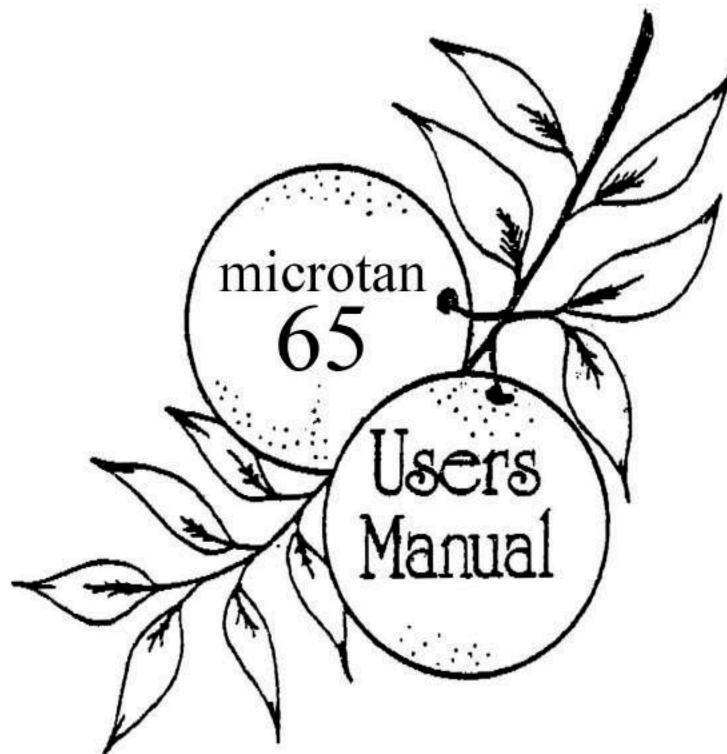
TANGERINE



TANGERINE
COMPUTER SYSTEMS LIMITED

FOREWORD

CHAPTER 1 Microprocessors and Binary Numbers.

CHAPTER 2 Constructional Notes.

CHAPTER 3 The Microtan 65.

CHAPTER 4 The Microtan System.

CHAPTER 5 The 6502 Microprocessor.
Tables of Machine Instructions.

CHAPTER 6 TANBUG V2

Copyright TANGERINE Computer Systems Ltd.
All Rights Reserved.

FOREWORD

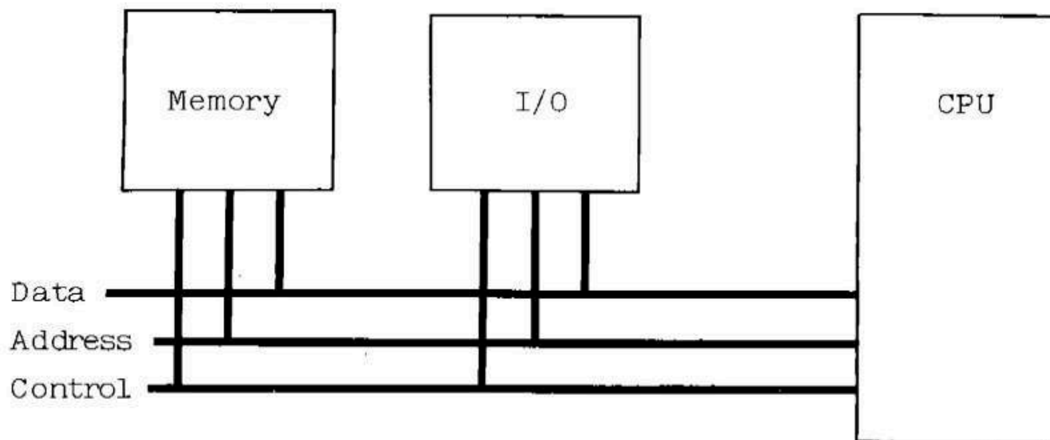
Unlike most other microcomputers, microtan 65 has been designed from the start with a small system in mind, therefore expansion of microtan 65 is a concept, not an after thought. We hope that this manual will be the first of many, as each Tangerine product is supplied with thorough and useful documentation, an absolute necessity with anything related to computers. The first chapter of this manual provides an introduction to microprocessors and the binary number system, it is by no means complete as this subject requires a complete book to be dedicated to it. Chapter two gives constructional notes for the kit version but purchasers of ready built microtan 65's are still advised to read it as it contains relevant information. Chapters three and four describe the microtan 65 and the microtan system respectively. The 6502 microprocessor is described in sufficient detail in chapter five which also contains complete tables of all of the 6502 machine code instructions. TANBUG, the most important item, is fully described in chapter six with a step by step guide and an example program to demonstrate the power of TANBUG's debugging aids. Also given in chapter six is a table of hex ASCII codes and the complete TANBUG listing.

Microprocessors and Binary Numbers

Microprocessors have been with us for a few years now. Their concept is not exactly new. The idea of a general purpose electronic processor controlled by a stored program is at least thirty years old. Only in the last few years has it been the size of a thumbnail, rather than a garden shed; and it costs a few pounds rather than a few hundred thousand. It has been integrated circuit manufacturing technology that has advanced and given the world the computer on a chip - the microprocessor. The revolution we hear about is the result of microprocessor applications. It may be the "sledge hammer to crack a walnut" tactic to use a computer to control a washing machine but if it is the cheapest, most reliable and the best way to do it, so what!

Programming a microprocessor to perform a certain task is not unlike using a programmable calculator. The microprocessor program is at a much more fundamental level though. In a programmable calculator there are usually a number of data registers and a separate program memory. Data flow is under control of the programmer when he/she writes the program. In a microprocessor system the program and data memories are not usually separate. It is possible to store an instruction or data at any particular address. The main difference is that a programmable calculator is dedicated to solving arithmetic problems and the user interfaces are specialised to that task. Results are displayed as decimal numbers and the machine responds at the press of a button to a particular, defined operation. The microprocessor is a general purpose device and must be designed into a system and programmed to perform the task required of it.

A simple computer system is shown diagramatically below.



The block marked as memory can be RAM or ROM and the block marked I/O is a special circuit for interfacing a peripheral device to the microprocessor address, data and control bus. Address and data buses are dedicated, as would be expected to memory and I/O address and data. The control bus controls and synchronises all other circuitry to that of the cpu. In its simplest form the cpu would contain only two registers; an accumulator and a program counter. The accumulator is the working register where all data is fetched and manipulated according to the instruction being executed. The program counter normally increments to the next instruction in memory but on occasions the program counter jumps to a new location, depending on the result of the previous operation, or to a subroutine.

Of course most computing systems, including microprocessors, are much more complicated than this, having several internal registers with specialised tasks inside the cpu. Some memory locations external to the cpu may also be assigned special tasks, such as interrupt vectoring. However complicated a computer is it, like all others, uses binary notation for all its data and instructions. This means that the programmer is working with data and machine operations at a very fundamental level.

The data used in a microprocessor consists of a group of binary bits; 8 in an 8 bit microprocessor. A binary bit can only take one

of two values, either Ø or 1. The numbering system that uses binary digits has a base of 2. To explain this first consider the all familiar decimal system. We start counting from Ø up to 9, at which point the digit returns to Ø and the next digit to the left increases by one. Each digit to the left is worth ten times as much as the digit to the right of it. In binary, the same thing happens except that a digit returns to Ø after only reaching 1, and each digit of increasing significance is worth twice as much as the digit to the right of it. Even when using binary numbers it is hard for a human being to leave decimal entirely behind. The left most column is worth only 1 unit in decimal, the next digit is worth 2, then 4, 8, 16, 32, 64, 128 and so on. Each of these numbers are in fact $2^{Ø}$, 2^1, 2^2, 2^3, 2^4, 2^5, 2^6, 2^7 and so on, (in decimal the digit positions represent $1Ø^{Ø}$, $1Ø^1$, $1Ø^2$, $1Ø^3$ and so on) and denote the respective bits (digits) bit Ø, bit 1, bit 2, bit 3 etc. Consequently, in an 8 bit microprocessor, these take the labels DØ-D7.

To represent a decimal number in binary, divide continuously by 2. The first time the division takes place the remainder becomes the first bit of the binary representation, i.e. bit Ø. The remainder from the next division becomes bit 1, and so on until division by 2 is not possible. For example, find the binary representation of the decimal number 57.

57 ÷ 2 = 28	remainder 1	– Bit Ø = 1
28 ÷ 2 = 14	remainder Ø	– Bit 1 = Ø
14 ÷ 2 = 7	remainder Ø	– Bit 2 = Ø
7 ÷ 2 = 3	remainder 1	– Bit 3 = 1
3 ÷ 2 = 1	remainder 1	– Bit 4 = 1
1 ÷ 2 not possible,	remainder 1	– Bit 5 = 1

Therefore decimal 57 = binary 111ØØ1

As another example, find the binary representation of the decimal number 26.

26 ÷ 2 = 13	remainder Ø	– Bit Ø = Ø
13 ÷ 2 = 6	remainder 1	– Bit 1 = 1
6 ÷ 2 = 3	remainder Ø	– Bit 2 = Ø
3 ÷ 2 = 1	remainder 1	– Bit 3 = 1
1 ÷ 2 not possible,	remainder 1	– Bit 4 = 1

and so decimal 26 = binary 11Ø1Ø

Adding together two binary numbers involves the same process as

adding two decimal numbers. The digits (in decimal) or bit (in binary) of equal weighting are added together, and if larger than 9 in decimal, or 1 in binary, a carry is made into the next column. For example adding 57 and 26 in decimal would be familiar in decimal as

```
5  7
2  6    +
8  3
1
```

in binary it is just the same.

```
 1 1 1 Ø Ø 1  +        57
   1 1 Ø 1 Ø           26
1 Ø 1 Ø Ø 1 1
1 1 1
```

Subtraction of binary numbers obeys the same rules as subtraction of decimal numbers. The problem for computers lies in the fact that its internal circuits can only recognise 1's and Ø's. There is no minus sign (-) to a computer. To overcome this we use an arithmetic technique called "two's complement". This involves treating the most significant bit of a binary number as a sign bit, Ø representing positive and a 1 representing negative. In an 8 bit microprocessor this will be D7. The two's complement of a binary number is formed by changing all Ø's to 1's and all 1's to Ø's, (this is known as inverting each bit) and then adding 1. Thus -1 becomes 11111111, -2 is 1111111Ø and so on. To understand this concept the equivalent in the decimal system can be considered. Imagine a car speedometer milage counter. Originally the counter is at ØØØØ. By driving forwards 4 miles the reading will become ØØØ4. Reverse 4 miles and the reading is back to ØØØØ. Reverse a further mile i.e. 1 mile backwards or -1 mile from the original position and the milage counter reads 9999. Reversing a further mile results in 9998 and so on. In two's complement binary with an 8 bit data word the largest positive number that can be represented is Ø1111111, or +127, and the largest negative number is 1ØØØØØØØ, or -128.

So how does this work in arithmetic, as an example take 26 from 57. The microprocessor would first find the two's complement of 26, as this is the number to be negated, even though the instruction would be a subtraction instruction. The two's complement of 26 is

11100110. Now add 57 and -26.

```
   0 0 1 1 1 0 0 1         57
   1 1 1 0 0 1 1 0   +    -26
 1 0 0 0 1 1 1 1 1         31
 1 1 1
```

The extra bit on the end is the carry bit and is a 1. This indicates that the result is positive. Now turn things over and subtract 57 from 26. The two's complement of 57 is 11000111.

```
   0 0 0 1 1 0 1 0         26
   1 1 0 0 0 1 1 1   +    -57
 0 1 1 1 0 0 0 0 1        -31
       1 1 1 1
```

No carry was generated and Bit 7 is a 1 indicating a negative number. This combination means that the number is negative and in two's complement form.

Obviously a user would not be too happy to have the result of calculations presented to him/her as binary numbers. A computer can of course use a program to convert between binary and decimal numbers. It is, however, quite simple and efficient to operate the computer in a decimal mode using binary coded decimal. Binary coded decimal is the grouping of four binary bits representing a decimal digit. As four binary bits enables the decimal numbers 0-15 to be represented, the binary codes representing 10-15 are not allowed. With an 8 bit data size computer each data word can represent two binary coded decimal blocks as shown

```
D7 D6 D5 D4    D3 D2 D1 D0
0  1  1  0     0  0  0  1
      |              |
      6              1
```

Larger numbers can be represented by cascading data words. For instance if it was required to represent 2925 in binary coded decimal, four blocks of four bits would be required, i.e. one block for each decimal digit.

```
0010   1001   0010   0101
  |      |      |      |
  2      9      2      5
```

By using the same idea as binary coded decimal, binary numbers can be represented by hexadecimal characters instead of long rows of 0's and 1's. As in BCD each hexadecimal character is formed

by a group of four binary bits. This time however, the four bit codes representing decimal 1Ø-15 are allowed and are labelled A-F respectively. For example

```
1100   0111
 |      |
 C      7
```

Obviously representing 11ØØØ111 as C7 is far more convenient. Hexadecimal is exceptionally useful when programs are written for microtan 65. Each program instruction consists of 1, 2 or 3, 8 bit words, or bytes as they are more commonly known. Therefore, if it were not for hexadecimal each program instruction would entail the entry, on a keyboard, of 8, 12 or 24 binary bits as opposed to 2, 4 or 6 hex characters.

Constructional Notes

Although this chapter is intended mainly for those who have purchased the microtan 65 in kit form, it is still recommended reading for those who have purchased a ready built microtan 65, as there is some relevant information given. Before you begin assembly please read the instructions carefully so that you have a good idea as to the sequence of operations you must perform. For assembly you will require a miniature soldering iron, thin multi-cored solder, pliers and wirecutters (both of the small variety). Provided you can solder reasonably well, you will have no problems in assembling the kit and should be up and running with TANBUG in a few hours. First, however, a few precautions:

1) The printed circuit board is of the plated through hole type. This means that every hole in the board has a layer of metallisation as a lining to the hole, electrically connecting the track on both sides of the board. This makes the board very expensive and also makes it very difficult to remove a component once it has been soldered in unless expensive equipment is used. Therefore make sure the correct component is inserted before soldering it in place. It is for this reason that sockets have been provided for the integrated circuits.
2) When soldering do not apply pressure to the printed circuit board, otherwise tracks may lift and break.
3) Although modern components are not so easily damaged by heat, a joint made quickly is less likely to go "dry".
4) MOS devices are used in the microtan 65 and these can be destroyed by static electricity. These devices also happen to be the most expensive! When it is time to handle these circuits take precautions against static such as; do not wear nylon clothes, ensure the soldering iron is properly earthed, use a sheet of aluminium foil to work on

and earth it to a radiator or a water pipe with a piece of wire.

5) Wash your hands. Dirt and grease on the printed circuit board makes soldering difficult and unreliable.
6) Solder only component leads on the track side.
7) DO NOT HURRY.

As a rough guide the following procedure for assembly is suggested:

a) Unpack the kit and check and identify all of the components as listed at the end of this chapter.
b) Fit and solder the I.C. sockets ensuring that they are the correct way round, the pin 1 identifier being at the end as marked on the printed circuit board. Note that the side of the board that the components are on is the side with the printed legend.
c) Fit and solder the keyboard socket in position A4.
d) Fit and solder the resistors in their correct positions.
e) Insert the four wire links LKNMI, LKRAM, LKROM and LKIO using the excess wire cut off of the resistors.
f) Fit and solder the capacitors. No electrolytics are used so there are no problems about polarity.
g) Fit and solder all of the diodes. Check polarity.
h) Fit and solder the two transistors. The leads are preformed and they will only fit into the board the correct way round.
i) Fit and solder the inductor, the UHF modulator and finally the crystal.
j) If you have purchased and edge connector, now is the time to fit and solder it onto the board.
k) Insert the integrated circuits, ensuring that they are the correct way round, into their respective sockets leaving the MOS devices to last. These are the 6502, the 2114's and if you have the graphics option, the 2102.

Assembly is now complete, but carefully double check and make sure that there are no solder blobs or bridges anywhere. The microtan

65 is now ready for connecting up to the t.v. receiver, keyboard and power supply.

Keyboard Connection

If you are using the keypad or a Tangerine alphanumeric keyboard, just plug them into the keyboard socket. If you are using some other alphanumeric keyboard, it will be necessary to wire it into a 16 pin DIL plug as follows. Pins 1 to 7 are the ASCII data inputs bits 1 to 7 respectively, bit 7 being the most significant bit. This data should be active high. Pin 15 is the keyboard strobe input. Microtan 65 recognises that a character has been typed by a positive edge at this input. The ASCII data should be stable when this edge occurs and remain stable for several milliseconds. It may be necessary to add inverters and/or latches to make your alphanumeric keyboard comply with microtan 65's requirements. Pin 8 is a GND connection and pin 16 provides +5 volts to power the keyboard. A reset button, or key, may be connected between pin 14 and GND. Microtan 65 must have a reset key fitted either on the keyboard socket or on the TANBUS connector.

Power Supply Connections

Microtan 65 requires only a single +5 volts power supply, 1 amp being ample for microtan 65 and a keyboard. If the user is contemplating expanding his system, then a power supply of higher rating would be desirable. If the microtan 65 is being used on its own then the power supply may be connected into the two positions marked on the printed circuit board.
WARNING: ALWAYS TURN OFF THE POWER SUPPLY WHEN REMOVING OR INSERTING INTEGRATED CIRCUITS, THE KEYBOARD, OR THE BOARD FROM A RACK. NEVER SOLDER TO A POWERED UP BOARD.

T.V. CONNECTION

The simplest procedure of all. Use a UHF lead with the correct connectors either end. Plug the lead into the t.v. aerial socket and into the UHF modulator output socket.

TURN ON THE POWER - DEPRESS THE RESET KEY.

TANBUG should be printed on the display together with a prompt to type in commands.

Graphics Option

The graphics option consists of five integrated circuits. These should be inserted into their respective positions.

Lower Case Option

The lower case option consists of two integrated circuits. These should also be inserted into their respective positions. This option provides lower case alphanumerics and symbols having hex ASCII codes of ØØ-1F and 6Ø-7F. If these codes are used without the lower case option the upper case equivalents will not be displayed in their place. What is more TANBUG's prompt character "█" will be replaced by a "?". Error indication is still a question mark.

Address Buffers

There is space on the microtan 65 for three integrated circuits which perform address buffering. These are only required when expanding the system but are provided nevertheless.

