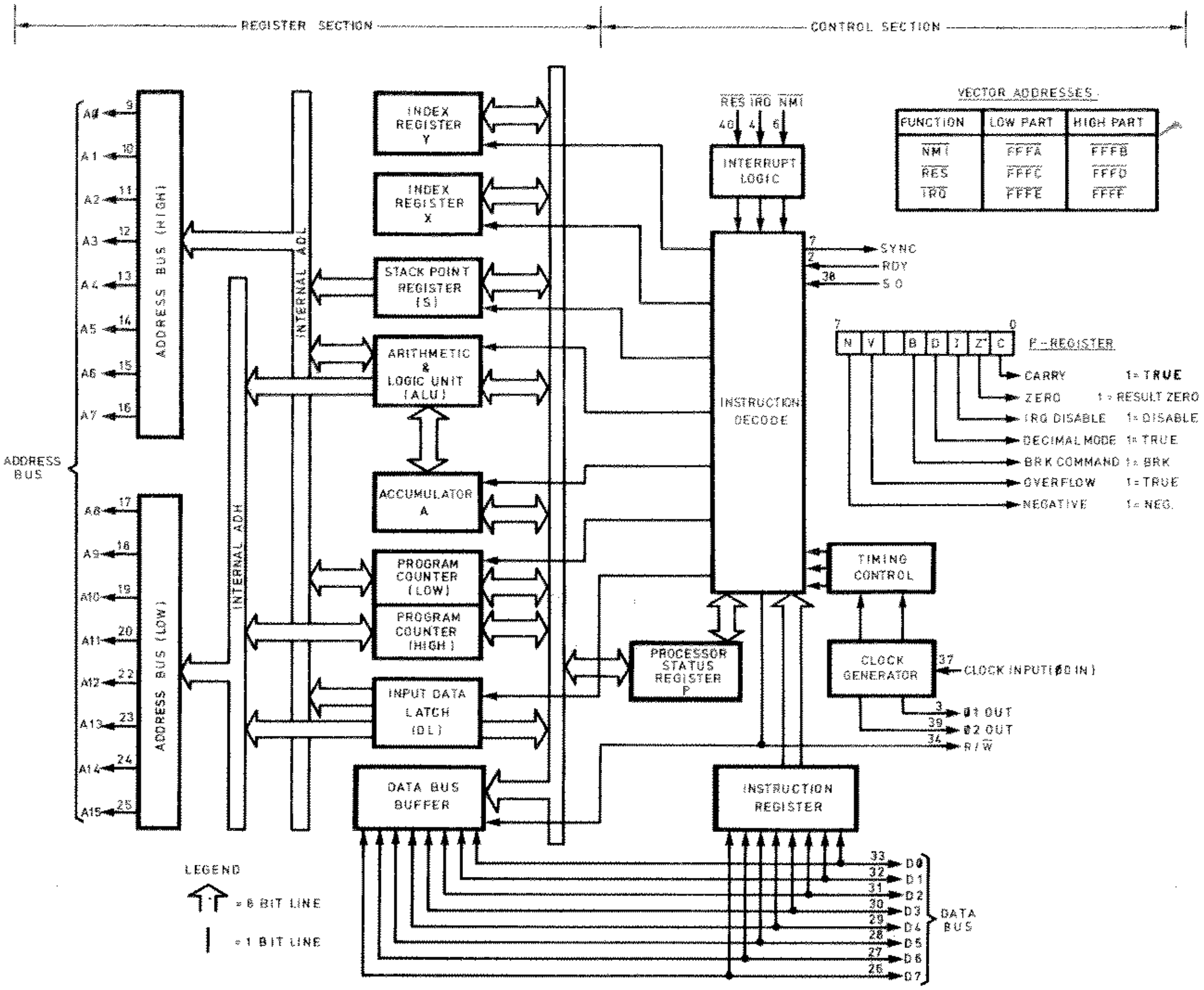


FIG. 2 JSR, RTS, OPERATION

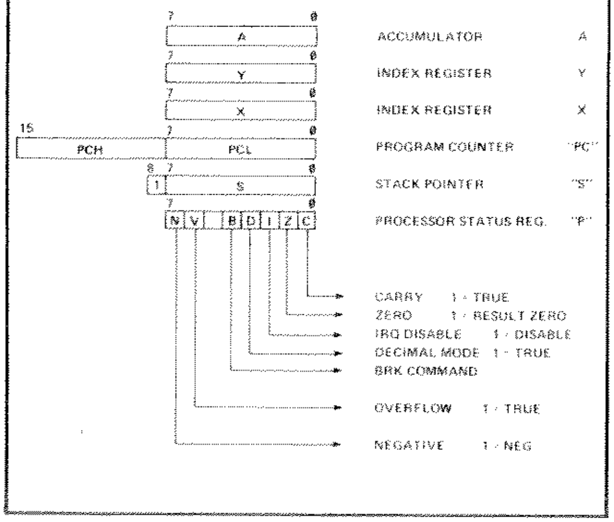


VECTOR ADDRESSES

FUNCTION	LOW PART	HIGH PART
NMI	FFFA	FFFB
RES	FFFC	FFFD
IRQ	FFFE	FFFF

ARCHITECTURE

PROCESSOR PROGRAMMING MODEL



ASCII CHARACTER SET (7-BIT CODE)

LSD \ MSD		∅ ∅∅∅	1 ∅∅1	2 ∅1∅	3 ∅11	4 1∅∅	5 1∅1	6 11∅	7 111
∅	∅∅∅∅	NUL	DLE	SP	∅	@	P		p
1	∅∅∅1	SOH	DC1	!	1	A	Q	a	q
2	∅∅1∅	STX	DC2	"	2	B	R	b	r
3	∅∅11	ETX	DC3	#	3	C	S	c	s
4	∅1∅∅	EOT	DC4	$	4	D	T	d	t
5	∅1∅1	ENG	NAK	%	5	E	U	e	u
6	∅11∅	ACK	SYN	&	6	F	V	f	v
7	∅111	BEL	ETB	'	7	G	W	g	w
8	1∅∅∅	BS	CAN	(	8	H	X	h	x
9	1∅∅1	HT	EM	)	9	I	Y	i	y
A	1∅1∅	LF	SUB	*		J	Z	j	z
B	1∅11	VT	ESC	+	;	K	[	k	
C	11∅∅	FF	FS	,	<	L	\	l	|
D	11∅1	CR	GS	-	=	M	]	m	
E	111∅	SO	RS	•	>	N	↑	n	
F	1111	SI	VS	/	?	O	←	o	DEL

INSTRUCTION SET — ALPHABETIC SEQUENCE

ADC	Add Memory to Accumulator with Carry
AND	"AND" Memory with Accumulator
ASL	Shift left One Bit (Memory or Accumulator)
BCC	Branch on Carry Clear
BCS	Branch on Carry Set
BEQ	Branch on Result Zero
BIT	Test Bits in Memory with Accumulator
BMI	Branch on Result Minus
BNE	Branch on Result not Zero
BPL	Branch on Result Plus
BRK	Force Break
BVC	Branch on Overflow Clear
BVS	Branch on Overflow Set
CLC	Clear Carry Flag
CLD	Clear Decimal Mode
CLI	Clear Interrupt Disable Bit
CLV	Clear Overflow Flag
CMP	Compare Memory and Accumulator
CPX	Compare Memory and Index X
