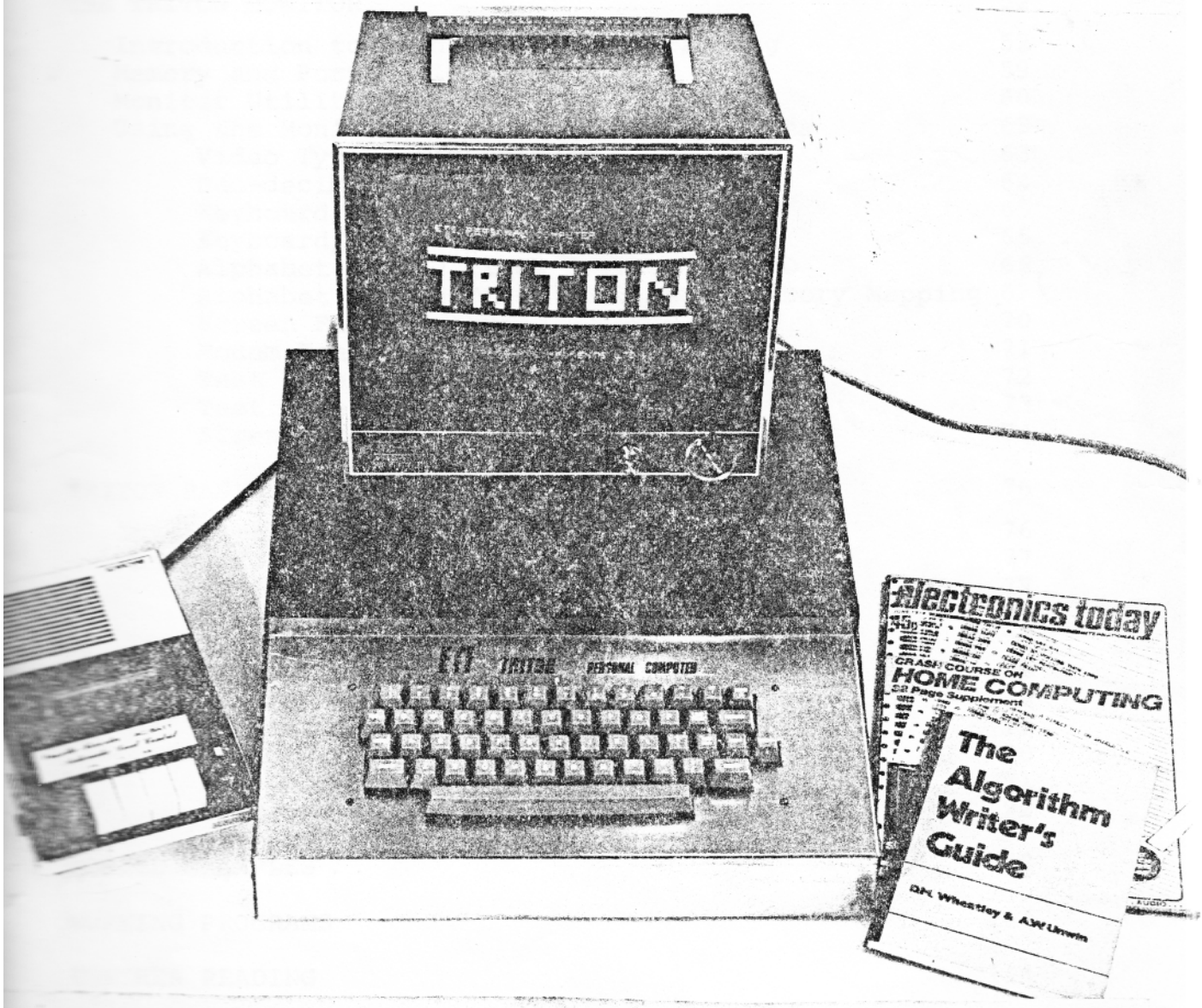
TRANSAM

01-4002-8137

THE TRITON MANUAL

ONE BOARD HOME COMPUTER

ETI, Transam and Mike Hughes, who designed the system, present the Triton — a one board computer that includes all the features expected in a machine providing the basis of a really powerful home system.



TRANSAM

CONTENTS

INTRODUCTION

Personal computing to date has been beyond the budget of many, and beyond the capabilities of many more. The dual skills of being able to construct one's own hardware and being able to create one's own meaningful software programs has, until now, limited the use of personal computers in this country.

The development of the TRITON computer now means that anyone can own, construct and use his own personal computer, no matter in which direction his particular skills may lie.

The TRITON Manual is designed in such a way as to remove the cloak of mystery surrounding the various aspects of computing - hardware, firmware and software - by explaining every aspect of the overall system in great detail, giving worked and documented examples of programs, both in machine code and in Basic wherever possible, and a clear and concise description as to exactly what is happening and why.

The TRITON project was conceived from the obvious demand for a single-board computer that was easy for the enthusiast to construct and that actually was "a computer" as opposed to a microprocessor evaluation kit. It was to be capable of being programmed in a high-level language, Basic, as well as machine code, have a full keyboard, graphics capability, output to UK domestic TV set and tape recorder for visual display and data storage, and to incorporate as many of the most recent advances in microcircuitry as possible to make it compact, portable and inexpensive.

In conjunction with Electronics Today International and ourselves, Mike Hughes, the designer of the project, has achieved this and more, the overall system being a very versatile and powerful computer with tremendous potential in homes, schools, colleges and industrial control and monitoring as well as business.

We hope that your interest in the TRITON will be well rewarded and that this Manual will help you and other enthusiasts to build, enjoy and learn more about personal computing, computers, how they work and how to use them.

TRANSAM COMPONENTS LTD
12 CHAPEL STREET
LONDON NW1
Tel 01-402 8137

TECHNICAL SPECIFICATION

SPECIFICATION

- SINGLE-BOARD CONSTRUCTION
- 8080A BASED SYSTEM
- BOARD HOLDS UP TO 4K ROM
- POWERFUL 1K MONITOR IN ROM
- POWERFUL 2K TINY BASIC IN ROM
- UP TO 3K USER RAM ONBOARD
- 64 GRAPHIC CHARACTERS
- AUDIO (STANDARD TTL) OUTPUTS
- COMPLETE POWER SUPPLY
- 1K VDU RAM ONBOARD
- 8 LEVELS OF INTERRUPT
- TAPE I/O AUTO START STOP
- 2 MODE VISUAL DISPLAY I/O OR MEMORY MAPPED
- FULL ALPHA NUMERICS
- UHF OUTPUT TO UK TV SET
- FULL 56 STATION KEYBOARD
- 42 LINE EXPANSION BUSBAR
- FULLY BUFFERED OUTPUT LINES FOR UP TO 64K MEMORY +256 I/O PORTS
- SPECIALLY DESIGNED CASE WITH ROOM FOR EXPANSION
- USES ALL LOW POWER SCHOTTKY TTL
- SPEED
  Master Clock 7.2MHZ
  Microcycle 1.25uS
  Instruction execution 6 to 13.75uS
  Tape I/O 300 Baud
  Print rate I/O Mode VDU 100 C.P.S.
  Display format 64 characters × 16 lines
- FIRMWARE RESIDENT IN ROM

1K MONITOR — 2 front panel interrupts allowing "Screen Clear/Cursor Reset" and reinitialisation without clearing memory. User machine code programming, memory check and modifications running of programs in machine code, listing, tape I/O for machine code or basic programs. Tape input with named file search facilities, clear and precise prompts and default statements — vectored jump to basic.

2K BASIC — provides the following commands and funtions

ABS	IF	NEW	RETURN	STEP
FOR	INPUT	NEXT	RND	STOP
GOSUB	LET	PRINT	RUN	TO
GOTO	LIST	REM	SIZE	VDU

ARITHMETIC AND LOGIC OPERATIONS (INTEGER ONLY)
DIVIDE MULTIPLY SUBTRACT ADD
GREATER THAN LESS THAN EQUAL TO
NOT EQUAL TO GREATER THAN OR EQUAL TO
LESS THAN OR EQUAL TO

A VERY VERSATILE POWERFUL COMPACT PERSONAL COMPUTER

PARTS LIST

RESISTORS (all ¼W 10% unless stated)

R1	82R ½W 10%
R2, 34, 44	10k
R3-19, 22-26, 30, 33, 35, 37, 38, 40, 46, 47	1k0
R20, 36	4M7
R21, 31	220R
R27	470R
R28, 29	330R
R32	15R
R39	4k7
R41, 45	220k
R42, 43	100k

POTENTIOMETERS

RV1	100R sub. min. horiz. preset
RV2, 3	10k sub. min. horiz. preset

CAPACITORS

C1, 15, 17	4 700u 25 V electrolytic
C2, 16, 18	470n polyester
C3-14	47n ceramic
C19	47p ceramic
C20, 23, 27, 32	47u 6V3 tantalum
C21	82p ceramic
C22	10u 6V3 tantalum
C24	100u 25 V electrolytic
C25, 29, 30	15n polyester
C26, 28	100n polyester
C31	220n polyester

SEMICONDUCTORS

IC1	LM323K
IC2	LM340T-12
IC3	LM320T-12
IC4	8224N
IC5	8080A
IC6	8228N
IC7, 8, 10, 49	74LS244
IC9	74LS245
IC11	74LS148
IC12	74LS240
IC13, 58	74LS00
IC14, 19, 67	74LS32
IC15	74LS154
IC16	74LS138
IC17, 18	74LS139
IC20	74LS02
IC 21, 22, 23, 24	MM2708Q
IC25-48	2111-2
IC50, 51, 68	74LS374
IC52	74LS75
IC53	74LS74
IC54	74S287
IC55	74LS132
IC56, 57, 71	74LS08
IC59, 60, 64, 65, 66	74LS157
IC61	SFC96364
IC62	74LS86
IC63	74LS163
IC69	RO-3-2513
IC70	74S472
IC72	74LS165
IC73-79	2102-2
IC80	AY-5-1013/TMS6011NC
IC81	555
IC82	MC14412VL
IC83	LM339N
D1-4	1N5400
D5-12	1N4001
D13, 14	1N4148
ZD1	5V1 400mW
Q1, 2, 3	BC148
LEDs 1-8	LD35Y (0.1" spacing)

TRANSFORMER

T1	12V+12V at 0.5A, 8V25 at 3A

SWITCH

PB1	DPDT Mains
PB2-5	SPST

CRYSTALS

X1	7.2000 MHz
X2, 3	1.0000 MHz

MISCELLANEOUS

PCB, Case, DIL Reed Relay type 15005, neon, 3A fuse plus holder, modulator (Astec type 1111E36), Full ASCII Keyboard, 64 way PCB plug and socket (optional) Type CS/CP64, 16 way inline PCB plug and socket Type A23-16, 8 way inline PCB plug and socket Type A23-8, edge connector to suit keyboard, 2 x 5 PIN DIN sockets, IC holders and heatsink (at least 90 x 100mm-matt black).

A complete set of components is available. For details, contact Transam, or see the Transam Catalogue.

Please note that the following ICs must be specially programmed:

IC 21,22,23,24
IC 54
IC 70

DESCRIPTION OF THE SYSTEM

Constructing a computer can be both challenging and educational. It certainly helps if you understand some of the terms and practicalities already but our aim has been to make TRITON a problem free and relatively easy to build computer.

Whether you are an electronics expert, a software specialist or a newcomer, you MUST know in general terms what the TRITON computer comprises otherwise you will not be able to get the best use from it when it is finally built. This manual is split up into different sections, each of which can be read independently and we suggest that you try the ones which sound most interesting first. You can then return to the constructional details.

The system itself is shown in Illustration A in block diagram form. For clarity, some of the single wire interconnections have been left out but nothing of significance has been omitted. The heart of the system is the 8080 microprocessor. The choice of this central processing unit was based on several major considerations.

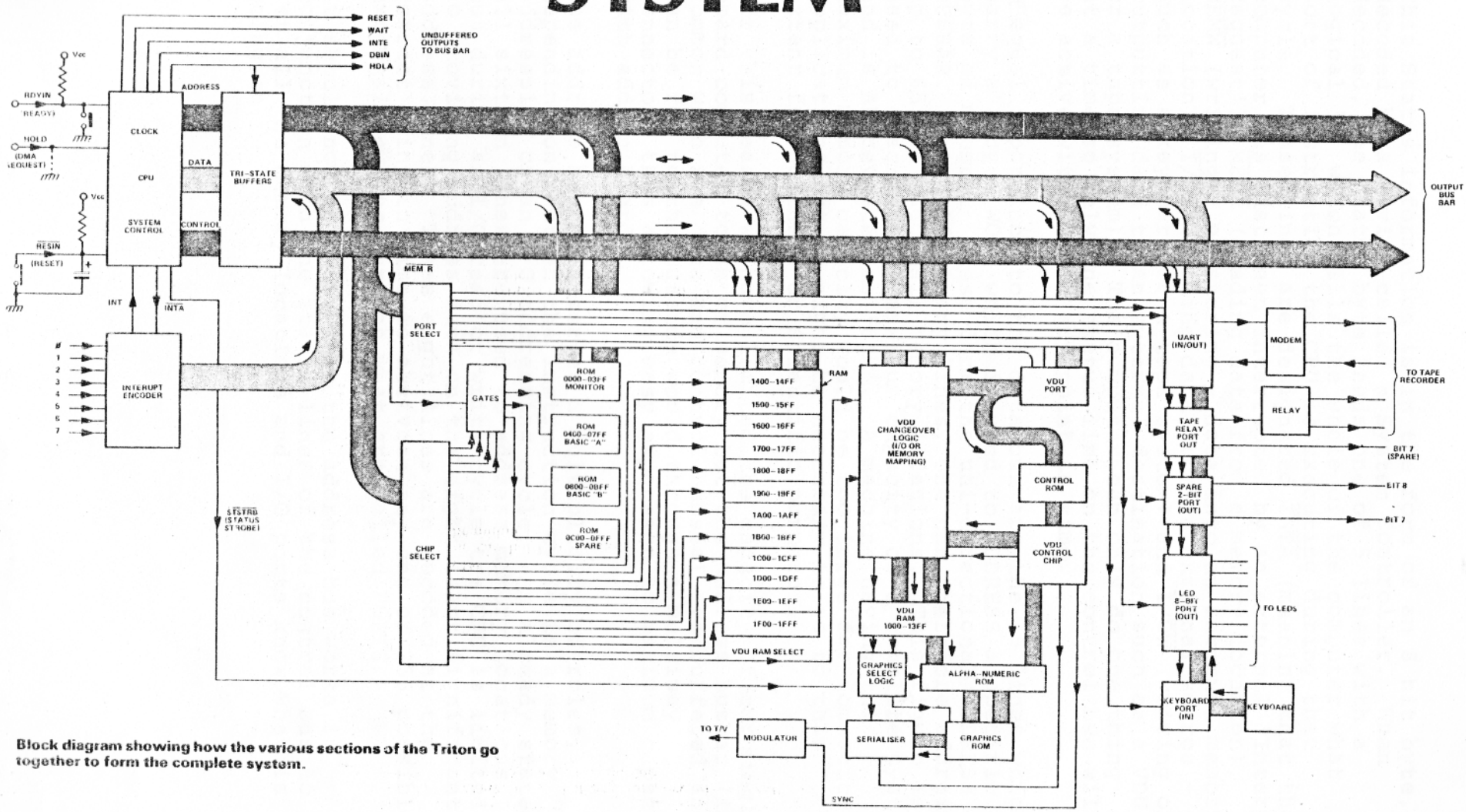
It has a very easily understood instruction set which is remarkably versatile for working at machine code level. There is also a great variety of software freely available to use with it. Last but not least, it is one of the cheapest CPUs available today.

The 8080 sequences through a list of instructions held in memory in 8 bit bytes and on receipt of each instruction, will carry out an operation which ranges from getting another byte of data from somewhere else in memory to carrying out simple logical or arithmetical operations on that data. It is not within the scope of this Manual to cover the inner workings of the MPU itself or for that matter, to explain every operation that the 8080 can offer. You can learn these when you have built the instrument and a further reading list is included in the reference section at the end of the Manual.

As it operates sequentially, the CPU needs a clock. In this case the master frequency is 7.20 MHz which is divided down to clock the CPU every 1.25 uS. This time is the duration of a microcycle and it takes from 4 to 11 microcycles for the CPU to complete an instruction.

The CPU has quite a large number of lines leading to it. The most significant are those carrying data to and from it. These number eight and are in the form of a bi-directional busbar (ie. can carry data to or from the CPU). To cut down on the number of wires coming from the CPU the data busbar serves a secondary purpose. It carries what is called 'STATUS' information at a certain point in time within an instruction cycle.

SYSTEM



Block diagram showing how the various sections of the Triton go together to form the complete system.

This Status information is in the form of an 8 bit byte, decoded by a device called a System Controller. When decoded, the Status byte feeds one of 5 lines with a logical '0' which tells the whole of the computer what sort of instruction the CPU is executing during that cycle. These lines are designated $\overline{\text{INTA}}$ (meaning that the computer has just been interrupted by an external 'Interput Request'), $\overline{\text{MEMR}}$ (reading data from a memory location), $\overline{\text{MEMW}}$ (writing - or storing - data into an internal memory location), $\overline{\text{I/OR}}$ (reading data from an external source - such as a keyboard or a tape system) and $\overline{\text{I/OW}}$ (writing or outputting data to an external destination such as a VDU or a tape system). The computer can only do one thing at a time so only one status line on the control bus will be active with '0' at any point in time.

Of equal importance to the data bus are the 16 lines which carry a 2 byte WORD which is used to ADDRESS a specific byte of memory. These form the uni-directional ADDRESS BUSBAR. Using 16 binary lines, one can therefore address up to 65,536 (decimal) memory locations. This figure is used to describe the maximum capacity of a complete system and is abbreviated as '64k'. A machine having this maximum memory capacity would be too large for domestic applications but it might be necessary to go to this extent for business or scientific purposes.

For this reason we have limited the capacity of our single-board computer to 8k of memory but the address busbar (in common with the data and control busses) are buffered and can be fed to the outside world through a multiway connector, thus allowing easy expansion to maximum capacity with add on boards.

The address bus also serves a multiplicity of roles, depending on whether the instruction cycle is a memory addressing or an I/O addressing cycle. As already stated, all sixteen lines are used to address memory locations, but during an I/O read or write cycle the CPU is limited to providing address data on the eight least significant address lines. These eight lines are decoded at the correct point in time to activate any one of 256 possible external devices through what are called PORTS.

It should now be clear that the address bus works in conjunction with the status lines of the control bus to let all the memory locations and I/O ports know what is going on.

Before moving on from the heart of the system, it is worth mentioning some of the single lines depicted on the illustration. When the computer is initially switched on, it is necessary to give it the right instruction to start with so that it can sequence on from there to complete the program in a sane manner. For this reason it is usual to have the first instruction at address location zero. This is logical and it is easy to make sure that the CPU looks at this location first by resetting the program counter to OOOOH (this means zero written as a four digit hexadecimal number. We can reset the program counter of the CPU by depressing a push button or as built into this machine, POWER ON RESET.

The line marked HOLD may be used for very special applications involving DMA (Direct Memory Access). Basically, this means that by making this line go to logic '1' it is possible to isolate the internal CPU from all three busbars (using the tristate facility of the buffers) and allow an external device to do what it will with the internal memory. We have strapped this line to '0' with a removable link so the facility is there if required. RDYIN is used if any memory or peripheral is incapable of responding as fast as the computer desires. The external device can make this line go to '0' for any period of time (usually set by a monostable) and when this happens the computer goes into a WAIT state and simply stops operating as long as this line is low. When the RDYIN signal is removed, it carries on as if nothing had happened.

The signal WAIT is issued during this time and the WAIT line can be seen designated as one of the unbuffered outputs. The RDYIN line provides a very useful option for those who do not wish to use it for its primary purpose. By connecting it via a push button switch to ground, the computer can be halted in the middle of any operation. This is particularly useful for stopping a BASIC LIST operation for cursory inspection. Facility for bringing this out to a push button is not made on the board but it is a simple matter to pick up the right point on the top side and take it via a single wire to the front panel - see the circuit diagram in this section.

The RESET output goes high momentarily when the reset button is pressed and can be used to carry out a synchronous reset on external equipment. The HDLA output indicates that the computer has gone into a HOLD (or DMA) state - if anyone takes the HOLD line high. The INTE line (Interrupt Enabled) indicates that the computer may be interrupted and the DBIN line indicates which way the computer expects data to be flowing on the bi-directional data bus. It goes high when CPU is expecting data to flow INTO it.

We are using the $\overline{\text{STSTRB}}$ (STATUS STROBE) signal for a very special purpose - to synchronise the memory mapping of the VDU - more is said about this in the relevant section.

As already inplied, the 8080 will allow itself to be interrupted in mid program provided that the program sets the Interrupt Enable flag. This provides the possibility of breaking into a program that is running to do something totally different. There is facility for 8 possible interrupts but only 7 can really be used on this machine (Interrupt 0 is redundant as it duplicates RESET).

An interrupt is entered into the machine on a single interrupt request line. Of the seven usable lines we are using, two within the machine to do a clearing operation on the VDU screen and to do a re-initialisation without clearing all the memory (which would otherwise happen if the reset button was pressed). There are five remaining lines, one of which is brought out to a spare push button on the front panel and the rest piped down the multiway socket along with the busbars. The interrupt request lines have to be encoded and formatted into an 8 bit date byte. When this is done the interrupt encoder tells the CPU with the INT signal that an interrupt has been received. When the CPU is ready to be interrupted, it issues an Interrupt Acknowledge signal $\overline{\text{INTA}}$ which is used to place the encoded byte onto the data bus. This byte enters the CPU and directs the computer to operate the desired subroutine. At the end of the routine the computer reverts to the main program continuing at the point at which it was interrupted.

The memory of TRITON is split into three types on the main board. There are locations for up to 4k of Read Only Memory (ROM) which is split between four 2708 Erasable ROMs. These occupy address locations OOOOH to OFFFH. The standard TRITON uses the first 1k to hold Monitor and Utility routines necessary to initialise the machine and re-vector interrupts. The next 2k holds a BASIC INTERPRETER and the fourth 1k block is left spare for future expansion or user designation.

There is 1k of Random Access Memory dedicated to the VDU. This starts immediately above the ROM area starting at 1OOOH. Normally this RAM is addressed in synchronism with the VDU line scan by the VDU control circuitry but the CPU can take over addressing under program control (in effect interrupting the VDU). This way the VDU can be built into the memory architecture of the main computer allowing quite spectacular animated visual displays. The VDU RAM can be written into only by the computer.

The rest of the memory is made up of RAM which is both read and write. This area is used to hold the stacks and tables of the MONITOR and BASIC INTERPRETER (512 bytes) and the main work area starts at 16OOH for a further 2½k ending at 1FFFH. This represents the full capacity of the on board memory. There is no reason however, why further read write memory should not be added externally starting from location 2OOOH.

The ROM and VDU RAM areas are blocked into units of 1k - to fall into line with the types of integrated circuits used. However the stack and work area RAMs are laid out in blocks of 256 bytes. The reason for the smaller blocks for the latter is to allow the user to choose to what extent he wishes to build up the on board memory. Those on a limited budget can extend the memory in fairly gentle stages, although to achieve any realistic use from the Basic Interpreter, it is essential to have at least five of these blocks.

The high order bits of the address busbar are used to decode which block is being addressed - this is done by the Chip Select decoder. Four lines can be seen going from this to the ROMs, the single line to the VDU 'Memory Mapping Changeover' logic and the remaining twelve lines to the Read Write blocks. Note that the ROM chip selects are gated with the $\overline{\text{MEMR}}$ signal from the Control Bus whereas this control signal and $\overline{\text{MEMW}}$ go straight to the RAM chips. This is because the 2111 Random Access Memory ICs used have internal chip select gating and output enables.

With the exception of the VDU which is 'hybrid' the rest of the system is made up from a variety of I/O stages. The most important of the latter is the Keyboard Input. The keyboard data and strobe lines are fed onto the data busbar via tri-state buffers which form the keyboard input port. Only when the computer's software addresses this port (by decoding the least significant 8 bits of the address bus) through the Port Select logic and issues a $\overline{I/OR}$ control signal will data from the keyboard be placed on the data busbar. Working in the opposite direction, the Output Port driving a bank of eight on board LEDs is a set of eight latches which catch and hold whatever data is on the busbar when they receive a coincident pair of signals from the port selector and the $\overline{I/OW}$ line of the control bus. These onboard LEDs do help to make the TRITON system more versatile and can be used for test purposes or in specialised development applications. The LEDs themselves could be discarded and the eight lines made available as a spare general purpose output port.

By making use of two redundant latches on the board it was possible to provide two spare output lines on one port and a spare line on the port which also feeds the tape recorder power control relay.

The UART (Universal Asynchronous Receiver/Transmitter) is the device which converts the 8 bit wide parallel data on the busbar to a specially formatted serial stream to feed the tape recorder modulator. It also carries out the complementary function of converting a received serial stream into parallel data bytes. The device operates as if it were two input ports and one output port. One of each sort of port would be obvious for a device which receives and transmits but the requirement for a second input port may not be so obvious. Because the device operates asynchronously from the main computer (it has its own clock operating a 300 baud) it is necessary to make the computer wait from time to time to allow the slower operating UART to complete a transmission cycle. This is indicated by the UART activating a flag which is regularly monitored by the second input port.

The VDU portion of the computer is based on the Thomson-CSF Control chip and operates in a unique manner for this integrated circuit. Not only is it possible to output to the VDU through an output port (in a similar manner to using a teletype) but the computer may also be used to inject data directly in the VDU's memory at extremely high speeds. This is most dramatically illustrated by programs written in low level Assembler when a character can be placed onto the screen in under 30 microseconds. A complete screen full of characters will appear as if the characters had all been outputted simultaneously. When used in conjunction with the special VDU function of the Basic Interpreter the speed is greatly reduced but in many instances is faster than the I/O mode with the advantage that characters can be placed anywhere on the screen without having to manipulate the cursor.

A further extension is the way the control chip is used to handle Graphics. Instead of the usual 6 bit wide RAM, 7 bits are used in this VDU application to enable the use of the complete set of ASCII codes. 64 extra character codes are therefore available by using those normally associated with lower case 'alpha' characters and all the control codes. Within the overall context of the computer, some of the control codes serve dual purposes and care has been taken in programming the VDU control ROM to inhibit printing a graphic when a control code is used for genuine control purposes.

The graphic select logic looks at the two most significant bits of the ASCII code and via an EXCLUSIVE OR function determines whether or not the symbol is graphic or alpha-numeric and then proceeds to select the standard alpha-numeric ROM or the specially programmed graphics ROM. There is a great deal of extra logic associated with this operation as well as the Memory Map/IO changeover which is dealt with in the section describing the circuit in detail.

For simplicity and reliability, the TRITON uses an ASTEC UHF modulator as a pre-assembled module. This operates on channel 36 with British modulation standards. For the benefit of those overseas we should mention that the television line and field standards used in this VDU are 50 Hz field rate with random interlace and 312½ lines per field (ie. 625 lines per randomly interlaced frame). The line and field rates are set by the crystal-controlled Thomson-CSF chip.