Component List and Identification

Integrated Circuits:

D1	74LS04	E1	74LS86
F1	74LS08	G1	74LS74
H1	74LS21	J1	74LS74
K1	6502	L1	74LS367
C2	74LS73	D2	74LS393
E2	74LS74 (lower case)	F2	74LS157
G2	74LS157	H2	74LS157
J2	74LS74	L2	74LS367
B3	74LS74	C3	74LS11
D3	74LS393	E3	74LS00
F3	2114	G3	2114
H3	2102 (graphics)	J3	74LS00
K3	TANBUG	L3	74LS367
B4	74LS244	C4	74LS374
D4	86S64BWF	E4	8678CAE or 86S64CAE (lower case)
F4	74LS251 (graphics)	G4	74LS374 (graphics)
H4	74LS374 (graphics)	J4	74LS138
L4	74LS74 (graphics)		

Resistors:

R1	1K	(brown,black,red)	R10	1K	(brown,black,red)
R2	1K	(brown,black,red)	R11	1K	(brown,black,red)
R3	10K	(brown,black,orange)	R12	1K	(brown,black,red)
R4	1K	(brown,black,red)	R13	1K	(brown,black,red)
R5	470R	(yellow,mauve,brown)	R14	1K	(brown,black,red)
R6	220R	(red,red,brown)	R15	1K	(brown,black,red)
R7	75R	(mauve,green,black)	R16	1K	(brown,black,red)
R8	1K	(brown,black,red)	R17	10K	(brown,black,orange)
R9	1K	(brown,black,red)			

Capacitors:

C1	10n	C10	47n	
C2	n10 or 100p	C11	47n	
C3	n10 or 100p	C12	47n	
C4	10n	C13	47n	
C5	47n	C14	47n	
C6	47n	C15	47n	
C7	47n	C16	47n	
C8	47n	C17	1n	
C9	47n	C18	47uF	+ve to "C18"

Miscellaneous:

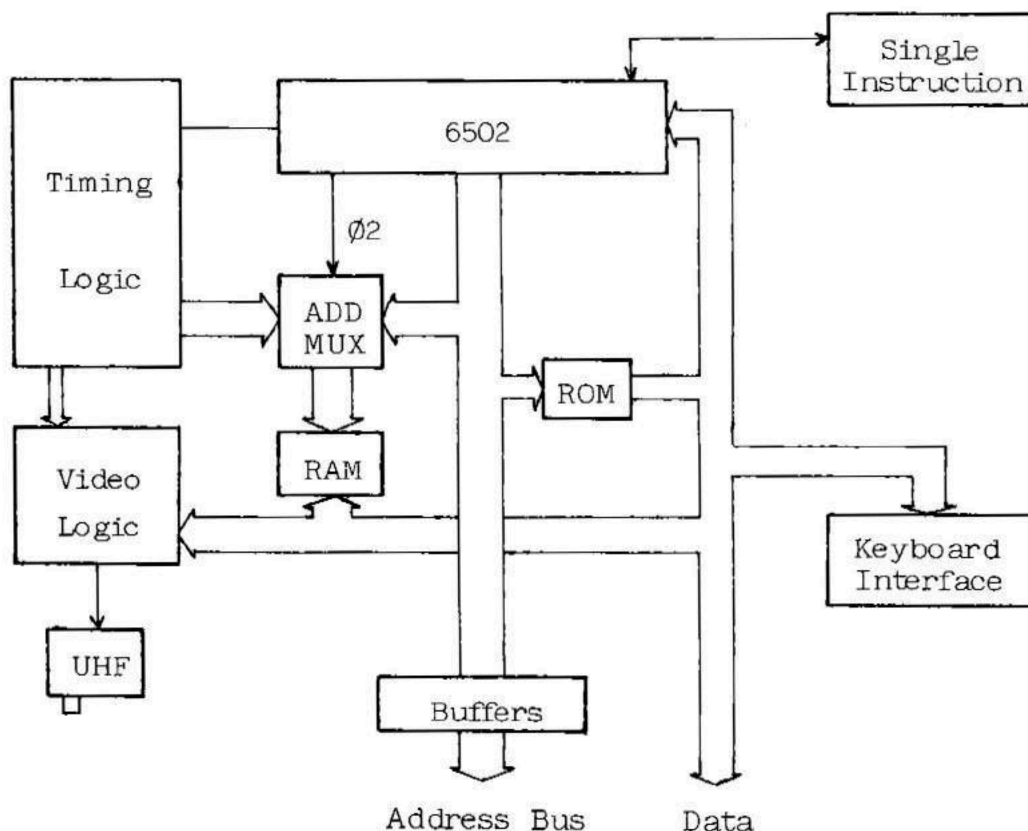
6MHz Crystal	Q1, Q2, Q3 - BC184
D1, D2, D3 - 1N4148	ZD1 - BZY88C6V8
CK1 Green Inductor	UM1111E36 Modulator
PCB - MT016	Edge Connector
16 off 14 pin sockets	12 off 16 pin sockets
2 off 18 pin sockets	4 off 20 pin sockets
1 off 24 pin socket	1 off 40 pin socket

1 off 16 pin R-N socket for keyboard connection.

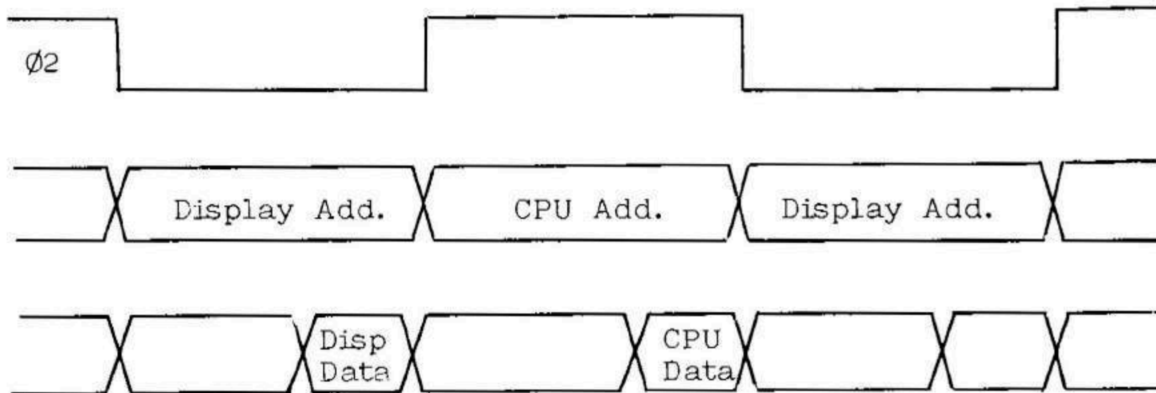
The Microtan 65

The microtan 65 is an exceptional microcomputer kit and the purpose of this section is to describe its design. It was considered that the majority of microkit users would initially benefit from a machine that was very easy to use rather than one that has parallel I/O, but was very awkward to use. Therefore microtan 65 is supplied with a VDU on-board, and because of this the ability to use an alphanumeric keyboard is an immediate asset. The 65Ø2 microprocessor was chosen mainly for its very simple yet elegant hardware structure. Of course, as most readers will know, the 65Ø2 also has a very powerful instruction set and addressing modes.

A fully expanded microtan 65 contains the 65Ø2 microprocessor, 1K ROM containing TANBUG, 1K RAM which is used for display memory and user program, keyboard interface, VDU logic and tri-state address buffers. The basic block diagram of the microtan 65 is shown below.



The most important aspect of the design is that the address multiplexer is switching at the processor clock rate. This can be done with the 65Ø2 as memory accesses will only occur on one phase of the clock i.e. when Ø2 is high. When Ø2 is low the memory is not selected. During this time the VDU logic reads the display memory, one location at a time and decodes memory contents as alphanumeric or graphics characters. You will also notice when using microtan 65 that the display is free from annoying speckles, spots and flashes. This is because there is no conflict of access to display memory between processor and display refresh logic. In fact you can run a program that actually resides in display memory and it will run at full speed without upsetting the display! The address and data bus timing is shown below to illustrate this.

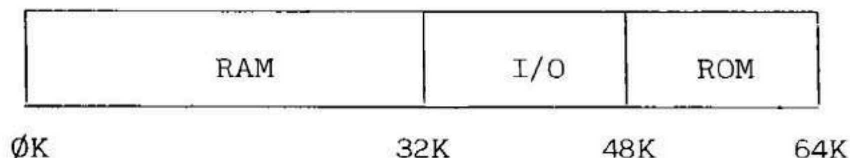


Memory mapped display: Pages 2 and 3 of the address space (see chapter on the 65Ø2) are used as the display memory, that is from hex location 2ØØ to hex 3FF. Location 2ØØ is the top left hand corner character position. Location 2Ø1 is the second character position in the top row and so on. The diagram below illustrates this more clearly.

Top Row	2ØØ	2Ø1	2Ø2		21E	21F
	22Ø	221				23F

To write a particular character into a character cell in the display the user must write the ASCII code, or whatever, for that character in the correct memory location. For example, if it was required to write the character "W" in the bottom row third column then the ASCII code 57 (for W) should be placed in memory location 3E2. If you use TANBUG's memory modify command M to load characters into the display remember that the display scrolls automatically and the character may be output onto a different row than intended.

Memory mapping of the microtan 65 is controlled on-board by three wire links, when used without the TANEX expansion board. As there is only 1K RAM, 1K ROM and 6 I/O locations on the microtan 65 each of the three are allowed to repeat through defined areas of the 64K address space to simplify design. The address map is represented diagramatically below.

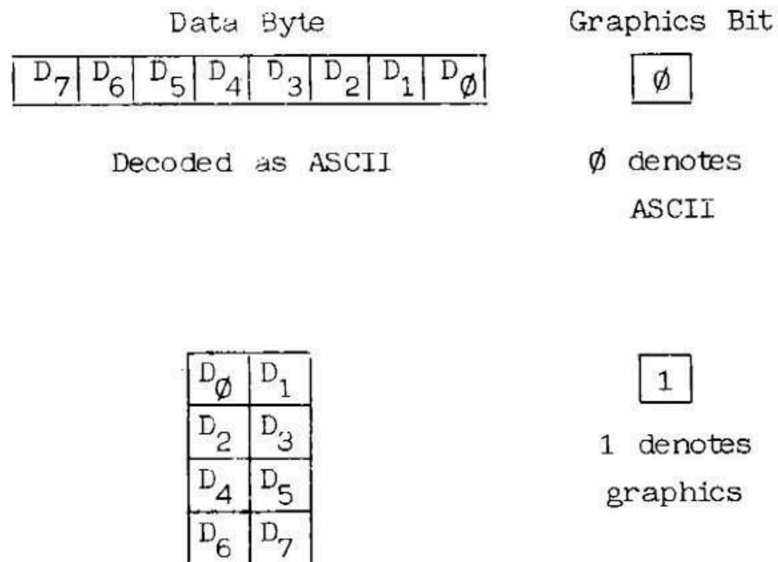


As can be seen the address space is divided into only three blocks and therefore only two bits of the address bus are necessary to control this map. The two bits used for memory mapping are the most significant address bits A15 and A14. Three links are used to wire in this address map on the microtan 65. When TANEX is used

however, all memory mapping is controlled on the TANEX board and not on the microtan 65. To modify the microtan 65 it is only necessary to cut these three links.

RAM and ROM hardware is of really no consequence to the user but a knowledge of the on-board I/O ports is useful. There are six peripheral ports used on the microtan 65 cpu board and these control graphics on and off, keyboard read and write, keyboard interrupt flag clear and delayed non-maskable interrupt (used for single instruction and breakpoints). Each one of these will be taken in turn:

a) Graphics on and off: The display refresh controller reads the display memory a byte at a time and interprets the bottom seven bits of each byte to be an ASCII coded character. There is however a ninth bit which can only be written to by the cpu. This ninth bit comes from the 21Ø2 1K x 1 bit RAM chip in the graphics option. If at a certain display location the ninth bit, or graphics bit, is a logic one the data byte read is not decoded as an ASCII character but is instead used to build up a graphics block of 2 x 4 pixels, in the character cell position.



Graphics pixels are illuminated if corresponding bit is a logic one.

The graphics on and graphics off I/O ports control an R-S flip-flop, the output of which is the data input to the 21Ø2 RAM chip. Therefore if the graphics flip-flop is in the on position each character written to the display memory will be decoded as a graphics character. Because the cpu cannot read the graphics bit directly it is not possible to scroll displays containing graphics. As displays containing graphics are rarely scrolled anyway this should not prove to be too much of a handicap.

The operations required to write graphics characters to the display is to firstly turn the graphics on by executing the assembly code instruction LDA BFFØ. Each character then written to display will be a graphics character until graphics are turned off by executing the assembly code instruction STA BFF3.

b) Keyboard read, write and interrupt clear: There is both an input and output port on the keyboard socket. The output port is used only for strobing the 20 key keypad. This output port is used by executing the assembly code instruction STA BFF2. The input port is used for both types of keyboard. When using the alphanumeric keyboard the strobe from the keyboard is used to clock a flip-flop (the keyboard interrupt flip-flop). The output of this flip-flop is the keyboard interrupt flag and forms the eigth input bit to the keyboard input port. If interrupts are enabled in the cpu then an interrupt will be generated by the keyboard strobe. Reading the keyboard input port will then allow the cpu to test whether or not the interrupt was generated by the keyboard or some other device. If interrupts are not enabled then the keyboard interrupt flag will still be set but will not generate an interrupt. It is possible to read the flag by reading the keyboard input port, thus a strobe from an alphanumeric keyboard can be

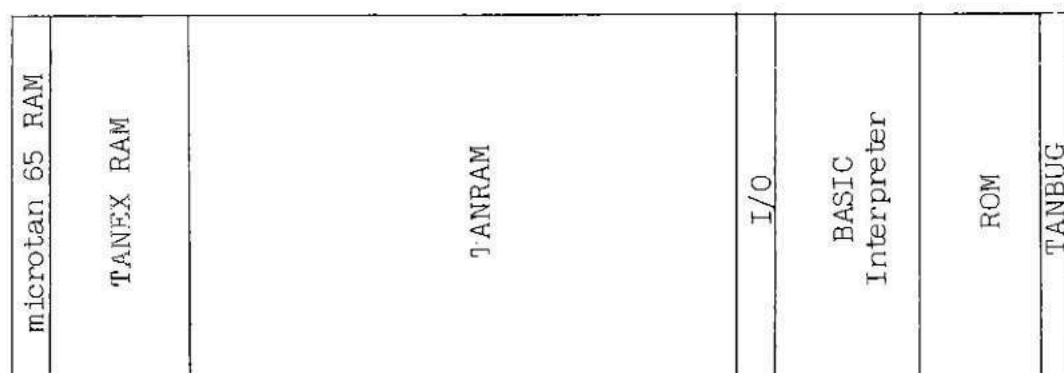
detected either by an interrupt or by software polling. Which ever technique is used though it will be necessary to reset the keyboard interrupt flag after it has been set and detected in order that it is ready to register a further keyboard strobe. To reset the keyboard interrupt flag the assembly instruction STA BFFØ should be executed. The keyboard port is read by the instruction LDA BFF3.

c) Delayed non-maskable interrupt: This facility is used by the monitor program TANBUG when executing single instructions and breakpoints. It should not be used in a user program. Users who will contemplate using the non-maskable interrupt in special applications will probably not be using TANBUG. In these cases the link LKNMI should be broken and the non-maskable interrupt may be driven via the board connector.

The Microtan System

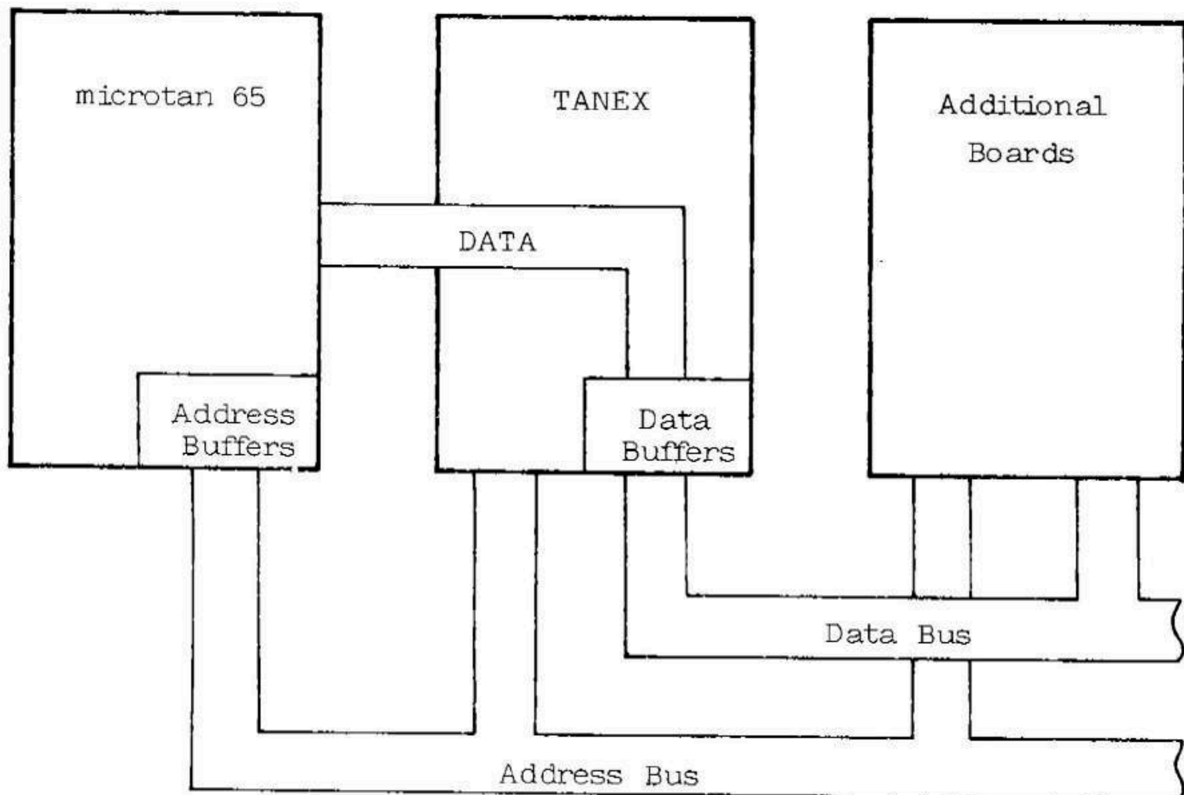
Microtan 65 expands into a full microcomputer system. All the printed circuit cards connect into a motherboard and use TANBUS as means to communicate to each other. TANBUS is a collective name for all of the signals that use the edge connector on each of the cards. The cpu card and expansion board have dedicated positions in the motherboard as there are slight differences between their requirements and that of other boards in the system. Because of this, these two positions are offset and only the correct boards will fit when used in the racking system. All the other positions are identical and boards designed to fit in these positions can be fitted in any one of them.

Expansion into a system is by way of the expansion board - hence its name. Its principle task is to completely control the memory mapping of the system. On its own the microtan 65 has a simple and hardware efficient memory map which is incompatible with a system. The expansion board separates the three blocks of memory space (RAM, ROM and I/O) of the microtan 65 into much smaller and efficiently used regions. This control is by way of the TANBUS signals RAME, ROME and IOE and produces the following memory map.



Details of the memory map pertaining to TANEX and TANRAM (4ØK bytes of static and dynamic memory) is in the manuals that are supplied with them. The 1K of I/O space includes any I/O on the microtan 65 and on TANEX. There is however plenty of space available for the users own I/O devices, addresses of which should start at the bottom of the I/O space (i.e. location BCØØ). Addresses for I/O devices specified by Tangerine start at the top of the I/O space and work downwards. It is very unlikely that the two will meet in the middle as 1K of I/O is quite large. (Micro-computers based on 8Ø8Ø and Z8Ø are limited to 256 I/O devices - microtan offers four times as many!).

The main signals on TANBUG are the address bus and data bus and as both of these need to be used with all the printed circuit cards likely to be used in the complete system they must be buffered in order to have a high driving capability. The address bus is buffered on the microtan 65 (along with the R/W line) using tri-state buffers so that the address bus may be tri-stated for DMA's (direct memory access). These buffers do not operate with the on-board RAM, ROM and I/O of the microtan 65 and so may not be accessed during DMA's. As there is no point in having a DMA to TANBUG or the small amount of RAM on the microtan 65 this is hardly a problem. Similarly with on-board I/O, there is no port on microtan 65 that can effectively be used in a DMA operation. Data bus buffering is performed on the expansion board and is bi-directional and tri-statable. Two control lines on the expansion board control these buffers. One turns them on and the other defines their direction i.e. is the cpu reading or writing. These buffers are turned on when the microtan 65 is performing a memory access in some other part of the system than itself. If it is performing an on-board access, such as to TANBUG or the RAM, then these buffers are turned off to prevent on-board circuits and off-board circuits both trying to drive the data bus. A block diagram of this scheme of buffering is shown below.



There are two other very important boards in the microtan system, TANRAM and TANDISC. TANRAM, as mentioned earlier is a 40K byte static and dynamic RAM board. 7K bytes of RAM are static devices using the popular 4K bit static memory device. The remaining 32K bytes of RAM are dynamic and use the 16K industry standard 4116. Refresh of the dynamic RAM is on-board and totally transparent to other parts of the system. It is also unaffected by DMA's. TANDISC is the all important disc controller which can handle up to four disc drives.

For those users who do not wish to expand to a full system there is the mini-motherboard and case which may be used to connect and house the microtan 65 and TANEX. Also included in the housing is a small power supply capable of meeting the needs of fully expanded boards. Although just two boards in a small housing this combination is very powerful. Fully expanded this mini system offers 8K RAM, 6K ROM, 8K BASIC interpreter, 32 I/O lines, 3 serial I/O (one with 16 baud rates and RS232/20mA) 4 counter timers and cassette interface.

TANBUS specification

A description of all signals on the backplane is now given with

indications as to their use, where applicable. The reader should refer to the chart of TANBUS connections at the end of this specification. Note that some connections are left blank. These have yet to be defined and most of them will almost certainly be used by future products for the microtan system, therefore the user is advised not to commit a custom design to specific connections on unused lines of TANBUS, it may produce incompatibility in the future.

