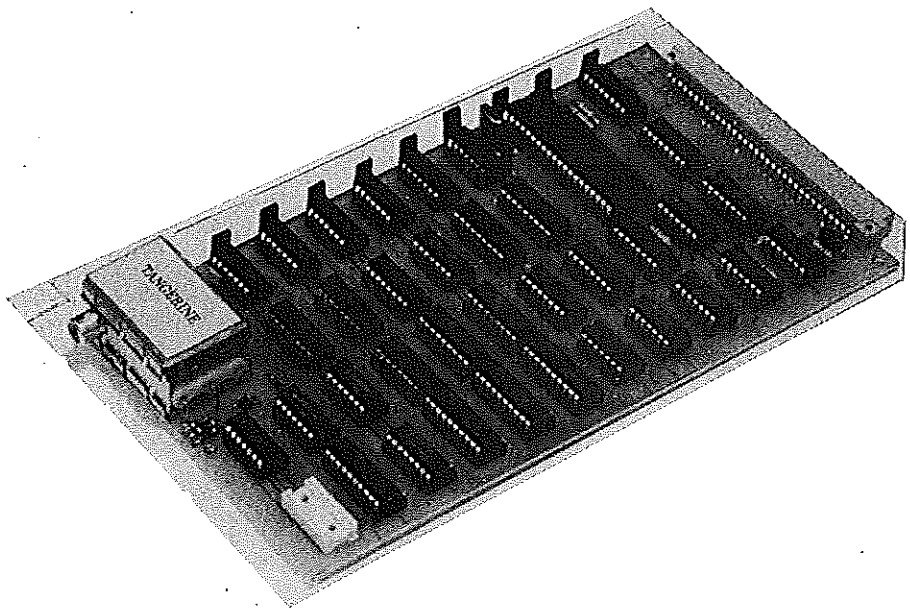
www.microtan.UKPC.net

TANGERINE



microtan 65

USERS MANUAL

(Original Version)

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. Games programs have been provided in chapter seven.

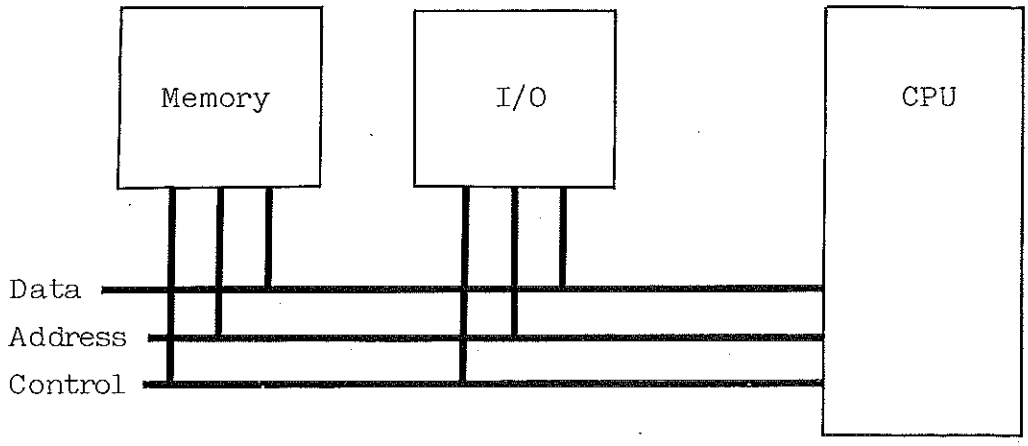
CHAPTER 1

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

111ØØ11Ø. Now add 57 and -26.

Ø Ø 1 1 1 Ø Ø 1		57
1 1 1 Ø Ø 1 1 Ø	+	-26
1 Ø Ø Ø 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 11ØØØ111.

Ø Ø Ø 1 1 Ø 1 Ø		26
1 1 Ø Ø Ø 1 1 1	+	-57
Ø 1 1 1 Ø Ø Ø Ø 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 nct 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 grcuping of four binary bits representing a decimal digit. As four binary bits enables the decimal numbers Ø-15 to be represented, the binary codes representing 1Ø-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 DØ
Ø 1 1 Ø	Ø Ø Ø 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.