CONSTRUCTION

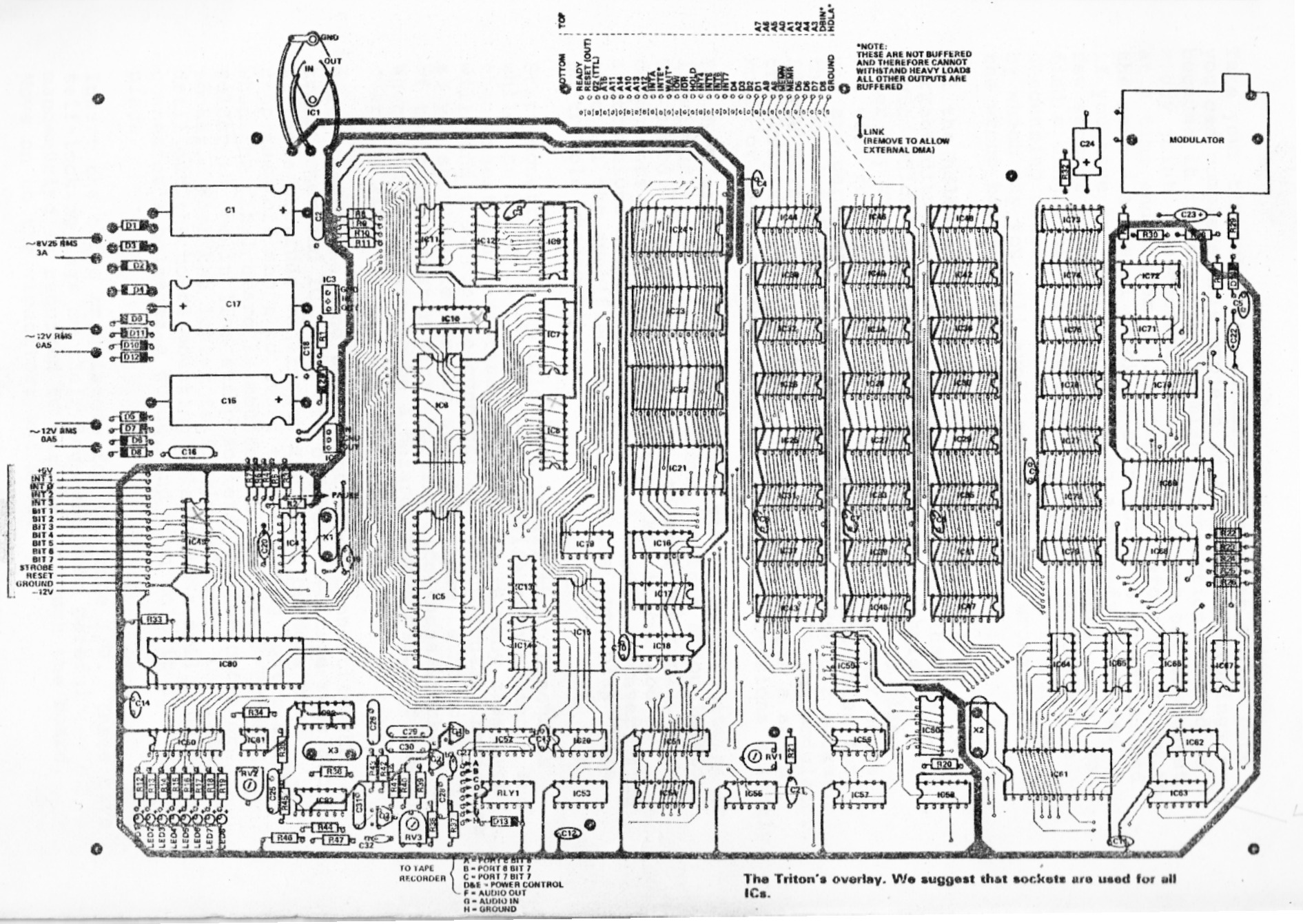
As the TRITON has been specially designed to work on a single printed circuit board, its construction is relatively easy.

Firstly, it is essential to use a top quality double-sided plated through board for the project. This, together with the complete kit of approved parts for the computer, is supplied exclusively by Transam Components Ltd (who retain full copyright). Unlike many other types of project, the PCB is likely to be the most expensive single item you have to invest in, but it is this component which brings the whole project into scope for the average constructor with no significant theoretical knowledge. You should notice the high quality surface of this board, due to the through hole plating process. Avoid finger contact and certainly keep it free from grease.

There are a large number of plated through holes, (approx. 2,500) and some of these are located beneath components. It would be wise to check the integrity of the through hole plating BEFORE any construction is started. It will probably be sufficient to check point to point with a continuity meter that the power address and data busbars are continuous throughout the board. This can be done by referring to the pin numbers of ICs on the schematic drawings and then picking these up on the topside illustration. An hour or two spent on this exercise is an invaluable insurance that the board has not been damaged in transit. Ideally, you should check the busbar integrity on the UNDERSIDE of the board as it is this side that you will be soldering.

The board has been designed to keep all the most intricate wiring on the top side - in particular the connections that run between IC pins. These are the most vulnerable to a heavy hand on the soldering iron, but this is not to say that any carelessness can be afforded underneath. Use the smallest soldering iron you have available - the bit must certainly be no greater than 3/32" diameter. As stated, ALL soldering operations should be carried out on the underside of the board - the through hole plating will route all necessary connections to the topside.

Wherever possible, try to reinforce the through hole plating by allowing molten solder to creep through the hole by capilliary action. To do this, hold the soldering iron in place long encough for the heat to flow through the hole and take the solder with it. A couple of seconds longer then your usual soldering time should suffice. You will notice that on the underside of the board there are hundreds of IC pin lands that do not appear to be connected to anything. These lands must be soldered in ALL positions because nearly all of them go somewhere on the top side.



The Triton's overlay. We suggest that sockets are used for all ICs.

Take your time with the soldering - even at a slow pace you can complete this project in a couple of evenings - because it is very easy to miss a connection or produce a dry joint. We recommend that you insert one component at a time and solder it in completely before moving on to the next - a visual check of each joint is essential and if you have any doubt, check with a magnifying glass. A few seconds taken up doing this can save hours trying to find a single missed connection. Insert IC sockets in descending order of size - ie. 40 pin, then 28 pin, etc, by which way you will avoid soldering the wrong socket in the wrong hole.

All the holes on the board have been pre-drilled to the correct diameters, but in the event of your having a device which will not quite go through the hole, do NOT under any circumstances attempt to drill out to size - you will ruin the through hole plating. The ONLY holes you are permitted to drill out are the fixing holes for the board and the mounting holes for the extender socket. If you have a stubborn component, for example the modulator, try scraping down the diameter of its lead with a sharp knife or use a needle file to reduce its dimensions slightly.

You MUST use sockets for all the integrated circuits as it is virtually impossible to remove ICs from a double-sided THP board. Transam Components Ltd will not replace components or be able to test your kit if the above instructions are not followed. You should, however, find that all the specified components slide into place with no trouble whatsoever.

Start construction by soldering in all the DIL sockets while the board is flat - it makes life much easier - and then insert all resistors and diodes. In many cases you will find that topside connections run underneath resistors so you should take care to see that there is no possibility of the bare resistor wire shorting onto these tracks. Where you feel this to be a hazard, use a piece of sleeving over the wire.

Next, insert the nine board pins which connect to the transformer and IC1 (the off off-board voltage regulator). If these protrude underneath the board by more than 1/16" crop them off short to avoid them shorting to the cabinet when the final board is mounted in its case. Proceed to solder in the in line strip sockets and the extender socket. When the latter is firmly soldered, CAREFULLY drill out the board mounting holes with a drill, using the connector's holes as a guide, and then bolt it firmly into place.

Insert the three transistors for the tape I/O. These have self-locking lugs on their legs and some careful manoeuvring is required to get them through the board. Move on to the capacitors and LEDs.

Leave the three large smoothing capacitors until last and take care to insert the LEDs the right way round - you will have to look very carefully at the solid tantalum capacitors to find their polarity. Then insert and solder in the three pre-set potentiometers.

Before progressing further, check the polarity of all the diodes and electrolytic capacitors you have inserted. It is often difficult to see the marking on 1N4148 devices so if in any doubt, use a test meter on them. Also make absolutely sure that you have put the zener diode in its correct position.

You can now insert and solder in the three crystals, making sure you have them correctly positioned. The crystals have their frequencies stamped on them (usually in kilohertz). Ideally, it is preferable to use sockets for the crystals, but as space is at a premium on the board, do be careful not to overheat them. This is the only instance where you are advised to use the minimum amount of time with the soldering iron and not to worry about capillary action.

Continue with construction by putting in the modulator and the two onboard regulators. Make sure you have the right regulator in the right position. One is for positive regulation, the other for negative and you can check their type numbers against the parts list and their designations on the board. Also ensure that you insert them the right way round - the metal fin should be on the face of them furthest away from the main smoothing capacitors.

Temporarily mount IC1 on its heatsink and run flying leads to the three pins allocated to it. Be very careful that you observe the correct sense of its two pins which are offset from the lateral centre line of the package. Hold your device against the layout drawing and turn it so that the pins are nearer the top than the bottom and you will then have the same sense as the drawing.

Before inserting any integrated circuits, give the power supply a dry run. Connect up the remaining six board pins to their corresponding terminals of the transformer and apply power. Use a voltmeter to ensure that you have the correct voltage rails present. You should get +5V and +12V at the output pins of ICs 1 and 2 respectively, and -12V at the output of IC3. You should read -5V at the junction between R1 and the zener diode. If all is well here, systematically check that you have the correct voltages at the sockets of EVERY integrated circuit. Use the schematic diagrams to help you identify the pin numbers. Finally, check that you have inserted the single wire link to the right of the extender socket.

Insert all the integrated circuits making ABSOLUTELY SURE that you have them orientated correctly and that they are in the correct locations. Again use a magnifying glass to read their type numbers as some of them look similar at a quick glance. Use the dot on the UART to locate pin 1 as the notch can be misleading. Note that the orientation of ICs varies a lot on the board and you must check each one individually. Insert the 2708 EROM chip that is marked MONITOR V4.1 into the socket for IC21, the one marked BASIC L4.1 'A' into the socket for IC22 and BASIC L4.1 'B' into IC23. The standard kit includes eight TMS 2111-2 devices (some may have different manufacturer's marks owing to availability from the standard suppliers). These should be inserted into IC locations 25 to 32 inclusive. The only gaps you should have on the board are for IC24 and ICs 33 to 48.

Do not install the keyboard at the moment but simply make up a coaxial lead linking the modulator to the aerial socket of a standard 625 line television set. Switch on the TV and allow it time to warm up, check that a rastor is just visible and tune it to approximately channel 36.

Set the three onboard potentiometers to their midway positions and apply power to the computer. You should see some change on the television screen even though you may not be spot on tune. Try adjusting the tuning over the whole range until a strong signal is locked in. You should now see the welcome message:

```
TRITON READY
FUNCTION? P G I O L W T
```

Once you can read this signal, you can rest assured that your computer is working. Switch the computer off, wait a few seconds and switch it on again. For a fraction of a second you will see some garble on the screen which will rapidly clear and the previous message will be repeated.

If you do not get this reaction you have made a mistake somewhere down the line which is almost certainly going to be one of the following:

Dry joints
Missed connections
Components in the wrong place
Solder splashes across tracks
EROMS plugged into the wrong sockets
RAMS plugged into the wrong locations
Faulty components
Faulty PCB

The last two possibilities are most unlikely - particularly the latter, if you carried out the recommended tests. Faulty components are a possibility if you use any from a doubtful source or if you try to use standard TTL instead of the low power devices specified. Avoid this problem altogether by buying best quality devices exactly as specified - they are much cheaper in the long run. If you purchased your components from Transam Components Ltd and can verify that you have checked all the other possibilities of failure, you should immediately contact Transam who will be operating a 'get it going' service for all customers. This will not apply to boards which are returned with careless soldering, soldered-in ICs or non-specified components used as substitutes.

We guarantee that the circuit board is correct and that the design works. Both the board and the resident firmware have been tried and tested over several months and, apart from soldering mistakes on the part of the constructor, no other faults were experienced during the period.

Presuming that all has gone well, you now have a nearly finished computer. Switch off and make up an umbilical cord of wires to go from the keyboard socket on the board to the keyboard and associated push switches. Use colour-coded wire and ensure that you make no mistake when connecting the relevant leads to the keyboard Cinch connector. It is double-sided and you must make sure to hold it with the correct orientation or you may have disastrous consequences with the power lines. Different types of keyboards have different connections - our drawing refers to the George Risk Model 756. We refer you also to the connection details supplied with the keyboard. The only comment we should make is that the specified keyboard, and some others, give you an option for bit 6 of the data. One option gives upper case characters (capitals) only, while the other gives both upper and lower case. This application needs the latter. The strobe is the static strobe which goes to '1' as long as a key is depressed.

The specified keyboard does not have any built-in direct function keys so these have to be provided by separate push buttons. These must be mounted on the front panel and are used to provide RESET, INT1 (Clear Screen), INT2 (Reset without clearing memory), and TAPE MANUAL OVERRIDE - ganged with PAUSE (see descriptions elsewhere). The first three push switches all have a common ground and are 'push to make' with a spring return. Use the Common lead and the respective signal leads to go to each of these switches. The fourth switch must be double pole 'push to make - push to break'. One pair of contacts should take the special 'PAUSE' line to ground when it is on. This line does not exist in the umbilical cord coming from the board socket but must be soldered to the end of R3 going to pin 3 of IC4. The other pair of contacts is connected across the tape power control pins of the respective DIN socket.

You can make up all the above on flying leads to test the unit fully before putting it into its cabinet. Power up again and get the initialisation message. Try pressing any key on the keyboard EXCEPT P G I O L W or T and the computer should respond by saying INVALID. Press CONTROL C and the screen should clear and re-initialise. Press RESET. When the button is released the same should happen. Try INT2 and the machine should, again, re-initialise. When you try INT1 the screen should clear without the message appearing. To get something back on the screen, press any keyboard key except those in the 'key character' message (P G I O L W T). You should once more get INVALID. Depress CONTROL C once more and your computer is re-initialised and ready for test.

We are assuming at this stage that you do not know anything about programming, so simply follow the instructions and check that you get what is described.

Depress P on the keyboard (there is no need to press shift because the Monitor program automatically gives you upper case). You will get:

```
P
PROG START=
```

(The computer is asking you to tell it the address of the part of memory you wish to inspect).

Type in ØØØØ followed by carriage return.
The display will now show:

```
P
PROG START= ØØØØ
ØØØØ 31
```

(31 is the data in location ØØØØ)

Depress carriage return repeatedly and you will get the following as you step through the Monitor program instructions:

```
P
PROG START= ØØØØ
ØØØØ 31
ØØØ1 8Ø
ØØØ2 14
ØØØ3 FB
etc...
```

Re-initialise with CONTROL C and then type L. The computer will again ask you for a start address but this time will list out the contents of 15 adjacent locations starting from that address. We can use this to test that the memory is there and working in the RAM area.

Answer the computer with the address 16ØØ and a carriage return. If you make a mistake before you press CR you can backspace with CONTROL H and change an entry, but you must then type through the rest of the line on the screen. The computer will list the contents against the memory addresses and then stop and ask for 'MORE?'.
If all is well you should see ØØ in all locations.
To continue, type Y and keep doing this, checking all the locations up to the highest order RAM on the board. Above that address the computer will read FF which indicates that there is no memory there. If you see any data above address 15FF (that is, anything other than 00 or FF) you can be sure you have a bad connection to the RAM IC which contains the data in question. This test only holds true immediately after first initialisation and cannot be used if you have attempted to write programs.

To get out of LIST, type any character other than Y and the computer will re-initialise. Carry out this or any of the other reset procedures already described and procede to check the G function. This is to facilitate running a machine code program. The computer will acknowledge:

```
G
RUN
PROG START=
```

This means it is ready to run but wants you to tell it from where in memory it should get its first instruction. Give it this information by typing 02B9 followed by CR. You will actually be running a re-initialisation program in the Monitor which should just acknowledge with:

```
FUNCTION? P G I O L W T
```

You are now back where you started so you can try typing W which turns the computer into nothing more than a video display typewriter. You can type away to your heart's content testing out all the alpha-numeric and graphics characters, using the keys in unshifted, shifted and control mode. Do this while inspecting the coding tables shown in the section describing the VDU and get used to the cursor move commands. Type a full line of characters and adjust VR1 for best line length. To get out of this mode of operation use CONTROL C or any of the other methods of resetting. It will be fun for you to see BASIC L4.1 in action so depress T. The computer acknowledges with:

```
T
BASIC L4.1
OK
```

Type in NEW followed by CR to ensure the memory is cleared and the computer re-acknowledges with the BASIC header.

Very carefully type in the following message line by line with a CR at the end of each line. Remember you can correct by backspacing with CONTROL H before you hit CR.

```
1Ø FOR A=1 TO 1Ø
2Ø PRINT 'YOU ARE MY MASTER'
3Ø NEXT A
RUN
```

You should not re-type the 'greater than' prompt signs - the computer is prompting YOU with these. When you press CR after typing RUN we hope you will be surprised - you have just written your first program.

You can now be confident that your computer is working correctly and it only remains to test and adjust the Tape I/O circuits. This must be done in stages.

First check the Tape Output software. Connect an audio monitor (simple amplifier or crystal earpiece) between the 'Tape Out' socket on the board and ground. You should hear a continuous tone. Call up BASIC by typing T and enter the above program again. Once you have done this, get back to the Monitor without erasing your BASIC program (use CONTROL C). Now press O to call up the Tape Output routine.

The computer will ask you for a TAPE HEADER which can be anything you like written in alpha-numerics. Preferably do not use a title longer than 20 characters as you might run out of input buffer space. We suggest you type in SLAVE ROUTINE. Follow this with CR while listening to the tone on the ear piece. Nothing will happen on the VDU but after a pause of between 5 and 6 seconds (this could be longer if you are using a master clock crystal lower than the 7.20 MHz as specified) you will hear about 1 second of regular high speed pulses followed by a few seconds of what can best be described as 'burble' (this is your program going out). The burble will stop and you will hear just the continuous tone you heard at the beginning. After a further 5 or 6 seconds, the VDU will confirm that the file has finished by displaying END followed by the re-initialisation heading.

Repeat this exercise but this time connect a continuity meter across the tape power control sockets on the board. The manual override switch must be open circuit. While you type in the tape header code the meter should show that the relay is open circuit, but as soon as you depress the CR to start the operation, the relay closes and stays closed until the VDU types END. It is obvious that the 5 or 6 second delay at each end of the routine is to allow a portion of blank tape to go by to reduce the chance of overlapping files or missing the start of the active tape at the beginning of a new cassette.

You must now set the Baud rate for your system. The simplest way is to use a frequency meter connected to pin 3 of IC81. Adjust VR2 until the meter reads exactly 4,800 Hz. A better way, and probably more viable for most constructors, is to use a standard test tape. This is available from Transam. It is better because different tape recorders might operate at different speeds which would influence the play-back baud rate of your system. This does not matter if you are only recording and playing back your own programs, but if you wish to use those from other sources, your OVERALL SYSTEM must operate at 300 baud. Using a standard test tape calibrates your overall system to 300 baud.

To carry out this test properly, you must have a master clock crystal having a frequency greater than 4.5 MHz, otherwise the VDU may not print out as fast as the data is coming in from the tape. You must also enter and run a special machine code program to facilitate the test. (How the program operates is explained in the section on the Monitor). The machine code program accepts any data on the tape and displays it, verbatim, on the VDU. If garble is received and decoded, garble will be printed. The test tape contains the alphabet followed by CR and Line Feed repeated many times over a period of a few minutes. All you have to do when the program is running, is set VR3 to its midway position and adjust VR2 until you get the alphabet reliably repeated on the screen. If, at the best setting of VR2, you still get occasional garble, try altering VR3 for sensitivity. You should of course, be using the phono output from your tape recorder, but if you do not have this, use the extension speaker socket with the volume set about 20% up from minimum.

Now carry out the following instructions EXACTLY.

Initialise the computer with RESET, type in P and enter the start address for the program as 16ØØ. For zero always use Ø and not O. Press CR and location 16ØØ will be shown to contain ØØ. Now use the memory change facility to start writing your program. Simply type in the following list of hexadecimal instructions - each pair of digits should be followed by CR. You will end up with a column showing address locations to the right of which is a column showing what WAS in that location (it should have been ØØ in all cases) and to the right of that the new data you have just typed in. When you have typed in the complete list of instructions, use CONTROL C to re-initialise, then type L, and list from location 16ØØ (as previously described). Check that the codes in each location correspond EXACTLY with those in the published program. Use CONTROL C to re-initialise and then type G. Enter 16ØØ without pressing CR at this stage. Make sure your tape recorder is properly connected to the board and switch on the recorder in PLAY mode.