Pin mnemonic	Description
+5	+5 volts power supply input.
+12	+12 volts power supply input.
-12	-12 volts power supply input.
GND	Earth return or Ø volts.
CLK	6MHz clock.
$\overline{\text{DMAREQ}}$	Used by peripheral devices to request control of TANBUS for direct memory access.
Ø1	Microprocessor clock phase 1.
Ø2	Microprocessor clock phase 2.
$\overline{\text{RST}}$	Used to reset the complete microtan system. When TANRAM, or any peripheral device with dynamic memories, is being used the reset line must only be active for about 1ØuS.
$\overline{\text{I/O}}$	An output from TANEX to indicate that the address bus is addressing an I/O device, i.e. the address is between BCØØ and BFFF.
A1-A15	Address Bus. Most of the time driven by the microtan 65 but handed over to a peripheral device when it performs a direct memory access.
$\overline{\text{ABE}}$	Address bus enable. An output from TANEX to the microtan 65. Used to disable the address buffers so that DMA's may be performed. Also disables the R/W buffer. Not a bussed signal.
$\overline{\text{DMAGNT}}$	An output from TANEX to indicate that the cpu has halted and the microtan 65 address buffers have been disabled so that the requesting peripheral may proceed with a DMA.

Pin mnemonic	Description
$\overline{\text{IRQ}}$	Interrupt request. An open collector line used by devices requesting an interrupt.
$\overline{\text{NMI}}$	Non-maskable interrupt. Used and driven by the delayed non-maskable interrupt circuitry on the microtan 65. If the user wishes to use non-maskable interrupts in specialist applications, then the link LKNMI on the microtan 65 should be broken and this line driven by a peripheral device with an open collector.
$\overline{\text{FB}}$	Field blanking of the television display. Driven from microtan 65. For future use.
$\overline{\text{DMAPOT}}$	Direct memory access priority output. A peripheral board drives this signal, which is read by a board lower in the chain, to indicate that it is producing a DMAREQ and that peripherals lower in priority must wait if they require to perform a DMA. Not a bussed signal.
$\overline{\text{DMAPIN}}$	Direct memory access priority input. Driven from a peripheral of higher priority and inhibits lower priority DMA's. Not a bussed signal. Note that DMAPOT and DMAPIN forms a daisy chain. Boards not using these signals should connect them together. The position nearest the microtan 65 has highest priority.
$\overline{\text{IOE}}$	TANEX output. Indicates that the address bus is addressing I/O on the microtan 65, locations BFFØ-BFFF.
$\overline{\text{RAME}}$	TANEX output. Indicates that the address bus is addressing RAM on the microtan 65, i.e. locations Ø-3FF.
$\overline{\text{ROME}}$	TANEX output. Indicates that the address bus is addressing ROM on the microtan 65, i.e. locations FØØØ-FFFF, or if the memory link on TANEX has been cut, locations F8ØØ-FFFF. ROM only exits on the microtan 65 from locations FCØØ-FFFF, i.e. TANBUG.

Pin mnemonic	Description
R/$\overline{W}$	Read not write. Driven by the microtan 65 to indicate whether the 65Ø2 is reading or writing to the data bus. Handed over to a peripheral device when DMA's are performed.
SYNC	The 65Ø2 sync signal, which indicates an instruction fetch cycle.
$\overline{HB}$	Horizontal blanking of the television display. Driven from microtan 65. For future use.
DØ-D7	Microtan 65 data bus which connects only to TANEX. Not a bussed signal.
DBØ-DB7	System data bus buffered from the microtan 65 on TANEX.
DP	Disc Present. This is a system control line that indicates that the disc controller card is present in the system.
65/$\overline{8Ø}$	TANDOS incorporates a Z80 microprocessor. This line defines which processor is active.
$\overline{NMI8Ø}$	This is a Z8Ø control signal for non-maskable interrupt and is independant from the 6502 non-maskable interrupt.
M1	This is a Z8Ø control signal that indicates the start of a new machine cycle and is equivalent to the 6502 sync signal.
$\overline{BE}$	Block Enable. This is a memory block control line to allow multiple banks of memory on the system motherboard. On the EXP slot this signal is active high. Slots Ø to 7 it is active low.
$\overline{INHRAM}$	Inhibit RAM. This signal allows, with the relevant decoding, any section of section of RAM between 2ØØØ and BBFF to be inhibited.
$\overline{INHROM}$	Similar to inhibit RAM but disables sections of ROM on the 32K ROMBOARD.

Notes for custom designers

If you are contemplating designing and building your own peripherals to connect to TANBUS then you should have a good knowledge of electronics and microprocessors. This being the case, the signals on TANBUS are all perfectly obvious as to their function. There are a few points that should be borne in mind however.

1) Buffer all signals before they are used by on-board circuitry especially Ø1, Ø2 and SYNC as these are direct from the 65Ø2.
2) I/O has been provided so that it is only necessary to decode I/O addresses in the 1K I/O space rather than the full 65K address space.
3) On boards that are connected into the DMA priority chain there must be a small amount of logic to implement the daisy chain. If a particular board is not requesting a DMA it must pass higher priority signals through the chain unmodified; it must also not affect lower priority requests if there are none at a higher priority. If the board does require a DMA it must inhibit lower priority requests and wait until all higher priority requests have ceased.

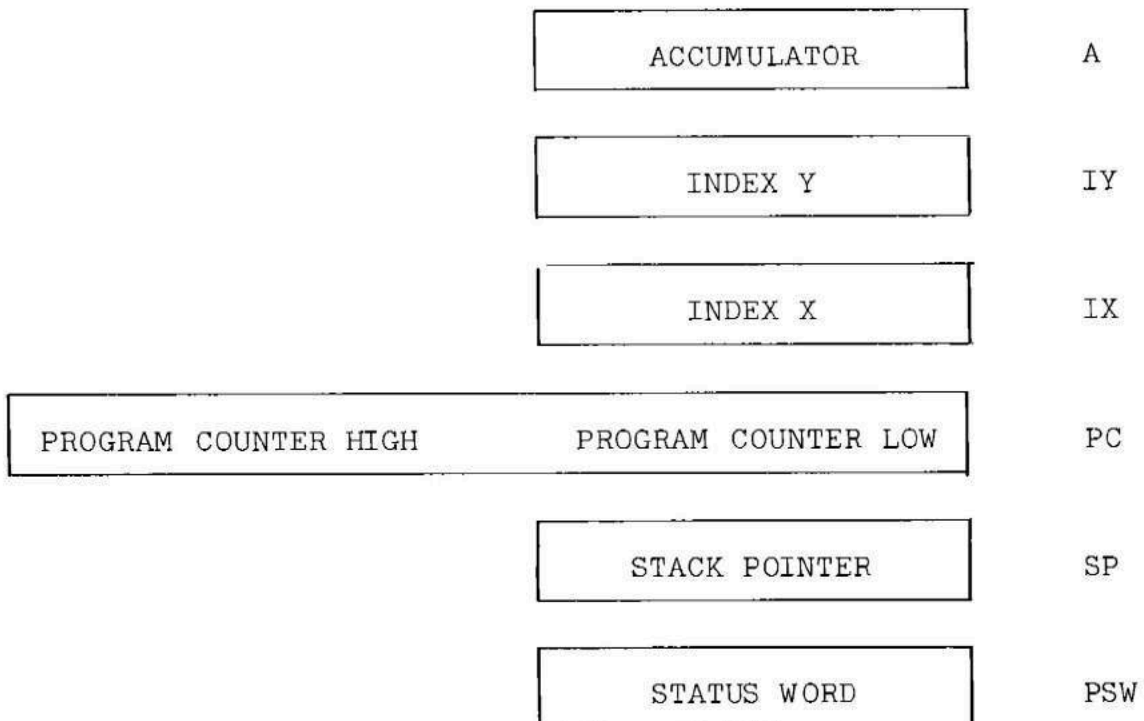
TANBUS CONNECTIONS

ADDITIONAL b	ADDITIONAL a		TANEX b	TANEX a		microtan 65 b	microtan 65 a
+5	+5	1	+5	+5	1	+5	+5
CLK	$\overline{\text{DMAREQ}}$	2			2	CLK	$\overline{\text{DMAREQ}}$
Ø1	Ø2	3			3	Ø1	Ø2
$\overline{\text{RST}}$	I/O	4		$\overline{\text{DMAREQ}}$	4	$\overline{\text{RST}}$	DP
A1	AØ	5	Ø1	Ø2	5	A1	AØ
A3	A2	6	$\overline{\text{RST}}$	I/O	6	A3	A2
A5	A4	7	A1	AØ	7	A5	A4
A7	A6	8	A3	A2	8	A7	A6
A9	A8	9	A5	A4	9	A9	A8
A11	A1Ø	1Ø	A7	A6	1Ø	A11	A1Ø
A13	A12	11	A9	A8	11	A13	A12
A15	A14	12	A11	A1Ø	12	A15	A14
$\overline{\text{DMAGNT}}$	$\overline{\text{IRQ}}$	13	A13	A12	13	SO	$\overline{\text{ABE}}$
65/$\overline{\text{80}}$	$\overline{\text{NMI}}$	14	A15	A14	14	$\overline{\text{FB}}$	$\overline{\text{IRQ}}$
$\overline{\text{DMAPOT}}$	$\overline{\text{DMAPIN}}$	15	$\overline{\text{DMAGNT}}$	$\overline{\text{ABE}}$	15	$\overline{\text{NMI8Ø}}$	$\overline{\text{NMI}}$
SO		16	SO	$\overline{\text{IRQ}}$	16	$\overline{\text{IOE}}$	$\overline{\text{RAME}}$
$\overline{\text{FB}}$	R/$\overline{\text{W}}$	17	$\overline{\text{FB}}$		17	ROME	R/$\overline{\text{W}}$
SYNC	$\overline{\text{HB}}$	18	$\overline{\text{IOE}}$	$\overline{\text{RAME}}$	18	SYNC	$\overline{\text{HB}}$
	DBØ	19	ROME	R/$\overline{\text{W}}$	19	DØ	
	DB1	2Ø			2Ø	D1	
	DB2	21	DBØ	DØ	21	D2	
	DB3	22	DB1	D1	22	D3	
	DB4	23	DB2	D2	23	D4	
	DB5	24	DB3	D3	24	D5	
	DB6	25	DB4	D4	25	D6	
$\overline{\text{INHROM}}$	DB7	26	DB5	D5	26	D7	
M1	$\overline{\text{NMI8Ø}}$	27	DB6	D6	27		
$\overline{\text{BE}}$	$\overline{\text{INHRAM}}$	28	DB7	D7	28	M1	65/$\overline{\text{8Ø}}$
-5	-5	29	$\overline{\text{INHRAM}}$		29		
+12	+12	3Ø	+12	+12	3Ø	+12	+12
-12	-12	31	-12	-12	31	-12	-12
GND	GND	32	GND	GND	32	GND	GND

The 6502 Microprocessor

The 65Ø2 is an 8 bit microprocessor, which means that the data operated upon in each instruction is 8 bits wide and the data path between cpu, memory and peripheral is also 8 bits wide. It has a reportoire of 56 basic instructions, can perform binary and BCD arithmetic and has thirteen addressing modes. Maskable and non-maskable interrupts are supported. The microprocessor is also, what is termed, stack orientated.

The first essential piece of information required is the programmers model. This indicates what register are to be found inside the cpu and their function.



Accumulator

The accumulator is the main working register of the cpu. All arithmetic and logical operations are performed between the accumulator and memory. Arithmetic operations may be binary or binary coded decimal. The mode used is controlled by the decimal flag in the processor status word (PSW).

Index Y

This is a special purpose register that is used in indexed addressing. It may also be used as a special register in a users program.

Index X

As index Y register.

Program Counter High - Program Counter Low

These two registers form a 16 bit program counter which enables the cpu to have an addressing range of 65K bytes. The high byte of the address indicates which page of memory is being accessed and the low byte indicates which location in that page. Therefore the memory space is divided into 256 pages each of 256 locations.

Stack Pointer

The stack pointer contains the address of the location in page 1 of the top of the stack. Data is pushed onto the stack by executing an appropriate instruction; the cpu automatically decrementing the stack pointer (the stack is of the push down variety). When data is required from the stack an appropriate instruction is executed which pulls the data from the stack; the cpu automatically incrementing the stack pointer. The stack is also used, automatically, when subroutines are called or interrupts serviced.

Processor Status Word

The processor status word provides an indication of the result of executing an instruction. Each bit of the status word is used for a particular function.

Bit Ø - Carry (C). This is effectively a ninth bit to the accumulator and is set or reset depending on the result of an arithmetic operation. For instance, if the addition of two binary numbers resulted in a number greater than 255, the carry bit would be set to a logic one. The carry bit can also be set and reset by the programmer.

Bit 1 - Zero Flag (Z). This flag indicates whether any data movement or calculation result involves the data being equal to zero. For example, if two equal numbers were subtracted from each other, or a zero was shifted into IX say, the zero flag would be set to a logic one, otherwise it would be set to a logic zero.

Bit 2 - Interrupt Disable (I). The interrupt disable flag is the output of a flip-flop, which is manipulated by both the programmer and the cpu. When set to a logic one, maskable interrupts are disabled. Non-maskable interrupts are unaffected.

Bit 3 - Decimal Mode (D). The state of this flag determines whether the cpu performs binary (D = Ø) or binary coded decimal (D = 1) arithmetic operations. Is manipulated by the programmer.

Bit 4 - Break Command (B). This flag is set only by the cpu and indicates the execution of a BRK instruction, which causes an interrupt to occur.

Bit 5 - Unused.

Bit 6 - Overflow (V). This flag is similar to the carry flag C. It operates in parallel with the carry flag but indicates the results of calculation if the numbers are considered as signed binary numbers. For example, if the result of adding two signed numbers results in a carry into the sign bit, this flag warns the programmer that sign correction must be carried out. Set to a logic one if a carry occurs, zero otherwise.

Bit 7 - Negative Flag (N). The N flag is set equal to the value of D7 in all data movement and calculation. Therefore, when using signed arithmetic, it is very simple to detect whether the data concerned is positive or negative.

Use of the Processor Status Word

The flags in the processor status word are used to indicate the status of the cpu after each instruction. These flags are only of any use if the programmer can test which state they are in. This is possible, and particularly powerful, by the use of the branch instructions. There are eight branch instructions each testing a particular flag state. Execution of a branch instruction auto-

matically tests the appropriate flag, there is no need for the programmer to write some code to do this himself/herself. Branch instructions are used to test processor status as program flow will need to jump form one segment to another depending on the results of an operation. As an example, the programmer may be in an iterative routine, the number of iterations being counted by an index register (IX say). By decrementing the register after each iteration the register will eventually be zero, this could indicate the end of the routine. By executing the branch on zero (BEQ) after the instruction to decrement index X (DEX) the program will branch when IX equals zero.

Addressing Modes

A powerful processor requires more than just a powerful instruction set, it requires powerful addressing modes as well. Addressing modes are the different number of ways that the cpu can access the data it requires, or work out where to jump to in branching. The 65Ø2 has thirteen addressing modes each one finding the effective address (the actual address required) in a different manner and having its own particular use.

1) Accumulator Addressing: This form of addressing is represented with a one byte instruction which implicitly indicates an operation on the accumulator.
2) Immediate Addressing: In immediate addressing the data is contained in the second byte of a two byte instruction. No further addressing is required.
3) Absolute Addressing: With absolute addressing the instructions are three bytes long. The effective address is formed by using the second byte as the low order byte of the address and the third byte as the high order byte. Thus a 16 bit address is formed enabling access to the full 65K addressing range.
4) Zero Page Addressing: This is exactly the same as absolute addressing except that the high order byte of the address is zero and therefore there is no need to have a third byte to the instruction. The effective address in page zero is obtained

from the second byte of the instruction.

5 & 6) Indexed Zero Page Addressing: This form of addressing is used in conjunction with the index registers IX and IY and is referred to as "Zero Page, X" or "Zero Page, Y" depending on which index register is used. The effective address is obtained by adding the second byte of the two byte instruction to the contents of the appropriate index register. This forms the low order byte of the address, the high order byte being held at zero. Note that no carry from the addition of these two numbers is transferred into the high order byte of the address, therefore no page boundary crossing can occur.

7 & 8) Indexed Absolute Addressing: Used in conjunction with the IX and IY index registers and referred to as "Absolute, X" and "Absolute, Y". The effective address is formed by adding the contents of the appropriate index register to the address (as in absolute addressing) contained in the second and third bytes of the three byte instructions. This type of addressing is very useful in scanning tables, that may reside anywhere in memory. The index register is used as a count value to indicate the position in the table or list being accessed, and the absolute address contained in the second and third bytes of the instruction used as a base address to indicate the start of the table or list.

9) Implied Addressing: In implied addressing the address to be accessed is implicitly stated in the instruction, such as "decrement IX" implies an operation on the IX register.

1Ø) Relative Addressing: This addressing mode can only be used with branch instructions, indeed it is the only addressing mode that can be used with branch instructions. The second byte of the two byte instruction is used as a signed binary displacement which is added to the program counter when it is pointing to the next instruction. Therefore the program can be made to jump forwards or backwards from its present position by

-128 to +127 memory locations (not instructions) TANBUG's offset command O calculates the offset to be added for you.

11) Indexed Indirect Addressing: This addressing mode makes use of the index X register and is referred to as "(Indirect, X)". The second byte of the two byte instruction is added to the contents of the IX register, a carry, if generated, being discarded. The result of this addition points to a location in page zero whose contents forms the low order byte of the effective address, the next memory location in page zero containing the high byte of the effective address.

12) Indirect Indexed Addressing: This addressing mode makes use of the IY register and is referred to as "(Indirect), Y". The second byte of the two byte instruction points to a location in page zero, the contents of which are added to the contents of the IY register, the result being the low order byte of the effective address. The carry generated by this addition is added to the contents of the next memory location in page zero, forming the high order byte of the effective address.

13) Absolute Indirect: The second byte of the three byte instruction forms a low order byte of a pointer address, the third byte containing the high order byte of the pointer address. The contents of the fully specified pointer address contains the low order byte of the effective address and the next memory location the high order byte.

Note the importance of the zero page in the memory map. This page should be used for addressing modes and commonly used constants in a program and not cluttered up with data or a program that can reside almost anywhere.

Subroutines

In programs there is often a program routine which needs to be performed quite often and in various places in the program. Instead of writing the program routine several times in different places, it can be written once as a subroutine, thus saving

valuable memory space. The subroutine code can be located anywhere in memory. When it is required to execute the subroutine in a program the programmer uses the "jump to subroutine" instruction, JSR. This is a three byte instruction using absolute addresing. In order to know where in the program it must return to, the cpu puts the program counter value of the next instruction after the subroutine call onto the stack. The cpu then jumps to the subroutine code and executes it. At the end of the subroutine there must be the "return from subroutine" instruction RTS. The program counter value of the next instruction after the subroutine call being pulled off of the stack. Subroutines may call other subroutines, and subroutines may also call themselves i.e. be re-entrant. The return addresses are all stored on the stack in the correct sequence.

Interrupts

Interrupts are a means by which a peripheral device may request the cpu to execute a program, not unlike a subroutine, that will service the peripheral in some way. The program code is generally referred to as an interrupt service routine. There are two types of interrupt available on the 65Ø2 microprocessor - maskable and non-maskable. Maskable interrupts can be disabled, that is the cpu will not recognise them, by setting the interrupt disable bit I of the processor status word. Non-maskable interrupts cannot be disabled. When an external device generates an interrupt the cpu completes the instruction it is currently executing, then places the program counter value of the next instruction and the processor status word onto the stack. The program counter is then loaded with the appropriate interrupt vector. For the maskable interrupt the vector is located in location FFFE (low order address byte) and FFFF (high order address byte) and for the non-maskable interrupt in locations FFFA and FFFB. This interrupt vector is the starting location of the interrupt service routine. At the end of the routine the user must execute the "return from interrupt" instruction RTI. The cpu then returns to the point where it was interrupted by pulling the old program counter and processor status word off the stack. When an interrupt occurs the interrupt disable flag I is set.

If more than one interrupting device is allowed on one or both of the interrupt types, then the appropriate interrupt vector must

point to a routine which tests each device, in order of priority, to see which caused the interrupt. This is done by reading the peripheral port which contains the individual interrupt flag of each device. The routine then directs the cpu to the appropriate interrupt service routine. Because interrupts use the stack to store the cpu state when an interrupt occurs, interrupts may be serviced during the servicing of a current interrupt, i.e. they may be nested.

Instruction Set

The 65Ø2 has 56 instructions. Each instruction may be 1, 2 or 3 bytes long. There now follows the tables of every instruction which fully explain its operation and available addressing modes. The tables use the following notation.