ØØ1Ø	1ØØ1	ØØ1Ø	Ø1Ø1
|	|	|	|
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 Ø'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

```
11ØØ    Ø111
 |       |
 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.

CHAPTER 2

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. As these are only required when the user expands his system they are supplied with TANEX.

Component List and Identification

I.C. sockets: 16 off 14 pins, 11 off 16 pins, 4 off 18 pins, 4 off 2Ø pins, 1 off 4Ø pins, 1 of keyboard socket is of different manufacture.

Resistors:

R1	22ØR	R8	1K
R2	47ØR	R9	1K
R3	75R	R1Ø	1K
R4	1ØK	R11	1K
R5	1K	R12	1K
R6	1K	R13	1K
R7	1K	R14	1ØK

Resistors are identified by their coloured bands which also serve to indicate their value. These bands are as shown:

1 2 3 4

Ignore band 4 as this is irrelevant, it is only bands 1, 2 and 3 that are of interest. The colour code for R1 (22ØR) is red, red,

brown; R2 (47ØR) is yellow, mauve, brown; R3 (75R) is mauve, green, black; R4 and R14 (1ØK) is brown, black, orange; and the rest R5-R13 (1K) is brown, black, red.

Capacitors:	C1	47n
	C2-C1Ø	decoupling capacitors
	C11	22n or 47n
	C12	1ØØp or n1Ø
	C13	1Øn

Capacitors are identified by their values which are printed on them. They are small flat plates, rather than tubes and their leads are such that they stand up rather than lay down.

Diodes:	D1-D3	IN4148 or equivalent
	D4	BZY88 C6V2 or C6V8