Press CR and procede to adjust VR2 as previously described. You should see:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOP     etc..
```

until the recording ends or you switch off the tape recorder. While this is happening your computer is locked within a program loop and you will NOT be able to get out of this with CONTROL C. You will have to use INT2 to re-initialise.

Here is the program you must type in:

Address location	Data you must enter
16ØØ	CD
16Ø1	27
16Ø2	Ø3
16Ø3	CD
16Ø4	1D
16Ø5	Ø3
16Ø6	CD
16Ø7	13
16Ø8	ØØ
16Ø9	C3
16ØA	Ø3
16ØB	16

Your TRITON is now completely set up and ready for use. You have made an extremely powerful computer whose applications are limited only by your own imagination and the development of more sophisticated software - coupled with extender boards to give you extra I/O functions (Floppy Disks, Line Printers, extra Tape Recorders, more Memory, etc). We have plans for further exciting applications and developments. Please contact us or watch our advertisements in the national magazines.

FINAL ASSEMBLY

We strongly recommend that you solder in all the sockets where there are spare locations on the board. This will make life much easier for you if ever you wish to increase the on board memory. When you have done this and have completed all the recommended tests you should procede to assemble the printed circuit board into the cabinet.

To make life easy we have arranged for the cabinet to be pre-drilled with all the major holes but you will have to drill out the fixing holes on the printed circuit board to suit the screw diameter. Carry out this operation extremely carefully, making sure that no swarf falls across any of the printed wiring. You should also take steps to avoid flexing the board while drilling - this is best accomplished by placing the whole board flat on an old wooden board using a hand drill from the top side. Make your holes slightly oversize to allow a little tolerance for fitting so that the board will not be under strain when the screws are tightened.

Use long screws inserted from the underside of the cabinet and fix these securely in place with nuts so that they will act as fixing columns. Run a spare nut on to each screw so that it stops about ¼" from the inside bottom of the cabinet - these will act as stand off spacers. You can then offer up the board to these pillars to check the fit but do not bolt it into place as yet. Remove the board and stick a sheet of self adhesive plastic film over the inside base of the cabinet where the pcb will be mounted. This will act as an insulator in the event of any untrimmed leads protruding too far beneath the board.

Use four small screws to mount the keyboard pcb to the sunken support bracket making sure that you sandwich a sheet of insulating board between it and the metal work (needless to say - you should remove the conductive foam pad from under the keyboard's integrated circuit). Do not mount the keyboard assembly into the cabinet at this stage.

Mount all the back panel components - DIN sockets, one and two,fuse holder and mains cable grommet and bolt the mains transformer into place. Make sure you place a solder tag under one of the fixing nuts for the transformer for an earth lead. Make up the bank of four push buttons for the front panel and screw into place. Three buttons should be push to make with a spring return and the fourth should be push to make/push to break. The first three are used for RESET, Interrupt 1 (Clear Screen Reset Cursor) and Interrupt 2 (Reset without clearing Memory) while the latter is for tape control manual override. Also mount the separate mains push button switch onto the front panel. Note that we do not supply a push button for Interrupt 3 because we feel that this will be of more use if brought out via one of the sockets on the back panel.

Get the mains wiring out of the way before proceeding further. The mains live lead should go to the front panel switch via the fuse whereas neutral goes direct to the switch. Do not forget to connect the earth wire to the solder tag under the transformer. We recommend that you use twisted pairs for as many of these mains connections as possible. Secure the mains cable with a clamp.

Mount IC1 and its heatsink to the outside of the back panel. No electrical insulation is necessary but we suggest you cut some short lengths of sleeving to go over the pins of the regulator to prevent accidental short circuits where they enter the cabinet. Place a solder tag under one of the fixing screws. To ensure good heat conduction the fixing screws should be well tightened (conductive grease is not necessary).

Mount the main pcb onto the supporting screws making sure that it is level and well clear of the base of the cabinet - in particular, ensure that the extender socket on the side is clear above the side lip of the acbinet. Use more nuts to fix the board securely in position. Hopefully you will never need to remove it again. Connect the nine power supply board pins to their respective positions on the transformer and IC1 taking particular care with the transofrmer markings and the orientation of the integrated circuit. DOUBLE CHECK ALL THESE.

Make up a loom of 8 wires to go from the eight pin inline socket on the board (next to RLY1) to the two DIN sockets. We suggest you use one socket to carry the tape in and out audio signals (with their common ground) and the two power control connections. The other socket can be used for the two bits from output Port 6, the spare bit from Port 7 and Interrupt 3 with a common ground. The choice of connections is entirely up to you and might depend on your application or type of tape recorder. It is a good idea to make a record of the connections and stick it on to the back of the cabinet for future reference.

If you have carried out the recommended tests, you will already have made up a loom of wires or have used ribbon cable to go from the sixteen pin inline socket to the keyboard. We cannot overstress the importance of getting the correct orientation of the keyboard's edge connector. The loom at the keyboard end should be split with the power lines, bits 1 to 7 and the strobe going to the keyboard. The reset line and Interrupts 1 and 2 go to the three push to make spring return buttons together with a common ground line taken off the keyboard edge connector.

Take the opportunity to add a pair of wires to this loom going from the tape control override switch to the two respective pins on the tape output DIN socket (loop these onto the two connections which have already been made from the eight pin in line socket). Note that Interrupt 0 is not used and that Interrupt 3 should be taken straight to a spare pin on the I/O DIN socket.

You can now fix the front panel neon taking the supply off the mains switch.

Finally use four screws to bolt the keyboard bracket assembly to the underside of the front panel ensuring that none of the keys foul the cut out. The keyboard edge connector can now be slid into position and we suggest you now use nail varnish to identify the mating ends of this connector as well as the board in line connectors.

It only remains to fix the cabinet cover into position using the self tapping screws, although before doing this it is worth giving the unit a run to check all is well and, if necessary, to carry out a final adjustment to the tape I/O baud rate.

If at a later date you wish to insert extra memory ICs into the spare board sockets, try as far as possible to support the board to prevent too much flexing.

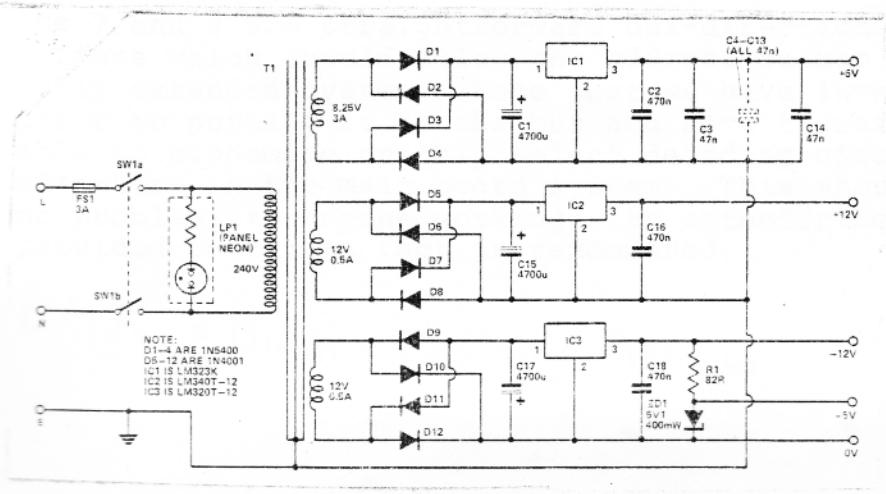
HOW IT WORKS

THE POWER SUPPLY

It needs no more than a few words to talk about the power supply. It has been kept as simple as possible, utilising three IC regulators to provide the main rail supplies which are +5V at 3A (the computer does not draw all this but there is very little to spare), +12V at ½A and -12V at ½A. A few milliamps are needed by the ROMs and the 8080 at -5V and this catered for by a simple zenar shunt off the -12V rail. You should check when building this stage, that R1 is of ½W rating.

The +12V and negative rails are straightforward. Dissipation by the regulators is quite reasonable to the extent that no heat sinks are necessary. The 470nF capacitors on the outputs of the regulators have been have been positioned as close to their respective devices as possible to prevent any possibility of parasitic oscillations. Note that the +5V rail has a dozen 47nF capacitors (C3 to C14) shunted across it. These are anti-spiking devices and have been placed in strategic places on the board. To test the prototype under adverse conditions we deliberately left these out and in daily operation over a period to two months, we did not get a single malfunction of the machine. To test the TRITON under even more severe conditions, we installed it within a heavy engineering plant where lathes, drills and grinding wheels were in constant use, as well as equipment operating with thyristor power control systems. At times the stray r.f. coming down the mains obliterated the television picture - but not once did the computer logic fail. Our conjecture that mains input filters on the power input were not necessary was in all cases fully justified.

To avoid excessive dissipation in the main +5V regulator (IC1) - which is still fairly high - we decided on a specially wound mains transformer, hence the rather unusual specification for an 8.25V winding. IC1 needs a large finned heatsink and to dissipate the heat from this we considered it wise to position it on the outside of the cabinet. There are no worries about electrical isolation as the can of the TO-3 package is connected to the chassis and the common rail.



The power supply section of the Triton is based on three terminal regulators.

IC4 is the master clock oscillator which contains divider circuits to provide the two-phase clock (Ø1 and Ø2) for the 8080. A TTL compatible output of Ø2 is available but not used on the main board - this is fed to the extender socket. The chip also contains gating circuits to synchronise the externally generated RDYIN command before feeding this to the CPU. An internal Schmitt Trigger on the reset input line (RESIN) allows a very simple charge up circuit comprising R2 and C20 to provide power on reset. Manual reset is carried out by momentarily taking RESIN to 0 volts via a push button. The clock receives a feedback signal (SYNC) from the CPU which is gated with Ø1 to give a STATUS STROBE pulse at the precise moment the data busbars are carrying the status byte. The pulse (STSTB) is fed to the System Control chip (IC6) to latch the status byte and is also used by the VDU to enable Memory Mapping changeover.

We will not attempt to describe the inner workings of the CPU (IC5) in this Manual but simply refer you to the reading list at the end of the book.

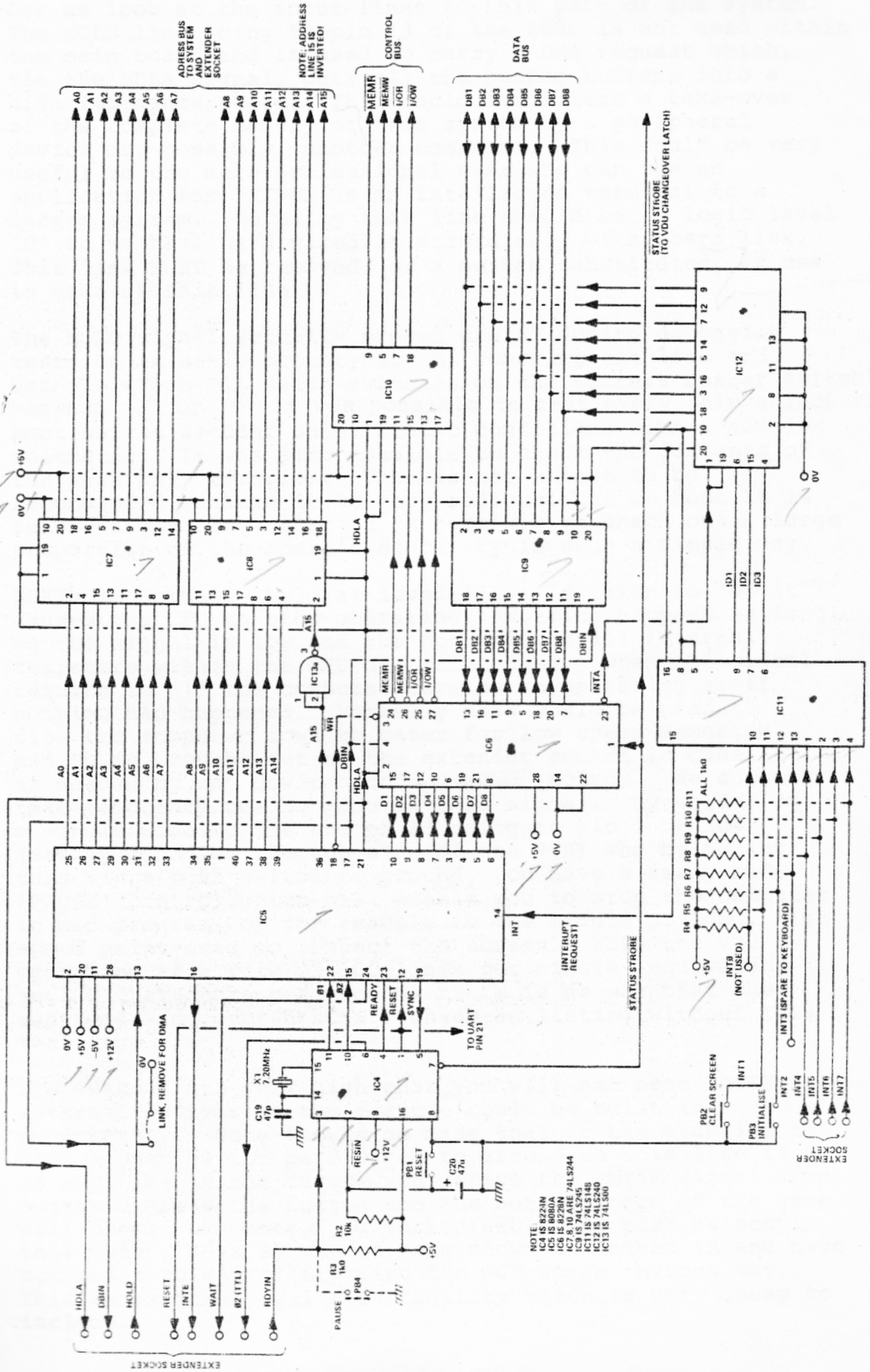
Certain outputs (namely HDLA, DBIN, INTE and WAIT) are taken to the extender socket directly from the CPU. These are unbuffered and account should be taken of this if you expand the system. Each line will adequately drive a single TTL load and more if you use low power devices.

IC6 is an 8228N 8080 System Controller which gates out the five main control busbar signals from the status byte at the time of STSTB and holds these on latches. The chip also comprises a set of bi-directional buffers for the data busbar - the direction of these buffers is controlled by DBIN and their outputs are disabled on the receipt of a DMA request by the HDLA signal. We were not happy that this buffer alone would be capable of supporting a fully extended system so felt it prudent to insert a further buffering stage in the shape of IC9 (74LS245). Like the System Controller, the latter chip is supervised by the HDLA and DBIN signals. Integrity of any DMA request is maintained on the data bus.

ICs 7 and 8 are straightforward uni-directional tri-state buffers which should allow the address busbar to feed a fully extended system. Note that we have inverted A15 prior to putting it on the bus and have therefore been able to economise on chip select decoding circuitry elsewhere in the main board system. This should present no problems to anyone working with extender boards provided that this fact is remembered.

DMA = Direct Memory Access

CPU


ADRESS BUS TO SYSTEM AND EXTENDER SOCKET
NOTE: ADDRESS LINE 15 IS INVERTED!
CONTROL BUS
DATA BUS
STATUS STROBE (TO VDU CHANGEOVER LATCH)
IC4 IS 8224N
IC5 IS 8080A
IC6 IS 8228N
IC7 & 10 ARE 74LS244
IC8 IS 74LS245
IC11 IS 74LS148
IC12 IS 74LS240
IC13 IS 74LS00
LINK, REMOVE FOR DMA
TO UART PIN 21
INTERRUPT REQUEST
EXTENDER SOCKET
PB1 RESET
PB2 CLEAR SCREEN
PB3 INITIALISE
INT3 (SPARE TO KEYBOARD)
INT8 (NOT USED)

Let us look at the input lines to this part of the system. The HOLD line going to pin 13 of the 8080 is not used within the main board and is used to carry a DMA request which, via the HDLA signal, puts all the busbar buffers into a high impedence state. This could facilitate a take-over of the complete memory of this system by a peripheral device or, possibly, another computer. This could be very useful to the more professional user who can see an application for TRITON as an intelligent terminal to a larger system. Normally this line should be at logic level 'O' so we have hard wired it accordingly with board link. This link MUST be removed, or a switch substituted, if use is made of this line.

The HOLD signal actually proved useful during debugging tests on an early prototype. By removing the link and using sixteen DIL slide switches on the address busbar switching between 'I' or 'O' it was possible to test every chip select and port select decoder under static conditions with a simple voltmeter. It was also possible to check the contents of the Read Only Memories. This is not a test to be recommended because it is very tedious, but we hope it is reassuring to know that it is possible to check out a large proportion of the board's circuitry in such a simple way.

RDYIN is normally held at level 'l'. If taken to 'O' it causes the CPU to stop operating. Nothing happens as long as the signal is low and the contents of all internal registers within the CPU are maintained. When the signal returns to 'l' the computer carries on operating as if nothing had happened. Normally this would be used to slow the computer down to cater for low speed memories and so is brought out to the extender socket in case it is needed by any devices on extender boards. We suggest that temporary use be made of this signal. By soldering a single wire to the end of R3 going to pin 3 of IC4 (easily found on the top side of the PCB) and by taking this via a push switch to ground, you have a ready made 'PAUSE' control which will enable you to stop the computer in mid program, or for example in the middle of long, high speed print-outs to inspect the screen. With the VDU operating at a rate of 100 lines per minute (equivalent to 100 characters per second) there is no way that one can read and comprehend a high-speed listing without this facility.

The chances are very high that you will not need RDYIN for external systems so the feature could be built in permanently. Note should be made that it bad practice to have a push switch hard wired to ground on this line if at any time in the future you derive the RDYIN signal from a gate. Press the button and the output stage of the gate will blow. Remember, you must disable the push switch in this mode - this is why we have shown it dotted in and have not built this facility into the PCB in an obvious way. This an extremely valuable facility which is very cheap to include.

You do not even need another push switch because you can use a spare pair of contacts on the Tape Control and Manual Overide. It does not normally matter if you press this button provided that the cassette recorder is switched off with its own control.

IC11 is the Interrupt Encoder which has eight lines going into it. These are normally held high by pull up resistors R4 to R11. The encoded three bit nibble is output in inverted form at pins 6, 7 and 9. If all the inputs are high, all the outputs are high and an '0' is placed on the Enable Output line at pin 15 (the latter is used to generate the INT signal - Interrupt Request - to the CPU). If any single input is pulled to 0 volts, via the push switches or external logic, an equivalent code to describe that line number is output as the Interrupt Data Nibble, and pin 15 goes high telling the CPU that an interrupt has been requested. The computer will carry on operating until it reaches a permissible point in its cycle to service the interrupt. When this point is reached, the CPU outputs an Interrupt Acknowledge signal ($\overline{\text{INTA}}$) through the status byte which is decoded and latched by the System Controller. This signal is used to activate the Output Enable of IC12 (an eight wide tri-state inverting buffer) which formats the ID nibble to make an eight bit Interrupt Data byte which is then accepted by the CPU as a RESTART instruction. The program counter jumps to one of eight fixed locations in memory - the location is defined by the ID byte - while the STACK preserves all current register data and status information. The computer then operates on the interrupt routine and returns to its main program when it comes to a RET instruction. Interrupts can be nested if the software permits it - this will depend on whether the EI (Enable Interrupt) instruction is used within the interrupting subroutine, because as soon as an interrupt is serviced, the computer will reset the EI flag.

Interrupt 0 should not be used even though it is available on the PCB. It simply duplicates the manual reset operation but would create major problems if used as the Monitor program contains an EI instruction early on in this routine. A very rapid build up of interrupt nests would occur which would fill up the stack in a fraction of a second. If you do not understand this, do not worry - just refrain from connecting the INT0 pin to a push button. INT1 is dedicated by the Monitor to provide a Clear Screen and Reset Cursor facility which can be carried out at any time, even while the VDU is printing data - quite useful for clearing garble from the screen. INT2 is also a dedicated function and is very important. Because the Monitor includes memory test facilities before it causes the computer to be initialised, use of the reset button will clear all memory - this can be very frustrating. To by-pass this problem we are using INT2 as a non-destructive reset which, as far as any running programs are concerned, is just like reset and the system will re-initialise but the memory will not be cleared.

ALWAYS use INT2 for reset unless one of your programs has corrupted the Monitor's stack. Only then should you press manual reset or carry out a Power On Reset by switching the machine off and on.

Finally, a word about clock frequency. You can use any frequency crystal for X1 but the ideal value is 7.20 MHz which should not be exceeded. Lower frequency devices are fine but the computer will operate proportionally slower. If you put in a higher frequency crystal, not only will you run into memory access time problems, but the computer will also be operating at a rate faster than the VDU can handle. The Monitor program has provided the maximum permissible print out rate for a clock frequency of 7.20 MHz.

PORT AND CHIPS

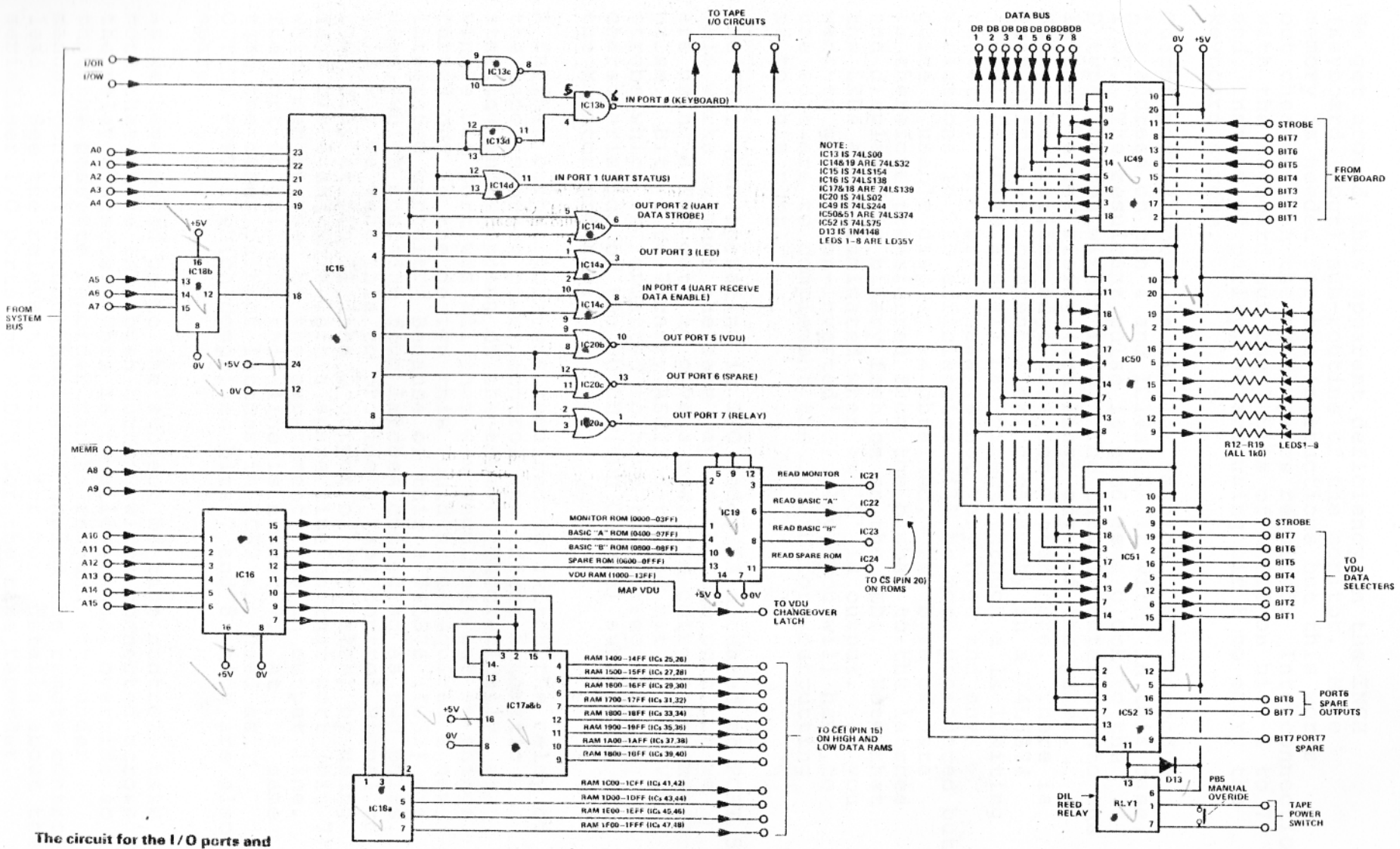
During an INPUT or OUTPUT instruction cycle the CPU will generate the address of the I/O port required on the 8 least significant bits of the address busbar. This has to be decoded to provide a single line signal which will activate the port. It is not sufficient to provide this address on its own because there is no way that the port can tell whether the select signal has come from a genuine port select instruction or whether it is the low order byte of a memory read/write cycle. Futhermore, there are times within the machine cycle when the address busbar can be in a transient or high impedance state which could cause indeterminate address information to be decoded by the port select circuits.

To prevent all these problems and also to differentiate between input and output ports, the decoded port line is gated with either the $\overline{I/OR}$ or $\overline{I/OW}$ control line. One or other of these lines goes to '0' AFTER the ports select address has been placed on the busbar and terminates BEFORE the address data changes. The pulse, so produced, is of the correct duration to strobe the I/O data on the data bus into or out of the port in question. Take, for example, the control of the Keyboard INPUT port. The port itself is simply an eight wide set of non-inverting tri-state buffers permanently connected to the data bus.

Pins 1 and 19 enable the putput of the port when they go to level '0'. Normally these pins are at '1' and held there by the output of IC13b, and keyboard data cannot affect the data bus. IC18b and 15 between them, allow 16 lines to be uniquely decoded from address bits 0 to 7. We use only 8 ports on the main board so part of this facility is redundant, hence not all the outputs from IC15 are used. IC18b is a 2 to 4 line decoder operating here as a 3 input NAND gate. When address OOH is present on the bus, pin 1 of IC15 goes low which points to Port 0 (the Keyboard). This signal is ORED with $\overline{I/OR}$ by IC13c, d and b, so when there is coincidence, IC49 receives '0' on pins 1 and 19. Whatever data is coming from the keyboard is transmitted onto the data bus and then accepted by the CPU as genuine input data. The only reason for using three NAND gates to provide the OR function is to use spare capacity in partly used ICs.

While on the subject of the keyboard port, those more experienced in computer techniques might wonder why we use only ONE port for the keyboard instead of having a second one to check the status. The reason is economy - both of components and space utilisation on the PCB.

KBD PORT



The circuit for the I/O ports and memory select.

We get around this apparent deficiency in the INCH (Keyboard Input) sub-routine of the monitor. More memory is needed for the instructions but this works out cheaper than the extra gates required. Interconnections with the keyboard put the 7 bits of ASCII on bits 1 to 7 and, instead of parity, we are using bit 8 to carry the keyboard strobe.

Output port 3 works in similar fashion. IC15 decodes its address on pin 4 and IC14a ORs it with, in this case, $\overline{I/OW}$. The resultant pulse is used as a clock to the D type latches within IC50. The data is entered into the latches on the rising (trailing) edge of the pulse. Using the trailing edge does not matter here. There is just sufficient current sinking capacity in a 74LS374 (IC50) to drive a small LED direct through a 1k limiting resistor. The brightness is not very high and any variations in LED efficiencies will be more pronounced but quite a usable display is obtained. The byte of data is therefore transferred from the busbar to the latches and displayed in binary fashion on the LEDs. Note that the LEDs are illuminated when an '0' is output. If you want to get a 'non-inverted' display, you will have to use software to complement the data before outputting it to this port.

The VDU, when operating in I/O mode, is situated at PORT 5. This works in much the same way as the LED port but we are using a NOR gate to give a positive going port enable pulse. Bits 1 to 7 carry ASCII data and bit 8, the VDU strobe which is formatted to have the correct timing characteristics by the OUTCH (VDU Output) subroutine of the Monitor program.

A further output port was required to switch the relay of the tape recorder power control (to effect automatic starting and stopping of the tape). Theoretically, a single bit port was all that was required, but as things turned out in the design this would have required a new integrated circuit. Because of this a 74LS75 (IC52) was used, which contains four latches connected as two pairs. This way we were able to provide a tape control signal to the relay at pin 11 (the $\overline{Q}$ output of one latch) by using data bit 8 and this left a spare line on that port (bit 7) which can be used by the experimenter as an output line. The port to call for this line is number 7. At the same time the other pair of latches in IC52 are used as OUTPUT PORT 6 which comprises bits 7 and 8 which are also spare.

As we have moved on to the subject of tape control, take note that there is a push button switch connected across the relay contacts. This is to allow manual override so that the cassette recorder can be rewound etc. under manual control without having to unplug the remote control lead. See the relevant section for more details about the serialiser I/O ports and MODEM for the tape recorder.

The memory of TRITON comprises four 1k blocks of ROM, one 1k block of VDU RAM and twelve 256 byte blocks of Read/ Write RAM. The high order addresses are used to decode individual lines which enable each block while low order addresses point to a specific location within the previously decoded block.

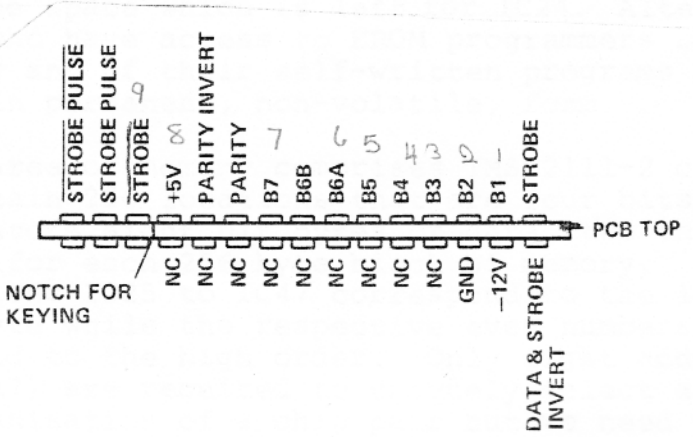
IC16 is a 3 to 8 line decoder but we are able to use it to decode, uniquely, eight individual blocks of 1k from the six most significant address lines. This is made possible by using A15 in inverted form and the internal gated select inputs of the 74LS138. The four lowest order selected lines correspond to memory blocks which start at 0000H, 0400H, 0800H and 0C00H respectively, and these hold the MONITOR, BASIC 'A' and BASIC 'B' read only memories. The block starting at 0C00H is a spare block reserved for ROM expansion. The line decoded at pin 11 or IC16 addresses the block of VDU RAM and the remaining three lines are fed to three 2 to 4 line decoders, ICs 17 and 18a, along with address bits A8 and A9.

The last three decoders break down the remaining 1k blocks into 12 blocks, each containing 256 bytes. Each of these 12 lines goes to a specific pair of random access memory integrated circuits that form the main work area of the computer.

Except for the ROMs, gating with $\overline{\text{MEMR}}$ and $\overline{\text{MEMW}}$ is carried out within the memories themselves. The 2708 read only memories contain only a chip select input and it is necessary to gate the $\overline{\text{MEMR}}$ control signal with each of the chip select lines prior to making connection with the appropriate pin. This gating is carried out by the quad 2 input OR gates contained within IC19.

Breaking the main RAM area down into 256 byte blocks made the design of this decoder slightly more complex than it would have been, had 1k blocks been used throughout, but the decision to do this was made on the grounds of economy as already explained.

Keyboard Connector (AS VIEWED FROM REAR OF BOARD)



We have excluded the VDU memory from this description because it is rather special and is covered in detail later.

The illustration has been abbreviated as most of the memory circuitry is a repetition of the same theme. You can clearly see the difference between the ROMs and Read/Write RAMs. There are four of the former - all 2708s - but in the standard machine only three are used immediately. A spare socket is left for the fourth (IC24) so that any expansion firmware can be simply plugged in. The 2708 is an ultra violet erasble ROM which contains 1,024 (decimal) bytes of memory each being 8 bits wide. To access a specific byte within it, you need a 10 bit address and A0 through A9 are used for this purpose. The eight output pins are tri-state which are enabled by an '0' on pin 20 (the chip select input). The respective outputs from each of the ROMS can therefore be commoned together on the data bus. The 'Programming Enable' pin (18) is only used when the devices are being programmed and therefore is left disconnected within the system. We use the block select signal gated with $\overline{\text{MEMR}}$ to provide the Chip Select strobe for the ROMs (this is described elsewhere).

The Monitor program is located within IC21 which starts at address location OOOOH so that the computer will always go through a firmware initialisation routine when switched on. The Power On Reset ensures that the first instruction the CPU reads will be the one located at OOOH. BASIC is located within ICs 22 and 23. Those new to computing should understand that these ICs MUST be placed in their correct respective sockets because there are many JUMP and CALL instructions which link the Monitor to BASIC and vice versa as well as internal jumps. Transam Components Ltd supplies the ROMs ready programmed and as the ICs all have the same type number, they will be supplied with self-adhesive labels identifying which is which. They should be designated MONITOR V4.1 for IC21, BASIC L4.1 'A' for IC22 and BASIC L4.1 'B' for IC23.

Transam will be offering many useful programs in the future to use the space which is left for IC24. Alternatively, readers who have access to EROM programmers can use this space for any of their self-written programs that are worth holding in permanent, non-volatile, form.

The RAM area of memory comprises TMS 2111-2 chips. These each contain 256 locations that are four bits wide. As we need to store eight bit bytes of data, two chips are required for each 256 byte block of memory. The odd number designations IC25 to IC47 correspond to the low order nibble of the byte while the respective even numbers (IC26 to IC48) correspond to the high order. Only eight address lines (A0 through A7) are required to uniquely select a byte within this organisation of a chip pair but we need to specify which pair by means of the Chip Select lines.

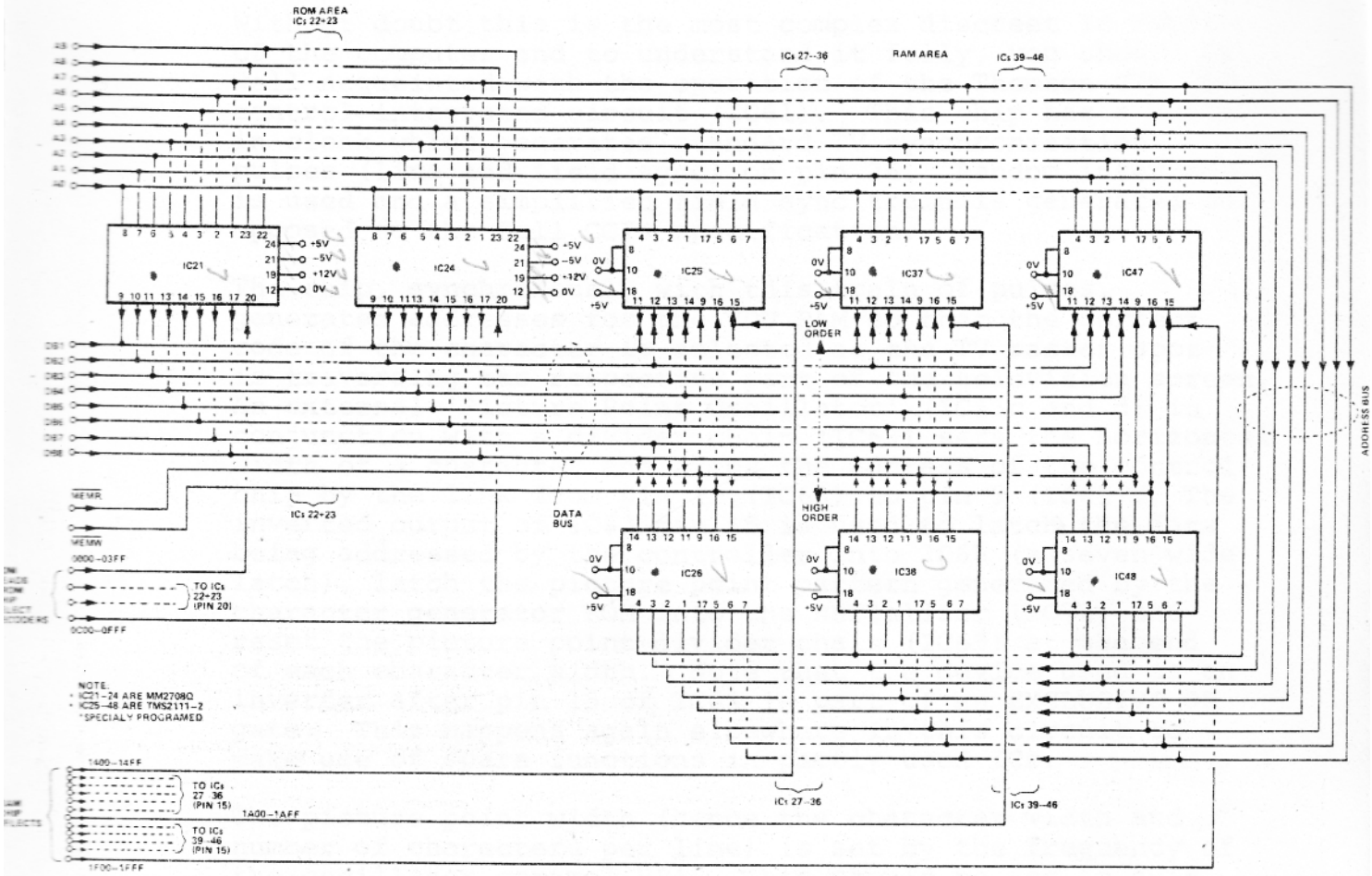
The 2111s have internal chip select and Read/Write gating so we are able to drive the MEMR and MEMW inputs direct from the control busbar.

ICs 25 and 26 form the first block of RAM which extends from address 1400H to 14FFH. This is used for the stack of the Monitor (set at 1480H) together with sixteen bytes (1400H to 140FH) as a 'SCRATCH' area and an unspecified length going upwards from 1410H as an input buffer. These locations are all specified by the standard Monitor and, of course, could be different if the user were to have a different Monitor. The rest of this block is taken up by part of the table and stack area for BASIC L4.1 which extends right the way through the next block (ICs 27 and 28).

With this information you can see that you could JUST operate the machine under the control of the Monitor only by having ICs 25 and 26 in place, leaving the rest of RAM blank. If you operate under the control of BASIC L4.1 your work area does not start until the third block of RAM (ICs 29 and 30). From hereon upwards you have flexibility in how many RAMs you care to insert (depending on your budget). Remember they have to be added in pairs and any pair that you add should go into the next highest IC designation - otherwise you will have gaps in your RAM sequence.

Transam supplies eight 2111 1Os (1k) in the standard kit of parts which is sufficient to provide all the necessary stack, tables and scratch areas with 512 bytes left as work area for BASIC. With this capacity you will be able to get the machine up and running and enjoy one or two short, experimental educational programs written in BASIC, but for the more exciting and challenging games, you will soon find you need to extend this area. By working in 256 byte blocks, this can be done in easy stages over a period of time.

REMEMBER, when you insert the RAMs, start at the designation for IC25 and work from there upwards as you fill up the RAM capacity of the board. Alternative manufacturer's devices may be used here but it is important to check that they have the same or shorter access time as the TMS 2111-2 which is specified. Note that there is not always uniformity in type numbering between different manufacturers.



Circuit diagram of the ROM and RAM circuitry. Note that in the basic machine IC 24 is omitted as are ICs 33-48.

Without doubt this is the most complex discreet IC region of the computer and to understand it fully, you should be well acquainted with the operation of the Thomson-CSF VDU control integrated circuit (IC61). This chip has a built in clock which generates standard TV synchronisation pulses (line and field sync) on pin 26. Random interlace is used and a simplified field sync train is generated as opposed to the full CCIR specification.