CPY	Compare Memory and Index Y
DEC	Decrement Memory by One
DEX	Decrement Index X by One
DEY	Decrement Index Y by One
EOR	"Exclusive-or" Memory with Accumulator
INC	Increment Memory by One
INX	Increment Index X by One
INY	Increment Index Y by One
JMP	Zump to New Location
JSR	Jump to New Location Saving Return Address
LDA	Load Accumulator with Memory
LDX	Load Index X with Memory
LDY	Load Index Y with Memory
LSR	Shift One Bit Right (Memory or Accumulator)
NOP	No Operation
ORA	"OR Memory with Accumulator
PHA	Push Accumulator on Stack
PHP	Push Processor Status on Stack
PLA	Pull Accumulator from Stack
PLP	Pull Processor Status from Stack
ROL	Rotate One Bit Left (Memory or Accumulator)
ROR	Rotate One Bit Right (Memory or Accumulator)
RTI	Return from Interrupt
RTS	Return from Subroutine
SBC	Subtract Memory from Accumulator with Borrow
SEC	Set Carry Flag
SED	Set Decmal Mode
SEI	Set Interrupt Disable Status
STA	Store Accumulator in Memory
STX	Store Index X in Memory
STY	Store Index Y in Memory
TAX	Transfer Accumulator to Index X
TAY	Transfer Accumulator to Index Y
TSX	Transfer Stack Pointer to Index X
TXA	Transfer Index X to Accumulator
TXS	Transfer Index X to Stack Pointer
TYA	Transfer Index Y to Accumulator

ADDRESSING MODES

ACCUMULATOR ADDRESSING — This form of addressing is represented with a one byte instruction, implying an operation on the accumulator.