The diodes look a bit like resistors but are generally smaller and glass encased. D1-D3 can be identified as the three diodes that are the same as each other. D4 is the one that is different and is a zener diode. Diodes must be connected into the board the correct way round. At one end is a band or a thicker band than any others. This end must be connected to the marked end of the legend on the printed circuit board.

Transistors: There are only two transistors and they are identical. They have three leads preformed triangularly. Type BC184C (Tr1 and Tr2).

Inductor: Only one. Looks like a resistor but has no coloured bands. Value - 100uH.

UHF Modulator: UM1111E36

Crystal: 6MHz.

Integrated Circuits:

C1 74LSØØ	D1 74LS32
E1 74LS73	F1 74LS11
G1 74LS21	H1 74LSØ4
K1 65Ø2	M1 74LS367 (TANEX)
C2 8678BWF	D2 8678CAE (Lower Case)
E2 74LS138	F2 74LSØØ
G2 Blue Spot	H2 White Spot
M2 74LS367 (TANEX)	A3 74LS74
B3 74LS74	C3 74LS86
D3 2102 (Graphics)	E3 2114
F3 2114	G3 74LS157
H3 74LS157	J3 74LS157
L3 74LS74	M3 74LS367 (TANEX)

B4	74LS244	C4	74LS251 (Graphics)
D4	74LS374 (Graphics)	E4	74LS374 (Graphics)
F4	74LS374	G4	74LSØ8
H4	74LS393	J4	74LS393
L4	74LS393	J4	74LS393
L4	74LS74 (Lower Case)	M4	74LS74 (Graphics)

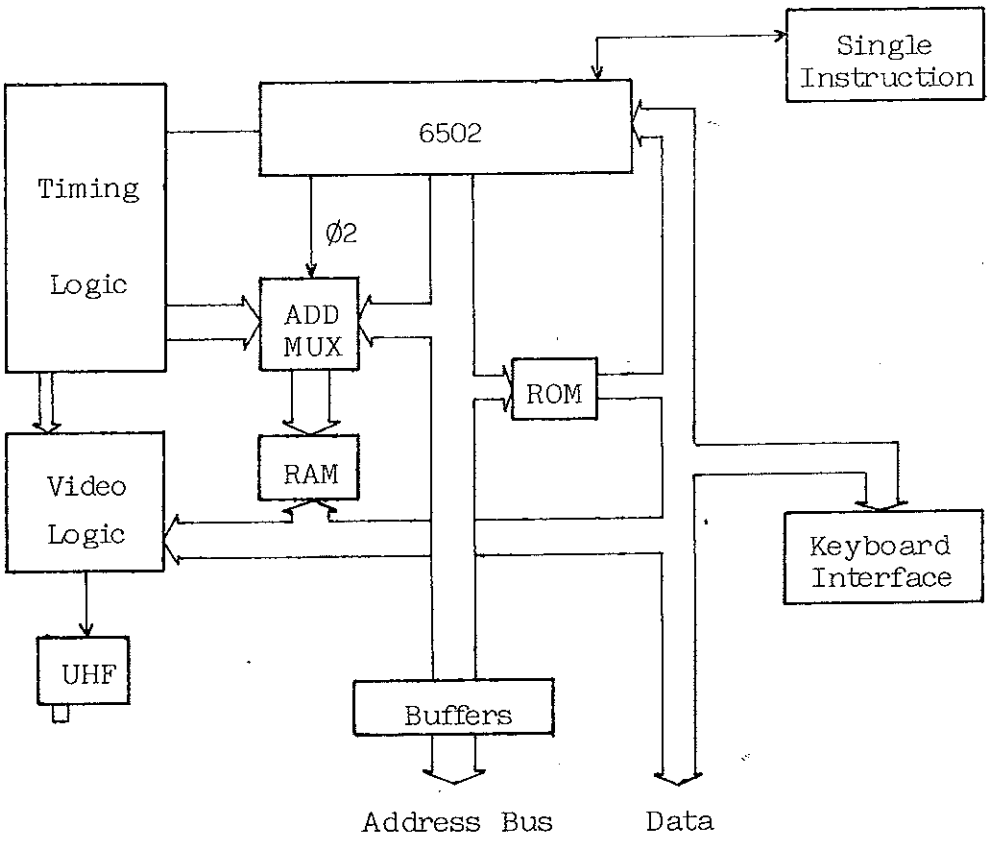
It should be obvious which devices are the integrated circuits, there is nothing else left! They should be fitted into the I.C. sockets with pin 1 nearest to the dot on the printed circuit legend. The end of the integrated circuit with pin 1 has a pip or indentation next to pin 1 and/or a piece taken out of the moulding at that end.

CHAPTER 3

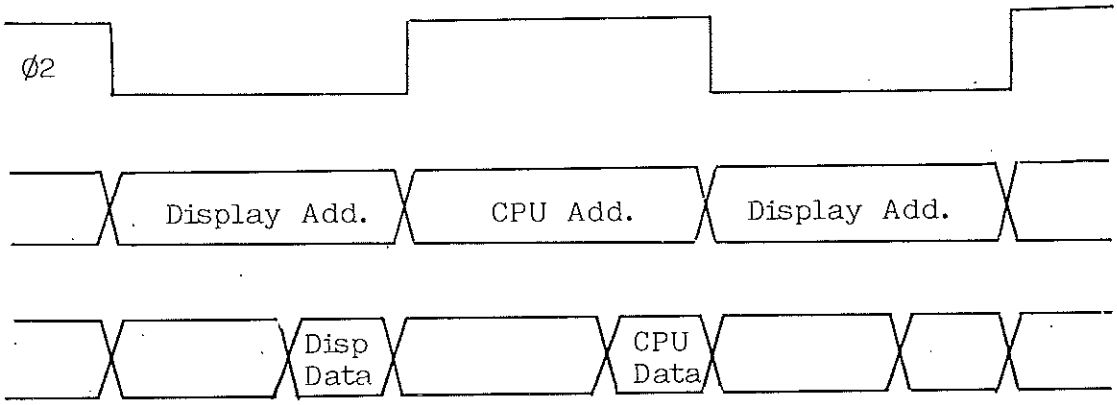
The Microtan 65

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.