The chip, synchronously with this train of pulses, generates addresses for the VDU RAM so that the correct code of the character is selected as the TV raster spot is traversing the respective part of the television screen. An external 'Picture Point Oscillator' (IC55c and d) in conjunction with a divider chain (IC63) sets the horizontal width of a character and steps the address of the control chip by the link from pin 12 (IC63) to pin 9 (IC61). The inverted output of IC63 pin 15 is used to latch the data being addressed by the controller into IC68 (a seven wide latch), latch the picture point pattern generated by the character generator ROM into the serialiser (IC72) and reset the picture point divider chain (IC63) at the end of each character width. Note that the device used as an inverter after pin 15 of IC63 is part of an EXCLUSIVE OR gate. This happens again elsewhere in this circuit to make use of spare functions in partly used ICs.

The picture point width (hence the character width and number of characters per line) is set by the frequency of the oscillator control VR1. This should be set to suit your television screen so that there is a small margin to the right of the 64th character - in this way, no characters are lost.

In the standard application of the Thomson-CSF chip ASCII, data held in the RAM output latches (IC68) is fed to the column address inputs of an RO-3-2513 character generator ROM. Note that our circuit does this but has been extended in this area. Firstly we are using a 7 bit wide RAM to hold the FULL ASCII code - we need this to provide capacity for graphics. The outputs of the latches feed both the standard alpha-numeric character generator (IC69) AND a specially programmed ROM (IC70) which contains picture point data for the 64 graphic symbols. We use the EXCLUSIVE OR function (IC62d) on bits 6 and 7 of the ASCII code to select either the graphics or alpha-numeric ROM. The select signals go through further gating (ICs 67a and d) to ensure that the integrity of the cursor generating pulse (pin 15 of IC61) is not corrupted.

Three further address lines from the VDU controller (pins 11, 12 and 13) address the picture point data ROMS in both ICs 69 and 69. Due to a limitation caused by the internal operation of the Thomson chip the row address code OOO is output for the top row AND the bottom four rows of the character cell. Normally rows O, 8, 9, lO and 11 are used to provide inter line gaps for alpha-numeric displays while rows 1 to 7 carry alpha-numeric picture point data. We have had to take this into account when designing the font of graphics symbols - some of which cannot fill the complete character cell rectangle on the screen. In simple terms, the limitation means that if a graphic demands a bright picture point in row O, there will have to be bright points on rows 8, 9, lO and 11 in the same column. Look at the table of graphics characters and you can see how we have adjusted the graphics to suit this restriction.

Further complications caused by this limitation are that a graphic must not appear on the topmost line of the tele-vision screen if that graphic contains picture points in its top row. The Thomson chip requires there to be zeros present here in order to derive field blanking. We could have got round this problem with extra gating but this would have been at the expense of simplicity. The restriction is, however, not much of a problem and in some instances can actually be used as a bonus. For example, by filling the top line of the screen with completely white character cells you will make the whole screen go white above the top line and below the 16th line of the screen, giving a frame to anything printed between the two extremes.

We have got over a similar problem (involving line blanking) by gating the video output with the INI function (pin 26 of IC61) in IC71b. Without this, any graphics symbol having a picture point in its most left-hand column would have caused a 'wrap around' white line that interferes with the d.c. level of the line sync pulse. The only problem that remains in this respect is that you will now get a single 'extra' picture point showing to the right of the 64th character down a line if you use a graphic in the most left-hand position of a line. This does not happen with all graphics - only those that have picture points in their most left-hand column.

The five outputs from the alpha-numeric ROM are wire ORED with five of the eight outputs from the graphics ROM and held high via pull up resistors R22 - 26. They are then fed to the correct positions in the serialiser shift register IC72. Note that the remaining three outputs from the graphics ROM have to be ANDED with a signal defining whether or not the character is a graphic (done by ICs 71a, c and d). This is to ensure that if alpha-numerics are printed there is a correct inter-character gap.

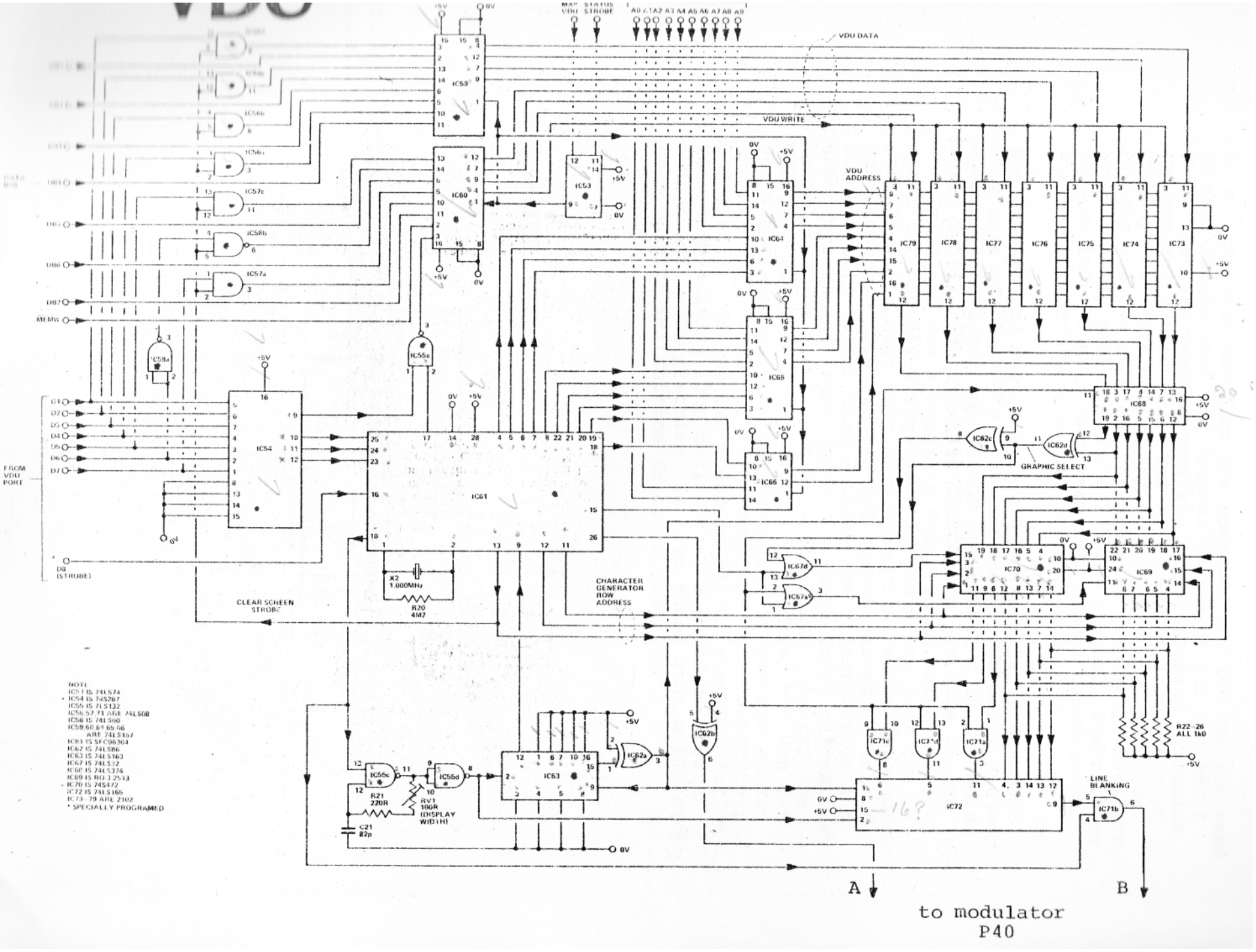
So far we have not explained how the VDU RAM is addressed by the control chip. We will deal with that now.

This is more complex than the standard application because we are allowing the CPU to memory map the VDU RAM to provide fast and possible animated graphics. To do this we have had to make allowance for the CPU to take over addressing control of the VDU RAM. This is done in a very simple way. We have taken all the address lines from IC61 and their equivalents from the computer busbar to a set of data selectors (ICs 64, 65 and 66). If the computer addresses the VDU memory location (any address between 1000H and 13FFH) the block select line (called MAP VDU in this illustration) is activated. This, of course, could happen if ever the address busbar went into a high impedance state (during HOLD, etc) so to prevent any spurious pulses affecting the operation, we gate the VDU block select line with $\overline{\text{STSTB}}$ which only occurs when valid address information is on the busbar. We do the gating in a D type latch so that during the complete cycle and at all other times, the data selectors hand over address control to IC61.

A similar transfer of responsibility takes place between the normal input data to the VDU (which gets to it via an output port) and the main computer data bus. In this case the data is selected by ICs 59 and 60. These also receive their changeover instruction from the changeover latch IC53. Note that we also have to do a changeover between the internally generated memory write command (pin 17 of IC61) and the computer's $\overline{\text{MEMW}}$ strobe. This is done within IC60.

It only remains to describe the gates on the VDUs internal data lines and IC54. The former are used to force the ASCII code for 'Space' onto the data lines when pin 13 (IC61) is at '0' in coincidence with a writing pulse to the VDU memory. This is to allow for the very useful internal function provided by the Thomson chip to clear the screen and reset the cursor in one operation.

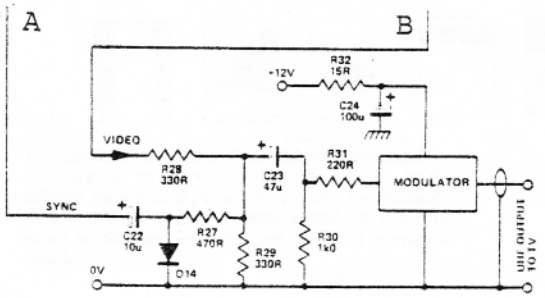
The VDU controller carries out a number of non writing functions as well as entering and addressing data within its memory. By using some of the ASCII codes as control, it is possible to do such things as move the cursor in steps to any position on the screen, reset the cursor, carry out a line feed or do a carriage return clearing ONLY the unused part of the line. There are also a couple of control codes that we wish the VDU to ignore - 00H and 04H respectively - these are NUL (or no operation) and EOT (end of text) flags. Recognition of all these special codes is carried out by the VDU CONTROL ROM (IC54). This has had to be specially programmed for the TRITON and the standard Thomson-CSF ROM (designed to go with their chip) will not do in this application.


VDU DATA
VDU WRITE
VDU ADDRESS
MEMW
FROM VDU PORT
CLEAR SCREEN STROBE
CHARACTER GENERATOR ROW ADDRESS
GRAPHIC SELECT
LINE BLANKING
X2 1.000MHz
R20 4M7
R21 220R
RV1 100R (DISPLAY WIDTH)
C21 82p
R22-26 ALL 1k0
A
B
to modulator P40
NOTE
IC53 IS 74LS74
IC54 IS 74S287
IC55 IS 74S132
IC56,57,71 ARE 74LS08
IC58 IS 74LS00
IC59,60,61,65,66 ARE 74LS157
IC61 IS SFC96364
IC62 IS 74LS86
IC63 IS 74LS163
IC67 IS 74LS32
IC68 IS 74LS374
IC69 IS RO-3-2513
IC70 IS 74S472
IC72 IS 74LS165
IC73-79 ARE 2102
* SPECIALLY PROGRAMED

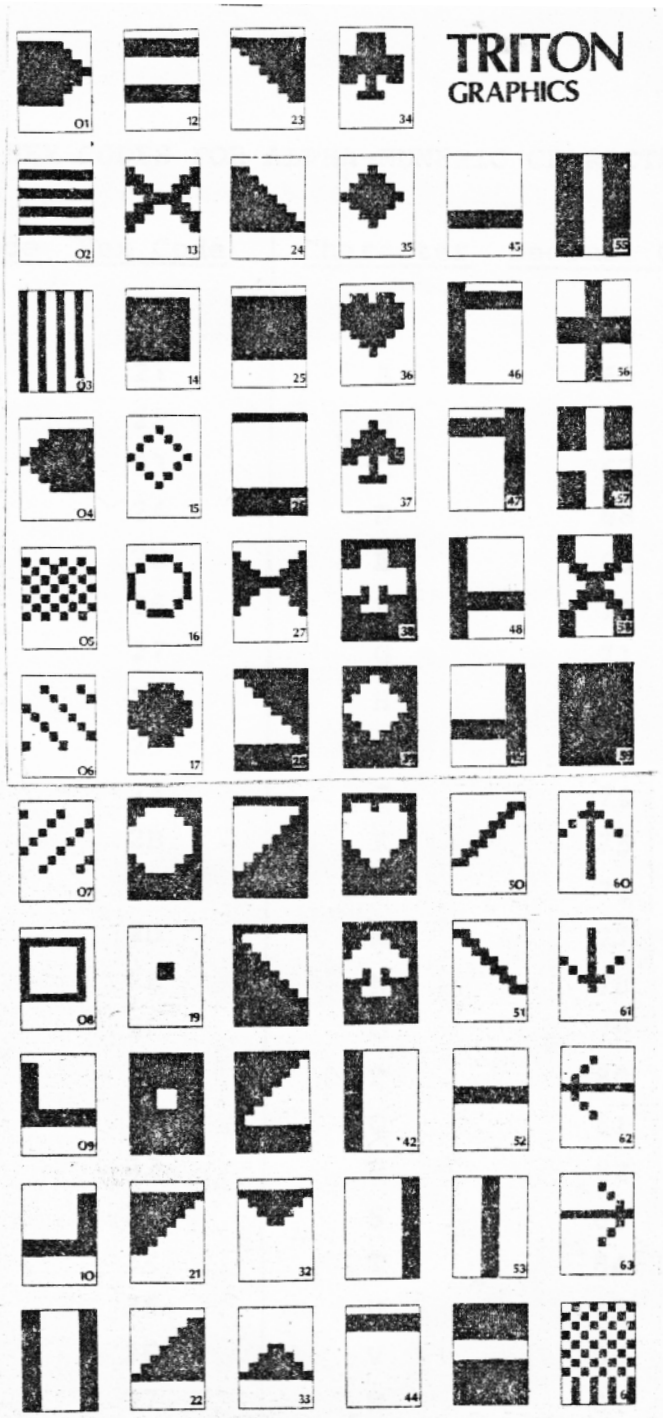
To get the best use from the computer and its VDU you need to know hexadecimal and decimal values of all the ASCII codes that are used to generate alpha-numerics, graphics and control characters. You also need to know which of the keyboard keys correspond to each graphic character. To help you, we show all the graphics with their respective codes and key names in the following tables. There is no need to learn them - just keep this Manual beside your TRITON computer whenever you use it.

Even though you may be a software specialist, you MUST take note of the VDU's printing speed. Normally you may output a character to the VDU for printing in I/O mode every 8.3mS. The standard TRITON monitor errs on the safe side and has a built-in delay which outputs a character roughly every 9mS. If you write your own software, you must take this speed limitation into account. Furthermore, there are two I/O operations which take a considerably longer time - these are 'Clear Screen and Home Cursor' and 'Home Cursor'. These instructions must be followed by a delay of at least 132mS. Again, the TRITON's monitor makes allowance for this, but you can get direct access to these functions if you use either the 'PRINT CONTROL' or 'VDU' commands which exist in BASIC L4.1. If you use these in BASIC, you MUST follow them with a delay loop having a time constant greater than 132mS. (In practice, we have found that a 200 step 'FOR - NEXT' instruction is quite safe.)

Modulator



GRAPHIC	DEC.	HEX	
1	0	00	
2	1	01	CONT A
3	2	02	CONT B
4	3	03	
5	4	04	
6	5	05	CONT E
7	6	06	CONT F
8	7	07	CONT G
9	8	08	
10	9	09	
11	10	0A	
12	11	0B	
13	12	0C	
14	13	0D	
15	14	0E	CONT N
16	15	0F	CONT O
17	18	10	CONT P
18	17	11	CONT Q
19	18	12	CONT R
20	19	13	CONT S
21	20	14	CONT T
22	21	15	CONT U
23	22	16	CONT V
24	23	17	CONT W
25	24	18	CONT X
26	25	19	CONT Y
27	26	1A	CONT Z
28	27	1B	
29	28	1C	
30	29	1D	
31	30	1E	CONT ↑
32	31	1F	CONT =
33	96	60	
34	97	61	a
35	98	62	b
36	99	63	c
37	100	64	d
38	101	65	e
39	102	66	f
40	103	67	g
41	104	68	h
42	105	69	i
43	106	6A	j
44	107	6B	k
45	108	6C	l
46	109	6D	m
47	110	6E	n
48	111	6F	o
49	112	70	p
50	113	71	q
51	114	72	r
52	115	73	s
53	116	74	t
54	117	75	u
55	118	76	v
56	119	77	w
57	120	78	x
58	121	79	y
59	122	7A	z
60	123	7B	{
61	124	7C	!
62	125	7D	}
63	126	7E	~
64	127	7F	



The table shows the decimal and hex codes associated with the Triton graphics and, where applicable, the key on the keyboard. The symbols may be used within a BASIC print statement or with the OUTCH monitor routine.

DECIMAL AND HEX CODES FOR ALPHA-NUMERIC CHARACTERS

Character	Decimal Code	Hex Code	Character	Decimal Code	Hex Code
Space	32	20	@	64	40
!	33	21	A	65	41
"	34	22	B	66	42
#	35	23	C	67	43
$	36	24	D	68	44
%	37	25	E	69	45
&	38	26	F	70	46
'	39	27	G	71	47
(	40	28	H	72	48
)	41	29	I	73	49
*	42	2A	J	74	4A
+	43	2B	K	75	4B
,	44	2C	L	76	4C
-	45	2D	M	77	4D
.	46	2E	N	78	4E
/	47	2F	O	79	4F
Ø	48	30	P	80	50
1	49	31	Q	81	51
2	50	32	R	82	52
3	51	33	S	83	53
4	52	34	T	84	54
5	53	35	U	85	55
6	54	36	V	86	56
7	55	37	W	87	57
8	56	38	X	88	58
9	57	39	Y	89	59
:	58	3A	Z	90	5A
;	59	3B	[	91	5B
<	60	3C	\	92	5C
=	61	3D	]	93	5D
>	62	3E	↑ or ^	94	5E
?	63	3F			

DECIMAL AND HEX CODES FOR NON-PRINTING CONTROL CHARACTERS

Control Function	Keyboard Key	Decimal Code	Hex Code
NUL (no-op for printer)	CONT @	0	00
EOT (End of Text)	CONT D	4	04
Backspace	CONT H	8	08
Step cursor RIGHT	CONT I	9	09
Line feed	Line feed or CONT J	10	0A
Step cursor UP	CONT K	11	0B
Clear screen/Reset cursor	CONT L	12	0C
Carriage return/clear line	CR or CONT M	13	0D
Roll screen (top line rolls round to bottom line etc)	ESC or CONT [	27	1B
Reset cursor	CONT \	28	1C
Carriage return (no clear)	CONT]	29	1D

TAPE I/O (FOR GOOD MEASURE)

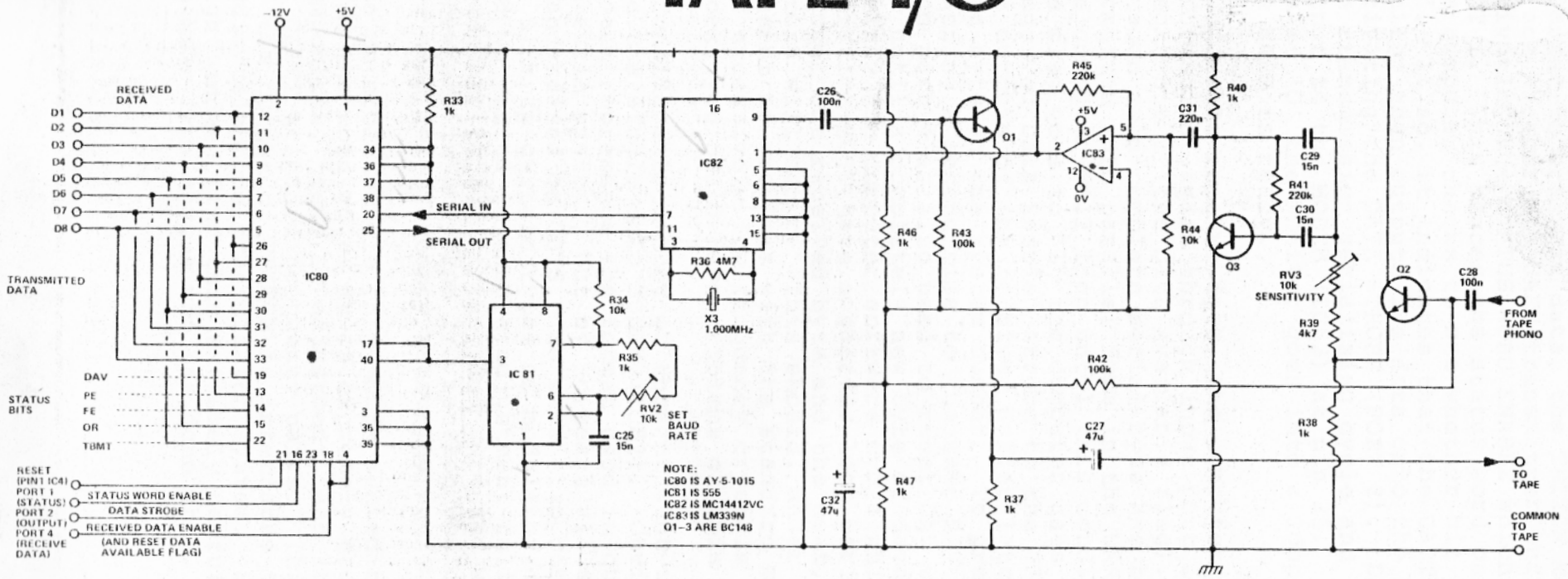
The AY-5-1013 Universal Asynchronous Receiver transmitter features tri-state outputs for received data and all status bits. This makes it ideal for busbar orientated systems and obviates the need for any other I/O ports to interface it to the main data bus of the TRITON. Note that respective bits of the data in and data out terminals of the chip are commoned together before joining the TRITON's data bus. The Status bits of the UART are similarly commoned with the DAV (Data Available) bit tied to bit 1 on the bus, PE (Parity Error) to bit 2, FE (Framing Error) to bit 3, OR (Over Run Error) to bit 4 and TBMT (Transmitter Buffer Empty) to bit 5. There is not reason why all these flags and error messages should not be used although TRITON's standard Monitor only samples DAV and TBMT. The monitor software, in its present form, does not check for errors even though parity is recorded - incidentally, ODD parity is used.

The DAV and TBMT flags are used to tell the computer when the UART has received and has ready a complete byte of new data or when the UART has finished a current serialising cycle and is ready to accept a new byte for transmission. In actual fact the UART will accept a second byte while it is still transmitting the first, due to the double buffering nature of its transmitter buffer.

The format of serialised data in TRITON is a START bit, 8 data bits, a parity bit and 2 STOP bits. These are transmitted at a rate of 300 baud set by the clock comprising IC81 (an NE 555). Baud rate is adjustable by about $\pm$ 50 percent by means of VR2 and, of course, it is important that this is accurantely set if tapes from other sources (recorded in TRITON's format) are to be played back. To obtain a rate of 300 baud, the oscillator must run at precisely 4,800 HZ and ideally this should be set on a frequency meter. We are aware that some people may have difficulty in gaining access to one of these and are prepared to supply specially recorded standard test tapes containing a few minutes of test data. If this is played into your computer with a simple machine language test routing typed into RAM you should be able to adjust VR2 until the data is reliably entered and displayed on the VDU. This will not guarantee that your clock is at 4,800 HZ because your recorder may be running at the wrong speed but it does ensure that your TRITON/Cassette Recorder pair operate TOGETHER at 300 baud.

While on this subject, you may experience problems when using battery powered tape recorders and some the cheaper varieties. For reliable operation, you should always use the SAME recorder/player and preferably a good quality mains powered machine. Batteries run down after sustained use and even the better quality recorder motors start to run more slowly when this happens. More about the tape recorder will be said later.

TAPE I/O



The tape I/O section of the Triton system.

For setting up and running at 300 baud, the TRITON calibration tape is available on cassette.

In order to transmit data, the TRITON Monitor first checks to see whether the UART transmitter buffer is empty by activating the STATUS WORD ENABLE which is, in effect, PORT 1. This places the status word on the data bus and the computer checks to see whether bit 5 (TBMT) flag is at '1'. If so, it indicates that the UART is ready and the Monitor then outputs its data onto the busbar while activating the DATA STROBE (PORT 2). DATA STROBE starts the transmission serialising cycle and the serial data is output to the MODEM (IC82) at pin 25. If the TBMT flag was at '0' the Monitor goes into a loop and waits until the UART is ready.

In order to receive data, the computer asks for status information, again through input port 1, but this time checks bit 1 (the Data Available flag). This goes high as soon as a complete serial byte has been received and formatted into parallel form in the UART's output latches. The computer will loop until this condition is met. When the flag goes to '1' the computer uses port 4 to send a 'Received Data Enable' strobe to the UART. This enables the outputs of the receiver buffer latches and places the data byte on the system busbar. To prevent the system reacting a second time to the same DAV flag, the pulse from port 4 is also used to reset DAV which then stays low until a completely new byte of data is received. Clearly the computer software cycle which carries out this operation MUST have a shorter loop period than the period between one received byte and the next, otherwise overun errors will occur. This should, under normal circumstances, present no problems with the standard monitor but if you use excessively low frequency crystals for the computer's clock, you could have problems of this nature.

A considerable amount of thought went into the decision to use the single chip MODEM (IC82). Obviously the CUTS system represents the format used in quite a number of domestic systems, but by no means all systems are standardised on CUTS. We were also concerned about the need to line up certain types of modulator/demodulator systems and we wanted to ensure adjustment reliability. It is necessary to adjust the baud rate and any further adjustment elsewhere in the Tape I/O section would inevitably require the need for frequency measuring equipment.

The Motorola single chip MODEM seemed highly attractive from the word go as it is extremely economical on external components, needs no adjustment, and it operates on USA or CCITT standard frequency pairs. This is important only if the output of TRITON is interfaced to an international radio/telephone transmission system. Some radio amateurs might indeed find this extremely useful and we would like to hear from anyone who has done this.

The MC14412VL is such a versatile chip that it was difficult to decide in which mode it should be used. Eventually, in order to have a frequency pair that would give best reliability with most tape recorders and to allow the MODEM to receive at up to 600 baud (not that this is used at present), we opted to go for the USA standard 'originate' mode in which the transmitted frequency pair is:

MARK ('1') = 1,270 Hz

SPACE ('0') = 1,070 Hz

Clearly we need to be able to demodulate the same pair of frequencies and so have to operate in Simplex mode - hence pins 2, 10 and 14 of IC82 are allowed to be '1'. Internal pull up resistors within the chip do away with the need for external pull ups hanging on these pins. Pin 2 actually is the 'Self Test' control input which makes the MODEM's receiver demodulate the same frequency pair that is being transmitted. Keeping this active prevents any ambiguaity as to whether one is 'originating' or 'answering'. It also allows you to carry out an ECHO self test on the MODEM by simply connecting the output to the tape recorder to the input from the tape recorder. To do this in practice would need a special test routine written in machine code as the Monitor does not contain such a test.

The MODEM interfaces directly with the UART and only needs a crystal and resistor to lock it to the correct frequency pairs. It is MOST important that a crystal of exactly 1.0000 MHz is used here otherwise you will be able to use pre-recorded tapes. The transmitted carrier of the MODEM is an eight level digitally synthesised sine wave of about 300mVrms which is buffered by TR1 before being fed via C27 to the tape recorder phono input.

It is quite a high level signal so avoid feeding it into microphone inputs, etc. The auxiliary, phono, radio or crystal cartridge inputs of most tape recorders should match satisfactorily and give flexibility in adjusting recording levels. We have found in pracice that the signal should be recorded in exactly the same way as you would handle hi-fidelity music. That is to say, you should not attempt to saturate the tape by deliberately over-recording (some systems recommend this). The output carrier is present at the output terminal all the time, so it is extremely easy to preset your recording level so that the VU meter is just indicating maximum modulation. The reason for treating the signal as Hi-Fi is that during playback the MODEM likes to see a near pure sine wave.

You should use an output FROM your tape recorder which provides in the order of 300mV on playback. There is considerable flexibility here and we have found that anything between about 40mV and 500mV will suffice. Preferably you should use the Phono Output from the recorder rather than the extension speaker socket. The former will always be at the same level whereas the latter depends on volume and tone control settings. With Philips equipment, and many others, this also means that you can monitor the signal over the loud-speaker as it is playing back and can manually count through the files while the computer is searching for a header. You can also keep the volume low without affecting the playback level.

To carry out a demodulation satisfactorily, the MODEM IC requires a very precise unity mark/space waveform at pin 1. The tolerance on the mark/space ratio has to be better than $\pm$ 4%. If the carrier being played back from the recorder carries any harmonic distortion, this will be difficult to convert to a square wave of the above specification - hence the need to record in Hi-Fi. To further purify the sine wave it is amplified and filtered by TR3. To some extent the input sensitivity can be adjusted by VR3 but under normal circumstances (within the range of input voltages mentioned above) this should always be set in its mid-point position. The high purity sine wave at the collector of TR3 is fed to IC83 which is a zero crossing comparator which will sense the zero crossing of a sine wave to within about 3mV. With a good input signal this results in a square wave which more than adequately meets the input specification of the MODEM. If the input signal strength falls too low, the 3mV tolerance of the comparator starts to become more significant and variance from unity mark/space ratio can occur. If you have to err to one side or the other, always aim to provide a higher, rather than lower, input signal - but do not overdo it.

No provision is made to combat oxide fall out, hence you should always use good quality tapes. This, coupled with the use of a good recorder and close adherence to the above instructions (including the correct setting of the baud rate) should give you a very reliable, simple to make and easy to adjust tape I/O system.

INTRODUCTION TO MACHINE CODE PROGRAMMING

If you have built the TRITON computer yourself, you will want to get the best out of it, and there are several modes in which it can be used. You can write and run programs in BASIC, and you may wish to record and recall these to and from tape. To do this you need to know something about the Monitor. You may wish to use the computer for control purposes for which you need to write machine language instructions. To do this it is essential to go through the Monitor - possibly making use of some of its in-built subroutines (see the section on Monitor Utilities.)

For those learning about computers it is a very good idea to get a grasp of machine code. It is not difficult to learn and can make life much more exciting as you will be able to get the computer to respond much more quickly than it will through a Basic Interpreter. Sometimes you can get several thousand times the speed for certain operations. To do this on the TRITON you must operate through the Monitor.

TRITON's standard Monitor is a program written in machine code which is held in ROM starting at address location 0000H. Its purpose is primarily to initialise the machine and to give it an elementary intelligence so that you can communicate with it. For this reason the machine must start off at the beginning of the monitor program every time it is switched on. The first machine code instruction that the computer sees in the Monitor sets the STACK without which it would be impossible to do much in the way of decision making via nested subroutines. It then proceeds to look at the next instruction which, in the case of TRITON, enables the interrupt operation (if ever it is needed) to be used.

The following instruction is a JUMP which leads on to the first of the main routines called SCANMEM. This routine points to address 1600H and writes FFH into it. It then reads back the value from that address and checks that the FFH was actually stored. Furthermore it writes 00H into the same location and checks that too. This is a check that the memory is there and working which, at the same time, leaves the memory location clear (i.e. containing 00H). The monitor routine then steps up one memory byte - to 1601H and does the same.

This process is carried out on every successive location from 1600H upwards until the computer finds an error. The address of the location where the error occurred is most likely to be the top of the RAM work area, but it could be the address of a faulty IC. In either event this top address is written into a pair of RAM bytes used by BASIC L4.1 to tell BASIC how much work space is available. The two bytes of memory used for this are 1481H and 1482H. As a general rule, whenever two byte instructions or data are written into memory, the 8080 microprocessor expects to see the least

significant byte in the lower value of the two addresses. Thus if 2000H was the location where memory ended - as found by SCANMEM - location 1481H would contain 00H while 1482H would contain 20H.

After checking the memory, the Monitor initialises the computer which acknowledges this on the screen with its standard message:

TRITON READY

FUNCTION? P G I O L W T

The Monitor then goes into a keyboard loop and the computer effectively waits for you to tell it what to do. This will depend on which key you depress. The letters it expects you to type are those shown in its acknowledgement, and are abbreviations for seven different primary operations you can do with the Monitor. These are:

P = Inspect any memory location and, if necessary, change or insert a byte of data. When the data is entered the computer automatically steps to the next address showing what is currently there and waits to see if you wish to change it.

G = Start running a program from any specified starting address location. The computer asks you, within this routine, what start location you want.

I = Input from tape recorder. The computer asks you for the header code of the file and then searches for it. When it has been found the data is written into the computer's memory starting at location 1600H. When the flag marking the end of the record has been found, the computer re-initialises with an abbreviated form of its initial 'switch on' message.

O = Output a program to the tape recorder. This, again, asks you to give your recording a header code. The routine automatically outputs programs written in BASIC and stops when it gets to the end of file address (the address written into bytes 1600H and 1601H by BASIC). For user-written machine code programs you have to manually enter the address immediately following your last instruction into these two bytes. Tapes are ALWAYS loaded and dumped with 1600H being the start location. When dumping has finished the computer re-initialises.

L = List the machine code content of all locations starting from any specified address. The computer asks for the first address then prints out the contents of this and the next 14. It then asks MORE? and expects you to type Y otherwise it re-initialises.

W = Typewriter mode. The computer behaves just as if it was a keyboard and VDU. Anything you type is displayed on the screen including graphics. Cursor control and special VDU functions (e.g. Clear Screen, Reset Cursor, etc) all operate, but the computer responds to nothing except CONTROL C which makes it re-initialise.

T = Jump to BASIC L4.1. This command causes the computer to jump out of the control of the Monitor into the control of BASIC. See the section which is entirely devoted to this. CONTROL C will jump back out of BASIC into the initialisation condition of the Monitor.

Note that CONTROL C will, in nearly all cases, get you out of an operation and back to the initialisation condition. The only times when it fails to do this are when you are locked in a user-written machine code program loop, searching tape for a non-existant header or outputting to the tape recorder. In these three cases you will have to use Interrupt 2 (which re-initialises without clearing memory) or RESET which goes through the SCANMEM routine and erases any data in memory.

When the computer asks for its initial instruction via one of the above letters you simply have to type the letter. No carriage return is needed. If you type the wrong letter, the computer replies 'INVALID' and waits for you to try again.

Here are some examples to try with the above functions:

P Function

Type P. The computer acknowledges with PROG START = and expects you to enter a four digit hexadecimal code as a start address. Try typing in 123M followed by a carriage return (CR). The computer says INVALID and re-initialises because you entered M as one of the digits. If you make a mistake typing in a character, you have a chance to correct it by backspacing with CONTROL H until you get to the error. Overtype the error, and continue overtyping the line until you get to where you were. Now press CR - the computer will confirm the start address by retyping it, and will then follow this with a space and the hexadecimal code of the contents at that location. In this case 3E. Depress CR again and you will step to the next location and so on all the way through memory if you so desire. This is the mode for manually inspecting memory locations.

Re-initialise with CONTROL C and type P again. This time we shall use the function to enter data, so we shall start at an address in RAM. Enter 16ØØ as the start address and depress CR. This time you will see the data currently in 16ØØ which you can change by typing two more hexadecimal

digits. If you type more than two it is only the first pair that the computer recognises, so if you make a mistake use the CONTROL H key to backspace. When you press CR the new data is entered into that location and the address is stepped on. When, at a later time, you come to enter a program in machine code, you should use this method.

Here is a simple machine code program that you can type in, starting at address 16ØØ, which we can use when we test the G function. The program instructs the computer to type the numbers Ø to 9 inclusive, and to keep repeating this sequence. The left-hand column is an aide memoire of the instruction's address (you only have to type this in on one occasion - when the computer asks for the start). The second column is the mnemonic of the machine code (dashes indicate that these locations carry data to go with the previous instruction). Some mnemonics are qualified with the name of the OPERAND - usually when the instruction requires the operand to be identified in the bytes following the instruction. The next column is the data that you should type in and the remarks describe what the instructions do in everyday language.

16ØØ	MVI A	"Ø"	3E	Enter the ASCII code for Ø
16Ø1	-		3Ø	into the accumulator.
16Ø2	CALL	OUTCH	CD	Call subroutine which outputs
16Ø3	-		13	code currently in accumulator
16Ø4	-		ØØ	to the VDU.
16Ø5	INR A		3C	Increment the accumulator.
16Ø6	CPI	3AH	FE	Check if code in accumulator is
16Ø7	-		3A	one greater than that for 9.
16Ø8	JNZ	16Ø2H	C2	If it is not, jump back to
16Ø9	-		Ø2	address 16Ø2 and loop until
16ØA	-		16	9 is reached.
16ØB	CALL	PCRLF	CD	When it is reached call
16ØC	-		33	carriage return/line feed
16ØD	-		ØØ	output subroutine.
16ØE	JMP	16ØØH	C3	Jump back to beginning of
16ØF	-		ØØ	program and keep repeating.
161Ø	-		16	

The two called subroutines OUTCH and PCRLF are utilities within the Monitor.

G Function

This function permits you to run a program from any start location you care to specify. If you have already entered the above program you can try running it. Make sure you are initialised (by using CONTROL C or Interrupt 2) and then type G. The computer acknowledges with RUN PROG START=. Type in 16ØØ (the start of the above program) and as soon as you press CR the program will execute. This particular program is typical of many user programs in that it stays locked in its own loop and you cannot get out of it by typing CONTROL C. You will have to use Interrupt 2.

Try writing and running the following simple program at 16ØØ that will jump immediately back into the Monitor to re-initialise.