Symbol	Meaning
A	Accumulator
X, Y	Index Registers
M	Memory
P	Processor Status Register
S	Stack Pointer
/	Change
–	No Change
+	Add
∧	Logical AND
-	Subtract
⊻	Logical Exclusive Or
↑	Transfer from Stack
↓	Transfer to Stack
→	Transfer to
←	Transfer to
V	Logical OR
PC	Program Counter
PCH	Program Counter High
PCL	Program Counter Low
OPER	Operand
#	Immediate Addressing Mode

ADC

Add memory to accumulator with carry

Operation: A + M + C → A, C

N Z C I D V
/ / / _ _ /

Addressing Mode	Assembly Language Form		OP CODE	No. Bytes	No. Cycles
Immediate	ADC	#Oper	69	2	2
Zero Page	ADC	Oper	65	2	3
Zero Page, X	ADC	Oper, X	75	2	4
Absolute	ADC	Oper	6D	3	4
Absolute, X	ADC	Oper, X	7D	3	4*
Absolute, Y	ADC	Oper, Y	79	3	4*
(Indirect, X)	ADC	(Oper, X)	61	2	6
(Indirect), Y	ADC	(Oper), Y	71	2	5*

* Add 1 if page boundary is crossed.

AND

"AND" memory with accumulator

Operation: A ∧ M → A

N Z C I D V
/ / _ _ _ _

Addressing Mode	Assembly Language Form		OP CODE	No. Bytes	No. Cycles
Immediate	AND	#Oper	29	2	2
Zero Page	AND	Oper	25	2	3
Zero Page, X	AND	Oper, X	35	2	4
Absolute	AND	Oper	2D	3	4
Absolute, X	AND	Oper, X	3D	3	4*
Absolute, Y	AND	Oper, Y	39	3	4*
(Indirect, X)	AND	(Oper, X)	21	2	6
(Indirect), Y	AND	(Oper), Y	31	2	5*

* Add 1 if page boundary is crossed.

ASL

Shift Left One Bit (Memory or Accumulator)

Operation: C ← 7 6 5 4 3 2 1 Ø ← Ø

N Z C I D V
/ / / _ _ _

Addressing Mode	Assembly Language Form		OP CODE	No. Bytes	No. Cycles
Accumulator	ASL	A	ØA	1	2
Zero Page	ASL	Oper	Ø6	2	5
Zero Page, X	ASL	Oper, X	16	2	6
Absolute	ASL	Oper	ØE	3	6
Absolute, X	ASL	Oper, X	1E	3	7

BCC

Branch on Carry Clear

Operation: Branch on C = Ø

N Z C I D V
_ _ _ _ _ _

Addressing Mode	Assembly Language Form		OP CODE	No. Bytes	No. Cycles
Relative	BCC	Oper	9Ø	2	2*

* Add 1 if branch occurs to same page.

* Add 2 if branch occurs to different page.

BCS

Branch on carry set

Operation: Branch on C = 1

N Z C I D V

\- - - - - -

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Relative	BCS Oper	BØ	2	2*

* Add 1 if branch occurs to same page.

* Add 2 if branch occurs to next page.

BEQ

Branch on result zero

Operation: Branch on Z = 1

N Z C I D V

\- - - - - -

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Relative	BEQ Oper	FØ	2	2*

* Add 1 if branch occurs to same page.

* Add 2 if branch occurs to next page.

BIT

Test bits in memory with accumulator

Operation: $A \wedge M$, $M_7 \rightarrow N$, $M_6 \rightarrow V$

N Z C I D V

M_7/_ _ _ M_6

Addressing Mode	Assembly Language Form		OP CODE	No. Bytes	No. Cycles
Zero Page	BIT	Oper	24	2	3
Absolute	BIT	Oper	2C	3	4

BMI

Branch on result minus

Operation: Branch on N = 1

N Z C I D V

\- - - - - -

Addressing Mode	Assembly Language Form		OP CODE	No. Bytes	No. Cycles
Relative	BMI	Oper	3Ø	2	2*

* Add 1 if branch occurs to same page.

* Add 2 if branch occurs to different page.

BNE

Branch on result not zero

Operation: Branch on Z = 0

N Z C I D V

_ _ _ _ _ _

Addressing Mode	Assembly Language Form		OP CODE	No. Bytes	No. Cycles
Relative	BNE	Oper	DØ	2	2*

* Add 1 if branch occurs to same page.

* Add 2 if branch occurs to different page.

BPL

Branch on result plus

Operation: Branch on N = Ø

N Z C I D V

_ _ _ _ _ _

Addressing Mode	Assembly Language Form		OP CODE	No. Bytes	No. Cycles
Relative	BPL	Oper	1Ø	2	2*

* Add 1 if branch occurs to same page.

* Add 2 if branch occurs to different page.

BRK

Force Break

Operation: Forced Interrupt PC + 2 ↓ P ↓

N	Z	C	I	D	V
_	_	_	1	_	_

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	BRK	ØØ	1	7

1. A BRK command cannot be masked by setting I.

BVC

Branch on overflow clear

Operation: Branch on V = 0

N	Z	C	I	D	V
_	_	_	_	_	_

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Relative	BVC Oper	5Ø	2	2*

* Add 1 if branch occurs to same page.
* Add 2 if branch occurs to different page.

BVS

Branch on overflow set

Operation: Branch on V = 1

N Z C I D V

_ _ _ _ _ _

Addressing Mode	Assembly Language Form		OP CODE	No. Bytes	No. Cycles
Relative	BVS	Oper	7Ø	2	2*

* Add 1 if branch occurs to same page.
* Add 2 if branch occurs to different page.

CLC

Clear carry flag

Operation: Ø — C

N Z C I D V

_ _ Ø _ _ _

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	CLC	18	1	2

CLD

Clear decimal mode

Operation: Ø → D

N Z C I D V
_ _ _ _ Ø _

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	CLD	D8	1	2

CLI

Clear interrupt disable bit

Operation: Ø → I

N Z C I D V
_ _ _ Ø _ _

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	CLI	58	1	2

CLV

Clear overflow flag

Operation: Ø → V

N Z C I D V
_ _ _ _ _ Ø

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	CLV	B8	1	2

CMP

Compare memory and accumulator

Operation: A → M

N Z C I D V
/ / / _ _ _

Addressing Mode	Assembly Language Form		OP CODE	No. Bytes	No. Cycles
Immediate	CMP	#Oper	C9	2	2
Zero Page	CMP	Oper	C5	2	3
Zero Page, X	CMP	Oper, X	D5	2	4
Absolute	CMP	Oper	CD	3	4
Absolute, X	CMP	Oper, X	DD	3	4*
Absolute, Y	CMP	Oper, Y	D9	3	4*
(Indirect, X)	CMP	(Oper, X)	C1	2	6
(Indirect), Y	CMP	(Oper), Y	D1	2	5*

* Add 1 if page boundary is crossed.

CPX

Compare memory and index X

Operation: X — M

N Z C I D V
/ / / _ _ _

Addressing Mode	Assembly Language Form		OP CODE	No. Bytes	No. Cycles
Immediate	CPX	#Oper	EØ	2	2
Zero Page	CPX	Oper	E4	2	3
Absolute	CPX	Oper	EC	3	4

CPY

Compare memory and index Y

Operation: Y — M

N Z C I D V
/ / / _ _ _

Addressing Mode	Assembly Language Form		OP CODE	No. Bytes	No. Cycles
Immediate	CPY	#Oper	CØ	2	2
Zero Page	CPY	Oper	C4	2	3
Absolute	CPY	Oper	CC	3	4

DEC

Decrement memory by one

Operation: M - 1 ⟶ M

N Z C I D V
/ / _ _ _ _

Addressing Mode	Assembly Language Form		OP CODE	No. Bytes	No. Cycles
Zero Page	DEC	Oper	C6	2	5
Zero Page, X	DEC	Oper, X	D6	2	6
Absolute	DEC	Oper	CE	3	6
Absolute, X	DEC	Oper, X	DE	3	7

DEX

Decrement index X by one

Operation: X - 1 ⟶ X

N Z C I D V
/ / _ _ _ _

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	DEX	CA	1	2

DEY

Decrement index Y by one

Operation: Y - 1 → Y

N Z C I D V
/ / _ _ _ _

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	DEY	88	1	2

EOR

"Exclusive - Or" memory with accumulator

Operation: A ⊻ M → A

N Z C I D V
/ / _ _ _ _

Addressing Mode	Assembly Language Form		OP CODE	No. Bytes	No. Cycles
Immediate	EOR	#Oper	49	2	2
Zero Page	EOR	Oper	45	2	3
Zero Page, X	EOR	Oper, X	55	2	4
Absolute	EOR	Oper	4D	3	4
Absolute, X	EOR	Oper, X	5D	3	4*
Absolute, Y	EOR	Oper, Y	59	3	4*
(Indirect, X)	EOR	(Oper, X)	41	2	6
(Indirect), Y	EOR	(Oper), Y	51	2	5*

* Add 1 if page boundary is crossed.

INC

Increment memory by one

Operation: M + 1 → M

N Z C I D V
/ / _ _ _ _

Addressing Mode	Assembly Language Form		OP CODE	No. Bytes	No. Cycles
Zero Page	INC	Oper	E6	2	5
Zero Page, X	INC	Oper, X	F6	2	6
Absolute	INC	Oper	EE	3	6
Absolute, X	INC	Oper, X	FE	3	7

INX

Increment index X by one

Operation: X + 1 → X

N Z C I D V
/ / _ _ _ _

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	INX	E8	1	2

INY

Increment index Y by one

Operation: Y + 1 → Y

N Z C I D V
/ / _ _ _ _

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	INY	C8	1	2

JMP

Jump to new location

Operation: (PC + 1) → PCL
(PC + 2) → PCH

N Z C I D V
_ _ _ _ _ _

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Absolute	JMP Oper	4C	3	3
Indirect	JMP (Oper)	6C	3	5

JSR

Jump to new location saving return address

Operation: PC + 2 ↓, (PC + 1) → PCL
(PC + 2) → PCH

N Z C I D V
_ _ _ _ _ _

Addressing Mode	Assembly Language Form		OP CODE	No. Bytes	No. Cycles
Absolute	JSR	Oper	2Ø	3	6

LDA

Load accumulator with memory

Operation: M → A

N Z C I D V
/ / _ _ _ _

Addressing Mode	Assembly Language Form		OP CODE	No. Bytes	No. Cycles
Immediate	LDA	#Oper	A9	2	2
Zero Page	LDA	Oper	A5	2	3
Zero Page, X	LDA	Oper, X	B5	2	4
Absolute	LDA	Oper	AD	3	4
Absolute, X	LDA	Oper, X	BD	3	4*
Absolute, Y	LDA	Oper, Y	B9	3	4*
(Indirect, X)	LDA	(Oper, X)	A1	2	6
(Indirect), Y	LDA	(Oper), Y	B1	2	5*

* Add 1 if page boundary is crossed.

LDX

Load index X with memory

Operation: M → X

N Z C I D V
/ / _ _ _ _

Addressing Mode	Assembly Language Form		OP CODE	No. Bytes	No. Cycles
Immediate	LDX	#Oper	A2	2	2
Zero Page	LDX	Oper	A6	2	3
Zero Page, Y	LDX	Oper, Y	B6	2	4
Absolute	LDX	Oper	AE	3	4
Absolute, Y	LDX	Oper, Y	BE	3	4*

* Add 1 when page boundary is crossed.

LDY

Load index Y with memory

Operation: M → Y

N Z C I D V
/ / _ _ _ _

Addressing Mode	Assembly Language Form		OP CODE	No. Bytes	No. Cycles
Immediate	LDY	#Oper	AØ	2	2
Zero Page	LDY	Oper	A4	2	3
Zero Page, X	LDY	Oper, X	B4	2	4
Absolute	LDY	Oper	AC	3	4
Absolute, X	LDY	Oper, X	BC	3	4*

* Add 1 when page boundary is crossed.

LSR

Shift right one bit (memory or accumulator)

Operation: Ø → 7 6 5 4 3 2 1 Ø → C

N Z C I D V
Ø / / _ _ _

Addressing Mode	Assembly Language Form		OP CODE	No. Bytes	No. Cycles
Accumulator	LSR	A	4A	1	2
Zero Page	LSR	Oper	46	2	5
Zero Page, X	LSR	Oper, X	56	2	6
Absolute	LSR	Oper	4E	3	6
Absolute, X	LSR	Oper, X	5E	3	7

NOP

No operation

Operation: No operation (2 cycles)

N Z C I D V
_ _ _ _ _ _

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	NOP	EA	1	2

ORA

"OR" memory with accumulator

Operation: A V M → A

N Z C I D V
/ / _ _ _ _

Addressing Mode	Assembly Language Form		OP CODE	No. Bytes	No. Cycles
Immediate	ORA	#Oper	Ø9	2	2
Zero Page	ORA	Oper	Ø5	2	3
Zero Page, X	ORA	Oper, X	15	2	4
Absolute	ORA	Oper	ØD	3	4
Absolute, X	ORA	Oper, X	1D	3	4*
Absolute, Y	ORA	Oper, Y	19	3	4*
(Indirect, X)	ORA	(Oper, X)	Ø1	2	6
(Indirect), Y	ORA	(Oper), Y	11	2	5*

* Add 1 on page crossing.

PHA

Push accumulator on stack

Operation: A ↓

N Z C I D V
_ _ _ _ _ _

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	PHA	48	1	3

PHP

Push processor status on stack

Operation: P ↓

N Z C I D V

\- - - - - -

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	PHP	Ø8	1	3

PLA

Pull accumulator from stack

Operation: A ↑

N Z C I D V

/ / - - - -

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	PLA	68	1	4

PLP

Pull processor status from stack

Operation: P ↑

N Z C I D V
From Stack

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	PLP	28	1	4

ROL

Rotate one bit left (memory or accumulator)

Operation: 7 6 5 4 3 2 1 Ø — C (M or A; C rotates back into bit 0)

N Z C I D V
/ / / _ _ _

Addressing Mode	Assembly Language Form		OP CODE	No. Bytes	No. Cycles
Accumulator	ROL	A	2A	1	2
Zero Page	ROL	Oper	26	2	5
Zero Page, X	ROL	Oper, X	36	2	6
Absolute	ROL	Oper	2E	3	6
Absolute, X	ROL	Oper, X	3E	3	7

ROR

Rotate one bit right (memory or accumulator)

Operation: (M or A) ⌐C → 7 6 5 4 3 2 1 Ø

N Z C I D V
/ / / _ _ _

Addressing Mode	Assembly Language Form		OP CODE	No. Bytes	No. Cycles
Accumulator	ROR	A	6A	1	2
Zero Page	ROR	Oper	66	2	5
Zero Page, X	ROR	Oper, X	76	2	6
Absolute	ROR	Oper	6E	3	6
Absolute, X	ROR	Oper, X	7E	3	7

RTI

Return from interrupt

Operation: P ↑ PC ↑

N Z C I D V
From Stack

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	RTI	4Ø	1	6

RTS

Return from subroutine

Operation: PC↑, PC + 1 → PC

N Z C I D V

_ _ _ _ _ _

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	RTS	6Ø	1	6

SBC

Subtract memory from accumulator with borrow

Operation: A - M - C → A

Note: C = Borrow

N Z C I D V

/ / / _ _ /

Addressing Mode	Assembly Language Form		OP CODE	No. Bytes	No. Cycles
Immediate	SBC	#Oper	E9	2	2
Zero Page	SBC	Oper	E5	2	3
Zero Page, X	SBC	Oper, X	F5	2	4
Absolute	SBC	Oper	ED	3	4
Absolute, X	SBC	Oper, X	FD	3	4*
Absolute, Y	SBC	Oper, Y	F9	3	4*
(Indirect, X)	SBC	(Oper, X)	E1	2	6
(Indirect), Y	SBC	(Oper), Y	F1	2	5*

* Add 1 when page boundary is crossed.

SEC

Set carry flag

Operation: 1 → C

N Z C I D V
_ _ 1 _ _ _

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	SEC	38	1	2

SED

Set decimal mode

Operation: 1 → D

N Z C I D V
_ _ _ _ 1 _

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	SED	F8	1	2

SEI

Set interrupt disable status

Operation: 1 → I

N Z C I D V
_ _ _ 1 _ _

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	SEI	78	1	2

STA

Store accumulator in memory

Operation: A → M

N Z C I D V

– – – – – –

Addressing Mode	Assembly Language Form		OP CODE	No. Bytes	No. Cycles
Zero Page	STA	Oper	85	2	3
Zero Page, X	STA	Oper, X	95	2	4
Absolute	STA	Oper	8D	3	4
Absolute, X	STA	Oper, X	9D	3	5
Absolute, Y	STA	Oper, Y	99	3	5
(Indirect, X)	STA	(Oper, X)	81	2	6
(Indirect), Y	STA	(Oper), Y	91	2	6

STX

Store index X in memory

Operation: X → M

N Z C I D V

– – – – – –

Addressing Mode	Assembly Language Form		OP CODE	No. Bytes	No. Cycles
Zero Page	STX	Oper	86	2	3
Zero Page, Y	STX	Oper, Y	96	2	4
Absolute	STX	Oper	8E	3	4

STY

Store index Y in memory

Operation: Y → M

N Z C I D V

_ _ _ _ _ _

Addressing Mode	Assembly Language Form		OP CODE	No. Bytes	No. Cycles
Zero Page	STY	Oper	84	2	3
Zero Page, X	STY	Oper, X	94	2	4
Absolute	STY	Oper	8C	3	4

TAX

Transfer accumulator to index X

Operation: A → X

N Z C I D V

/ / _ _ _ _

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	TAX	AA	1	2

TAY

Transfer accumulator to index Y

Operation: A → Y

N Z C I D V
/ / _ _ _ _

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	TAY	A8	1	2

TYA

Transfer index Y to accumulator

Operation: Y → A

N Z C I D V
/ / _ _ _ _

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	TYA	98	1	2

TSX

Transfer stack pointer to index X

Operation: S → X

N Z C I D V
/ / _ _ _ _

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	TSX	BA	1	2

TXA

Transfer index X to accumulator

Operation: X → A

N Z C I D V
/ / _ _ _ _

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	TXA	8A	1	2

TXS

Transfer index X to stack pointer

Operation: X → S

N Z C I D V
_ _ _ _ _ _

Addressing Mode	Assembly Language Form	OP CODE	No. Bytes	No. Cycles
Implied	TXS	9A	1	2

ØØ - BRK
Ø1 - ORA - (Indirect,X)
Ø2 - Future Expansion
Ø3 - Future Expansion
Ø4 - Future Expansion
Ø5 - ORA - Zero Page
Ø6 - ASL - Zero Page
Ø7 - Future Expansion
Ø8 - PHP
Ø9 - ORA - Immediate
ØA - ASL - Accumulator
ØB - Future Expansion
ØC - Future Expansion
ØD - ORA - Absolute
ØE - ASL - Absolute
ØF - Future Expansion
1Ø - BPL
11 - ORA - (Indirect),Y
12 - Future Expansion
13 - Future Expansion
14 - Future Expansion
15 - ORA - Zero Page,X
16 - ASL - Zero Page,X
17 - Future Expansion
18 - CLC
19 - ORA - Absolute,Y
1A - Future Expansion
1B - Future Expansion
1C - Future Expansion
1D - ORA - Absolute,X
1E - ASL - Absolute,X
1F - Future Expansion
2Ø - JSR
21 - AND - (Indirect,X)
22 - Future Expansion
23 - Future Expansion
24 - BIT - Zero Page
25 - AND - Zero Page
26 - ROL - Zero Page
27 - Future Expansion
28 - PLP
29 - AND - Immediate
2A - ROL - Accumulator
2B - Future Expansion
2C - BIT - Absolute
2D - AND - Absolute
2E - ROL - Absolute
2F - Future Expansion
3Ø - BMI
31 - AND - (Indirect),Y
32 - Future Expansion
33 - Future Expansion
34 - Future Expansion
35 - AND - Zero Page,X
36 - ROL - Zero Page,X
37 - Future Expansion
38 - SEC
39 - AND - Absolute,Y
3A - Future Expansion
3B - Future Expansion
3C - Future Expansion
3D - AND - Absolute,X
3E - ROL - Absolute,X
3F - Future Expansion