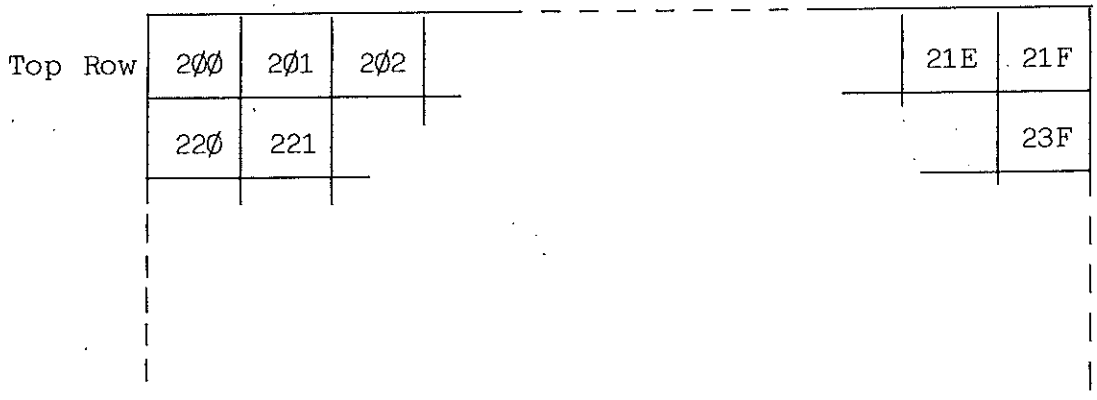
A fully expanded microtan 65 contains the 65Ø2 microprocessor, 1K ROM containing TANBUG, 1K RAM which is used for display memory and user program, keyboard interface, VDU logic and tri-state address buffers. The basic block diagram of the microtan 65 is shown below.

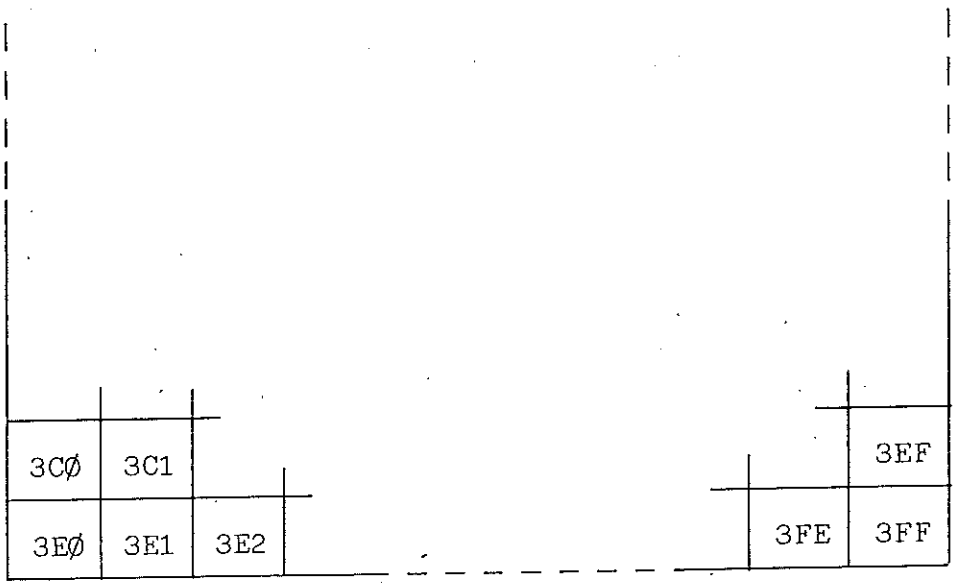


The most important aspect of the design is that the address multiplexer is switching at the processor clock rate. This can be done with the 65Ø2 as memory accesses will only occur on one phase of the clock i.e. when Ø2 is high. When Ø2 is low the memory is not selected. During this time the VDU logic 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.