IMMEDIATE ADDRESSING — In immediate addressing, the operand is contained in the second byte of the instruction, with no further memory addressing required.

ABSOLUTE ADDRESSING — In absolute addressing, the second byte of the instruction specifies the eight low order bits of the effective address while the third byte specifies the eight high order bits. Thus, the absolute addressing mode allows access to the entire 65K bytes of addressable memory.

ZERO PAGE ADDRESSING — The zero page instructions allow for shorter code and execution times by only fetching the second byte of the instruction and assuming a zero high address byte. Careful use of the zero page can result in significant increase in code efficiency.

INDEXED ZERO PAGE ADDRESSING — (X, Y indexing) — This form of addressing is used in conjunction with the index register and is referred to as "Zero Page, X" or "Zero Page, Y". The effective address is calculated by adding the second byte to the contents of the index register. Since this is a form of "Zero Page" addressing, the content of the second byte references a location in page zero. Additionally due to the "Zero Page" addressing nature of this mode, no carry is added to the high order 8 bits of memory and crossing of page boundaries does not occur.

INDEXED ABSOLUTE ADDRESSING — (X, Y indexing) — This form of addressing is used in conjunction with X and Y index register and is referred to as "Absolute, X", and "Absolute, Y". The effective address is formed by adding the contents of X or Y to the address contained in the second and third bytes of the instruction. This mode allows the index register to contain the index or count value and the instruction to contain the base address. This type of indexing allows any location referencing and the index to modify multiple fields resulting in reduced coding and execution time.

IMPLIED ADDRESSING — In the implied addressing mode, the address containing the operand is implicitly stated in the operation code of the instruction.

RELATIVE ADDRESSING — Relative addressing is used only with branch instructions and establishes a destination for the conditional branch.
The second byte of the instruction becomes the operand which is an "offset" added to the contents of the lower eight bits of the program counter when the counter is set at the next instruction. The range of the offset is −128 to +127 bytes from the next instruction.

INDEXED INDIRECT ADDRESSING — In indexed indirect addressing (referred to as (Indirect,X)), the second byte of the instruction is added to the contents of the X index register, discarding the carry. The result of this addition points to a memory location on page zero whose contents is the low order eight bits of the effective address. Both memory locations specifying the high and low order bytes of the effective address must be in page zero.

INDIRECT INDEXED ADDRESSING — In indirect indexed addressing (referred to as (Indirect),Y), the second byte of the instruction points to a memory location in page zero. The contents of this memory location is added to the contents of the Y index register, the result being the low order eight bits of the effective address. The carry from this addition is added to the contents of the next page zero memory location, the result being the high order eight bits of the effective address.

ABSOLUTE INDIRECT — The second byte of the instruction contains the low order eight bits of a memory location. The high order eight bits of that memory location is contained in the third byte of the instruction. The contents of the fully specified memory location is the low order byte of the effective address. The next memory location contains the high order byte of the effective address which is loaded into the sixteen bits of the program counter.

COMPUKIT UK101 Price List

COMPUKIT UK 101 in Kit form	£219.00 + VAT
COMPUKIT UK 101 Fully assembled	£269.00 + VAT

SPARE PARTS

P.C.B.		55.00	
Manual (construction and Software)		7.90	
Keyboard (complete, all switches and Keytops)		39.50	
Replacement Keytops		.25p	each
Keyswitches		.60p	each
ROM (Basic) 4 chips		40.00	
Monitor (2K) chip		10.00	each
Character generator		12.50	each
TTL chip-set (everything included except processor, memory and ACIA)		24.50	
UHF Modulator UM 1233 (8MHz Bandwidth)		4.90	
Socket set (complete)		10.00	

INDIVIDUAL SOCKETS

40 pin socket		.45p	each
24 pin socket		.35p	each
18 pin socket		.25p	each
16 pin socket		.20p	each
14 pin socket		.18p	each
8 pin socket		.12p	each
Crystal (8MHz)		2.80	each
3 Amp Regulator LM323K		4.90	each
Heat Sink		.65p	each
Electrolytic capacitor 3300 mfd 40v		.45p	each
3 Amp diode		.25p	each
Resistors, capacitors and small diodes pack (these parts are not sold separately)		5.90	
6502 processor		7.90	each
6850 ACIA		4.50	each
2114 Memory chips	8 For	49.00	
Fuse Holder		.10p	each
Fuse 2.5 Amp		.06p	each
Transformer		8.90	
Colour Add-On Board		T.B.A.	(Oct.)
RAM/I/O Expansion Board		T.B.A.	(Nov.)