40 - RTI
41 - EOR - (Indirect,X)
42 - Future Expansion
43 - Future Expansion
44 - Future Expansion
45 - EOR - Zero Page
46 - LSR - Zero Page
47 - Future Expansion
48 - PHA
49 - EOR - Immediate
4A - LSR - Accumulator
4B - Future Expansion
4C - JMP - Absolute
4D - EOR - Absolute
4E - LSR - Absolute
4F - Future Expansion
50 - BVC
51 - EOR - (Indirect),Y
52 - Future Expansion
53 - Future Expansion
54 - Future Expansion
55 - EOR - Zero Page,X
56 - LSR - Zero Page,X
57 - Future Expansion
58 - CLI
59 - EOR - Absolute,Y
5A - Future Expansion
5B - Future Expansion
5C - Future Expansion
5D - EOR - Absolute,X
5E - LSR - Absolute,X
5F - Future Expansion

60 - RTS
61 - ADC - (Indirect,X)
62 - Future Expansion
63 - Future Expansion
64 - Future Expansion
65 - ADC - Zero Page
66 - ROR - Zero Page
67 - Future Expansion
68 - PLA
69 - ADC - Immediate
6A - ROR - Accumulator
6B - Future Expansion
6C - JMP - Indirect
6D - ADC - Absolute
6E - ROR - Absolute
6F - Future Expansion
70 - BVS
71 - ADC - (Indirect),Y
72 - Future Expansion
73 - Future Expansion
74 - Future Expansion
75 - ADC - Zero Page,X
76 - ROR - Zero Page,X
77 - Future Expansion
78 - SEI
79 - ADC - Absolute,Y
7A - Future Expansion
7B - Future Expansion
7C - Future Expansion
7D - ADC - Absolute,X
7E - ROR - Absolute,X
7F - Future Expansion

8Ø - Future Expansion
81 - STA - (Indirect,X)
82 - Future Expansion
83 - Future Expansion
84 - STY - Zero Page
85 - STA - Zero Page
86 - STX - Zero Page
87 - Future Expansion
88 - DEY
89 - Future Expansion
8A - TXA
8B - Future Expansion
8C - STY - Absolute
8D - STA - Absolute
8E - STX - Absolute
8F - Future Expansion
9Ø - BCC
91 - STA - (Indirect),Y
92 - Future Expansion
93 - Future Expansion
94 - STY - Zero Page,X
95 - STA - Zero Page,X
96 - STX - Zero Page,Y
97 - Future Expansion
98 - TYA
99 - STA - Absolute,Y
9A - TXS
9B - Future Expansion
9C - Future Expansion
9D - STA - Absolute,X
9E - Future Expansion
9F - Future Expansion

AØ - LDY - Immediate
A1 - LDA - (Indirect,X)
A2 - LDX - Immediate
A3 - Future Expansion
A4 - LDY - Zero Page
A5 - LDA - Zero Page
A6 - LDX - Zero Page
A7 - Future Expansion
A8 - TAY
A9 - LDA - Immediate
AA - TAX
AB - Future Expansion
AC - LDY - Absolute
AD - LDA - Absolute
AE - LDX - Absolute
AF - Future Expansion
BØ - BCS
B1 - LDA - (Indirect),Y
B2 - Future Expansion
B3 - Future Expansion
B4 - LDY - Zero Page,X
B5 - LDA - Zero Page,X
B6 - LDX - Zero Page,Y
B7 - Future Expansion
B8 - CLV
B9 - LDA - Absolute,Y
BA - TSX
BB - Future Expansion
BC - LDY - Absolute,X
BD - LDA - Absolute,X
BE - LDX - Absolute,Y
BF - Future Expansion

C∅ - CPY - Immediate
C1 - CMP - (Indirect,X)
C2 - Future Expansion
C3 - Future Expansion
C4 - CPY - Zero Page
C5 - CMP - Zero Page
C6 - DEC - Zero Page
C7 - Future Expansion
C8 - INY
C9 - CMP - Immediate
CA - DEX
CB - Future Expansion
CC - CPY - Absolute
CD - CMP - Absolute
CE - DEC - Absolute
CF - Future Expansion
D∅ - BNE
D1 - CMP - (Indirect),Y
D2 - Future Expansion
D3 - Future Expansion
D4 - Future Expansion
D5 - CMP - Zero Page,X
D6 - DEC - Zero Page,X
D7 - Future Expansion
D8 - CLD
D9 - CMP - Absolute,Y
DA - Future Expansion
DB - Future Expansion
DC - Future Expansion
DD - CMP - Absolute,X
DE - DEC - Absolute,X
DF - Future Expansion

E∅ - CPX - Immediate
E1 - SBC - (Indirect,X)
E2 - Future Expansion
E3 - Future Expansion
E4 - CPX - Zero Page
E5 - SBC - Zero Page
E6 - INC - Zero Page
E7 - Future Expansion
E8 - INX
E9 - SBC - Immediate
EA - NOP
EB - Future Expansion
EC - CPX - Absolute
ED - SBC - Absolute
EE - INC - Absolute
EF - Future Expansion
F∅ - BEQ
F1 - SBC - (Indirect),Y
F2 - Future Expansion
F3 - Future Expansion
F4 - Future Expansion
F5 - SBC - Zero Page,X
F6 - INC - Zero Page,X
F7 - Future Expansion
F8 - SED
F9 - SBC - Absolute,Y
FA - Future Expansion
FB - Future Expansion
FC - Future Expansion
FD - SBC - Absolute,X
FE - INC - Absolute,X
FF - Future Expansion

TANBUG
V2

Notes for Users Familiar with TANBUG V1

TANBUG version 2 has been designed such that all your programs written under TANBUG V1 will run identically under TANBUG V2. TANBUG V2, which occupies 2K instead of TANBUG V1s 1K, contains extra features as follows:

- Basic Warm Start
- Parallel Printer Driver
- Serial Printer Driver
- Link to External (user) Software Driver
- RS232 Input to the Monitor
- Additional Subroutines, Including Memory Management
- Basic Clear Screen

TANBUG V2 is compatible with Microsoft Basic V1 and XBUG V5. However, XBUG V5 Translator format was specifically designed for screen output and gives an overprinted format. The best method of obtaining a listing is to use the Instruction dissassembler to list code in memory. Later versions of XBUG will have this problem rectified.

The TANBUG monitor program is located in 2K bytes of read only memory (ROM) at the top of the address space i.e. pages 248 - 255. It contains facilities to enter, modify, run and debug programs. This chapter of the manual gives full details of the command facilities and subroutines available to the user.

TANBUG will only operate in the memory map of the Microtan system, it is not a general purpose 65Ø2 software package and has been specifically written for Microtan. Locations F7F7, F7F8 and F7F9 are reserved for a jump to an expansion monitor ROM which is positioned on the expansion board, more about this later.

Locations 2ØØ-3FF i.e. pages 2 and 3 are the visual display memory - TANBUG writes to these locations whenever a command is typed to the monitor. Locations BFFØ-BFF3 are the addresses of the peripheral attachments, e.g. keyboard, graphics function flip-flop etc. Locations 1ØØ-1FF i.e. page 1, are used as the stack by the microprocessor. Since the stack is of the push down variety it follows that the whole of the area will not be used as stack storage in the majority of programs. TANBUG requires to use locations 1FØ-1FF as stack storage (only 16 locations). The rest of this area is free for user programs. Locations 4Ø-FF are also available as user RAM, the preceeding locations Ø-3F being reserved for use by TANBUG. User programs which do not use the stack may therefore be loaded anywhere i.e. the area 4Ø-1EF. For user programs which do use the stack, the user must calculate how many stack locations are required and reduce the upper limit accordingly.

TANBUG contains coding to automatically identify whether the keypad or full ASCII keyboard is connected to the keyboard socket. This coding is executed every time a reset is issued, and thereafter a sequence of code, particular to the keyboard type in use, is executed. Reset must therefore always be issued after changing the keyboard type.

When using an ASCII encoded alphanumeric keyboard, monitor commands are typed in as shown in this chapter. There is however no reset key on an ASCII keyboard, one must be fitted as shown in the chapter describing assembly of the Microtan kit. TANBUG drives this type of keyboard in the interrupt mode.

The keypad is used somewhat differently, its layout being shown below.

<table>
<tr><td>SHIFT</td><td>DEL
LF</td><td>SP
CR</td><td>RST</td></tr>
<tr><td>M
C</td><td>G
D</td><td>S
E</td><td>N
F</td></tr>
<tr><td>P
8</td><td>ESC
9</td><td>B
A</td><td>L
B</td></tr>
<tr><td>4</td><td>0
5</td><td>C
6</td><td>R
7</td></tr>
<tr><td>0</td><td>1</td><td>2</td><td>,
3</td></tr>
</table>

TANBUG interrogates the keypad for a depressed key, then translates the matrix encoded signal into an ASCII character which it puts up on the visual display just as if the equivalent key were depressed on an ASCII encoded keyboard. Because of the limited number of keys it has been necessary to incorporate a shift function on the keypad. So to obtain the character P for example, the user presses and releases SHIFT, then depresses and releases P.

The SHIFT key contains a self cancelling facility - if the user presses SHIFT twice in succession the pending shift operation is cancelled. So as an example, using the two keys SHIFT and 8, the operation SHIFT P yields P on the display. SHIFT SHIFT P yields 8 on the display. Other special purpose keys on the keypad are RST, which issues a reset to the Microtan, and DEL which delete the last character typed. Repeated deletes erase characters back to the beginning of the line.

Keyboard Protocol

If you have a serial KB enabled, the system disables the keypad at startup, i.e. you cannot use it (the keypad) at all.

If a serial KB is disabled, then the system looks for the two types of keyboard as before. (ASCII keyboard and keypad). If you are using the serial KB and have an ASCII KB plugged in, the serial KB is disabled as soon as you hit a key on the ASCII KB.[1]

From now on in this chapter, the Microtan will be treated as having one type of keyboard only, since all functions required can be derived by depressing the appropriate key or keys on whichever is used - keyboard or keypad.

Having described some of the background to TANBUG, it is now possible to describe the commands and syntax of TANBUG i.e. how to use it. An example is shown later on. All numerical values of address, data and monitor command arguments are in hexadecimal. The symbol < CR > means on depression of the carriage return key, < SP > the space key or bar, < ESC > the escape key (ALT on some keyboards) and < LF > line feed. In all examples, text to be typed by the user will be underlined, while TANBUG responses will not. █ indicates the cursor. <ADDR> means a hexadecimal address, ARG means hexadecimal data and <TERM> means one of the terminators <CR> , <SP> , <ESC> , or <LF> .

1 - See Appendix.

All commands are of the form

<COMMAND> <TERM>
or <COMMAND> <ARG> <TERM>
or <COMMAND> <ARG>, <ARG> <TERM>
or <COMMAND> <ARG>, <ARG>, <ARG> <TERM>

where <COMMAND> is one of the mnemonic commands and <ARG> is a hexadecimal arugment application to the command being used. The requirement argument is defined for each command. It should be noted at an early stage that the longest argument will contain 4 hexadecimal characters. If more are typed all but the last 4 are ignored. As an example consider the memory modify command M12340078 <CR>. In this case location 0078 will be modified or examined as all but the last 4 characters are ignored.

<TERM> is one of the terminating characters <CR>, <SP>, <LF> or <ESC>. In fact TANBUG accepts any of the "control" characters (HEX code 0-20) as terminator. TANBUG will reply with a ? if an illegal command is encountered.

Starting the Monitor TANBUG:

Press the RST key on the keypad or the reset key or button connected to the Microtan. TANBUG will scroll the display and respond with

TANBUG
█

On a system rack Micron, a reset is automatically executed on power-up. Note: that on initial power up the top part of the display will be filled with spurious characters. These will disappear as new commands are entered and the display scrolls up. On subsequent resets the previous operations remain displayed to facilitate debugging. Note: that if your Micron is not fitted with the lower case option, then your prompt will be a ? and not the block █.

Memory Modify/Examine Command M:

The M command allows the user to enter and modify programs by changing the RAM locations to the desired values. The command also allows the user to inspect ROM locations, modify registers etc. To open a location, type the following

M <ADDR> <TERM>

TANBUG then replies with the current contents of that location. For example to examine the contents of RAM location 1ØØ type M1ØØ<CR> TANBUG then responds on the display with

M1ØØ,ØE,█

assuming the current contents of the location were ØE.

There are now several options open to the user. If any terminator is typed the location is closed and not altered and the cursor moves to the next line scrolling up the display by one row. If, however, a value is typed followed by one of the terminators <CR>, <LF> or <ESC> the location is modified and then closed. For example using <CR>

M1ØØ,ØE,FF
█

location 1ØØ will now contain FF. If however <SP> is typed, the location is re-opened and unmodified.

M1ØØ,ØE,FF
MØ1ØØ,ØE,█

This facility is useful if an erroneous value has been typed. The terminators <LF> and <ESC> modify the current location being examined, then opens the next and previous locations respectively i.e. using <LF>

M1ØØ,ØE,FF
MØ1Ø1,AB,█

and using <ESC>

M1ØØ,ØE,
MØØFF,56,█

Using <LF> makes for very easy program entry, it only being necessary to type the initial address of the program followed by its data and <LF>, then responding to the cursor prompt for subsequent data words.

Note: that locations 1FE and 1FF should not be modified. These are the stack loctions which contain the monitor return addresses. If they are corrupted TANBUG will almost certainly "crash" and it will be necessary to issue a reset in order to recover.

The Modify memory command only accepts one byte of information at a time, while programming convention dictates that all bytes of an instruction are written on one line. For example, a program may be printed as

```
Ø1ØØ A9ØØ LDA#Ø
Ø1Ø2 854Ø STA 4Ø
```

This would be entered via TANBUG as

```
M1ØØ,ØE,A9<LF> (First byte of first instruction)
M1Ø1,FF,ØØ<LF> (Second byte of first instruction)
M1Ø2,AB,85<LF> (First byte of next instruction)
M1Ø3,ØØ,4Ø<LF> (etc.)
```

List Command L:

The list command allows the user to list out sections of memory onto the display. It is possible to display the contents of a maximum of one hundred and twenty consecutive memory locations simultaneously. To list a series of locations type

L <ADDR>, <NUMBER> <TERM>

where ADDR is the address of the first location to be printed and NUMBER is the number of lines of eight consecutive locations to be printed. TANBUG pauses briefly between each line to allow the user to scan them. For example, to list the first 16 locations of TANBUG (which resides at F8ØØ-FFFF) type LF8ØØ,2<CR>. The display will then be

```
LF8ØØ,2
F8ØØ  4C  51  F9  4C  B2  F9  4C  9B
F8Ø8  F9  4C  79  FE  A9  ØD  4C  75
█
```

If zero lines are requested (i.e. <NUMBER> = Ø) then 256 lines will be given.

Go Command G:

Having entered a program using the M command and verified it using the L command, the user can use the G command to start running his own program. The command is of the format G <ADDR> <TERM>. For example, to start a program whose first instruction is at location 1ØØ type G1ØØ <CR>. When the user program is started the cursor disappears. On a return to the monitor it re-appears.

The G command automatically sets up two of the microprocessors internal registers

a) The program counter (PC) is set to the start address given in the G command.

b) The stack pointer (SP) is set to location 1FF.

The contents of the other four internal registers, namely the status word (PSW), index X (IX), index Y (IY) and accumulator (A), are taken from the monitor pseudo registers (described next). Thus the user can either set up the pseudo registers before typing the G command, or use instructions within his/her program to manipulate them directly.

Register Modify/examine Command R:

Locations 15 to 1B within the RAM reserved for TANBUG are the user pseudo registers. The user can set these locations prior to issuing a G command. The values are then transferred to the microprocessors internal registers immediately before the user program is started. The pseudo register locations are also used by the monitor to save the user internal register values when a breakpoint is encountered. These values are then transferred back into the microprocessor when a P command is issued, so that to all intents and purposes the user program appears to be uninterrupted.

The R command allows the user to modify these registers in conjunction with the M command. To modify/examine registers type R <CR> and the following display will appear (location 15 containing ØØ say).

R
MØØ15,ØØ,█

Now proceed as for the M command.

Naturally the M command could be used to modify/examine location 15 without using the R command - the R command merely saving the user the need to remember and type in the start location of the pseudo registers. Pseudo register locations are as follows.

Location	Function
15	Low order byte of program counter (PCL)
16	High order byte of program count (PCH)
17	Processor status word (PSW)
18	Stack pointer (SP)
19	Index X (IX)
1A	Index Y (IY)
1B	Accumulator (A)

Two typical instances of the use of the R command are:-

a) Setting up PSW, IX, IY and A before starting a user program.

b) Modifying registers after a breakpoint but before proceeding with program execution (using the P command) for debugging purposes.

Note that when modifying registers in case (b) care must be taken if PCL, PCH or SP are modified, since the proceed command P uses these to determine the address of the next instructions to be executed (PCL, PCH) and the user stack pointer (SP).

Single Instruction Mode S:

Single instruction mode is a very powerful debugging aid. When set TANBUG executes the user program one instruction at a time, re-entering the monitor between each instruction and printing out the status of all of the microprocessor's internal registers as they were after the last instruction executed in the user program. The S command is used in conjunction with the proceed command P and the normal mode command N. Examples are given in the description of the P command.

Normal Mode Command N:

The N command is the complement of the S command and is used to cancel the S command so that the microprocessor executes the user program in the normal manner without returning to the monitor between each instruction. Reset automatically sets the normal mode of operation.

Proceed Command P:

The P command is used to instruct TANBUG to execute the next instruction in the user program when in single instruction mode. Pseudo register contents are transferred into the microprocessor's internal registers and the next instruction in the user's program is executed. The monitor is then re-entered. P may also be used with an argument thus P <NUMBER> <CR> where NUMBER is less than or equal to FF. In this case the program executes the specified number of instructions +1 before returning to the monitor.

Each time the monitor is re-entered after execution of an instruction or instructions, the status of the microprocessor internal registers, as they were in the user program, are printed across the screen in the following order:

Address of next instruction to be executed.
Processor status word.
Stack pointer.
Index register X.
Index register Y.
Accumulator.

Note that these are in the same order as the pseudo registers, described earlier.

Whenever the user program is entered, the cursor is removed from the display. Whenever the monitor is entered, the cursor returns to the display as a user prompt. While in the monitor between user instructions, any monitor command can be typed. A program must always be started by the G command, then P used if in single instruction mode. A P command used before a G command is issued, is likely to cause a program "crash" and should not be attempted.

As an example, consider the simple program which repeatedly adds 1 to the accumulator.

Address	Data	Mnemonic	Comment
1ØØ	69	ADC 1	: add 1 to acc.
1Ø1	Ø1		
1Ø2	4C	JMP 1ØØ	
1Ø3	ØØ		
1Ø4	Ø1		

Set the single instruction mode and start the program. The user may wish to initially set the accumulator to ØØ by using the M command.

```
S
G1ØØ
Ø1Ø2 2Ø FF ØØ Ø1
```

TANBUG then responds with the characters shown above.

Ø1Ø2 is the address of the next instruction to be executed.

2Ø is the processor status word value.

FF is the low byte value of the stack pointer. The high byte is always set to 1, the stack is therefore pointing at location 1FF.

ØØ is the value of the index X register.

ØØ is the value of the index Y register.

Ø1 is the value of the accumulator. It is a 1 as 1 has been added to the accumulator and it is assumed that the user cleared the accumulator before starting the program.

Since the cursor has re-appeared, TANBUG is ready for any monitor command. For example, registers or memory locations can be modified, or the program may be re-started from scratch by typing G1ØØ <CR> again. If the user wishes to continue then type P <CR>. The resulting display is

```
S
G1ØØ
Ø1Ø2  2Ø  FF  ØØ  ØØ  Ø1
P
Ø1ØØ  2Ø  FF  ØØ  ØØ  Ø1
█
```

Since the instruction at location 1Ø2 was "Jump to 1ØØ", the status print out shows that this has indeed occurred. Registers, since they were not modified by any program instruction, remain unchanged. To proceed further type P <CR> again.

```
S
G1ØØ
Ø1Ø2  2Ø  FF  ØØ  ØØ  Ø1
P
Ø1ØØ  2Ø  FF  ØØ  ØØ  Ø1
P
Ø1Ø2  2Ø  FF  ØØ  ØØ  Ø2
█
```

The add instruction has been executed again, so the accumulator has incremented by 1 to become 2. Now typing P4 <CR> gives a display.

```
S
G1ØØ
Ø1Ø2  2Ø  FF  ØØ  ØØ  Ø1
P
Ø1ØØ  2Ø  FF  ØØ  ØØ  Ø1
P
Ø1Ø2  2Ø  FF  ØØ  ØØ  Ø2
P4
Ø1Ø2  2Ø  FF  ØØ  ØØ  Ø4
█
```

TANBUG allowed execution of 4 instructions before again returning to the monitor. The 4 instructions were 2 add instructions and 2 jump instructions thus giving the accumulator the value 4.