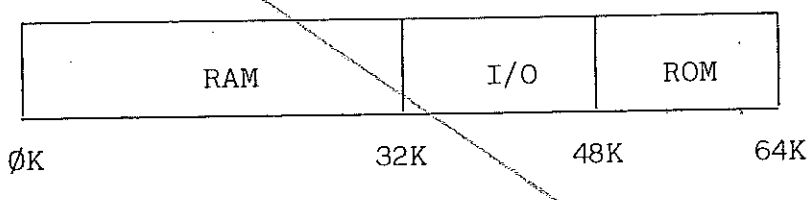




To write a particular character into a character cell in the display the user must write the ASCII code, or whatever, for that character in the correct memory location. For example, if it was required to write the character "W" in the bottom row third column then the ASCII code 57 (for W) should be placed 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.

Not applicable with TANEX connected



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 2102 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.

Data Byte

D_7	D_6	D_5	D_4	D_3	D_2	D_1	$D_\emptyset$

Decoded as ASCII

Graphics Bit

$\emptyset$

$\emptyset$ denotes ASCII

$D_\emptyset$	D_1
D_2	D_3
D_4	D_5
D_6	D_7

1

1 denotes graphics

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.

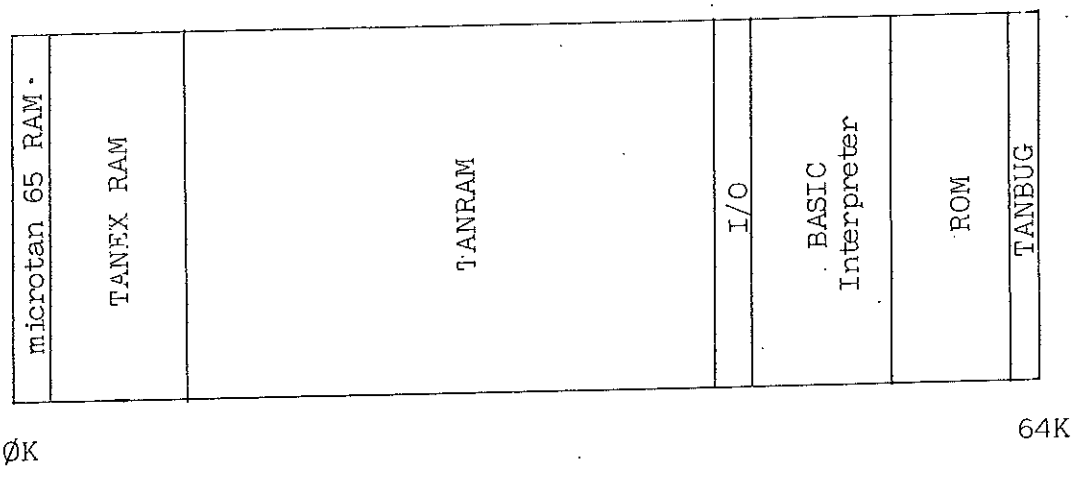
c) Delayed non-maskable interrupt: This facility is used by the monitor program TANBUG when executing single instructions and breakpoints. It should not be used in a user program. Users who will contemplate using the non-maskable interrupt in special applications will probably not be using TANBUG. In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

CHAPTER 4

The Microtan System

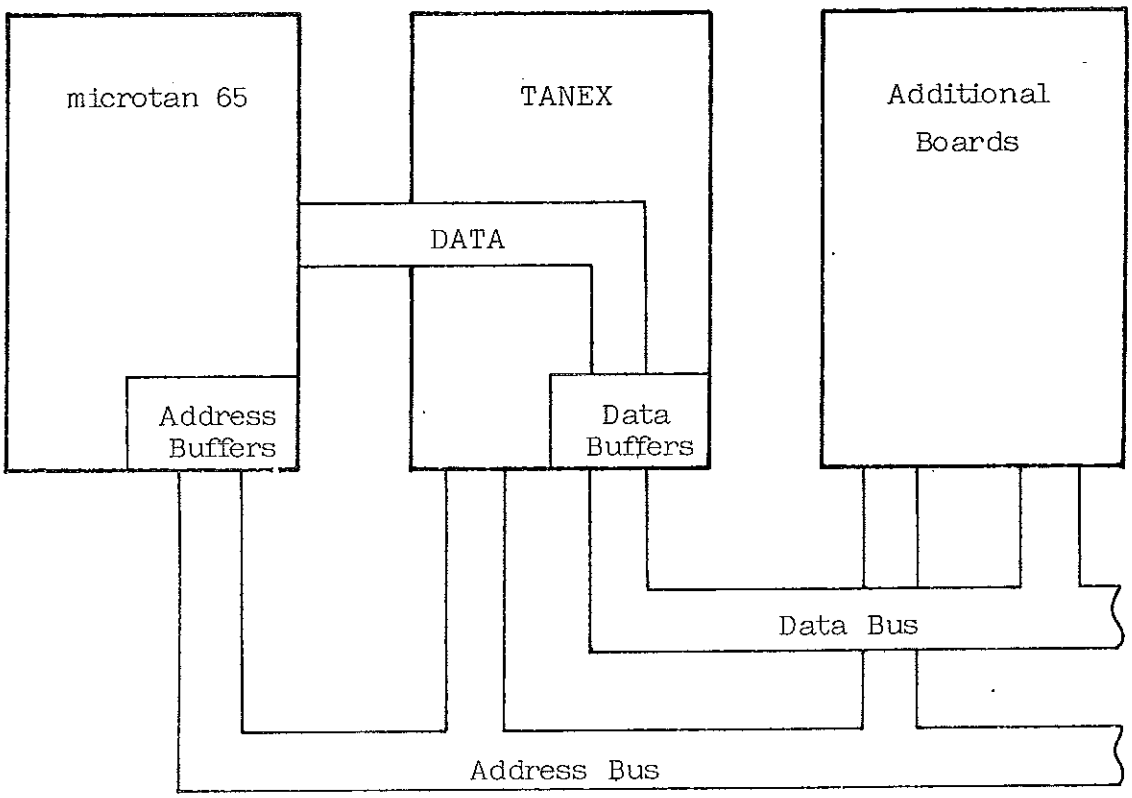
Microtan 65 expands into a full microcomputer system. All the printed circuit cards connect into a motherboard and use TANBUS as means to communicate to each other. TANBUS is a collective name for all of the signals that use the edge connector on each of the cards. The cpu card and expansion board have 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. (Microcomputers 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 4ØK 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 RS232/2ØmA) 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