```
16ØØ   JMP   RSTØ   C3   Jump to restart Ø
16Ø1    -            ØØ
16Ø2    -            ØØ
```

Just enter these three instructions using P and run them using G. The computer will act as if it had been reset to a switch on condition (i.e. will clear all memory and completely re-initialise). This is an instruction which should normally be avoided or you will delete your program!

O Function

We shall deal with this before covering tape input. O transfers control to the tape output routine and it can be used for dumping either BASIC or machine code programs to tape. The only stipulation is that all programs MUST start at 16Ø2H. The reason is that the routine is designed for BASIC, and to save space we have deliberately avoided writing a second routine to deal with user programs. There are, however, no major problems.

First of all type the following short program into RAM starting at address 16Ø2. The program uses another of the Monitor utilities (PDATA) which prints out any string of ASCII codes in memory starting at a specified location. We shall use it to print out repeatedly the TRITON's initialisation message.

```
16Ø2   LXI D   Ø377H   11   Load DE register pair with
16Ø3     -             77     start address of string.
16Ø4     -             Ø3
16Ø5   CALL    PDATA   CD   Call subroutine which prints
16Ø6     -             23     string of characters and
16Ø7     -             ØØ     returns when it sees Ø4.
16Ø8   JMP     16Ø2H   C3   Jump back to start of program
16Ø9     -             Ø2     to repeat.
16ØA     -             16
```

Note that the address immediately following this program is 16ØB - we shall be making use of this in a moment. Before that, verify that the program runs by using G and then Interrupt 2 to get out of the loop (DO NOT use RESET or you will erase your program!)

During the tape output routine the software needs to know how long the program is that it is dumping. This is defined by the bytes in address locations 16ØØ and 16Ø1. The locations must contain the low order and high order bytes of the address immediately after the end of the prgram to be saved. This is done automatically by BASIC but you have to do this manually if you wish to save user programs. Use the

P function to enter this information manually in the two bytes starting at 16ØØ. These should be ØB and 16 respectively (note that the low order byte goes in first). When you have done this, re-initalise with CONTROL C and you are ready to save your program.

Connect up the audio leads of your recorder to the TRITON (including remote control if you have one). Depress the manual override button on TRITON (if necessary) and find a suitable place on a tape - preferably the beginning! Carry out a dummy run record to set the recording level against the carrier frequency of TRITON - just maximum on the VDU meter. Rewind, and get ready to start. Take your finger off the manual override button and then type O into TRITON. It will acknowledge with TAPE HEADER=. You should now enter a header code of your own choosing - but don't forget what it was! We suggest you use PRINT ROUTINE 1. Note that you can use free text alpha-numerics in the header but the length should not exceed 20 characters. Before pressing CR set the recorder to RECORD and then you can press CR. It will take about 15 seconds to record this program - including about 5 to 6 seconds at either end as guard gaps.

The recording format is as follows:

5 to 6 seconds of unmodulated carrier

64 carriage return codes as start file code

The header code

Contents of every memory location starting at 16ØØ byte by byte

5 to 6 seconds of unmodulated carrier

When recording has finished, the computer will type END and re-initialise and, if the remote control lead is used, the tape recorder will switch off. YOU must then switch off the tape recorder's own control. Hopefully you will now have your program on tape. Rewind the recorder and play back the record over its loudspeaker to check that at least something went on! If you are satisfied, you can switch the computer off at the mains. Switch it on again, and then proceed to try the Tape Input function. Note that, if you intend to save a program written in BASIC, you must do exactly the same operations except that you can omit manually entering the end of file address. To save BASIC, you must jump out of BASIC via CONTROL C in order to pick up the O function.

I Function

The procedure to input from a tape is the same for machine code programs as for BASIC. Connect up the audio leads and find the start of the file manually (if you can). If the

file you want is somewhere in the middle of the tape you will have difficulty doing this, and will have to rely on the Input function's ability to search. If you have a long tape this could take quite some time. Do not record too many files onto a tape - 4 or 5 averaging 1 minute each is sufficient (and it is easier to index the tapes). Switch on the computer, and when it is initialised type in I. The computer will respond with TAPE HEADER= and you should then type in the header code EXACTLY, including all spaces and punctuation if there were any. Before pressing carriage return, make sure that the tape is rewound to BEFORE the place where your file starts. It does not matter if you are currently sitting in the middle of the previous file, because the input routine will ignore it. Set the recorder to play, and press CR. If you can, monitor the sound of the recording. Files having the wrong header code will be skipped over, but when the computer has made a perfect match with the code entered it will accept the input data, and when it comes to an end the computer types END, switches the tape off (if remote is being used) and re-initialises.

All you have to do is run the program. If it was a machine code program you must type G followed by 16Ø2 and then CR. For BASIC, type T and then RUN, followed by CR.

L and W Functions

These have been described fully already. L lists what is in adjacent memory locations in blocks of 15 bytes (so that you can inspect the screen before it continues). At the end of a block it asks MORE? and continues if you type Y (no carriage return is needed). Any other character will be flagged INVALID, and the computer will re-initialise.

The W function has also been described elsewhere; it turns the machine into a video typewriter. No key characters are recognised, and no default conditions can arise. CONTROL C is the only way of re-initialising apart from Interrupt 2 and RESET.

MEMORY AND PORT DESIGNATIONS

To help those who wish to get involved in machine code programming at an early stage, here are the addresses of memory blocks and ports. It is assumed that you will be operating under the control of the standard Monitor program, so we also list the addresses of its main Utilities.

Memory Start Addresses

0000H - 03FFH	1k EROM	(Holds ~~standard~~ MONITOR) HUMBUG V5.1A
0400H - 07FFH	1k EROM	(Holds BASIC ~~L4.1~~ L5 "A")
0800H - 0BFFH	1k EROM	(Holds BASIC ~~L4.1~~ L5 "B")
0C00H - 0FFFH	1k EROM	(~~Spare location~~) HUMBUG V5 1.B
1000H - 13FFH	1k RAM	(VDU memory - can only be written into by computer)
1400H - 14FFH	¼k RAM	(Holds stack and tables for Monitor from 1400H to 147FH; 1480H upwards through this block will be used by BASIC ~~L4.1~~ L5 tables, otherwise is free as M/C code work area)
1500H - 15FFH	¼k RAM	(Completely reserved for BASIC ~~L4.1~~ L5 stack, otherwise is free for the user)
1600H - 1FFFH	2½k RAM	(Work area for BASIC ~~L4.1~~ L5 or user programs. Note that locations 1600H and 1601H are made use of by tape I/O routines to store End of File address - hence user programs for saving/loading to and from tape should always start at 1602H - see the section describing the Monitor)
2000H - FFFFH	56k	Available for "Off Board" extensions

Port Designations

00H	Keyboard INPUT (note special routine is necessary)
01H	Tape I/O UART Status INPUT
02H	Tape I/O UART Data Strobe (start transmission) OUTPUT
03H	LEDs OUTPUT (note LEDs are on for "0")
04H	Tape I/O UART Receive Data Enable (receive data) INPUT
05H	VDU OUTPUT (note strobe - bit 8 - has to be specially formatted by software)
06H	Spare OUTPUT (note only bits 7 and 8 are output)
07H	Relay OUTPUT (note bit 8 is used to drive tape control relay out, bit 7 is used as a spare line)
08H - FFH	Available for "Off Board" extensions

MONITOR UTILITIES

0000H	RST0	Reset address. Enables Interrupt and checks and clears memory writing number of bytes available into locations 1481H (low order byte) and 1482 (high order byte). With full main board memory in place this should read 2000H. Then goes on to initialise computer.
0008H	RST1	Reacts to Interrupt 1 clearing VDU and resetting cursor.
000BH	INCH	A CALL routine which inputs character from keyboard. Stays in keyboard loop until key is depressed.
0010H	RST2	Reacts to Interrupt 2, clearing VDU and initialising computer without clearing memory. Used for normal reset operation.
0013H	OUTCH	A CALL routine which outputs character to VDU.
0018H	RST3	Reacts to Interrupt 3 and re-vectors to 1618H for user routine.
001BH	INDATA	A CALL routine which allows string of characters to be entered to any place in memory. Start location of string pointed to by DE register pair which must be pre-loaded before calling INDATA. Returns on Carriage Return.
0020H	RST4	Reacts to Interrupt 4 and re-vectors to 1620H for user routine.
0023H	PDATA	A CALL routine which allows string of characters to be printed from any place in memory. DE register pair must be pre-loaded with start address before calling. Routine returns when it sees EOT terminator (04H) but DE steps to one address beyond terminator allowing immediate recall for further string to be printed.
0028H	RST5	Reacts to Interrupt 5 and re-vectors to 1628H for user routine.
002BH	PSTRNG	A CALL routine identical to PDATA, except that it outputs Carriage Return/Line Feed prior to printing string.
0030H	RST6	Reacts to Interrupt 6 and re-vectors to 1630H for user routine.
0033H	PCRLF	A CALL routine which prints Carriage Return followed by Line Feed, and then returns with the original contents of accumulator intact.
0038H	RST7	Reacts to Interrupt 7 and re-vectors to 1638H for user routine.

The above are fixed location utilities whereas the following are within the main body of the Monitor which are liable to be relocated if new generation monitors are written. They are, nonetheless, very useful and can save a lot of redundant instructions elsewhere. Note that you can only guarantee using these if you have MONITOR V4.1 on board!

0053H	INIT	Destination of Interrupt 2. Clears screen and initialises computer. Also resets stack, so should only be used as destination of a JUMP instruction.
00AAH	BASICIN	Checks to see if a key has been depressed on keyboard. If it has, it returns with the data - otherwise it returns immediately with the accumulator cleared and flags set accordingly. Useful for real time interactive programs.
010DH	DLY	When called introduces approximately 3mS time delay and returns with all registers (except flags) intact.
0116H	FIVSEC	When called introduces a delay of between 5 and 6 seconds. Returns with registers and flags intact.
0134H	CLRSCN	Clears screen and resets cursor. Approximately 200mS time delay is built into this routine. Returns with registers and flags intact.
02B9H	REINIT	This should only be used as a JUMP destination. It resets the stack and re-initialises the computer WITHOUT clearing the screen. A useful instruction at the end of user programs to avoid having to use the HLT instruction.
0327H	TPEON	When called switches the tape control relay on (relay contacts could be used for other control purposes by the user). Note that routine returns with 80H in accumulator.
032CH	TPEOFF	When called switches the tape control relay off. Routine returns with 00H in accumulator.
03A1H	ACKA	Start address of string which prints INVALID. Use should be made of PSTRNG utility to insert Carriage Return and Line Feed.
03B5H	ACKB	Start address of string which prints START=. (Use via PSTRNG)
03C4H	ACKC	Start address of string which prints HEADER=. (Use via PSTRNG)
03CFH	ACKD	Start address of string which prints END. (Use via PSTRNG)

USING THE MONITOR IN MACHINE CODE PROGRAMS

There are 6 fixed location utility subroutines located in a jump table in the early part of the Monitor. These are designated INCH, OUTCH, INDATA, PDATA, PSTRNG, and PCRLF. We have used a couple already. (Their addresses and descriptions are given in the section on Monitor Utilities.) There are also 2 interrupts which are brought out to push buttons on the front panel. These are dedicated, but a third is available for the user. INT1 clears the screen and resets the cursor, but can only be used once unless a keyboard input follows it. (Interrupts are re-enabled within the keyboard input routine.) INT2 is the reset which does not clear memory. INT3 is spare and contains a jump instruction to 1618H which can be used for your own routine. It is NOT ADVISABLE to use INT3 unless you have a properly formatted interrupt routine sitting in that location, as you could corrupt any other programs in memory! Interrupts 4 to 7 are brought out on TRITON's extender socket. See the table for the re-vectored addresses of these. You can use INT1 as a subroutine to be called. It jumps to the CLRSCN (Clear Screen) routine elsewhere in the Monitor and this is terminated by a RETURN. Here is an example of a user-written interrupt routine using INT3. Note that it starts at the re-vectored start address 1618:

1618	LXI D	161FH	11	Load start of string saving
1619	-		1F	I AM INTERRUPT 3
161A	-		16	
161B	CALL	PSTRNG	CD	Print carriage return
161C	-		2B	followed by string.
161D	-		ØØ	
161E	RET		C9	Return to main program.
161F	DATA	I	49	String data starts here and
162Ø	-	SPACE	2Ø	terminates with end of
1621	-	A	41	text marker Ø4.
1622	-	M	4D	
1623	-	SPACE	2Ø	
1624	-	I	49	
1625	-	N	4E	
1626	-	T	54	
1627	-	E	45	
1628	-	R	52	
1629	-	R	52	
162A	-	U	55	
162B	-	P	5Ø	
162C	-	T	54	
162D	-	SPACE	2Ø	
162E	-	3	33	
162F	-	EOT	Ø4	

Load this program then re-initialise and momentarily connect INT3 to ground. Next press W and type on the screen, activating INT3 from time to time. The interrupt should announce its presence. We hope that this simple example will show you that, contrary to popular belief, interrupts are quite easy to write programs for.

Apart from the fixed location utilities there are a few other useful ones within the Monitor. These are listed and briefly described in the table. Use the table to see if you can analyse the following machine code programs which are designed to lead you through most of them.

We know that some readers will be completely new to computing and may never have come across machine code programming before; others will have had some experience of BASIC, and some will be very experienced. In order to make this Manual of value to all parties, we have designed some experimental programs which, hopefully, will be sufficient to give the more experienced all they need to know about the Monitor, so that they can quickly go on to write their own software. We cannot possibly hope to illustrate all permutations and combinations of the wide range of instructions in the 8080 machine code set, but by frequent use of a few we will try to show, by example, how these are used. Those with less or no experience should work their way through the examples, and by seeing the same sort of pattern emerging - particularly with JUMPS and CALLS - they will soon start to see how a program can be built up. It is advisable, however, to do a lot more reading to get to grips with the full 8080 instruction set (a list of further reading is given at the end of the Manual). There are many excellent books on the subject - one of the advantages of using an 8080 in TRITON!

Video Typewriter Program

16ØØ	LXI D	1612H	11	Load start address of message
16Ø1	-		12	string in DE register pair.
16Ø2	-		16	This is required by PSTRNG subroutine.
16Ø3	CALL	PSTRNG	CD	Call subroutine which prints
16Ø4	-		2B	string starting at address
16Ø5	-		ØØ	held in DE register pair.
16Ø6	CALL	PCRLF	CD	Call subroutine which prints
16Ø7	-		33	out carriage return and
16Ø8	-		ØØ	line feed.
16Ø9	CALL	INCH	CD	Call subroutine which inputs
16ØA	-		ØB	data from keyboard and
16ØB	-		ØØ	holds this in accumulator.
16ØC	CALL	OUTCH	CD	Call subroutine which prints
16ØD	-		13	out ASCII character from
16ØE	-		ØØ	data in accumulator.
16ØF	JMP	16Ø9H	C3	Jump back to address 16Ø9H and
161Ø	-		Ø9	wait for next character from
1611	-		16	keyboard.