By typing N <CR> then P <CR> removes the single instruction mode and causes the program to proceed. It now does not return to the monitor but continues to race around this small program loop continually adding and jumping back. There is no way to exit from this trivial program except by a microprocessor reset or, if using an alphanumeric keyboard, by typing ESC.

It can be seen that the S and P commands are particularly useful when tracing a program which contains instructions that transfer program control e.g. jumps, branches and sub-routines, since these commands allow the user to interrogate the order of execution of his/her program.

Breakpoint Command B;

A breakpoint is a complementary debugging aid to single instruction mode. Instead of stepping singly through all instructions in a program, the breakpoint facility allows the user to specify the address at which he requires the monitor to be re-entered from his/her program. As an example, consider a long program in which a fault is suspected to exist near the end. It would be very tedious and time consuming to single step through the program to the problem area. A breakpoint can be set just previous to where the fault is suspected to exist and the program started with the G command. Normal execution occurs until the breakpoint is reached, then the monitor is re-entered with the same status print-out as for single instruction mode. Any monitor commands can then be used and the program continued.

The format of the breakpoint command is

B <ADDR>, <NUMBER> <CR>

where <ADDR> is the address of any instruction OPCODE (but not argument), <NUMBER> is any number from Ø - 7 defining one of 8 breakpoints. B <CR> removes all breakpoints. As an example consider the following program

```
100  E8           LOOP: INX
101  C8                 INY
102  69  01             ADC#1
104  4C  00  01         JMP  LOOP
```

Firstly set index X, index Y and the accumulator to ØØ using the R command. To set breakpoint Ø at the jump instruction and start the program type B1Ø4,Ø <CR> . The display will then be

```
B1Ø4,Ø
G1ØØ
Ø1Ø4  2Ø  FF  Ø1  Ø1  Ø1
```

The jump instruction was reached and the breakpoint redirected control back to TANBUG. If it were required, single instruction mode could be set for further debugging. However, assume that we wish to execute the loop again by typing P<CR>.

```
B1Ø4,Ø
G1ØØ
Ø1Ø2  2Ø  FF  Ø1  Ø1  Ø1
P
Ø1Ø4  2Ø  FF  Ø2  Ø2  Ø2
```

The proceed command P has gone once through the breakpoint and then re-entered the monitor. If P <NUMBER > <CR> was typed it would have proceeded through the breakpoint <NUMBER > times.

Up to 8 breakpoints can be set at 8 different locations. The B <CR> command removes all breakpoints. A single breakpoint can be removed by setting its address to Ø. For example, imagine a breakpoint is set as follows: B1Ø2,2, and it is subsequently wished to remove it but leave any others unaltered; type BØ,2<CR> to remove it.

<u>Caution</u>. The breakpoint system works by replacing the user's instruction with a special instruction (BRK) whose opcode is ØØ. Replacement is carried out when G or P is typed.

On return to the monitor the original opcode is replaced. It is therefore possible to corrupt the user program under some circumstances. The following points should therefore be observed:

a) Breakpoints must only be set at the opcode part of a user instruction and nowhere else.

b) If the user program utilises the BRK instruction as part of the user code, then the user must have his own special interrupt routine and cannot use breakpoints.

c) If breakpoints are set in the user program and a reset is issued while the microprocessor is executing the user program rather than the monitor, the breakpoints are lost and those locations at which breakpoints were set in the user program will be corrupted. These locations must be re-entered using the M command before restarting the user program.

d) Setting more than one breakpoint at the same address causes the user program to be corrupted.

e) To use breakpoints, the user must not have modified the interrupt link, i.e. the interrupt code within TANBUG must be executed.

The status of breakpoints may be inspected by using the M command to examine the breakpoint status table. This is located in RAM locations 2Ø-2F and are as follows:

Address	Contents
2Ø	PCL BØ
21	PCH BØ
22	PCL B1
23	PCH B1

Address	Contents
24	PCL B2
25	PCH B2
26	PCL B3
27	PCH B3
28	PCL B4
29	PCH B4
2A	PCL B5
2B	PCH B5
2C	PCL B6
2D	PCH B6
2E	PCL B7
2F	PCH B7

For example, typing M2Ø<CR> followed by <LF> gives

```
M2Ø,ØØ
MØØ21,Ø1,█
```

This indicates that breakpoint Ø is set to location 1ØØ by taking the contents of location 2Ø as PCL and of location 21 as PCH. If the breakpoint is set at location Ø then this particular breakpoint is disabled.

Offset Command O:

The offset command O is a program writing aid. It calculates branch offsets for the user for incorporation as arguments in branch instructions. Consider the example:

```
1ØØ  E8    LOOP:  INX
1Ø1  C8           INY
1Ø2  69           ADC#1
1Ø3  Ø1             .
                    .
                    .
12Ø  DØ           BNE LOOP
121               (branch argument)
```

To calculate the number to enter into location 121 is quite tedious without a facility such as the O command. It is used with the following format.

O<ADDR. OF BRANCH OPCODE><ADDR. OF DEST.>< CR >

and in this case it would be necessary to type Ø12Ø,1ØØ <CR>. The display would be

Ø12Ø,1ØØ = DE

FØ is the number that should be entered into location 121 such that if the BNE instruction is true the program counter will jump to the label LOOP.

Note that the maximum branch range is 7F forwards and backwards. If the range is exceeded a ? is displayed.

Copy Command C:

The copy command allows copying of the contents of one block of memory to another. Its format is

C<START ADDR. SOURCE><END ADDR. SOURCE><START ADDR. DEST.>

Suppose it is required to copy the block of data in locations FCØØ-FDØØ into a block starting at location 2ØØ. This may be achieved by typing CFCØØ,FDØØ,2ØØ <CR> . The display will be

CFCØØ,FDØØ,2ØØ

As 2ØØ is the starting address of the display memory, the user will notice that the top half of the screen has been over written with all sorts of weird and wonderful characters. What this example has done is to take the first 256 bytes of TANBUG and copy them into the top half of the display.

The display then scrolled having the top 7 rows filled with these characters.[2]

Breakpoints and the ESC Key

If an alphanumeric keyboard is being used, depression of the ESC key (ALT on some keyboards) will cause a re-entry into the monitor from the user program. This is possible because the alphanumeric keyboard is interrupt driven. For example, if the trivial program

```
100  69      LOOP:  ADC#1
101  01
102  4C             JMP LOOP
103  00
104  01
```

has been started by typing the G command, the program continues to loop around continuously with no exit path to the monitor, except by issuing a reset. Instead of a reset the user can press the ESC key, TANBUG responding thus

```
0100  20  FF  01  01  01
█
```

Using the ESC key has caused a breakpoint to be executed and the monitor invoked. The register print-out above is only typical, the value of each being that when the ESC was depressed. Any monitor command may now be typed, for example P causes the user program to proceed once again.

The ESC facility is most useful in debugging where the user program gets into an unforseen loop where breakpoints have not been set. It enables the user to rejoin the monitor without using reset and losing the breakpoints that have been set.

2 = See Appendix

Notes:

a) The ESC facility is only implemented on interrupt driven keyboards, i.e. alphanumeric ASCII keyboards. and is not implemented on the keypad.

b) Interrupt must be enabled for the ESC facility to operate. TANBUG enables interrupts when entering a user program, therefore do not disable interrupts if the ESC facility is required.

c) The user must not have modified the interrupt jump link. TANBUG's interrupt code must be executed.

Input/Output Control

TANBUG V2 contains subroutines, accessible from machine code or user subroutines, as well as directly via the keyboard, to allow input/output to user peripherals.

Information about which devices are in use is stored in the printer status word, which is at location Ø in RAM. The word is made up as follows:

bit 7 ... bit Ø

BAS WARM	SCN DIS	SER O/P ON	PAR O/P ON	EXT O/P ON	DC FLAG	SPECIAL PRINT MODE	SER I/P ON

In more detail:

BIT Ø SER I/P ON - set by the monitor on initialisation if a serial keyboard is connected to Tanex. Cleared if not, or if a keypad is connected to the Microtan, or if an ASCII keyboard interrupts.

BIT 1 SPECIAL PRINT - set to Ø by monitor on initialisation. When Ø, a line of output to a parallel device is terminated by LF only, while to a serial device CR LF is output. This may be set to 1 by the user so that, if his printers require them, the serial interface terminates with LF only while the parallel interface gives CRLF.

BIT 2 DC FLAG - used by the monitor to denote output control codes for printers on/off.

BIT 3 EXT OUTPUT ON - zeroed by the monitor on initialisation. If set to 1 by the user, can be linked to a user output driver subroutine.

BIT 4 PAR O/P ON - zeroed by the monitor on initialisation. If set to 1 by any of the various control commands, initialises the printer and directs output to it.

BIT 5 SER O/P ON - as Bit 4, but for serial printer interface on Tanex.

BIT 6 SCN DIS - set to Ø by initialisation. If set to a 1 by control commands, inhibits output to the screen.

BIT 7 BAS WARM - Used by the monitor for BASIC warm start.

Important Notes:

a) If you wish to change the value of any of the bits in the printer status word, you should leave the BAS WARM and DC FLAG bits set to their current values.

b) The SCN DIS facility is available for your use. However, the monitor subroutines require the screen to be enabled for command storage. Therefore, if you are using a teletype for input, you should leave the screen area enabled, even though you may not have a TV display connected to the Microtan.

The bits in the printer status word, as well as being user programmable, are changed by certain monitor commands, and also by control codes output via TANBUG. These are described individually for each printer.

Every time that the subroutine OUTALL or OUTRET (or OUTPCR, OPCHR) is called, either from the monitor, BASIC, or a user program, the printer status word is examined and output is routed to all those devices which are enabled. Thus you can use your printers with all existing software merely by controlling the output bits as described below.

Parallel Printer

TANBUG V2 contains software to drive the optional 6522 on Tanex in a Centronics-type parallel output mode.

Table 1 shows the pin connections for the interface cable.

Fig. 1

Function	Printer Connector Pin	Tanex Skt No & Pin	Function
Data 1	2	C1 - 2	PAØ
Data 2	3	C1 - 3	PA1
Data 3	4	C1 - 4	PA2
Data 4	5	C1 - 5	PA3
Data 5	6	C1 - 6	PA4
Data 6	7	C1 - 9	PA5
Data 7	8	C1 - 10	PA6
Data 8	9	C1 - 11	PA7
$\overline{\text{Strobe}}$	1	C1 - 12	CA2
$\overline{\text{Ack}}$	10	C1 - 13	CA1
$\overline{\text{INIT}}$	31	D1 - 2	PBØ
BUSY	11	D1 - 3	PB1
$\overline{\text{ERROR}}$	32	D1 - 4	PB2
GND	19	C1 - 7	OV
GND	21	C1 - 8	OV
GND	23	D1 - 7	OV
GND	25	D1 - 8	OV

There are two subroutines for the parallel printer - one to initialise it and one to output data. Timing diagrams for a typical printer are shown in Figs. 2 and 3.

Fig. 2 - Initialisation

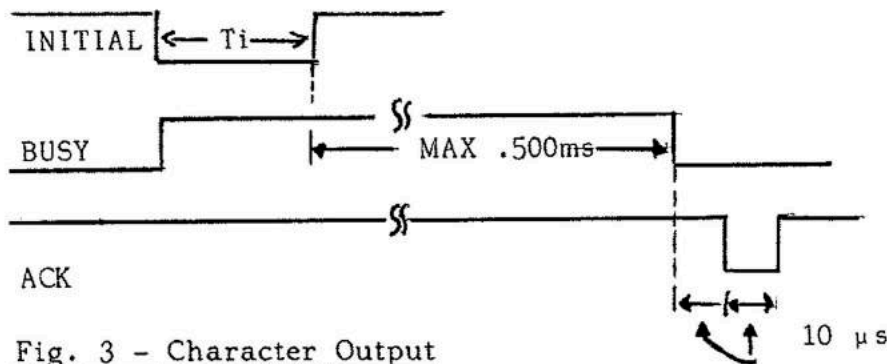
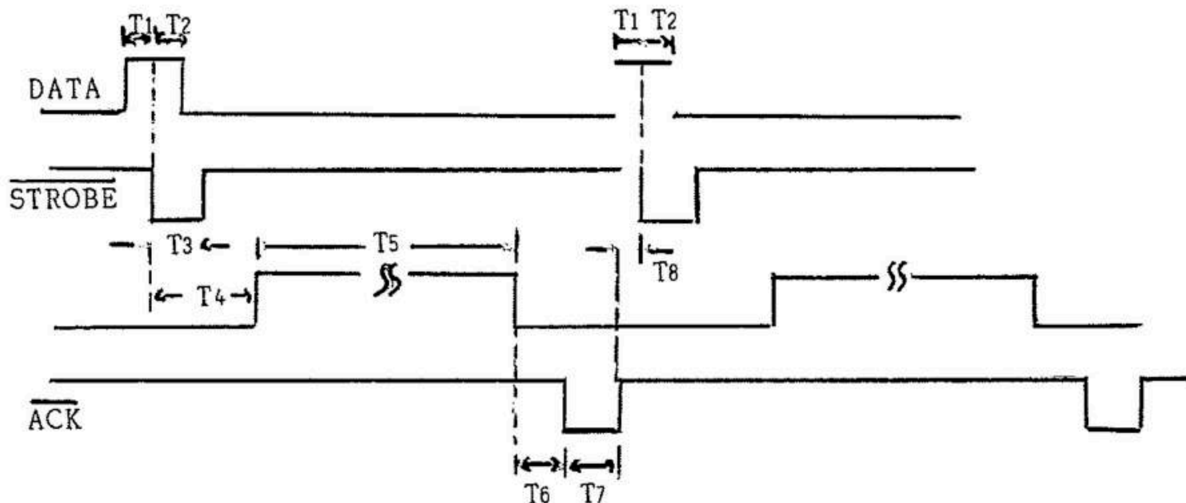


Fig. 3 - Character Output



A) T1, T2 ≥ 0.5 (μs). Data signal must be stable during T1 and T2 centering on falling-edge of $\overline{\text{STROBE}}$.

B) 1 (μs) $\leq$ T3 $\leq$ 90 (μs) .... Pulse width of $\overline{\text{STROBE}}$.

C) 30 (μs) $\leq$ T4. Delay time between $\overline{\text{STROBE}}$ input and rising-edge of BUSY signal is over 30 μs.

D) 60 (μs) $\leq$ T5. Minimum length of T5 of BUSY signal is 60 μs. T5 varies with every input data. When print command is input, BUSY signal becomes "HIGH" until completion of print-out. (Max. 3s).

E) T6, T7 = 10 (μs). $\overline{\text{ACK}}$ is output by falling-edge of BUSY with the timing of T6 and T7.

F) 0 $\leq$ T8. Right after the rising-edge of $\overline{\text{ACK}}$, $\overline{\text{STROBE}}$ is allowed to be input.

Fig 4 - Signal Descriptions

DATA 1 - 8	8 bit parallel data - logic H = 1.
$\overline{\text{STROBE}}$	A low pulse strobes in data.
$\overline{\text{ACK}}$	A low pulse signifies data received.
$\overline{\text{INIT}}$	A low pulse initialises the printer.
BUSY	A high indicates the printer is busy.
$\overline{\text{ERROR}}$	A Low indicates an error in the printer. (e.g. no power, paper out).

In addition some printers have an output select pin (output from Tanex) and paper out pin (input to Tanex). These are not implemented in Tanbug V2, though of course you may connect these via the 6522 and your own software if required.

Controlling the Parallel Printer

The parallel printer can be controlled in several ways:

a) When running the monitor, or the BASIC interpreter, the printer can be turned on or off by typing ↑P (hold down the CTRL key and hit P). Successive operations of this kind alternately turn the printer on and off. As an example, consider listing a BASIC program (the printer is off).

Program entry

10 PRINT "THIS IS AN EXAMPLE"<CR>
LIST ↑P <CR>

The printer is turned on and your program listed.
The ↑P is not printed.

b) From within a user program, you can turn the printer on by using the OUTALL subroutine to output the code (DC1) 2. Thereafter, all output transmitted by the OUTALL subroutine is also transmitted to the printer (see section on subroutines).

Example, to turn the printer on:

```
LDA #$ 11
JSR OUTALL          ;output DC1
LDA #$2
JSR OUTALL          ;output 2
```

The printer can be turned off by outputting (DC1) 3.

c) You can output directly to the parallel printer without going through OUTALL by using the OUTPAR subroutine, but note that the printer must be initialised first. Refer to the detailed description of subroutines.

d) The printer is turned off by a RESET.

Printer Errors

If your printer is in an error condition, the system will fail to respond for 10 seconds while a timeout check takes place. The message "PRINT ERROR" will then be displayed on the TV screen, and the printer will be disabled by TANBUG. Rectify the error and repeat.

Note that the many printers work in line mode, i.e. characters are stored up until a line terminator (LF) occurs, then the whole line is output in one shot.

Using Other Types of Parallel Printers

Other types of parallel printers with a Centronics - compatible interface should operate without modification. Note, however, that some printers use <LF> as a buffer terminator - Tanbug V2 outputs a line of text terminated by <LF> only - no carriage return is output. If your printer requires the sequence <CR LF> then you can set the "special print" bit in the printer status word, which will cause this sequence to be output. Use the following code, which must be executed after every reset:

```
LDA $Ø        ;get status word
ORA #2        ;set print bit
STA $Ø        ;store it
```

If you wish to use a non-Centronics parallel printer, you must add logic to produce the interface signals shown in Figs. 2 and 3.

IMPORTANT NOTE

If the data transfer rate to your printer is slower than your cassette data rate, you must disable your printer before using the XBUG E or F commands, otherwise filename errors will occur.

Serial Printer

Tanbug V2 contains software to drive a serial printer via the UART on Tanex. The interface may either be V24 or 20mA current loop - refer to the Tanex manual for selection.

Fig. 5 shows the printer connections.

Fig. 5 - Serial Printer Connections

Function	Tanex Connection
Printer Ground	E1 - 7
Printer Drive (V24)	E1 - 3
or	
Printer Drive (20mA) +	E1 - 1
Printer Drive (20mA) -	E1 - 2
Printer Enable	E1 - 8 connect to E1 - 7

Note that the modem control pin 8, printer enable, must be grounded to operate - you can ground this at the printer end if required so that an error is given if the printer is not connected.

Operation

Whenever a reset is executed, the serial printer interface is set up to the following specification:

110 baud
Internal clock Rx
8 bits/word
2 stop bits
Parity disabled
Non-echo
Interrupt disabled
$\overline{RTS}$ Low
Enable Rx/Tx

This allows connection of a normal 110 baud teletype printer.[3]

Output can be controlled by the following methods:

a) From any monitor command, or from BASIC, repeatedly typing ↑V (hold down CTRL and hit V) alternately turns the printer on and off. When on, any output from BASIC, from the monitor, or via the OUTALL or

3 = See Appendix

OPCHR subroutines, is directed to the serial printer as well as any other output device which is enabled.

b) From within a user program the printer can be turned on by using the OUTALL subroutine by outputting (DC1) Ø and off by (DC1)1.

See example under parallel printer.

c) You can output directly to the serial printer without affecting other devices by using the OUTSER subroutine, See section on subroutines.

d) The printer is turned off by a reset.

Printer Errors

If the printer is disabled for hardware reasons, the message "PRINT ERROR" is displayed on the screen, and TANBUG V2 turns the printer off.

IMPORTANT NOTE

If you are using a printer at 110 baud, you must turn the printer off before using XBUG V5 "E" and "F" commands, otherwise due to the slow transfer rate filename errors will occur. Cassette handling should be executed via the screen.

Using Other Serial Printers

TANBUG V2 initialises the serial printer as stated above. If you wish to use printers with other specifications (for example, a different baud rate) you can modify the UART status registers BFD2 and BFD3 via the monitor "M" command, referring to the Tanex Manual for the functions of each status bit. Note that you must do this after each reset.

TANBUG V2 outputs CR LF at the end of each line, this sequence being required for most serial devices. By setting the "SPECIAL PRINT" bit in the Printer Status Word (described in more detail in the Parallel Printer Section) you can output LF only as a terminator.