$\overline{IRQ}$	Interrupt request. An open collector line used by devices requesting an interrupt.
$\overline{NMI}$	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.
$\overline{FB}$	Field blanking of the television display. Driven from microtan 65. For future use.
$\overline{DMAPOT}$	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.
$\overline{DMAPIN}$	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.
$\overline{IOE}$	TANEX output. Indicates that the address bus is addressing I/O on the microtan 65, locations BFFØ-BFFF.
$\overline{RAME}$	TANEX output. Indicates that the address bus is addressing RAM on the microtan 65, i.e. locations Ø-3FF.
$\overline{ROME}$	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.

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

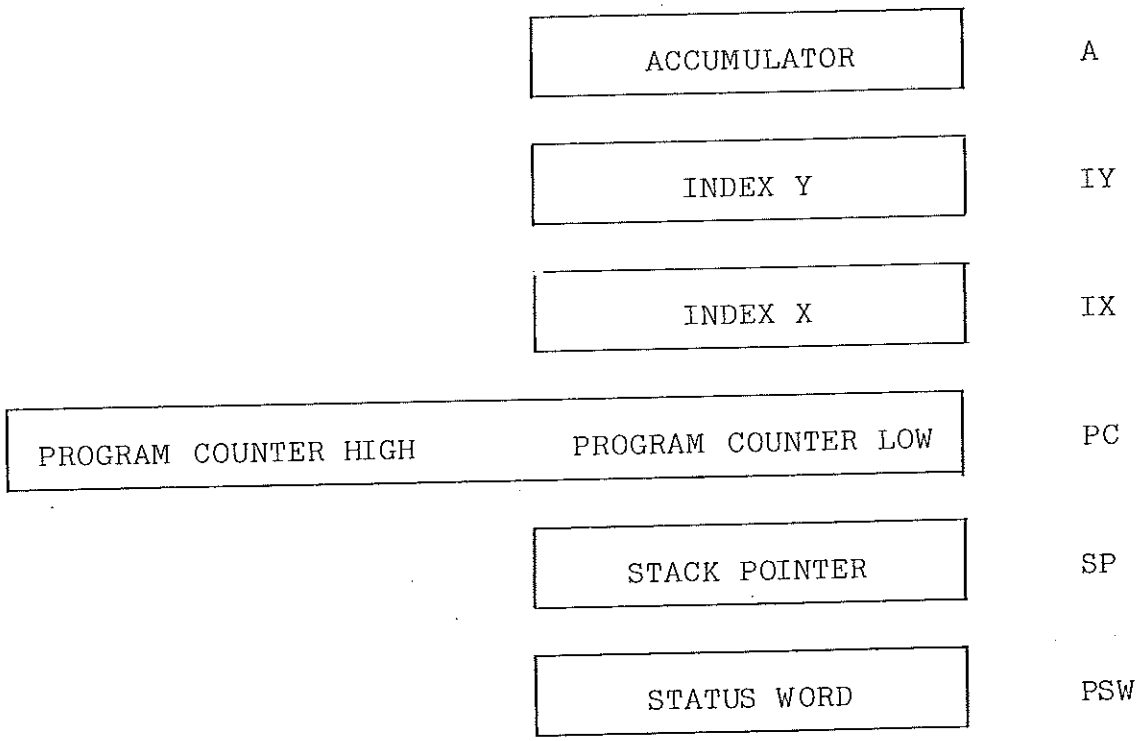
ADDITIONAL			TANEX			microtan 65	
b	a		b	a		b	a
+5	+5	1	+5	+5	1	+5	+5
CLK	$\overline{\text{DMAREQ}}$	2			2	CLK	$\overline{\text{DMAREQ}}$
Ø1	Ø2	3			3	Ø1	Ø2
$\overline{\text{RST}}$	$\overline{\text{I/O}}$	4		$\overline{\text{DMAREQ}}$	4	$\overline{\text{RST}}$	
A1	AØ	5	Ø1	Ø2	5	A1	AØ
A3	A2	6	$\overline{\text{RST}}$	$\overline{\text{I/O}}$	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		$\overline{\text{ABE}}$
$\overline{\text{FB}}$	$\overline{\text{NMI}}$	14	A15	A14	14	$\overline{\text{FB}}$	$\overline{\text{IRQ}}$
$\overline{\text{DMAPOT}}$	$\overline{\text{DMAPIN}}$	15	$\overline{\text{DMAGNT}}$	$\overline{\text{ABE}}$	15		$\overline{\text{NMI}}$