```
1612   DATA    O       4F   Start address of message string.
1613           /       2F
1614           K       4B
1615           SPACE   2Ø
1616           T       54
1617           Y       59
1618           P       5Ø
1619           E       45
161A           !       21
161B           EOT     Ø4   End of text terminator code.
```

When you run this program, it acknowledges with the message O/K TYPE! and you can then use the TRITON as if it were in the W function operating mode (i.e. it becomes nothing more than a video typewriter). You can escape from the program by depressing CONTROL C (this applies to any program which repeatedly goes through the INCH routine).

Duo-decimal String Program

The following program starts by stepping onto the next line of the VDU and waiting for you to type in your own message string. This can be of any length (limited only by the amount of memory you have on board) and you can use any alpha-numeric or graphic characters, but there are a few exceptions. You should not use CONTROL D as this will enter the EOT code. Carriage return should only be depressed at the end of your message as it is used to step the program onto its second phase. You CAN use control codes in the string to step the cursor down and do a non-line erasing carriage return (see relevant table). When you depress the normal carriage return key the computer repeats your message 12 more times and then re-initialises without clearing the screen.

```
16ØØ   CALL    PCRLF    CD   Print carriage return and
16Ø1    -               33      line feed.
16Ø2    -               ØØ
16Ø3   LXI D   1618H    11   Set DE register pair to point to
16Ø4    -               18      start address of input buffer
16Ø5    -               16      required by INDATA subroutine.
16Ø6   CALL    INDATA   CD   Call subroutine which allows stri
16Ø7    -               1B      to be input to buffer starting
16Ø8    -               ØØ      at address held in DE register
                                pair.
16Ø9   MVI B   ØCH      Ø6   Set the decimal number 12 (ØCH)
16ØA    -               ØC      in register B. This specifies
                                the number of times line is
                                to be repeated.
16ØB   LXI D   1618H    11   Reset DE register pair to point
16ØC    -               18      to start of input buffer.
16ØD    -               16
16ØE   CALL    PSTRNG   CD   Print string starting at address
16ØF    -               2B      held in DE register pair.
161Ø    -               ØØ
```

```
1611   DCR B              Ø5   Decrement contents of register
                               B by one.
1612   JNZ     16ØBH      C2   If register B is not decremented
1613    -                 ØB      to zero jump back to 16ØBH
1614    -                 16      and print string again.
1615   JMP     REINIT     C3   If B is at zero program has
1616    -                 B9      finished, so jump to REINIT
1617    -                 Ø2      to return control to Monitor
                                  without clearing screen.
1618   BUFFER                  Start address of input buffer.
```

Note that the REINIT subroutine is NOT at a fixed location except in MONITOR V4.1. Nonetheless it is very useful as it enables you to jump out of a user program back into the Monitor without clearing the screen.

Keyboard/LED Program Version 1

```
16ØØ   CALL    INCH       CD   Input data from keyboard
16Ø1    -                  ØB      to accumulator.
16Ø2    -                  ØØ
16Ø3   CMA                 2E   Complement contents of
                                   accumulator.
16Ø4   OUT     PORT Ø3H    D3   Output contents of accumulator
16Ø5    -                  Ø3      to port 3 (LED port).
16Ø6   JMP     16ØØH      C3   Jump back to 16ØØH for next
16Ø7    -                  ØØ      input from keyboard.
16Ø8    -                  16
```

This program enables you to test out both the LED port and the keyboard by outputting the binary code from the keyboard to the LEDs. The program complements the accumulator to compensate for the fact that on TRITON the LEDs go on for level 'O'. In this program they will go on if a bit from the keyboard is '1'. Because we are going through the Monitor's INCH routine, you will find that a shift inversion takes place. You get upper case alpha codes when unshifted and lower case alpha codes when shifted. This is designed in to make the keyboard more convenient to use. Numerical keys are not affected by the shift inversion. Again, because we are using INCH you can escape from this program loop with CONTROL C. You will notice that the most significant LED (bit 8) is permanently off - this is because bit 8 is used for the input strobe and this bit is masked off by the Monitor as data is entered. Notice also that the data is latched onto the LEDs after the key has been released.

Keyboard/LED Program Version 2

This program does exactly the same as the one just described, but is written without the use of any subroutines. We are, in effect, writing into the program a simplified form of the INCH subroutine. We say 'simplified' because we are NOT going to introduce a keyboard inversion this time. The

program also introduces the newcomer to some machine code logic followed by a decision. The logic in question checks whether a key has been depressed. If it has not, the program waits in a loop, and only after a depression will it continue.

```
16ØØ   IN       PORT ØØH    DB   Input data from keyboard
16Ø1    -                    ØØ      port to accumulator.
16Ø2   MOV B,A               47   Copy contents of accumulator
                                     to B register AND
16Ø3   ANI       8ØH         E6      accumulator with 10000000
16Ø4    -                    8Ø      (binary), to see if bit 8
                                     is '1' or 'O'.
16Ø5   JZ        16ØØH       CA   If it is 'O' jump back to
16Ø6    -                    ØØ      16ØØH and wait for key
16Ø7    -                    16      to be depressed.
16Ø8   MOV A,B               78   If it is '1' restore data
                                     in accumulator.
16Ø9   CMA                   2F   Complement accumulator.
16ØA   OUT       PORT Ø3H    D3   Out put contents of accumulat
16ØB    -                    Ø3      to port 3 (LED port).
16ØC   JMP       16ØØH       C3   Jump back to 16ØØH and wait
16ØD    -                    ØØ      for next key to be
16ØE    -                    16      depressed.
```

Bit 8 is not masked off this time, and will be permanently at '1'. Note that with the program as it stands you cannot escape via CONTROL C and will have to use Interrupt 2. You can, however, modify it to escape on CONTROL C by changing and adding a few instructions starting at 16ØCH. Try adding the following:

```
16ØC   CPI       Ø3H         FE   Compare contents of
16ØD    -                    Ø3      accumulator with Ø3H
                                     (ASCII code for CONTROL C)
16ØE   JZ        REINIT      CA   If comparison is valid
16ØF    -                    B9      jump to REINIT and
161Ø    -                    Ø2      escape from program loop.
1611   JMP       16ØØH       C3   Otherwise jump back to 16ØØH
1612    -                    ØØ      and wait for next key
1613    -                    16      depression.
```

Alphabet Twelve Times Over Using I/O

This program should be compared with the one following, as they both do the same thing - print the alphabet twelve times and then re-initialise. This first program uses the conventional I/O techniques, whereas the second makes use of the powerful memory mapped option to the TRITON's VDU. We hope you will recognise the tremendous difference in speed of operation between the two methods. Note that in both programs we make use of the RST1 instruction at the beginning. This is one of8 special re-start instructions which are fixed destination CALL instructions. Using RST1 will call the subroutine at location ØØØ8H, which, in the case of TRITON's Monitor, is then re-vectored with a jump

to Ø134H. The routine in question is the one which clears theVDU screen and resets the cursor. The advantage of using an RST instruction is that you do not have to specify the address of the subroutine, hence saving two bytes in your program. You should only use RST instructions if the subroutine being called terminates in a RETURN command.

```
16ØØ   RST1           CF   Clear screen via special
                            restart instructions.
16Ø1   MVI B   ØCH    Ø6   Set decimal value 12 (ØCH)
16Ø2    -             ØC      in B to specify number
                              of alphabets required.
16Ø3   MVI A   41H    3E   Set ASCII code for 'A' in
16Ø4    -             41      accumulator.
16Ø5   CALL    OUTCH  CD   Print contents of accumulator.
16Ø6    -             13
16Ø7    -             ØØ
16Ø8   INR A          3C   Increment ASCII code in
                              accumulator by one.
16Ø9   CPI     5BH    FE   Compare it with ASCII code
16ØA    -             5B      which is one greater than Z.
16ØB   JNZ     16Ø5H  C2   If not greater than Z jump back
16ØC    -             Ø5      to 16Ø5H and repeat until
16ØD    -             16      complete alphabet is printed.
16ØE   CALL    PCRLF  CD   If alphabet is completed output
16ØF    -             33      carriage return and line
1610    -             ØØ      feed.
1611   DCR B          Ø5   Decrement value in B register
                              by one.
1612   JNZ     16Ø3H  C2   If it is not zero we do not
1613    -             Ø3      have 12 alphabets, so jump
1614    -             16      back to 16Ø3H and repeat.
1615   JMP     REINIT C3   If it is zero, re-initialise.
1616    -             B9
1617    -             Ø2
```

When you run the program try and judge the time it takes to display the 12 alphabets and then go on to the next example.

Alphabet Twelve Times Over Using Memory Mapping

Although this program is longer than the one just described, you will see an element of similarity in the way the alphabet is formed (by incrementing the accumulator) and we keep track of the number of alphabets in the B register. Instead of using the I/O OUTCH routine, we use the HL register pair to point to memory locations which are within the block of the VDU's RAM. This starts at 1ØØØH and finishes at 13FFH. We then use the MOV M,A instruction which copies whatever is in the accumulator to the memory location being addressed by the HL register pair. By using the INX H instruction we can increment the latter to display the next character etc. Notice that carriage returns and line feeds are not needed in the main body of the program because we

are using addressing to tell the computer exactly where to place each character. When one alphabet is finished we have to compute the address of the start of the next by adding the hex number 26 to the address currently in the HL register pair (this is done at instruction 161lH). Notice that, after this operation, we have to make allowance for a carry by calling an INR H subroutine.

```
16ØØ   RST1              CF   Clear screen with special
                               restart instruction.
16Ø1   LXI H             21   Load HL register pair with
16Ø2   -                 FF      address one less than start
16Ø3   -                 ØF      of VDU RAM.
16Ø4   MVI B   ØCH       Ø6   Set number of alphabets
16Ø5   -                 ØC      required in register B.
16Ø6   MVI A   41H       3E   Set ASCII code for 'A' in
16Ø7   -                 41      accumulator.
16Ø8   INX H             23   Increment HL register pair
                                 by one.
16Ø9   MOV M,A           77   Copy contents of accumulator
                                 to memory.
16ØA   INR A             3C   Increment contents of accumulator.
16ØB   CPI     5BH       FE   Compare contents of accumulator
16ØC   -                 5B      with code one greater than Z.
16ØD   JNZ     16Ø8H     C2   If it is not greater than Z
16ØE   -                 Ø8      jump back to 16Ø8H and
16ØF   -                 16      repeat.
161Ø   MOV A,L           7D   If it is, copy contents of L
                                 to accumulator.
1611   ADI     26H       C6   Add 26H to this value.
1612   -                 26
1613   CC      1629H     DC   If addition causes a carry,
1614   -                 29      call subroutine which
1615   -                 16      increments H register by one.
1616   MOV L,A           6F   Replace new low byte address
                                 in register L.
1617   DCR B             Ø5   Decrement register B by one.
1618   JNZ     16Ø6H     C2   If it is not zero we do not have
1619   -                 Ø6      12 alphabets, so jump back
161A   -                 16      to 16Ø6H and repeat.
161B   MVI B   ØCH       Ø6   Set decimal value 12 into
161C   -                 ØC      B register.
161D   MVI A   ØAH       3E   Set accumulator to ASCII code
161E   -                 ØA      for line feed.
161F   CALL    OUTCH     CD   Output line feed -o VDU to step
162Ø   -                 13      cursor down (to get it clear
1621   -                 ØØ      of last alphabet).
1622   DCR B             Ø5   Decrement register B by one.
1623   JNZ     161FH     C2   If it is not zero we have not
1624   -                 1F      stepped cursor to below the
1625   -                 16      last line of alphabet, so
                                 jump back to 161FH and repeat.
1626   JMP     REINIT    C3   If it is zero, re-initialise.
1627   -                 B9
1628   -                 Ø2
1629   INR H             24   Subroutine to increment register
162A   RET               C9      H in event of a carry.
```

When memory mapping the VDU you must remember that the Clear Screen/Reset Cursor operation does more than is immediately apparent to the eye. The addresses of different positions on the VDU screen must correspond to specific places on the screen (1ØØØH is the top left-hand corner and 13FFH is the bottom right-hand corner). If, as a result of previous activity, the VDU screen has been scrolling, these absolute address values do not correspond with the absolute positions on the screen. To make sure that addresses correspond to positions in an absolute manner, you must carry out a cursor reset operation. This can be on its own or combined with the clear screen operation. A similar operation must be carried out if you use the VDU function when under the control of BASIC L4.1. Note that it takes the VDU 132mS to carry out a home cursor operation, so if ever you output the raw instruction in your own software, you must introduce a time delay greater than this before outputting anything else to the VDU. The Clear Screen/Reset Cursor utility in the Monitor has a delay of about 200mS built into it, so you can call it without introducing any further delay.

Note that when this program runs the cursor stays stationary in its reset position (top left-hand corner of the screen). If we allowed it to stay there the re-initialisation message would overprint our alphabets, so we have included some extra instructions (starting at 161BH) which step the cursor down twelve positions immediately prior to re-initialisation.

To further demonstrate the flexibility of memory mapping you can alter the layout on the screen by altering the value added to the address to get the next line. Alter the data at location 1612H to 29 and re-run the program. The lines should have start points staggered by 3 character positions. Similarly you can alter the start location for the display as a whole by altering the data in locations 16Ø2H and 16Ø3H. Try making these 2Ø and 1Ø respectively and alter the data at 1612H to 25. When you run the program the rows should slant the other way.

When memory mapping the VDU you must always be careful to ensure that your memory pointer (HL register pair) cannot exceed the highest address of VDU (13FFH), otherwise you will start overwriting the input buffer and stack area of the Monitor. If this happens all sorts of strange things will begin to take place, and you will probably find that you completely lose control through the keyboard. If this should happen, you must resort to the RESET button and end up with a cleared memory!

Screen Flash Program

To further demonstrate the speed of memory mapping the VDU, you might like to try the following program, which alternately makes the whole screen clear and then white (all but the top line, that is). By selecting different values for the timing byte in location 16ØFH, you can make the screen flash on and off at different speeds. Note that we use the INX H instruction to sequence through all the relevant VDU locations (at location 16Ø6), and immediately follow it by a check to see that we have not exceeded the VDU area. We do this by comparing the high order byte of the pointer register with 14. If a comparison is true, the program breaks out of its incrementing loop.

The newcomer should take note of the time delay routine between address locations 16ØDH and 1615H. Routines of this type are very commonly used in programs and they usually work by pre-setting a value in a register or, as in this case (to get a long delay), a register pair. The value is then repetitively decremented by a looping program until the value in the register/s becomes zero. The higher the number that is pre-set the greater will be the delay. Try typing the program in and running it.

16ØØ	MVI B	2ØH	Ø6	Set ASCII code for SPACE
16Ø1	-		2Ø	in register B.
16Ø2	LXI H	1Ø4ØH	21	Set memory pointer to address
16Ø3	-		4Ø	of start of second line
16Ø4	-		1Ø	on VDU.
16Ø5	MOV M,B		7Ø	Copy contents of register B
16Ø6	INX H		23	to memory location pointed to by HL register pair.
16Ø7	MOV A,H		7C	Increment HL and copy H to accumulator.
16Ø8	CPI	14H	FE	Compare value taken from H
16Ø9	-		14	register with 14H.
16ØA	JNZ	16Ø5H	C2	If comparison is true we are
16ØB	-		Ø5	pointing to one location beyond end so VDU RAM.
16ØC	-		16	If not we jump back to 16Ø5H and repeat.
16ØD	LXI D	6ØØØH	11	If VDU screen is filled, load
16ØE	-		ØØ	DE register pair with
16ØF	-		6Ø	timing byte value 6ØØØH.
161Ø	DCX D		1B	Decrement DE register pair by one
1611	MOV A,D		7A	Copy value in D to accumulator.
1612	ORA E		B3	OR it with value currently in E register.
1613	JNZ	161ØH	C2	If it is in zero continue
1614	-		1Ø	through program, otherwise
1615	-		16	jump back to 161ØH and keep decrementing.
1616	MOV A,B		78	Copy contents of B to accumulator
1617	CPI	2ØH	FE	Compare with ASCII code for
1618	-		2Ø	SPACE.

```
1619    JNZ     16ØØH   C2    If it does not compare,
161A    -               ØØ      jump back to beginning of
161B    -               16      program and clear screen.
161C    MVI B   7AH     Ø6    If it does compare, set ASCII
161D    -               7A      code for "WHITE CELL" in
                                register B
161E    JMP     16Ø2H   C3    Then jump back to 16Ø2H
161F    -               Ø2      and make screen white.
162Ø    -               16
```

You will notice that each time a memory mapping operation takes place you see a number of pulses flash up on the TV screen. These represent the moments of time the CPU takes over control of the VDU's addressing lines. They are particularly noticeable in this experiment because we are sequentially addressing nearly 1,000 bytes of VDU memory. In most instances one would not be changing data in so many places in so short a period of time, and the effect of the odd pulse now and again is hardly noticeable.

MODEM Echo Test Program

The internal MODEM on the TRITON is primarily for recording programs onto tape, but it can also be used for data transmission. The principles are exactly the same. There is a subroutine in the Monitor called TPT (short for Tape Out) which simply places whatever data is in the accumulator into the transmitter buffer of the UART, which serialises it and modulates the carrier of the MODEM. A bonus exists within TRITON in that the UART/MODEM pair will transmit and receive simultaneously. One can therefore connect the MODEM's output back to its input, and link the TPT routine to its complement TPIN, which takes received data from the UART and places it into the accumulator. By placing these subroutines within a looped program, one can output data from the accumulator through the MODEM, feed it back to the MODEM's input, and then decode it in the UART and place it back into the accumulator. If all goes well the received data should be identical to the transmitted data. If it is not, then you have something wrong with your tape I/O hardware. The following program outputs the ASCII code for A through the MODEM and then clears the accumulator in preparation for the received data. When the latter arrives back it is output to the VDU. You should see a continuously repeated printing of letter "A".

```
16ØØ   MVI A   41H    3E   Set accumulator to ASCII code
16Ø1    -              41      for "A".
16Ø2   CALL    TPT     CD   Call subroutine which serialises
16Ø3    -              AD      and modulates data from
16Ø4    -              Ø2      accumulator onto tape
                               recorder output line.
16Ø5   SUB A           97   Destroy data in accumulator
                               by clearing it.
16Ø6   CALL    TPIN    CD   Call subroutine which demodulates
16Ø7    -              1D      incoming data and forms it
16Ø8    -              Ø3      into parallel word, finally
                               placing it in accumulator.
16Ø9   CALL    OUTCH   CD   Print contents of accumulator
16ØA    -              13      on VDU.
16ØB    -              ØØ
16ØC   JMP     16ØØH   C3   Jump back to 16ØØH and keep
16ØD    -              ØØ      repeating.
16ØE    -              16
```

Before running the program, short the tape out and tape in sockets together on the computer's DIN socket. Run the program, and try adjusting the baud rate control - you should see a noticeable difference in the speed of reception of the letter A. Remove the shorting wire and reception should stop. When you replace it reception should start again. With a slightly more sophisticated program you could arrange for one TRITON to communicate with another down a pair of wires or via a radio link. An even simpler pair of programs would allow you to type directly onto tape and play back what you have typed onto the VDU. Why not try and write these for yourself using the INCH and OUTCH utilities as well as the TPT and TPIN routines shown above. Note that TPT and TPIN are not fixed location routines so you can only call them at addresses Ø2ADH and Ø31D respectively when using MONITOR V4.1.

Test Tape Playback Program

In the section on how to make the TRITON, we suggested you use Transam Components Ltd's test tape to adjust the baud rate of your UART - the program was listed, and instructions were given on how to use it. By repeating it here with mnemonics and instruction descriptions, we hope that you will be able to follow it through without any further explanation.

```
16ØØ   CALL   TPEON    CD   Call subroutine which switches
16Ø1    -              27     on tape control relay.
16Ø2    -              Ø3
16Ø3   CALL   TPIN     CD   Call subroutine to input data
16Ø4    -              1D     from tape and store in
16Ø5    -              Ø3     accumulator.
16Ø6   CALL   OUTCH    CD   Print contents of accumulator
16Ø7    -              13     on VDU.
16Ø8    -              ØØ
16Ø9   JMP    16Ø3H    C3   Jump back to 16Ø3H for next
16ØA    -              Ø3     byte from tape recorder.
16ØB    -              16
```

Note that TPEON can only be called at address Ø327H when using MONITOR V4.1.

Test Tape Record Program

In case you have lost your test tape, or want to make another copy to be on the safe side, you can write the program written by Transam to produce our master copy. Your system must have been set up to the correct baud rate prior to making your own test recording, otherwise you will only be calibrating your own system against itself.