Screen Output Suppression

TANBUG V2 allows you to suppress output to the screen display - this is useful, for example, in situations where you wish to input data via the keyboard and display and output different data on the printer.[4]

Screen output is turned off and on as follows:

a) In the monitor or BASIC, or via the JPLKB or POLLKB subroutines, by typing ↑S (hold down CTRL key and hit S). Successive operations turn the display off and on.

b) From within a user program the screen is turned on by outputting (DC1) 4 via the OUTALL subroutine, and off by (DC1) 5. See example under parallel printer.

c) Output can be made to the screen without affecting other enabled devices by calling the OUTSCR subroutine. See descriptions of subroutines.

d) The screen is turned on by a reset.

IMPORTANT NOTE. The monitor subroutine HEXPCK, and also XBUG use the screen as data storage. Therefore, it is necessary to enable the screen when using the monitor even though you may not have a TV display connected.

4 = See Appendix

External Output Devices

TANBUG V2 allows you to link in your own output device to work with the Monitor and Microsoft BASIC. If bit 3 in the printer status word, EXT O/P ON, is set, then the zero-page locations INTSL2,3 are used as a jump location to link in your own handler subroutine. As an example:

```
        LDA #Ø              ;user program code
        STA INTSL2
        LDA #4Ø
        STA INTSL3          ;user subroutine at 4000
        LDA PSTAT           ;get status word
        ORA #8
        STA PSTAT           ;turn on ext printer
  4ØØØ                      ;user subroutine


        RTS
```

Note that the subroutine must be in memory before the external drive is enabled.

The external printer may be turned off by the code

```
        LDA PSTAT
        AND #F7
        STA PSTAT
```

It is also turned off by a reset.

NOTE If you enable the external printer, you cannot link in extra interrupts using the INTSL1 facility (described in the interrupt section). Use the INTFS facility instead.

```
        INTSL2 = 0011
        INTSL3 = 0012
```

When a (DC1)(Number) code is output, these do not appear at the output device. Should you wish to output a DC1 code directly, then you should write DC1 twice:

```
LDA#11
JSR OUTALL
LDA#11
JSR OUTALL
```

which will cause one DC1 to be printed.

Any illegal codes (DC1) (Illegal code) will cause just the illegal ASCII code to be printed.

Note that the Translator and Instruction Disassembler in XBUG V5 is primarily for use with the TV display, and give an abridged format on each type of printer. Later versions of XBUG will give the correct printing format.

Using an External Keyboard

TANBUG V2 will accept serial input from the UART on TANEX, and feed it to the monitor and BASIC programs. It is accepted via the JPLKB (POLLKB) subroutine.

Fig. 6 shows the serial input connections.

Function	Tanex Connection
Ground	E1 - 7
$\overline{DCD}$	E1 - 10
$\overline{DSR}$	E1 - 11
Serial in V24	E1 - 12
or	
20mA in +	E1 - 13
20mA in -	E1 - 7

To enable the input, $\overline{DCD}$ and $\overline{DSR}$ must be tied to ground. This can be done on the input plug so that serial input is only recog-

nised when a keyboard is plugged in.

Method of Operation

On a reset, TANBUG V2 looks to see if a keyboard is connected by determining whether $\overline{DCD}$ and $\overline{DSR}$ are tied to ground. If they are then the serial interface is set up as described under serial printer but in addition the UART input interrupt is enabled, and the serial printer is turned on. The JPLKB subroutine (used by the Monitor and Microsoft Basic) recognises interrupts from the serial keyboard, and passes them to the monitor.

If, while the serial keyboard is enabled, an interrupt from the Microtan keyboard port occurs, the serial keyboard is disabled and future input must come from the Microtan keyboard until another reset occurs.

You should always press reset after plugging in an external keyboard since this can cause an error interrupt.

If you are using a hex keypad to initialise the Micron, you may leave the pad plugged in. The monitor will accept input only from the keypad until the following sequence of operations is typed:-

Press return key on teletype
Press return key on keypad

Input will now be accepted from the teletype.

Once the serial input is disabled, either by the user or via an interrupt from the Microtan keyboard port, bit Ø of the Printer Status Word (SER I/P ON) is cleared. Under this condition, the monitor interrupt routine will not recognise a UART interrupt. This enables the user to configure the system, via the software interrupt link, to handle UART interrupts for his own purposes.

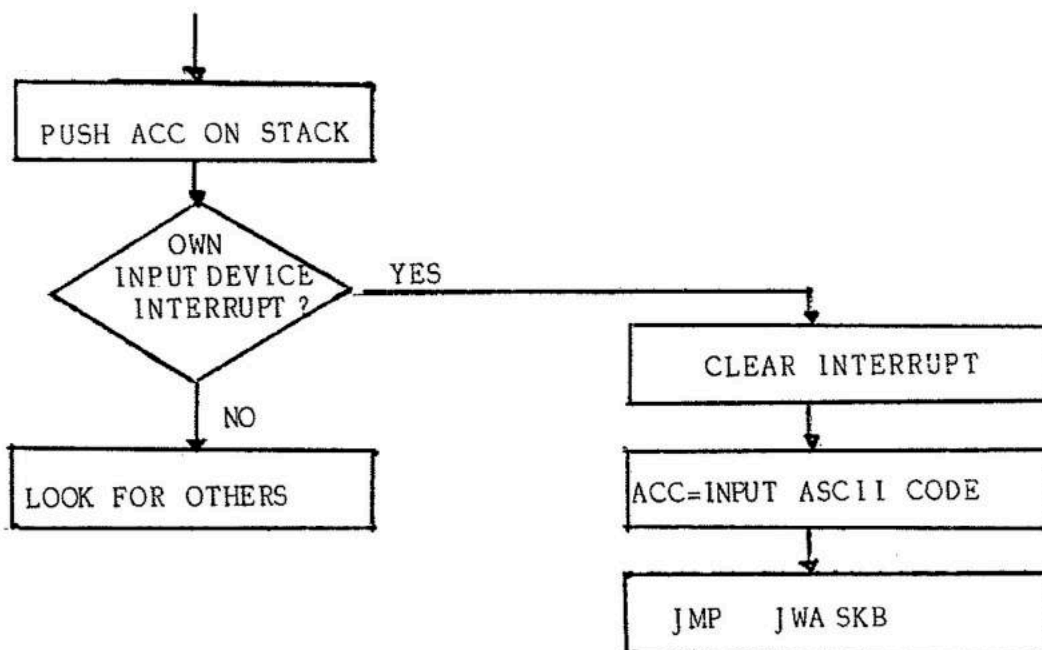
NOTE that if you are using serial input/output with no display, you must use the ↑V command to turn off the printer while reading cassette tapes, otherwise due to the slow printing speed, Filename errors will occur.

As with the serial printer, you can modify the UART status words BFD2 and BFD3 to allow keyboard input at different baud rates, different word lengths etc. Refer to the Tanex manual for status word designations.

The input and output speeds must be the same baud rate - different input and output speeds are not allowed.

External Input to Monitor

External input devices running under interrupt can be linked to the monitor. Interrupt link INTSL1, 2, 3, (see section on interrupts) should be set to jump to user code of the form



User Subroutines

Certain input/output subroutines are available to the user. Since these rely on a standard display format, this will be described first, followed by the user subroutine descriptions.

Tanbug V2 contains more user subroutines than Tanbug V1. Note that you can use your software written for Tanbug V1 without modification, since care has been taken to preserve the same locations for subroutines in Tanbug V2. Tanbug V2, however, contains a jump table at the low end of ROM, and this should now be used to access subroutines in future programs.

Display Format

Tanbug V2 is equipped with a "screen clear, cursor home" command. Monitor commands can be input on any line of the screen, but must begin at the left-hand edge. The cursor moves towards the right as characters are entered. When a line is filled, or a carriage return is output, the cursor moves down one line unless it is on the bottom line, when the display is scrolled (all lines shift up one row) and the bottom line becomes available for more output. However, there is no reason why users should restrict themselves to this mode of operation unless they intend to use Tanbug's subroutines to control the display in their own programs. It should be noted that the display memory is read/write memory and may be used as a character buffer prior to processing thus saving RAM locations for a user program.

Subroutine JPLKB

Subroutine JPLKB is used to interrogate the keyboard for a typed key. (Appropriate software for the type of keyboard in use is automatically set-up by TANBUG when a reset is issued). On exit from the subroutine the RAM location labelled ICHAR (address ØØØ1) contains the ASCII code of the character typed, whether it is typed on the keypad or on an alphanumeric keyboard. When using the alphanumeric keyboard, interrupts must be in the enabled state. As an example use the code

```
1)    CLI                    ;enable interrupts
2)    JSR JPLKB              ;poll the keyboard
3)    LDA ICHAR              ;load acc. with character
```

The sequence of operations here are

1) Enable interrupts so that alphanumeric keyboard may be interrogated.

2) The program loops around within the JPLKB subroutine until a key is pressed.

3) The program exits from JPLKB with the ASCII code for the key pressed in the location labelled ICHAR. The accumulator is loaded with this value.

Notes: Address of JPLKB is F81D. Address of ICHAR is ØØØ1. The registers IX, IY and A are corrupted, therefore, the user must save and restore their values if necessary.

Subroutine OUTRET

This subroutine outputs a carriage return to all the output devices, which react if they are enabled. It also re-instates the cursor, which is switched off when a user program is started. This subroutine should be called in a user program prior to any display input or output to clear the bottom line.

Notes: Address of subroutine OUTRET is F80B. Registers IX and IY are unaffected. Register A is corrupted and must be saved if required. This subroutine is equivalent to OUTPCR in Tanbug V1.

Subroutine OUTALL

This subroutine is called to output a character held in the accumulator, to all output devices which react if they are enabled. The cursor, obliterated on a user program start, is re-instated. As an example, consider the code

```
.
.
.
LDA#3Ø
JSR OUTALL
LDA#31
JSR OUTALL
.
.
.
```

Since 3Ø is the ASCII code for the character "Ø" and 31 is the ASCII code for the character "1", the result (assuming this is the first call to this subroutine) on the current line of the display is

Ø1█

Repetitive calls of OUTALL will fill the current line of the display with the appropriate characters. When the end of the line is reached, OUTALL moves on to the next line on the display. Carriage return is not output to printers, though you can of course output a carriage return via OUTALL.

Notes: Address of subroutine OUTALL is F8ØE. Registers IX and IY are unaltered. Register A is corrupted and must be saved if required. This subroutine is equivalent to OPCHR in Tanbug V1.

Subroutine JHXPN

Subroutine JHXPN takes a binary value from the accumulator and outputs it as two hexadecimal characters to all output devices. Consider the code

.
.
.

```
          PHA                       ; save A on stack
          JSR OUTRET                ; scroll display
          PLA                       ; recover A
          JSR JHXPN                 ; output A in hex
          JSR OUTRET                ; scroll display
                        .
                        .
                        .
```

This code will display the contents of the accumulator as two hex characters. For example if the accumulator contained the value 2C the resulting display would be

2C

Notes: Address of subroutine JHXPN is F81A. Register IY is unaltered. Registers IX and A are corrupted and must be saved if required. This subroutine is equivalent to HEXPNT in Tanbug V1.

Subroutine JHXPK

This subroutine reads hex characters from the current line of the display and packs them up into two eight bit binary values, enabling a sixteen bit word to be assembled. It is useful for incorporation into programs which require numerical keyboard input. Usually JPLKB is used in conjunction with OUTALL to enter data to the display, then JHXPK called when a carriage return is encountered. The following user code could be used to do this

```
                        .
                        .
                        .
          JSR OUTRET                ; scroll display
NXTCHR:   JSR JPLKB                 ; wait for character
          LDA ICHAR                 ; put it in A
          CMP# ØØ                   ; carriage return ?
          BEQ GOPACK                ; yes, pack it
```

```
        JSR OUTALL              ; else store in display
        JMP NXTCHR              ; get next character
1) GOPACK: LDY#ØØ               ; set IY to first char.
2)      JSR JHXPK               ; pack it
3)
                    .
                    .
                    .
```

In this example the subroutine is used in the following way:

1) Set IY with the character position at which packing is to start. The left most location of the current line corresponds to setting IY to Ø. The next location corresponds to IY equal to 1 etc.

2) Call JHXPK. Characters are packed until a character other than 0-9 or A-F is encountered; an exit then occurs.

3) Continue into the user code where the values of HXPKL and HXPKH will be read.

For example, packing 1 CR gives HXPKL = 1 and HXPKH = Ø. Packing FEDC CR gives HXPKL = DC and HXPKH = FE. Packing FEDCBA CR gives HXPKL = BA and HXPKH = DC, i.e. if more than four hexadecimal characters in succession are encountered then the last four are packed. Additionally, two flags in the processor status word (PSW) are used to indicate exit conditions. The zero flag Z is clear if the terminating character is the cursor (ASCII code FF), set otherwise. The overflow flag V is set if there was one or more hex characters, clear if the first character encountered by the subroutine was not a hexadecimal character.

Notes: Address of subroutine JHXPK is F817. Address of HXPKL is ØØ13 and HXPKH is ØØ14. Registers IX, IY and A are all corrupted and must be saved if necessary. This subroutine is equivalent to HEXPCK in Tanbug V1.

Subroutine JCURSF

Subroutine JCURSF is used to obliterate the cursor from the screen. It writes a space into the location pointed to by ICURS + ICURSH and VDUIND, where the cursor would be displayed by the monitor and OUTRET, OUTALL subroutines. Address of JCURSF is F829. No registers are corrupted.

Subroutine JCURSN

Subroutine JCURSN is the converse of JCURSF, that is it writes the cursor symbol 7F to the location pointed to by ICURS + ICURSH and VDUIND.

The above two subroutines can be used to control the Microtan cursor. Refer to the display address map on page 3.2 of the Microtan manual to obtain the addresses of the start of each line. A program to place the cursor at the end of the second line on the display would be as follows:

```
JSR JCURSF       ; turn the cursor off, wherever it is
LDA #20          ; set low part of line address
STA ICURS
LDA #2           ; set high part of line address
STA ICURSH
LDA #$1F
STA VDUIND       ; set how many chars long line
JSR JCURSN       ; switch the cursor on
```

JCURSN is located at F826. No registers are corrupted.

Subroutine RETMS

Calling subroutine RETMS (via JSR or JMP) can be used to return to the monitor after executing your program, without a Tanbug message being printed. The stack pointer is reset to 1FF.

Address of RETMS is F820. The monitor corrupts X, Y and A.

Subroutine RETMON

This subroutine performs as RETMS, except that the stack is not reset. You can therefore return to the monitor, via the instruction JMP RETMS, without changing your programs stack. By calling JSR RETMON, you can if required call the monitor as a subroutine in your program.

RETMON is located at F823. The monitor corrupts X, Y and A.

Subroutine JMNRW

Subroutine JMNRW is included to allow simple manipulation of the memory management system. (For hardware operation, refer to the section on memory management). The subroutine itself is located in the Monitor area, with its variables in zero page. These areas are not affected by operation of the memory management. Subroutine JMNRW can therefore be called from code resident in bank Ø to access other pages. on exit, page Ø is reselected by the subroutine.

The following zero-page locations are used by the subroutine:

MEMSEG(40): The least significant 4 bits are set by the user to contain the page number to be written to. If they are set to Ø a write operation is not executed. The most significant 4 bits operate similarly for a read. (They correspond directly to the memory management status word). The subroutine is thus multipurpose in that it can read, write, or read old contents and write new contents in one operation.

MEMDAW(41): The user loads this location with data to be written before calling JMNRW.

MEMDAR(42): The subroutine loads this location with data read from the required location.

MEMLO(43): Low byte of address to be manipulated.

MEMHI(44): High byte of address to be manipulated.

Note that the subroutine does not change the contents of any of these locations, therefore if you wish to repeatedly read and write from a particular location you only need to carry out the setup procedure once.

Example - to read from location 6000 in page 1, and then write a different value:

```
LDA #11          ; set up for read and write
STA 4Ø           ; to page 1
LDA #FF
STA 41           ; data to be written
LDA #Ø
STA 43
LDA #6Ø          ; set up address
STA 44
JSR JMNRW        ; execute
LDA 42           ; read data loaded to acc.
```

Address of JMNRW is F811. No registers are corrupted.

<u>Subroutine JMNRW1</u>

This subroutine is exactly equivalent to JMNRW, but after the operation has been executed, the address held in locations 43, 44 is incremented by 1, providing a convenient means for block operations.

Address of JMNRW1 is F814.

<u>Subroutine PRPUP</u>

Subroutine PRPUP powers up the parallel printer interface, and initialises the printer ready to receive data. See also the section on input/output control.

Note that this subroutine does <u>not</u> set the "PAR ON" bit in the Printer Status Word, but merely sets up the hardware to initialise the printer.

PRPUP is located at F8ØØ. The accumulator is corrupted.

Subroutine OUTPAR

Subroutine OUTPAR outputs the character held in OCHAR (location 2) to the parallel printer interface, irrespective of whether the "PAR ON" bit in the printer status word is set. The printer should be initialised via PRPUP at the start of your program.

OUTPAR is located at F8Ø3. A. X and Y are corrupted.

Subroutine OUTSER

As OUTPAR, but outputs via the V24 UART on Tanex. There is no need to initialise this, as it is done by Tanbug on a Reset.

OUTSER is located at F8Ø6. A, X and Y are corrupted.

Subroutine OUTSCR

Subroutine OUTSCR takes the ASCII value in the accumulator, and outputs it to the display screen but neither printer.

OUTSCR is located at F8Ø9. A, X and Y are corrupted.

Interrupts

TANBUG uses the maskable and non-maskable interrupts. However, means have been provided to access the interrupts via both hardware and software. Of necessity user interrupts may, in some cases, place restrictions on certain monitor commands.

The Maskable Interrupt

When TANBUG is initialised by a reset, certain RAM locations are set up to link through the interrupts for monitor use. These locations are labelled INTFS1, INTFS2, INTFS3 and INTSL1. When a maskable interrupt occurs, the following sequence of events is obeyed (assuming the RAM locations mentioned above have not been modified).

a) The program jumps to INTFS1 in RAM.

b) The locations INTFS1, INTFS2 and INTFS3 contain the instruction JMP KBINT. The program therefore jumps to KBINT which resides in the monitor ROM.

c) The monitor software looks to see what caused the interrupt. If a BRK instruction, then the breakpoint code is executed. If a keyboard interrupt, location ICHAR is updated with the new ASCII character which is read from the keyboard I/O port.

d) If the interrupt is caused by anything other than a BRK instruction, then the monitor jumps to location INTSL1.

e) Normally INTSL1 contains an RTI instruction - the program would then return to where it was interrupted.

It can therefore be seen that the user can implement his/her own interrupt service routines in two ways.

1) A fast interrupt response by modifying the locations INTFS1, INTFS2 and INTFS3 to jump to the user interrupt service code. In this case breakpoints and the ESC command cannot be used unless the user program jumps back to the monitor service routine after executing its own code.

2) A slower interrupt response by modifying INTSL1, INTSL2 and INTSL3 to jump to user service routine, after executing the monitor service routine. The RAM locations INTSL1, INTSL2 and INTSL3 would be modified to contain the instruction JMP USER. This method places no restrictions on monitor commands.

   The slow interrupt facility cannot be used if the external output link is in operation. See section on printers.

A number of things should be noted when using interrupts:

a) An RTI instruction must always occur at the end of user code to return the program to the point at which it was interrupted, unless the user code jumps back to the monitor service routine.

b) If a reset is issued, the INTFS and INTSL locations are set back to their monitor values by TANBUG, and the user has to reset them.

c) If any microprocessor internal registers are used in the user interrupt service routine, they must be saved before modification, and restored before the RTI instruction, i.e. on return to the monitor the registers IX, IY and A must contain the same values as they had on entry to the user routines.

d) The interrupt jump locations should be modified by instructions in the user program at run time and not by the use of the M command. This is because TANBUG software uses keyboard interrupts. If using an alternative link at INTFS1, no breakpoints can be set.

e) Addresses of RAM locations are: INTFS1 = ØØØ4, INTFS2 = ØØØ5, INTFS3 = ØØØ6, INTSL1 = ØØ1Ø, INTSL2 = ØØ11, INTSL3 = ØØ12.

The Non-maskable Interrupt

The non-maskable interrupt vector is accessed in the same way as explained for the maskable interrupt. The user can obtain access by modifying locations NMIJP, NMIJP1, and NMIJP2. Note that single instruction mode will be inoperative and that breakpoints will be destructive, i.e. they are destroyed when they have been executed once and replaced with the original code. Addresses of RAM locations are: NMIJP = ØØØ7, NMIJP1 = ØØØ8 and NMIJP2 = ØØØ9.