$\overline{\text{IOE}}$	$\overline{\text{RAME}}$	16		$\overline{\text{IRQ}}$	16	$\overline{\text{IOE}}$	$\overline{\text{RAME}}$
$\overline{\text{ROME}}$	R/$\overline{\text{W}}$	17			17	$\overline{\text{ROME}}$	R/$\overline{\text{W}}$
SYNC	$\overline{\text{HB}}$	18	$\overline{\text{IOE}}$	$\overline{\text{RAME}}$	18	SYNC	$\overline{\text{HB}}$
	DBØ	19	$\overline{\text{ROME}}$	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	
	DB7	26	DB5	D5	26	D7	
		27	DB6	D6	27		
		28	DB7	D7	28		
		29			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

CHAPTER 5

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.

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 ∧ M, M_7 → N, M_6 → 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

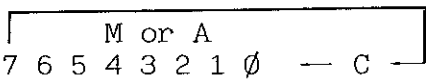
Operation: P ↑ (From Stack)

N Z C I D V

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

ROL

Rotate one bit left (memory or accumulator)

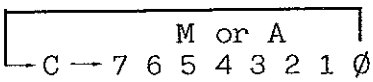
Operation: 

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

Rotate one bit right (memory or accumulator) **ROR**

Operation: 

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

Return from interrupt **RTI**

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

4Ø - 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
5Ø - 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

6Ø - 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
7Ø - 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

CØ - 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
DØ - 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

EØ - 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
FØ - 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