```
16ØØ   MVI A  41H      3E   Set ASCII code for "A"
16Ø1    -              41     in accumulator.
16Ø2   CALL   TPT      CD   Call subroutine which outputs
16Ø3    -              AD     contents of accumulator
16Ø4    -              Ø2     to tape.
16Ø5   INR A           3C   Increment ASCII code in
                              accumulator.
16Ø6   CPI    5BH      FE   Compare it against code which
16Ø7    -              5B     is one greater than "Z".
16Ø8   JNZ    16Ø2H    C2   If it is not, jump back to
16Ø9    -              Ø2     16Ø2H and keep repeating.
16ØA    -              16
16ØB   MVI A  ØDH      3E   If "Z" has been reached and
16ØC    --             ØD     recorded, set accumulator
                              to ASCII for CR.
16ØD   CALL   TPT      CD   Output carriage return code
16ØE    -              AD     to tape.
16ØF    -              Ø2
161Ø   MVI A  ØAH      3E   Set accumulator to ASCII code
1611    -              ØA     for line feed.
1612   CALL   TPT      CD   Output line feed code to tape.
1613    -              AD
1614    -              Ø2
1615   JMP    16ØØH    C3   Jump back to beginning of
1616    -              ØØ     program and keep repeating.
1617    -              16
```

The program repeatedly outputs the alphabet (followed by carriage return and line feed) to tape for as long as you care to let it run.

Siren Program

In the previous program examples we have tried to demonstrate most of the Monitor's utilities and at the same time exercise the more important I/O facilities of TRITON. We have not yet used the spare output port, and this program will demonstrate how you can go about using it to communicate with the outside world (in this case with a crystal earpiece or external amplifier). Port 6 is only a two bit port - outputting any data on the two most significant data bus lines. We shall use bit 8 to generate an audio frequency square wave output, but to make the program a bit more interesting we shall use software to make the frequency increase over a period of time.

16ØØ	MVI B	8ØH	Ø6	Set register B to initial
16Ø1	-		8Ø	note divisor value (128 decimal).
16Ø2	MVI C	2ØH	ØE	Set register C to value to
16Ø3	-		2Ø	give note duration.
16Ø4	MOV D,B		5Ø	Copy current divisor from B into D.
16Ø5	CMA		2F	Complement accumulator.
16Ø6	DCR D		15	Decrement register D by one.
16Ø7	JNZ	16Ø6H	C2	If register D is not zero,
16Ø8	-		Ø6	jump back to 16Ø6H
16Ø9	-		16	and repeat.
16ØA	OUT	PORT Ø6H	D3	If it is, output value in
16ØB	-		Ø6	accumulator to port 6.
16ØC	DCR C		ØD	Decrement register C by one.
16ØD	JNZ	16Ø4H	C2	If it is not zero, jump
16ØE	-		Ø4	back to 16Ø4H and repeat.
16ØF	-		16	
161Ø	DCR B		Ø5	If it is, decrement register B by one.
1611	JNZ	16Ø2H	C2	If register B is not zero,
1612	-		Ø2	jump back to 16Ø2H
1613	-		16	and repeat.
1614	JMP	16ØØH	C3	If it is, jump back to start of program and keep repeating.

It is the CMA instruction which makes the output signal alternate between "I" and "O". The time taken between transitions is controlled by the value of the D register which, in the first instance, gets its data from register B. Each time a note is played and held for its duration (set by the value in C) register D is decremented so that the next note will have a marginally higher frequency. The program loops until the value in D is zero (infinite frequency!) and then starts all over again.

To keep the program short and simple we have omitted to compensate for the fact that we have one shortening timing loop nested within another. In practice this means that as the note divisor value diminishes so will the duration of the note. For this program you can hardly notice it, but if you use TRITON to synthesise music you must make allowances for this.

With an even simpler program than this you could use TRITON as a high stability square wave generator.

Without doubt you will get most enjoyment from machine code programming when you start to write more lengthy programs than those we have used as illustrations. Space does not permit us to go any further at this stage, but we hope that the examples will get you going on the road to writing your own. Good luck!

INTRODUCTION TO PROGRAMMING IN BASIC

Most low-cost home computers to date are only capable of being programmed in machine code - this is a language of 'numbers' forming a powerful but difficult language in which to address your machine. Machine codes are interpreted by the microprocessor as specific executable instructions and are performed extremely rapidly. Each microprocessor has its own quite distinct machine codes, and these codes have to be learnt by the programmer in order to achieve execution of any task by the microprocessor, and, what is more, a program written in machine code is not executable on any machine using a different microprocessor.

The TRITON computer can be programmed both in machine code and in a more useful language called 'Basic'.

Before any computer is able to function as a tool, it must be instructed by the programmer exactly what to do. Communication between humans in English is only possible after years of training and even adults sometimes misunderstand each other's remarks. Clearly, ambiguities are not acceptable in computing, and a clear, precise language must be the key to communicating with a machine. These 'languages' are called programming languages, and there are several in use today, e.g. Fortran, Cobol, Algol and Basic. We shall concern ourselves only with Basic in this discussion, as this is the most common language in use in the personal computer arena.

The 'Tiny Basic' used in the TRITON personal computer is easy to understand, simple to use and yet a very powerful aid to the programmer for everyday applications.

We trust that the information which follows will enable you to become more aware of the tremendous facilities now affordable to the personal computer enthusiast, and wish you every success with your programming.

THE BASIC INTERPRETER

The TRITON BASIC is a BASIC Interpreter specially configured for the TRITON Computer. It contains many of the common BASIC commands and most small BASIC programs will be easily converted to run on the system. The following is a description of the BASIC Interpreter.

VARIABLES

All variables and numbers are stored as 16 bit integers and therefore must lie in the range -32767 to 32767. There are 26 variables, each denoted by a single letter A to Z. There is one array denoted by @. This array is automatically dimensioned to make use of any memory space left unused by your Basic Program. The number of bytes of memory space in this array can be obtained at RUN time using the SIZE function. Now try Example 1 (see Page 93).

FUNCTIONS

There are three functions available.

ABS(X)	which gives the absolute value of the variable X.
RND(Y)	which gives a random number between 1 and Y inclusive.
SIZE	which gives the number of bytes left unused by your program.

Hence the maximum index for the array @ ( ) is SIZE/2.
Now try Example 2.

ARITHMETIC OPERATORS

+	Add
-	Subtract
*	Multiply
/	Divide

+, -, * and / operations must result in a value in the range -32767 to 32767 and as they are also integer, any division is rounded down. e.g. 5/2 gives 2, 2/3 will give 0.
Now try Example 3.

COMPARE OPERATORS

>	greater than
<	less than
=	equal to
#	not equal to
>=	greater than or equal to
<=	less than or equal to

The compare operators are usually used with the IF command but can also be used in expressions. The result of any comparison is 1 if true and 0 if not true (false).
Try Example 4.

EXPRESSIONS

Expressions are formed from number, variables and functions.

e.g.

```
10 LET A = 10          A is set to 10
20 LET B = A           B is set to contents of A i.e. 10
```

Arithmetic operators are used in expressions and are evaluated from left to right, except that * and / are always evaluated first. Spaces between numbers, variables and functions are ignored. Spaces embedded in command words are not allowed.

```
10 LET A = -10         A is set to -10
20 LET B = A/2         B is set to -5

10 LET A = 4*2 + 3     A is set to 11
10 LET A = 3 + 4*2     A is against set to 11
                       (multiply evaluated first)
```

Parentheses can be used to change the order of evaluation:

```
10 LET A = (3+4)*2     A is now set to 14
```

Parentheses can be nested, the maximum depth being limited by the size of the stack:

```
10 LET A = 14
20 LET B = ((A-1)*2 - 7)*3    B is set to 57
```

Conditional operators are usually found with the IF command:

```
10 IF A = 1 B = B +1
```

In this statement when A is equal to 1, the expression B = B+1 is executed and 1 is added to the contents of B. Conditonal expressions can be combined to form multiple conditions and can also be used in arithmetical expressions. Try Example 5.

STATEMENTS

A BASIC statement consists of a statement number between 1 and 32767 followed by one or more commands. If a statement contains more than one command, each command is separated by a semi-colon ; and the statement is ended by a carriage return.

```
10 LET A = 10
20 LET B = A
30 LET C = A+B
```

This can be written:

```
10 LET A = 10; LET B = A; LET C = A+B
```

It should be noted that the latter method will be harder to change or correct. The commands GOTO, STOP and RETURN must be last command in any statement.
Try Example 6.

COMMANDS

The following commands are available in the TRITON BASIC L4.1

LET

LET is used to set a variable to the result of an expression.

`10 LET A = 10`	The variable A is set to 10
`20 LET B = (A-1)*2`	The variable B is set to the result of the expression (A-1)*2 i.e. 18
`30 LET @(3) = B/3`	The fourth element of the array @ is set to 6 (The first element is @(0))

The expression need not be an arithmetical expression.

```
10 LET A = 1, B = 2, c = 3
```

each part being separated by a comma. We can therefore rewrite an earlier example:

```
10 LET A = 10, B = A, C = A+B
```

Try Example 7.

REM

The REM (Remark) Command allows the programmer to comment his program. The interpreter will ignore the rest of the line.

```
100 REM THIS IS THE START OF THE SUBROUTINE Y = A*A+B
```

Try Example 8.

PRINT

The PRINT command is used to print numbers, variables, expressions and text.

```
10  print A                 will print the contents of
                            variable A
10 PRINT A*2                prints twice the contents of
                            variable A
10 PRINT "THIS IS A TITLE"  prints THIS IS A TITLE
```

Several variables etc. can be printed at once. Each item to be printed is separated by a comma.

```
10 PRINT A,B,C              will print the contents of A
                            followed by B and C on the
                            same line
```

Text can be used to qualify printout.

```
10 PRINT 'THE RESULT IS', A
```

Try Example 9.

Text can be contained by either single or double quotes, this allows the other type of quote to be printed.

```
10 PRINT 'ABC"CBA',  "123'321" will print
ABC"CBA123'321
```

Try example 10.

Numerical values are printed with leading spaces (Right Justified) in a field of width 8 characters. The field width can be altered using a # sing followed by the new width (i.e. # 3 gives a width of 3). The field width will then remain effective until another or the end of the current PRINT statement.

```
10 PRINT A,# 3, B,# 1, C will print A in a width of 8
                           characters, B in a width of 3
                           and C in a width of 1.
```

1 will result in C being printed Left Justified and any following printout will be shifted to the right if C is greater than 9.

The field width can also be an expression

PRINT # I, A will print A in a field width equal to the contents of variable I

The maximum field width is 63.
Try Example 11.

Note that negative numbers required an extra character in the field width for the minus sign. Extra spaces can be generated by repeated commas.

```
PRINT # 3,A,,,B
```

will print a 3 character A, 2 spaces and a 3 character B

Several PRINT statements can be made to print on the same line by ending the statement with a comma.

```
10 PRINT  'THIS WILL',
20 PRINT  'ALL BE PRINTED',
30 PRINT  'ON ONE LINE'
```

will print THIS WILL ALL BE PRINTED ON ONE LINE

Try Example 12.

PRINT on its own will space one line.

Grpahic characters can be printed using the PRINT statement.

The description of the graphics font lists those graphics which can be contained in quotes and will result in graphics being printed.
Try Example 13.

The PRINT statement can also be used to issue control characters to the display chip.

`10 PRINT ^H`	will issue a control H which will backspace the cursor
`10 PRINT ^I`	will issue a control I which will forward space the cursor
`10 PRINT ^J`	moves cursor down
`10 PRINT ^K`	moves cursor up
`10 PRINT ^L`	will clear the whole screen and reset the cursor. Note that this command must be followed by a delay before the next command (FOR I=1 TO 250; NEXT I)
`10 PRINT ^M`	will reset the cursor to the start of the line.

Try Example 14.

INPUT

The INPUT command is used to read an expression from the keyboard. Normally the keyboard input is just an integer value between -32767 and 32767.

```
10 INPUT A
```

When this statement is executed, the BASIC will first print A followed by a space and then wait for keyboard input. The input is terminated by carriage return. The input is then stored in the variable A

```
10 INPUT A,B
```

will print A,space, then wait for input, it will then print B, space, and wait for input again

Try Example 15.

Instead of just allowing the machine to prompt you with the variable, it is much better to ask a specific question. This is done by enclosing the text of the question in quotes.

```
10 INPUT  'HOW MANY EGGS HAVE YOU LEFT?' I
```

The machine will print HOW MANY EGGS HAVE YOU LEFT? and then wait for a number to be typed in. If during RUN time, the typed input is not a valid expression, the prompt will be repeated and then the machine will wait again. It is also possible to reprint only part of the prompt.

```
10 INPUT  'WHAT IS', 'A+B?'C, 'A-B?'D
```

The first time the printout will be WHAT IS A+B? and after an invalid input it will just print A+B?
Try Example 16.

The BASIC interpreter uses its expression evaluation routine to decode the input and therefore the programmer or user can enter an expression using variables already set up.

```
10  LET A=3, B=2
20  INPUT C
30  PRINT C
```

Instead of entering a value for C, the user can enter an expression such as A+B. The expression will then be evaluated by the interpreter and the result 5 stored in the variable C. The machine will then print 5.
Try Example 17.

It is also possible to enter single characters as a reply by making use of the expression input.

```
10  LET  Y=0, N = 1
20  INPUT  'DO YOU WANT TO CONTINUE? Y OR N' A
30  IF A=1  STOP
```

If the user replies Y, A will be set to the contents of Y, i.e. zero. If the user replies N, A will be set to 1 and program will STOP.

IF

The IF command is used to compare expresssions, using the compare operators. If the result of this comparison is true (non zero), the rest of the statement is executed. If the result of the comparison is false (zero), the rest of the statement is skipped and execution resumes on the next statement.

```
10  IF A=0 PRINT 'A IS ZERO'
```

The machine will print A IS ZERO only when A is zero. Note that unlike other BSIC interpreters and compilers, the word THEN is not used.
Try Example 18.

Either side of the compare can be an expression.

```
10  IF A=B*2  PRINT 'A IS TWICE B'
20  IF A*3=B*2  PRINT 'A = B*2/3'
```

Try example 19.

A compare operator need not be used in the IF statement but this practice should be avoided where possible as it can make the program very hard to follow.

```
10  IF A-1  PRINT 'A IS NOT ONE'
```

The PRINT command is only skipped when the result of the expression in the IF command is zero.
Several commands can follow the IF command.

```
10  IF A=0  PRINT 'A IS ZERO'; GOTO 50
```

When A is zero, the machine will print A IS ZERO and then jump statement 50.

GOTO

You will probably be fairly familiar with the GOTO command already as it has appeared in several of the examples for the other commands. The GOTO command is used to break the sequential processing of the BASIC interpreter and causes the interpreter to jump either forward or backwards to the specified statement number.

```
50  GOTO 10
```

When the interpreter executes this statement it will jump back up the program to statement 10 and continue its processing from statement 10.

Again, the statement number following the GOTO can be an expression.

```
20  GOTO A*2
```

It will jump to the statement number calculated from the expression A*2. If the expression gives a non-existent statement number, the BASIC will give an error report.

Using a simple expression for a GOTO is useful where different routines may be required as a result of an input. Try example 20.

The problem with using the 'computed' GOTO is that the statement numbers of the routines must follow the sequence defined by the expression and this can cause problems when adding code unless careful planning has been carried out. Another method of using a computed GOTO is to use the array variable and index it.

```
10  LET @(1) = 100, @(2) = 200, @(3) = 100, @(4) = 25
20  INPUT I
30  GOTO @(I)
```

If the input for I is 1 the interpreter will jump to statement 100

```
for I = 2  it will jump to 200
for I = 3  to 100 again
for I = 4  to 25
```

It is advisable when using the computed GOTO to check the variable for valid values, e.g. in the above example it would be advisable to insert:

```
25  IF I < 1  GOTO 20
27  IF I > 4  GOTO 20
```

This will only allow an input of 1 to 4, any other input will result in a repeat request for input.
Try Example 21.

GOSUB AND RETURN

The GOSUB command, although similar to the GOTO command, is used to exit from a statement and jump to a routine starting at the specified statement number. Execution continues from the specified statement number until a RETURN command, whereupon the BASIC returns to the command following the original GOSUB.

```
10   PRINT 'LETS EXECUTE ROUTINE 100'
20   GOSUB 100; PRINT 'WE HAVE NOW RETURNED'
30   STOP
100  PRINT 'THIS IS ROUTINE 100'
120  PRINT 'I WILL RETURN WHEN I HAVE FINISHED'
130  RETURN
```

This will result in the following print-out:

```
LETS EXECUTE ROUTINE 100
THIS IS ROUTINE 100
I WILL RETURN WHEN I HAVE FINISHED
WE HAVE NOW RETURNED
```

The GOSUB 100 command causes the BASIC to junp to statement 100 but to remember where it is in statement 20. It now executes from statement 100 until it reaches the RETURN command. It then returns to statement 20 and continues processing it.

The GOSUB command is used in a program where several lines of program appear several times. These lines need only be written once and a GOSUB used when these are required. Try Example 22.

FOR AND NEXT COMMANDS

The FOR command is a very powerful command. It is used to make the BASIC interpreter loop 'FOR' a specified number of times, the end of the loop being defined by the NEXT command.

```
10  FOR I=2  TO 10 STEP 2
20  PRINT I
30  NEXT I
```

I is set to 2 when the FOR statement is first encountered. It will then remain at 2 until the NEXT command is encountered. On reaching the NEXT command 2 is added to I and the Basic returns to the command following the FOR command. This is repeated until I becomes greater than 10 whereupon execution continues with the command following the NEXT command.

Hence, the machine will print:

```
2
4
6
8
10
```

On exit from the loop, I remains at its next value, i.e. 12.

If statement 10 had been:

```
10 FOR I = 2 TO 11 STEP 2
```

I will be left at its first value greater than 11, i.e. 12.

Try Example 23.

Negative indexing is allowed as long as the first value is greater than or equal to the second and the step is negative.

```
10  FOR I=10  TO 1 STEP - 1
50  NEXT I
```

I will start at 10 and step down to 1 in increments of 1. If STEP is omitted, a step of 1 is assumed.

```
10  FOR I =1 TO 100
```

I will start at 1 and step up to 100 in increments of 1. Once more, expressions can be used in all three positions instead of numbers. The expressions are evaluated when the FOR command is executed and any following changes to the variables used will not affect the loop.

```
10  LET  I = 10
20  FOR  I = I TO I + 5
50  NEXT I
```

The initial value of I is evaluated as 10, the final value is 15. Within the loop, I will index from 10 to 15 in steps of 1.
Try Example 24.

FOR and NEXT commands can be 'nested' within each other, the limit being that of the size of the stack.

```
10  FOR I = 1 TO 10
20  FOR J = 1 TO 5
30  PRINT I * J
40  NEXT J
50  NEXT I
```

This will result in the machine printing I*J when

```
I=1 and J=1 2 3 4 5    then for

I=2 and J=1 to 5       etc...

until I=10
```

Try Example 25.

When a NEXT command is executed, the BASIC interpreter checks that the variable specified is the same as that used by the most recent FOR. If they are not the same, the FOR is terminated and the previous FOR examined. This continues until a match is found.

```
10  FOR I = 1 TO 10
20  FOR J = 1 TO 10
30  IF  J = 5 GOTO 50
40  NEXT J
50  NEXT I
```

Each time J gets to 5, the BASIC jumps to statement 50. This cancels the J FOR loop leaving J at 5 and continues with the I for loop. If within a FOR loop, another FOR loop using the same variable is encountered, the first FOR loop is terminated.
Try Example 26.

STOP

The STOP command stops the execution of the program when it is executed. Any number of STOP commands can be included within a program.

```
10  GOSUB 100
20  GOSUB 200
30  GOSUB 300
40  STOP
```

This is an example where STOP is useful.

VDU

The VDU command allows the programmer direct access to the VDU control chip and its memory, hence allowing a wide range of graphic applications.

The VDU command has two parameters, the first being the VDU memory address, the second being the desired graphic symbol specified as a decimal number.

```
10  VDU 5, 126
```

This will result in the graphic → being placed in the fifth byte of the VDU memory. The VDU memory is arranged as 16 rows each containing 64 bytes therefore addresses 1 to 64 are on the first row, 65 to 128 on the second, etc. Due to the function of the VDU control chip, care should be taken when using the first row and the first column as certain graphics characters will produce strange effects.

Try Example 27.

You will notice from this example that X is used for column addressing and Y for row addressing. By leaving Y constant, the same row is always used.

Formula Z = X + Y * 64 converts the X and Y values into an address Z. It also avoids using the top row. To use the top row as well, use the following formula:

```
Z = X + (Y-1) * 64
```

To allow the programmer to use all the VDU control commands, address zero has been allocated:

```
10  VDU 0, 12
```

This does not use memory location zero, instead the value 12 is output to the VDU controller. 12 is the command to clear the screen and reset the cursor. Note that commands 12 and 28 require an extra delay while the command is executed. A FOR loop should be used (FOR I=1 TO 150; NEXT I) before the next PRINT, VDU or INPUT command.

Try Example 28.

Other useful VDU commands are as follows:

8	Backspace cursor one character
9	Forward space cursor one character
10	Line Feed (Move cursor down one line)
11	Move cursor up one line
12	Reset cursor to top and clear screen
13	Carriage Return - Reset cursor to start of line clearing rest of line
27	Line Feed
28	Reset cursor to top without screen clear
29	Reset cursor to start of line without rest of line clear

When using the memory mapping option, care must be taken to make sure that the memory address is between 1 and 1024 inclusive. If you exceed 1024 it is possible to overwrite the stack and your program.

It is possible to make your BASIC program modify itself using VDU but this is fairly difficult and not really worth the trouble it can cause.

Before using memory mapping it is advisable to use either command 12 or 28 to reset the cursor. If the screen has been scrolling, row 1 will not be at the top of the scan unless this is done.

The graphic symbol specified in the second parameter is a decimal number between 0 and 255 inclusive. If a larger number is specified, only the least significant byte is used.

The graphics and character code are given elsewhere in the Manual but some of the useful are listed below.

0 to 31	see Graphic Font
32	Space
33 - 47	! " # $ % & ' ( ) * + , - . /
48 - 57	0 to 9
58 - 64	: ; < = > ? @
65 - 90	A to Z
91 - 95	[\ ] ↑ ←
96 - 127	see Graphics Font
128 - 225	Is a repeat of 0 to 127 (The high order bit is ignored)

To print a variable between 0 and 9 using VDU, just add 48.

```
VDU 0, I + 48
```

This will print the value of I if it lies between 0 and 9.

To produce moving graphic, it is necessary to use FOR loops to index the memory mapping. In the sample program BOUNCE, the old location of the character is blanked out using a space (32) and the character mapped into its new location.

By leaving the blanking instruction out, the sample program PATTERN gradually draws a pattern.

DIRECT COMMANDS

The following are direct commands to the BASIC interpreter. They are obeyed as soon as they are entered.

RUN

RUN will start the execution of the program at the lowest statement number.

LIST

LIST will print out all statements in ascending numerical order.

LIST 100 will print out all the statements starting at statement 100.

LIST 50,10 will print 10 lines starting at statement 50.

NEW

NEW will delete all program statements ready for a new program. Control C will return you (at any time) to the Monitor.

Any BASIC command can be entered as a Direct Command by leaving off the statement number. The statement is then executed immediately and not stored as part of the program.

This feature is very useful when your program stops due to an error report (see Error Reports).

ABBREVIATIONS

All the commands can be abbreviated as follows. It is advisable only to abbreviate when you are tight on memory as the abbreviated program can be extremely difficult to follow.

FUNCTIONS

A.	=	ABS
R.	=	RND
S.	=	SIZE

COMMANDS

Implied	=	LET	i.e. A=B+C, D=E+F etc.
REM	=	REMARK	
P.	=	PRINT	
IN.	=	INPUT	
I.	=	IF	
G.	=	GOTO	
GOS.	=	GOSUB	
R.	=	RETURN	
F.	=	FOR	
TO	=	TO	
S.	=	STEP	
N.	=	NEXT	
S.	=	STOP	
V.	=	VDU	

EDIT
READ
WRITE
CALL
PEEK
POKE

DIRECT COMMANDS

L.	=	LIST
R.	=	RUN
N.	=	NEW

ERROR REPORTS

It is quite probable that you can have already seen some of the error reports generated by the BASIC Interpreter. Although there are only three different error messages (WHAT? HOW? and SORRY) the BASIC will insert a question mark at the point where the error occurred.

WHAT?

This means the interpreter has come across a command or expression that cannot interpret.

```
WHAT?
300 I? PUT A
```
- INPUT is spelt wrongly

```
WHAT?
40 A + 300/(B+C?
```
- The close parenthesis is missing

HOW?

This means the interpreter cannot execute the command.

```
HOW?
60 A = 300*500?
```
- The result is greater than 32767

```
10 A=5,B=0

HOW?
20 C=A/B?
```
- You cannot divide by zero

```
HOW?
40 GOTO 37?
```
- Statement 37 is missing

SORRY

This means that there is not enough memory. This can occur during typing in a program or during the execution when the array is used - @ ( ). It is worth checking the variable or expression if the array is involved to make sure that it is a sensible value.

```
 SORRY?
210 A = @(I*J+K)
```

If this occurred during typing in of the program then there is not enough memory. If this occurred during execution (RUN) then either there is not enough memory for @ or the expression I*J+K may be incorrect. To check this type:

```
PRINT I,J,K,I*J+K
```

and the values of I J K and I*J*K will be printed. You can now check if the result is correct. If the result of the PRINT is OK then another check is:

```
PRINT SIZE
```
- this will give how much memory space (in bytes) is left.

TRITON BASIC EXAMPLES

These examples should be tried as the description is read. To enter BASIC commands enter T as answer to the TRITON READY message. The machine will reply:

```
BASIC   L4.1
OK
```

Before each example, unless other wise stated, you should now type

```
NEW Return
```

This will clear any old program already in memory. The machine will again reply:

```
BASIC   L4.1
OK
```

It is now ready for BASIC code. Each line of code is keyed in and is then entered by pressing the Return Key.
Control C can be entered to return to the monitor at any time.

EXAMPLE 1

Variables

```
10  LET  A=1
20  LET  B=-8
30  PRINT A,B
RUN
```

The machine should print 1 -8

EXAMPLE 2

Functions

```
10  LET A=  -8
20  LET B=  ABS(A)
30  LET C=  RND(10)
40  PRINT   A,B,C
RUN
```

The machine should print -8 8 ? The ? will be a random number between 1 and 10. If you type RUN again, you will get a different third number. To find the number of bytes of memory used by this program, or any program, enter

PRINT SIZE followed by CR - try this with the other examples.

EXAMPLE 3

Arithmetic Operators

```
10  LET    A = 3 + 4
20  LET    B = 5 - 6
30  LET    C = 3 * 15
40  LET    D = 5/3
50  PRINT  A,B,C,D
RUN
```

The machine should print 7 -1 45 1

EXAMPLE 4

Compare Operators

```
10  LET    A = 0
20  LET    B = 0
30  LET    C = 0
40  IF     1 > 2 LET A = A + 1
50  IF     3 # 4 LET B = B + 1
60  IF     4 >=3 LET C = C + 1
70  LET    D = 2 < 1
80  PRINT  A,B,C,D
RUN
```

The machine should print 0 1 1 0

The explanation is as follows:

```
Statement 40   1 is not greater than 2 so A remains at zero
Statement 50   3 is not equal to 4 so B has 1 added to it
Statement 60   4 is greater than 3 so C has 1 added to it
Statement 70   2 is not less than 1 so D is set to zero
```

EXAMPLE 5

Expressions

```
10  LET A = -10
20  LET B = A/2
30  LET C = ((A-1)*2+B)*3
40  PRINT A,B,C
RUN
```

The machine should print -10 -5 -81

EXAMPLE 6

Statements

```
10  LET A = 10; LET B=A; LET C=A+B
20  PRINT A,B,C
RUN
```

The machine should print 10 10 20

EXAMPLE 7

The LET command

```
10  LET A = 10
20  LET B = (A-1)*2
30  LET @(3) = B/3
40  PRINT A,B,@(3)
RUN
```

The machine should print 10 18 6

EXAMPLE 8

The REM command

```
10  LET A = 5
20  REM - EVALUATE AREA OF CIRCLE
30  LET B =(314*A*A)/100
40 PRINT B
RUN
```

The machine should print 78

EXAMPLE 9

The PRINT command

```
10  LET A = 5
20  PRINT 'A=',A,'A*A=',A*A
RUN
```

The machine should print A= 5 A*A= 25

Note that although the machine has printed 25 for A*A, the value of A is still 5.

EXAMPLE 10

```
10  PRINT "HOW MANY 2'S in 4"
20  PRINT "THIS IS WRITTEN IN  BASIC  LANGUAGE"
RUN
```

The machine should print HOW MANY 2'S IN 4
THIS IS WRITTEN IN BASIC LANGUAGE

EXAMPLE 11

```
10  LET A = 3
20  LET B = 100
30  PRINT # 3,A,B,A,B
RUN
```

The machine should print bb3100bb3100.
The character b is a blank.

Try this example using different values for A and B and changing 3. Now try this:

```
10  LET A = 0
20  LET A = A + 1
30  PRINT # A,A
40  IF A < 10 GOTO 20
RUN
```

The machine should now print:

```
 1 2 3 4 5 6 7 8 9 10
```

The field width is set to the same value as the variable A, hence the numbers are slowly moved one space to the right each time.

EXAMPLE 12

```
10  LET A = 0
20  LET A = A + 1
30  PRINT # 1,A,
40  IF A < 20 GOTO 20
RUN
```

The machine should print:

```
 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

The commas following the A in the PRINT statement causes all the numbers to be printed on the same line.