Error Linking

It will be noted that TANBUG displays a question mark whenever an illegal command is typed. In order to allow future expansion of the monitor, an error link to memory external to the monitor ROMs is incorporated.
When an error occurs the following sequence of events is initiated:

a) The program jumps to F7F7.

b) With no expansion board (TANEX) present the address F7F7 (outside TANBUG space) is decoded as address FFF7 (inside TANBUG space).

c) A question mark is printed.

With TANEX present, a special link is incorporated to return the program to the monitor. The user may remove this link and insert an EPROM in the position which includes the address F7F7 containing the code JMP USERCODE at address F7F7, where USERCODE may contain software to deal with any extra commands the user wishes to add to the monitor. Note that this facility will be used by future TANGERINE software.

There are two methods of returning to the monitor from external code:

1) The instruction RTS at the end of the user code returns to the monitor, gives a carriage return then continues looking for commands.

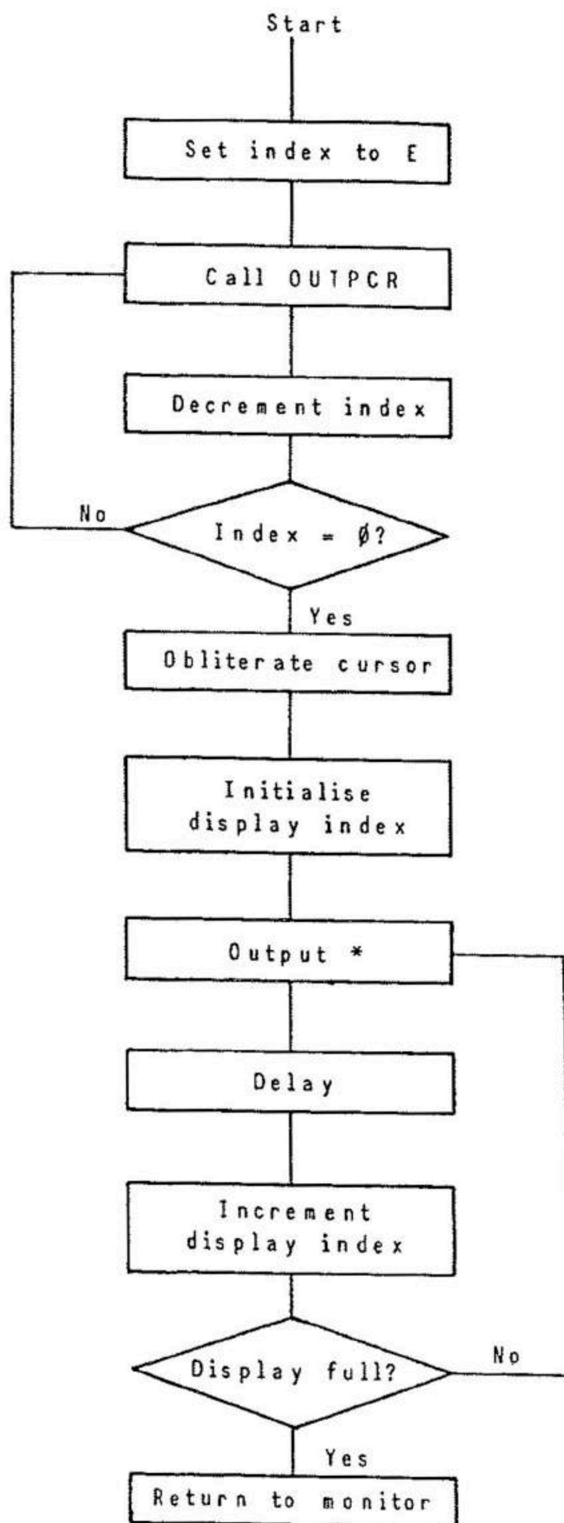
2) The instruction JMP FFF7 returns to the monitor, giving a question mark on the display.

Example of TANBUG's Use

The following simple example program clears the screen by calling OUTPCR F times, then slowly fills the screen with asterisks. It is used as an example to demonstrate the use of some of TANBUG's commands.
Deliberate errors are later written into the program to demonstrate TANBUG's fault finding capabilities.

The first step in writing a program is to produce a flowchart of program execution. The second step is to write the program in assembly language code using the instruction mnemonics. The third step is to look up and write the op-codes and arguments for each instruction. At this stage the branch code arguments will be left blank and TANBUG's O command used.

The flowchart and program listing now follows.


Start
Set index to E
Call OUTPCR
Decrement index
No
Index = Ø?
Yes
Obliterate cursor
Initialise
display index
Output *
Delay
Increment
display index
Display full?
No
Yes
Return to monitor

Example program listing

```
ØØ5Ø  ØØ  ØØ            VDUIND: Ø               ;display index
ØØ52  AØ  ØF             START: LDY# F          ;set Y index
ØØ54  2Ø  73  FE         SCRAG: JSR  OUTPCR     ;carriage return
ØØ57  88                        DEY             ;do E times
ØØ58  1Ø  (arg 1)               BPL  SCRAG
ØØ5A  A9  2Ø                    LDA# 2Ø         ;load A ascii space
ØØ5C  8D  EØ  Ø3                STA  3EØ        ;obliterate cursor
ØØ5F  A9  ØØ                    LDA# Ø          ;set display index
ØØ61  85  5Ø                    STA  VDUIND
ØØ63  A9  Ø2                    LDA# 2
ØØ65  85  51                    STA  VDUIND+1
ØØ67  AØ  ØØ              CONT: LDY# Ø          ;clear Y index
ØØ69  A9  2A                    LDA# 2A         ;set ascii *
ØØ6B  91  5Ø                    STA (VDUIND),Y
ØØ6D  A2  ØF                    LDX# F          ;delay loop
ØØ6F  AØ  FF                    LDY# FF
ØØ71  88                 DECIT: DEY
ØØ72  DØ  (arg 2)               BNE  DECIT
ØØ74  CA                        DEX
ØØ75  DØ  (arg 3)               BNE  DECIT
ØØ77  18                        CLC             ;inc display index
ØØ78  E6  5Ø                    INC  VDUIND
ØØ7A  DØ  (arg 4)               BNE  NOMSB
ØØ7C  E6  51                    INC  VDUIND+1
ØØ7E  A5  51             NOMSB: LDA  VDUIND+1   ;top of display?
ØØ8Ø  C9  Ø3                    CMP# 3
ØØ82  DØ  (arg 5)               BNE  CONT       ;no - continue
ØØ84  A5  5Ø                    LDA  VDUIND
ØØ86  C9  FF                    CMP# FF
ØØ88  DØ  (arg 6)               BNE  CONT       ;double prec. cmp.
ØØ8A  ØØ                        BRK             ;return to monitor
```

Program entry is performed using the M command. For the time being set the branch arguments (arg 1 - arg 6) to ØØ, these can be altered when calculated, using the O command.

Once the program is entered the branch offsets are calculated. The first is arg 1 which has an opcode address of ØØ58 and branches to the label SCRAG at location ØØ54. By typing O58,54 CR TANBUG prints out the value of arg 1 as FA. This may now be placed in location ØØ59 using the M command. By repeating the exercise for the other five arguments, it will be found that location ØØ73 should contain FD, ØØ76 should contain FA, ØØ7B should contain Ø2, ØØ83 should contain E3 and ØØ89 should contain DD.

The program will now run if it has been entered correctly. To start the program type G52 CR since the first instruction of the program is at location ØØ52. When the screen is full of asterisks the program exits to the monitor. Alternatively, if an alphanumeric keyboard is being used, depression of the ESC key causes an exit to the monitor. If the program does not run correctly, then it may be necessary to issue a reset in order to regain control. The program can be listed by typing L5Ø,8 CR yielding a display of

```
L5Ø,8
ØØ5Ø  ØØ  ØØ  AØ  ØF  2Ø  73  FE  88
ØØ58  1Ø  FA  A9  2Ø  8D  EØ  Ø3  A9
ØØ6Ø  ØØ  85  5Ø  A9  Ø2  85  51  AØ
ØØ68  ØØ  A9  2A  91  5Ø  A2  ØF  AØ
ØØ7Ø  FF  88  DØ  FD  CA  DØ  FA  18
ØØ78  E6  5Ø  DØ  Ø2  E6  51  A5  51
ØØ8Ø  C9  Ø3  DØ  E3  A5  5Ø  C9  FF
ØØ88  DØ  DD  ØØ  XX  XX  XX  XX  XX
█
```

providing the program has been correctly entered (XX indicates any value as these locations are not part of the program). If the program failed to run, carefully check the listing from the L command with the program listing and correct any errors with the M command.

Having got the program working it is now possible to introduce a deliberate error to demonstrate the use of breakpoints and the single instruction mode. The error to be introduced is to put the wrong value for the branch argument on the first occurrence of the instruction BNE DECIT; instead of location 73 containing FD change it to FB. Now the register IY will never be zero and the program will loop here. If the program is started now only one asterisk will be printed and then nothing else will happen. Debugging steps are as follows:

a) Regain control to the monitor by issuing a reset.

b) The first part of the program is being executed correctly as the display scrolls. Furthermore, it is at least getting to location 6B because an asterisk is printed. It would be very tedious to single instruction this far from the beginning because the OUTPCR routine is called sixteen times. Therefore, set a breakpoint at location 6D by typing B6D,Ø<CR>.

c) Start the program again by typing G52<CR>. The display scrolls and the status message

```
ØØ6D  31  FF  ØF  ØØ  2A
```

is displayed. Control is now back in the monitor.

d) Set single instruction mode by typing S<CR>.

e) Repeatedly typing P <CR> causes single instructions to be executed followed by a status print-out. The following sequence of instructions will be observed.

```
ØØ6F  21  FF  ØF  ØØ  2A
ØØ71  A1  FF  ØF  FF  2A
ØØ72  A1  FF  ØF  FE  2A
ØØ6F  A1  FF  ØF  FE  2A
```

Now if the code were correct the program could not go back to location 6F. In fact, since IY is shown to be FE, the program should have jumped back to location 71. The branch instruction is probably at fault, therefore examine it and its argument using the M command.

```
M72,DØ,
MØØ73,FB,█
```

The value in location 73 should be FD, therefore, change it by typing FD<CR>.

f) Remove single instruction mode and breakpoints by typing N<CR> the B<CR>.

g) Restart the program by typing G52<CR> . The program should now run correctly.

Note that when using an alphanumeric keyboard, debugging is slightly easier. When the program sticks in a loop ESC can be used to return to the monitor (provided interrupts have not been disabled). Single instruction mode can then be set to determine the loop in which the program was running.

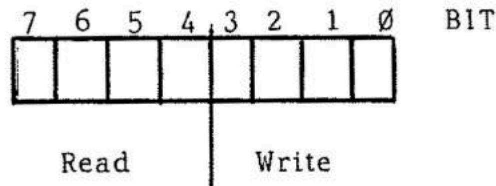
Memory Management Control

The memory management system allows selection of alternative banks of certain areas of memory:-

Area	Address	
Z Page	Ø – 1ØØ	Unaffected by memory management
Stack	1ØØ – 1FF	Unaffected by memory management
Display	1FF – 3FF	Unaffected by memory management
TANEX RAM	3FF – 1FFF	Unaffected by memory management
	2ØØØ – BBFF	Alternative banks within this range can be selected
I/O ROM	BBFF –	Unaffected by memory management

This is carried out by means of a write-only status word at location FFFF - located within the monitor space, but since the monitor ROM is read-only, a write to this location does not affect the monitor, and saves address space.

The control register is allocated as follows:



Bits 0,1 and 2 (for write) and 4, 5 and 6 (read) control which expansion slot within the motherboard is to be accessed. Bit 3 (write) and 7 (read) must be Ø to access the system rack containing the Micron, 7 to access an expansion rack. It can be seen that the ROM area, the I/O area, plus the 7K of RAM on TANEX, are unaffected by the memory management settings. Thus any of the programs in ROM, plus those written for the TANEX 7K, can access any page of additional memory simply by setting the required bits in the Memory Management word.

However, when a program is written to reside in RAM upwards of address 2000, it is not possible to use code located within this space to read data from another page because, as soon as the memory management status read page is changed, the instruction fetch will also occur at that page.

To allow data to be written/retrieved from other pages in this case, two memory management subroutines, fully described in the subroutine section, are supplied.

It is strongly recommended that page Ø only of additional RAM (2ØØØ-BBFF) is used to store code (machine or BASIC) and that other pages are accessed via the memory management subroutines.

Tanbug V2 and Microsoft Basic

Tanbug V2 has been designed to be completely compatible with existing versions of Microsoft Basic. In addition, extra facilities have been included to enhance BASIC's features.

Basic Initialisation and Warm Start

Instead of typing GE2ED <CR> to start BASIC, you should now type

BAS <CR>

This starts BASIC for you, and initialises the system so that you can recover with a warm start. (GE2ED does not do this).

Now, if you exit from BASIC using THE RESET key, you can re-enter it, preserving the program you had entered in BASIC, by typing

WAR <CR>

you can, in fact, execute other monitor functions (Modify Memory, Translate, Instruction disassemble etc.) in between leaving BASIC and re-entering it, provided you do not corrupt any of locations (hex) 80 - 15F, or any of your program locations (from 400 upwards, the limit depending on the length of your program).

Your can NOT use WAR when:

a) Reset was hit while BASIC was dumping to cassette.

b) Reset was hit while BASIC was loading from cassette.

c) If you have not first initialised BASIC with a BAS command.

A failure-to-run will be denoted by a breakpoint status error printout, from which it will be necessary to hit RESET and type BAS.

Printer Control from Microsoft BASIC

There are two methods of printer control via TANBUG V2, direct mode and program mode. (See also section on printers).

In direct mode, printers can be controlled as follows:

CTRL P	-	repeated operations alternately turn the parallel printer on and off.
CTRL V	-	repeated operations alternately turn the serial printer on and off.
CTRL S	-	repeated operation alternately turn the TV display on and off.

In program mode, certain character pairs can be output to control the printer as follows (decimal numbers):

17, Ø	Serial output on
17, 1	Serial output off
17, 2	Parallel output on
17, 3	Parallel output off
17, 4	Screen on
17, 5	Screen off

As an example, the following program asks a question on the display, prints the answer on the printer but not on the display, then asks another question on the display (printer is off, screen on assumed at start).

```
10    INPUT "WHAT IS YOUR NAME"; A$

20    PRINT CHR$(17); CHR$(5); CHR$(17); CHR$(2)

30    PRINT A$

40    PRINT CHR$(17); CHR$(3); CHR$(17); CHR$(6)

50    INPUT "WHAT IS YOUR ADDRESS"; A$
etc.
```

Clear Screen

In direct mode, typing CTRL L (except when in EDIT) clears the display and puts the cursor in the top left hand corner of the screen. Subsequent operations then work down the screen.

A screen clear can also be called from program mode by the instruction

```
PRINT CHR$(12)
```

Memory Management Control

Page Ø of expansion RAM (address 2ØØØ upwards) only, can be used for BASIC program expansion. Other memory pages can be used for data storage by calling the memory management subroutines via the USR command. For a full description of these subroutines, see the soubroutine section and memory management section of this manual.

As an example, consider a BASIC subroutine to read the contents of memory location 6ØØØ (hex) in page 1, and then write 1 to it:- (the comments are for clarification and are not part of the program).

```
4000    POKE 34, 17
4010    POKE 35, 248     ; set up subroutine address (F8 hex)
4020    POKE 64, 17      ; set up which page (11hex), read and
                           write
```

```
4030    POKE 65, 1      ; data to be written
4040    POKE 67, 0      ; mem address (6000 (hex))
4050    POKE 68, 96
4060    X = USR(I)      ; execute
4070    X = PEEK (66)   ; read data into X (before write)
```

Since all locations except the read value are unaffected (unless you use the INPUT command) subsequent writes need only consist of

```
4080    POKE 65, 2      ; new data
4090    X = USR(I)      ; write
```

Note that you can also call the MMINC subroutine (POKE 34,20 POKE 35, 248) which automatically increments the memory address in locations (decimal) 67 and 68. This facility enables writing to or reading from sequential locations with minimal code overhead.

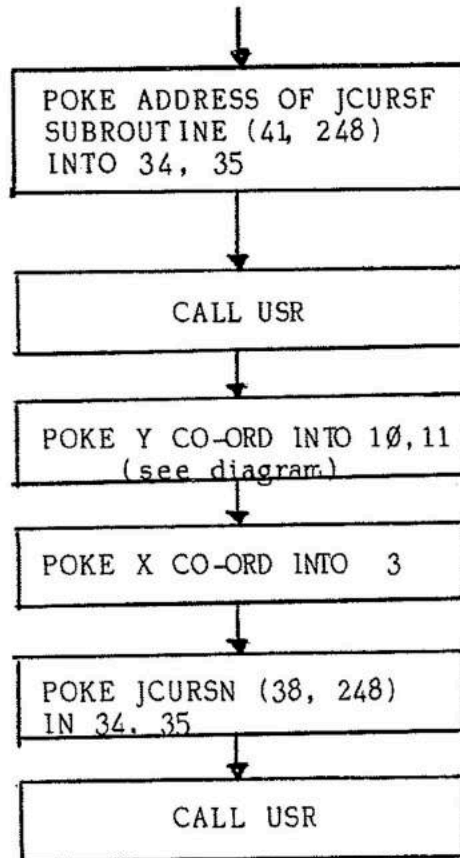
Note that decimal locations 64 - 68 are corrupted by use of the input command, so if this is used between memory management operations, these locations must be reset by POKES.

Cursor Control

The cursor may be placed anywhere on the screen by calling the JCURSF and JCURSN routines in Tanbug V2, via the USR call. The screen can be divided as follows:

	0	1	2		30	31	X coord (decimal)
02 00							
02 32							
02 64							
etc.							

A flow diagram for the cursor control subroutine is as follows:



The cursor will, of course, obliterate the character over which it appears.

The example program below make the cursor appear in the top right-hand quadrant of the screen, (it is necessary to type CTRLC to exit from this program).

```
10 POKE 34, 41
20 POKE 35, 248
30 X = USR(0)
40 POKE 10, 32
50 POKE 11, 2
60 POKE 3, 30
70 POKE 34, 38
80 POKE 35, 248
90 X = USR (0)
100 GOTO 100
OK
```

TABLE OF HEX ASCII CODES

ØØ	NUL		
Ø1	Control A	-	Home
Ø2	Control B		
Ø3	Control C		
Ø4	Control D		
Ø5	Control E		
Ø6	Control F		
Ø7	Control G	-	Bell
Ø8	Control H	-	Backspace
Ø9	Control I	-	Horizontal Tab - Cursor Right
ØA	Control J	-	Line Feed
ØB	Control K		
ØC	Control L	-	Page Clear - Form Feed
ØD	Control M	-	Carriage Return
ØE	Control N		
ØF	Control O		
1Ø	Control P		
11	Control Q		
12	Control R		
13	Control S		
14	Control T		
15	Control U		
16	Control V		
17	Control W		
18	Control X		
19	Control Y		
1A	Control Z	-	Vertical Tab - Cursor Up
1B	S1		
1C	S2		
1D	S3		
1E	S4		
1F	S5		

Note that the codes ØØ - 1F produce special symbols when used in display memory.

TABLE OF HEX ASCII CODES (CONTINUED)

2Ø	Space	4Ø	@	6Ø	'
21	!	41	A	61	a
22	"	42	B	62	b
23	£ or #	43	C	63	c
24	$	44	D	64	d
25	%	45	E	65	e
26	&	46	F	66	f
27	'	47	G	67	g
28	(	48	H	68	h
29	)	49	I	69	i
2A	*	4A	J	6A	j
2B	+	4B	K	6B	k
2C	,	4C	L	6C	l
2D	-	4D	M	6D	m
2E	.	4E	N	6E	n
2F	/	4F	O	6F	o
3Ø	Ø	5Ø	P	7Ø	p
31	1	51	Q	71	q
32	2	52	R	72	r
33	3	53	S	73	s
34	4	54	T	74	t
35	5	55	U	75	u
36	6	56	V	76	v
37	7	57	W	77	w
38	8	58	X	78	x
39	9	59	Y	79	y
3A	:	5A	Z	7A	z
3B	;	5B	[	7B	{
3C	<	5C	\	7C	¦
3D	=	5D	]	7D	}
3E	>	5E	^	7E	~
3F	?	5F	_	7F	█ or Rubout

TANGERINE
COMPUTER SYSTEMS LIMITED
Forehill Works Forehill Ely Cambs England