EXAMPLE 13

```
10  PRINT ' zzzzzzz'
20  PRINT ' zezzzez'
30  PRINT ' zzezezz'
40  PRINT ' zzzezzz'
50  PRINT ' zzezezz'
60  PRINT ' zezzzez'
70  PRINT ' zzzzzzz'
RUN
```

The machine should print the Five of Clubs.

Note that z is typed by pressing shift and Z - it will appear on the screen as ▮

EXAMPLE 14

```
10  PRINT 'AB',^J,'CD',^K,'EF'
20  PRINT
RUN
```

The machine should print AB EF
                            CD

EXAMPLE 15

```
10  INPUT A
20  PRINT 'A IS ',A
30  INPUT B,C
40  PRINT 'B IS',B, ' C IS',C
RUN
```

The machine should print A. If you type a number followed by Return, the machine will print A IS nnn, where nnn is the number entered. The machine will now ask for B, and then C, and will then print B IS C IS yyy, where xxx and yyy are the values entered for B and C.

EXAMPLE 16

```
10  INPUT 'WHAT IS', 'YOUR AGE?' A
20  INPUT 'WHAT IS', 'YOUR HEIGHT?' B
30  LET   C = B*12/A
40  PRINT 'YOU HAVE GROWN', #4,C,'INCHES PER YEAR'
RUN
```

The machine should print WHAT IS YOUR AGE? now enter your age.
The machine should print WHAT IS YOUR HEIGHT? type an invalid input e.g. 6 FEET.
The machine should print WHAT
followed by YOUR HEIGHT?
Now enter you height to the nearest foot.

EXAMPLE 17

```
10  INPUT A,B
20  INPUT C
30  PRINT C
RUN
```

Enter any numbers for A and B then enter an expression such as A*B for C. The machine should then print the result of the expression.

EXAMPLE 18 TAPE HEADER = COUNT DOWN_

```
10  LET A=11
20  PRINT 'COUNT DOWN'
30  LET A=A-1
40  PRINT A
50  IF A#1  GOTO  30
60  PRINT 'IGNITION'
70  PRINT 'TRITON HAS LIFTOFF'
RUN
```

The IF command at statement 50 will cause the programme to jump back to statement 30 until A is equal to 1 when it will skip to statement 60 and finish the program.

EXAMPLE 19

```
10  INPUT A
20  IF A > 1000 PRINT 'A IS VERY LARGE'
30  IF A = 100  PRINT 'A IS ONE HUNDRED'
40  IF A < 3    PRINT 'A IS SMALL'
50  IF A # 0    GOTO 10
RUN
```

Enter several values for A, i.e. 2000 then 100 then 2. The program will keep asking for another input until zero is entered.

EXAMPLE 20

```
10  INPUT  A,B
20  INPUT  'ADD OR SUBTRACT - ENTER 1 OR 2' C
30  GOTO   C*50
50  LET    D=A+B
60  GOTO   200
100 LET    D=A-B
200 PRINT  'THE RESULT IS ', D
RUN
```

EXAMPLE 21

```
10  PRINT  'COMBINATION'
20  LET    X=RND(5), Y=RND(5), Z=RND(5)
30  LET    A=0
40  LET    A=A+1
50  GOTO   30+A*30
60  INPUT  'WHAT IS THE 1ST NUMBER?' B
70  IF     B=X GOTO 40
80  GOTO   30
90  INPUT  'WHAT IS THE 2ND NUMBER?' B
100 IF B=Y GOTO 40
110 GOTO   30
120 INPUT  'WHAT IS THE 3RD NUMBER?' B
130 IF B=Z GOTO 40
140 GOTO   30
150 PRINT  'WELL DONE'
RUN
```

The machine will generate 3 random numbers between 1 and 5 inclusive - you have to guess them. If you get one wrong, you start again.

Note the 'computed' GOTO at statement 50.

If A is 1 the machine jumps to statement 60.

If A is 2 it jumps to 90, etc.

EXAMPLE 22

```
10  GOSUB 100
20  INPUT 'HEIGHT' A
30  GOSUB 100
40  INPUT 'WEIGHT' B
50  GOSUB 100
60  INPUT 'AGE' C
70  STOP
100 PRINT 'PLEASE TYPE IN YOUR',,
110 RETURN
RUN
```

The machine should print:

```
PLEASE TYPE IN YOUR HEIGHT
PLEASE TYPE IN YOUR WEIGHT
PLEASE TYPE IN YOUR AGE
```

Note the print command ends in a comma.

EXAMPLE 23

```
10  FOR I=1 TO 12 STEP 3
20  PRINT I
30  NEXT I
40  PRINT 'I IS LEFT AT', I
RUN
```

The machine should print:

```
1
4
7
10
I IS LEFT AT 13.
```

EXAMPLE 24

```
10  INPUT A,B,C
20  LET  C=ABS(C)
30  IF B<A LET C = -C
40  FOR I=A TO B  STEP C
50  PRINT I
60  NEXT  I
RUN
```

A is the start value, B is the end value and C is the step. The program contains a check to ensure the correct sign for the step.

```
First try  A=1   B=10  C=3
Now try    A=10  B=1   C=-2
Then       A=50  B=-5  C=10
           A=4   B=20  C=-4
```

Type RUN for each execution.

EXAMPLE 25

```
10  FOR I = 1 TO 12
20  FOR J = 1 TO 12
30  PRINT #4, I*J,
40  NEXT J
50  PRINT
60  NEXT I
```

The machine should print the first 12 multiplication tables.

EXAMPLE 26

```
10  INPUT  'ENTER A NUMBER BETWEEN 1 AND 10' A
20  FOR I = 1 TO 10
30  IF I = A GOTO 60
40  PRINT  I
50  NEXT   I
60  FOR I = 10 TO A  STEP -1
70  PRINT  I
80  NEXT   I
RUN
```

The machine will loop inside the first FOR loop until A is reached, it then jumps to statement 60 and encounters another FOR using the variable I, the first FOR is terminated and execution continues with the second FOR loop.

EXAMPLE 27

```
10  LET X = 1 , Y = 8
20  FOR I = 48 TO 90
30  LET Z = X + Y * 64
40  VDU Z, I
50  X = X + 1
60  NEXT I
RUN
```

ADD 55 VDU Z, 32
70 GOTO 20

EXAMPLE 28

```
10   VDU  0, 12
20   FOR I=1 TO 150;NEXT I
30   LET Y = 4*64
40   FOR X = 10 TO 18
50   LET Z = X + Y
60   VDU Z, 42
70   VDU Z+Y, 42
80   NEXT X
90   LET X = 10
100  FOR Y = Y TO Y+Y  STEP 64
110  LET Z = X+Y
120  VDU Z, 42
130  VDU Z+8, 42
140  NEXT Y
RUN
```

This should clear the screen and draw a square of *'s. Note the delay after the clear screen (Statement 20).

Finally, to insert an extra line or modify an existing line in any program, it is only necessary to type the line number followed by the new text. They do not have to be in numerical order as they are sorted by BASIC. With Example 28 entered, enter the following:

```
40   FOR X = 10 TO 55
130  VDU Z+45, A
70   VDU Z+Y, A
150  NEXT A
120  VDU Z, A
60   VDU Z, A
5    FOR A = 1 TO 127
```

List this and you will find all statements now in numerical order. Run the program and see the changes.

WORKING PROGRAMS

BOUNCE (1K)

DESCRIPTION

BOUNCE is a small program written to show the use of the VDU command in memory mapping mode. When RUN is entered, the screen should clear and then the letter O will appear to bounce about the screen.

HOW IT WORKS

Statement 10 is used to clear the screen and reset the cursor. Five variables are then initialized, I and J being the horizontal and vertical co-ordinates for the ball position on the screen. K and L are the increments for these positions and N is the previous position. Statement 30 is the start of the main loop of the program. Here the increments are added to the horizontal and vertical positions. Statements 40 and 50 check that the horizontal position is still within the bounds of the screen. If it is greater than 64 or less than 1, the sign of the increment K is changed and added twice to bring the position back onto the screen.

Statements 70 and 80 similarly check the vertical position is on screen. Statement 100 then calculates the actual byte number M which I and J point to. The previous position is then blanked by Statement 110 and the new position displayed by Statement 120. Variable N is used to save the old position and the program jumps back to Statement 30 to continue. The initial values of K and L can be changed to produce different speeds and directions.

```
 10 VDU 0,12
 20 LET I=0,J=0,K=1,L=1,N=1
 30 LET I=I+K,J=J+L
 40 IF I>64 GOTO 60
 50 IF I>=1 GOTO 70
 60 LET K=-K,I=I+K+K
 70 IF J>16 GOTO 90
 80 IF J>=1 GOTO 100
 90 LET L=-L,J=J+L+L
100 LET M=I+(J-1)*64
110 VDU N,32
120 VDU M,79
130 LET N=M
140 GOTO 30
```

PATTERN (1K)

DESCRIPTION

PATTERN is a very similar program to BOUNCE. It uses the VDU command in memory mapping mode to draw a pattern on the screen.

HOW IT WORKS

PATTERN is almost identical to BOUNCE. The first difference is Statement 20 where K and L are different values to produce a different index.

Statement 110 is the next change - the previous position is no longer blanked out and hence a trail is left as the asterisk 'bounces' about the screen. Eventually the asterisk will start to follow the same path and will hence disappear.

Try different values of K and L to produce different patterns.

```
 10 VDU 0,12
 20 LET I=0,J=0,K=2,L=3
 30 LET I=I+K,J=J+L
 40 IF I>64 GOTO 60
 50 IF I>=1 GOTO 70
 60 LET K=-K,I=I+K+K
 70 IF J>16 GOTO 90
 80 IF J>=1 GOTO 100
 90 LET L=-L,J=J+L+L
100 LET M=I+(J-1)*64
110 VDU M,42
120 GOTO 30
```

POOLS (1K)

DESCRIPTION

POOLS is a small program written to help you pick your 'Win of a Life Time'. You tell the program how many teams to select from, how many teams to pick and how many times to repeat the selection. The program will then print the required number of teams, having selected them in random. This is repeated for the number of times required. Good Luck.

HOW IT WORKS

The program prints the title POOLS and then asks HOW MANY TEAMS ALL TOGETHER. This gives the total from which the selection can be made. Next the program then sets up a FOR loop to produce this number of selections at Statement 60.

The array is then zeroed for the total number of teams and another FOR loop set up for the required number of teams to be selected. The function RND is used to select a random team number from the total number and the array is checked to see if this has already been selected. If it has, RND is jumped back to try for another number. If the number has not been slected, the appropriate position in the array is set to one to indicate the number has been selected.

Statement 120 is the end of the inner most FOR loop and causes the program to jump back to Statement 80 until the required number of teams has been selected. After all the teams have been selected, the array is scanned and the index of all non zero elements printed giving the randomly selected teams.

```
 10 PRINT'POOLS'
 20 INPUT'HOW MANY TEAMS ALL TOGETHER'A
 30 INPUT'HOW MANY TO PICK'B
 40 IF B>A GOTO 30
 50 INPUT'HOW MANY TIMES'C
 60 FOR X=1 TO C
 70 FOR I=1 TO A;@(I)=0;NEXT I
 80 FOR Y=1 TO B
 90 Z=RND(A)
100 IF @(Z)<>0 GOTO 90
110 @(Z)=1
120 NEXT Y
130 FOR I=1 TO A
140 IF @(I)=0 GOTO 160
150 PRINT #4,I,
160 NEXT I
170 PRINT;PRINT
180 NEXT X
```

MISSILE (1K)

DESCRIPTION

Missile is a game which utilizes the graphics and memory mapping features available to the TRITON.

The object of the game is to shoot down each enemy aircraft before it gets past your defences (across the screen). To do this you command three missile sites.

Unfortunately, your radar station has been knocked out and the aircraft can come in at any height between 1 and 16 miles high.

The missile stations are sited at 16, 32 and 48 miles from the coast (left edge of the screen). Both the aircraft at 8 miles high, station 1 must fire with a delay of 8 seconds.

There are 10 aircraft coming in and before each, you must set the missile fuses by specifying the time delay before ignition.

Good Luck.

```
 10 N=0
 20 FOR J=1 TO 10
 30 @(2)=976,@(5)=992,@(8)=1008
 40 @(3)=976,@(6)=992,@(9)=1008
 50 PRINT'MISSILE'
 60 INPUT'1ST'@(1),'2ND'@(4),'3RD'@(7)
 70 VDU 0,12
 80 FOR I=1 TO 150:NEXT I
 90 Y=(RND(16)-1)*64,Z=1
100 FOR X=1 TO 64
110 VDU Z,32
120 W=Y+X
130 VDU W,62
140 Z=W
150 FOR I=1 TO 7 STEP 3
160 IF X<@(I) GOTO 250
170 VDU @(I+2),32
180 IF @(I+1)<1 GOTO 250
190 VDU @(I+1),94
200 @(I+2)=@(I+1)
210 @(I+1)=@(I+1)-64
220 IF @(I+2)=W GOTO 400
250 NEXT I
300 NEXT X
310 GOTO 500
400 VDU W,42
410 N=N+1
500 NEXT J
510 PRINT'HITS',N
```

REVERSAL (1K & 2K)

DESCRIPTION

The REVERSAL GAME is a game of skill played with the computer. The computer will arrange the numbers 1 to 9 in a random sequence and your job is to arrange them into ascending order in the minimum number of moves.

For example:

The machine might print the following sequence -

8 3 7 9 1 4 2 6 5

You now have to get these arranged in ascending order. To do this, you are allowed to reverse the order of any number of digits starting at the left hand side.

The machine will ask

NUMBER TO REVERSE

If you enter 4 the result will be

9 7 3 8 1 4 2 6 5

If you now enter 9 you will get

5 6 2 4 1 8 3 7 9

You now have 9 in the correct place and can set about getting 8 7 6 etc. in their correct positions.

After a few games, some short cuts will become apparent. Go on - see how good you are.

THE REVERSAL GAME (RUNS IN 1K)

```
 10 PRINT 'THE REVERSAL GAME'
 20 FOR I=1 TO 9
 30 LET A=RND(9)
 40 IF I=1 GOTO 80
 50 FOR J=1 TO I-1
 60 IF @(J)=A GOTO 30
 70 NEXT J
 80 LET @(I)=A
 90 NEXT I
 95 LET B=0
200 PRINT #2,@(1),@(2),@(3),@(4),@(5),@(6),@(7),@(8),@(9)
210 INPUT'NUMBER TO REVERSE'J
220 IF J<1 GOTO 240
230 IF J<10 GOTO 260
240 PRINT'INVALID - TRY AGAIN'
250 GOTO 210
260 LET K=(J+1)/2
270 FOR I=1 TO K
280 LET A=@(I),@(I)=@(J+1-I),@(J+1-I)=A
290 NEXT I
295 LET B=B+1
300 FOR I=1 TO 9
310 IF @(I)#I GOTO 200
320 NEXT I
330 PRINT'TOTAL',#3,B
340 GOTO 10
```

ALTERNATIVE ENDING FOR ABOVE (RUNS IN 2K)

```
320 NEXT I
400 IF B<6 PRINT 'EXCELLENT';GOTO 500
410 IF B<11 PRINT 'VERY GOOD';GOTO 500
420 IF B<16 PRINT 'GOOD';GOTO 500
430 IF B<21 PRINT 'NOT BAD';GOTO 500
440 PRINT 'BETTER LUCK NEXT TIME'
500 PRINT 'YOUR TOTAL WAS',#4,B
510 GOTO 10
```

BATTLE (2K)

DESCRIPTION

The BATTLE game is a simple version of the old game of battleships. In this game, both you and the computer have a grid of squares five by five, number 1 to 25.

YOURS					COMPUTERS				
1	2	3	4	5	1	2	3	4	5
6	7	8	9	10	6	7	8	9	10
11	12	13	14	15	11	12	13	14	15
16	17	18	19	20	16	17	18	19	20
21	22	23	24	25	21	22	23	24	25

The computer will hide four single square ships somewhere in its grid. It will ask you to hide your four ships in your grid by specifying their positions.

The computer will now ask which square you wish to fire at, check if you have scored a hit and then select one of your squares at random and shoot back.

The winner is the first to hit all four ships.

Be careful that you don't waste your shots by picking the same square twice.

```
 10 PRINT 'BATTLE GAME'
 15 LET N=4
 20 FOR I=1 TO 50;@(I)=0;NEXT I
 30 FOR I=1 TO N
 40 LET J=RND(25)
 50 IF @(J+25)#0 GOTO 40
 60 LET @(J+25)=1
 70 NEXT I
 80 PRINT 'WHAT ARE YOUR',#3,N,' POSITIONS'
 90 FOR I=1 TO N
100 INPUT J
114 IF J<1 PRINT 'INVALID POSITION - TRY AGAIN';GOTO 100
115 IF J>25 PRINT 'INVALID POSITION - TRY AGAIN';GOTO 100
120 IF @(J)#0 PRINT 'THIS POSITION IS USED - TRY AGAIN';GOTO 100
125 LET @(J)=1
130 NEXT I
140 LET A=N,B=N
200 INPUT 'WHERE DO YOU WANT TO FIRE AT'J
204 IF J<1 PRINT 'INVALID POSITION';GOTO 400
205 IF J>25 PRINT 'INVALID POSITION';GOTO 400
210 IF @(J+25)=2 GOTO 400
220 IF @(J+25)=1 GOTO 500
230 PRINT 'YOU MISSED ME - MY TURN NOW'
240 LET @(J+25)=2
250 LET J=RND(25)
260 IF @(J)=2 GOTO 250
270 IF @(J)=1 GOTO 600
280 PRINT 'I MISSED YOU - I PICKED',#3,J,' - YOUR TURN'
290 LET @(J)=2
300 GOTO 200
400 PRINT 'YOU WASTED THAT ONE - MY TURN NOW'
410 GOTO 250
500 LET A=A-1
510 PRINT 'YOU HIT ME - I HAVE',#2,A,' LEFT'
520 IF A=0 GOTO 700
530 PRINT 'MY TURN NOW'
540 GOTO 240
600 LET B=B-1
610 PRINT 'I HIT YOU - I PICKED',#3,J
620 PRINT 'YOU HAVE',#2,B,' LEFT'
630 IF B=0 GOTO 800
640 PRINT 'YOUR TURN'
650 GOTO 290
700 PRINT 'YOU HAVE JUST WON - I WILL GET YOU NEXT TIME'
710 GOTO 900
800 PRINT 'I HAVE WON AGAIN'
900 GOTO 10
```

NUMBER GAME (2K)

DESCRIPTION

This is a computer version of the popular game. Instead of coloured pegs, the machine will select a four digit number between 1111 and 8888 inclusive.

The idea of the game is to gues this random number in the minimum number of moves. To help you in this task, after each move, the machine prints the number of digits, correct and in the correct place (Black) and the number of digits correct but in the wrong place (White).

For instance, suppose the machine selected

4732

If your guess was

1725

the machine will print

BLACK = 1 WHITE = 1

The seven is correct and in the right place, hence BLACK = 1

The two is correct but in the wrong place, hence WHITE = 1

If your next gues is

7451

the machine will print

BLACK = 0 WHITE = 2

Seven and four are correct but in the wrong places.

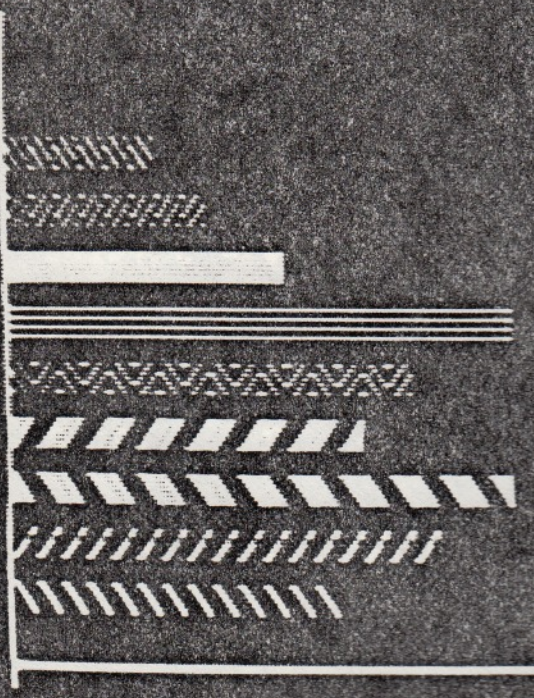
See what the minimum number of moves you can get the number in.

```
 10 LET @(15)=64,@(16)=174,@(17)=284,@(18)=394
 20 LET @(19)=165,@(20)=265,@(21)=365,@(22)=75
 30 LET @(23)=275,@(24)=375,@(25)=85,@(26)=185
 40 LET @(27)=385,@(28)=95,@(29)=195,@(30)=295
 50 PRINT'NUMBER GAME'
 60 LET K=0
 70 LET @(11)=RND(8),@(12)=RND(8),@(13)=RND(8),@(14)=RND(8)
 80 FOR I=1 TO 4;LET @(I)=@(I+10);NEXT I
 90 LET K=K+1
100 INPUT 'WHAT IS YOUR GUESS'D
110 LET A=D/1000,D=D-A*1000
120 LET B=D/100,D=D-B*100
130 LET C=D/10,D=D-C*10
140 LET @(7)=A,@(8)=B,@(9)=C,@(10)=D
150 LET @(5)=0,@(6)=0
160 FOR I=15 TO 30
170 LET A=@(I),B=A/100,C=A/10-B*10,D=A-B*100-C*10
180 IF @(B+1)=@(C+1) LET @(D+1)=@(D+1)+1,@(B+1)=10,@(C+1)=20
190 NEXT I
200 PRINT #2,'BLACK=',@(5),' WHITE=',@(6)
210 IF @(5)#4 GOTO 80
220 IF K<3 PRINT 'EXCELLENT';GOTO 270
230 IF K<6 PRINT 'VERY GOOD';GOTO 270
240 IF K<9 PRINT 'GOOD';GOTO 270
250 IF K<12 PRINT 'NOT BAD';GOTO 270
260 PRINT 'BETTER LUCK NEXT TIME'
270 PRINT 'YOUR TOTAL WAS',#4,K
280 GOTO 50
```

FURTHER READING

MICROPROCESSOR INTERFACING TECHNIQUES (expanded edition)	£7.45
BASIC CONCEPTS (Vol 1: Introduction to Microcomputers)	£5.75
UNDERSTANDING MICROCOMPUTERS	£7.45
BEGINNING BASIC	£7.12
STEP BY STEP INTRODUCTION TO 8080 MICROPROCESSOR	£5.70
HOME COMPUTERS - Beginner's Glossary and Guide	£4.95
HOME COMPUTERS - Q and Answers Vol 1: Hardware	£5.70
HOME COMPUTERS - Q and Answers Vol 2: Software	£4.95
8080 MACHINE LANGUAGE PROGRAMMING for beginners	£5.10
UNDERSTANDING COMPUTERS	£6.50
6800 COOKBOOK	£7.15
8080 COOKBOOK	£7.15
AN INTRODUCTION TO PERSONAL AND BUSINESS COMPUTING	£5.75
8080 PROGRAMMING FOR LOGIC DESIGN	£5.95
SOME COMMON BASIC PROGRAMS	£5.95
8080 PROGRAMMERS' POCKET GUIDE	£2.25
8080 HEX CODE CARD	£2.25
PCC REFERENCE BOOK	£4.95
WHAT TO DO AFTER YOU HIT RETURN	£7.00
HOBBY COMPUTERS ARE HERE	£3.95
NEW HOBBY COMPUTERS	£3.95
INSTANT BASIC	£4.95
YOUR HOME COMPUTER	£3.95
MY COMPUTER LIKES ME WHEN I SPEAK BASIC	£1.65
BASIC COMPUTER GAMES (revised 101 Basic games)	£5.50
COMPUTER PROGRAMS THAT WORK IN BASIC	£2.55
MICROPROCESSORS from Chips to Systems	£7.45

Data sheets are available on most ICs from Transam Components Ltd. Please contact us for information if required.



MANY DIFFERENT
VARIETIES OF BAR GRAPH
GRAPHICS CAN BE USED
ON THE TRITON. THESE
ARE JUST A FEW!_

EFFECT OF ALPHA KEYS IN CONTROL MODE

CONTROL A =
CONTROL B =
CONTROL E =
CONTROL F =
CONTROL G =
CONTROL N =
CONTROL O =
CONTROL P =
CONTROL Q =
CONTROL R =
CONTROL S =
CONTROL T =

CONTROL U =
CONTROL V =
CONTROL W =
CONTROL X =
CONTROL Y =
CONTROL Z =
CONTROL H = STEP CURSOR LEFT
CONTROL I = STEP CURSOR RIGHT
CONTROL J = STEP CURSOR DOWN
CONTROL K = STEP CURSOR UP
CONTROL L = CLEAR SCREEN/RESET CURSOR
CONTROL M = CARRIAGE RETURN/LINE CLEAR

CONTROL C REINITIALISES AND CONTROL D IS EOT MARKER _

SHIFT A = ♠ SHIFT G = ▽ SHIFT M = ┌ SHIFT T = |

SHIFT B = ♦ SHIFT H = ▯ SHIFT N = ┐ SHIFT U = ▄

SHIFT C = ♥ SHIFT I = | SHIFT O = ├ SHIFT V = ||

SHIFT D = ♠ SHIFT J = | SHIFT P = ┤ SHIFT W = ┼

SHIFT E = ▯ SHIFT K = ¯ SHIFT Q = / SHIFT X = ||

SHIFT F = ▽ SHIFT L = _ SHIFT R = \ SHIFT Y = X

EFFECT OF KEYS
IN "SHIFT ALPHA" MODE

SHIFT S = _ SHIFT Z = ▌

SOFTWARE CAN BE USED IN CONJUNCTION
WITH GRAPHICS TO PRODUCE LARGE ALPHA-
NUMERIC DISPLAYS LIKE THIS.