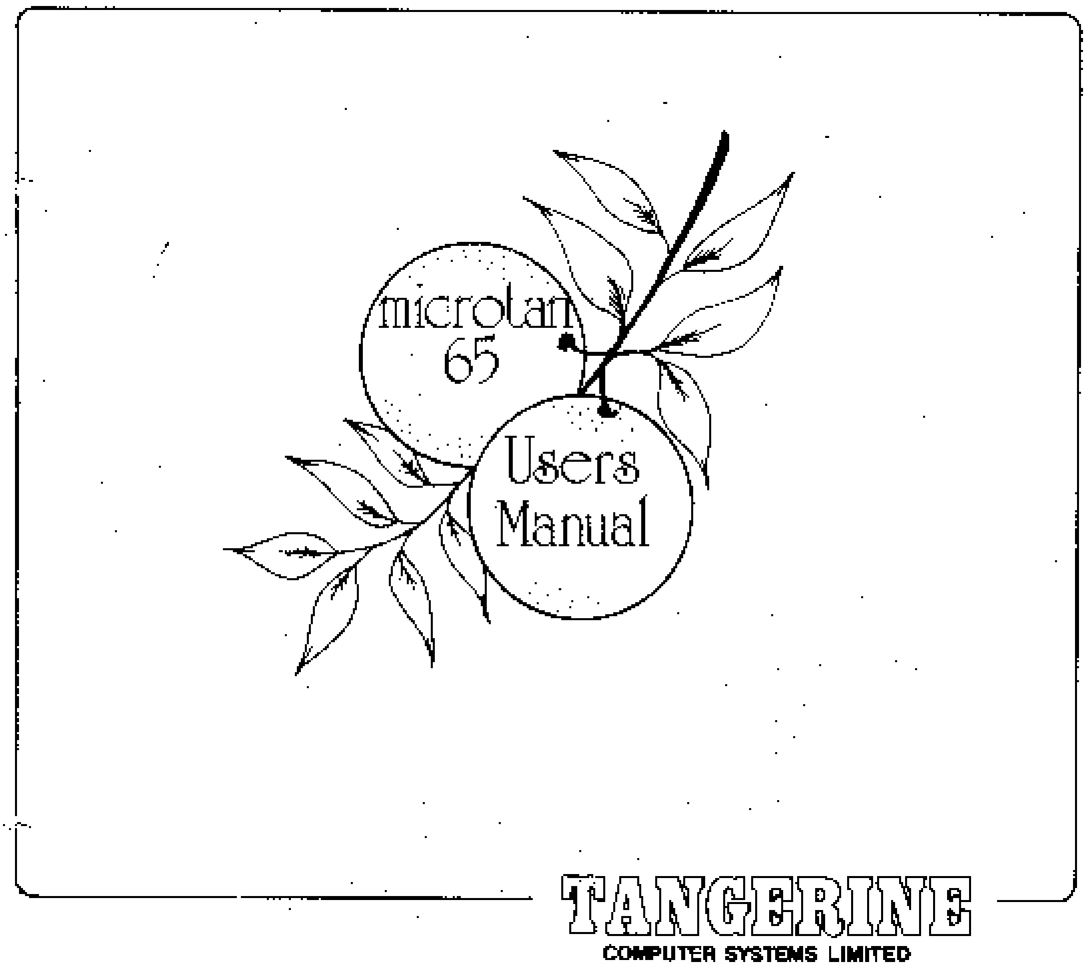


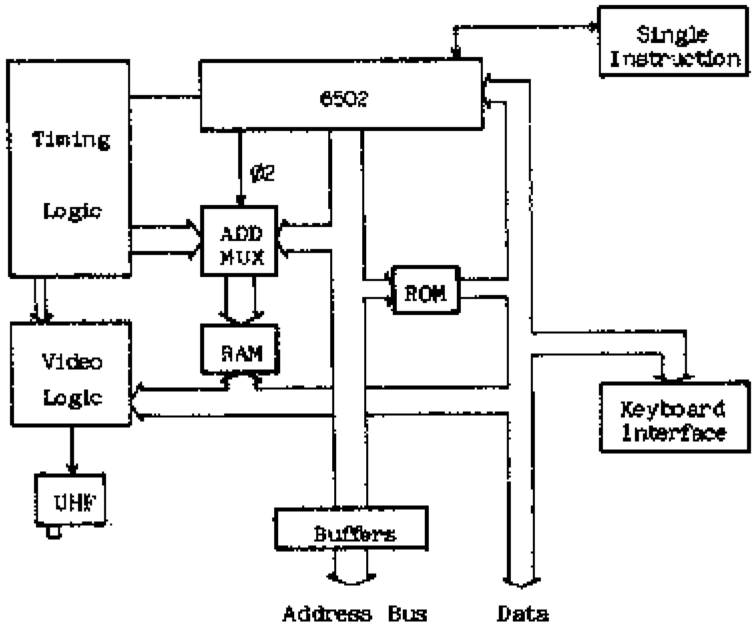
FOREWORD

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 contructional 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 3: The Microtan 65

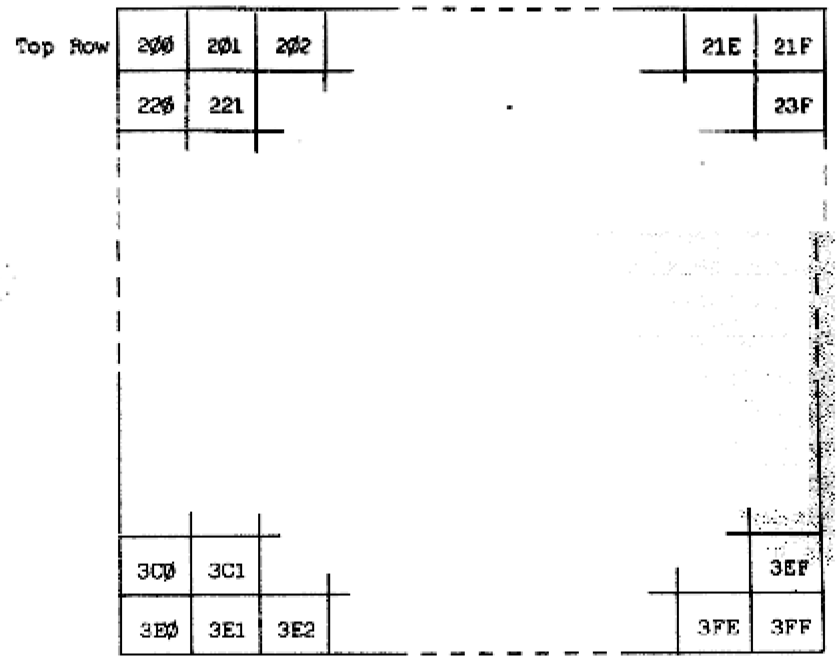
The microtan 65 is an exceptional microcomputer kit and the purpose of this section is to describe its design. It was considered that the majority of microkit users would initially benefit from a machine that was very easy to use rather than one that has parallel I/O, but was very awkward to use. Therefore 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 6502 microprocessor was chosen mainly for its very simple yet elegant hardware structure. Of course, as most readers will know, the 6502 also has a very powerful instruction set and addressing modes.

A fully expanded microtan 65 contains the 6502 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 6502 as memory accesses will only occur on one phase of the clock i.e. when 02 is high. When 02 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 6502) are used as the display memory, that is from hex location 200 to hex 3FF. Location 200 is the top left hand corner character position. Location 201 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 loaction 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 exapansion 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 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.

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 2102 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 BFF0. 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 eight 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 keayboard 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 interrupt 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 is has been set and detected in order that is its ready to register a further keyboard strobe. To reset the keyboard interrupt flag the assembly instruction STA BFF0 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 used 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 6 : TANBUG

The TANBUG monitor program is located in 1K bytes of read only memory (ROM) at the top of the address space i.e. pages 252-255. 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 6502 software 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 200-3FF 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 100-1FF 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 page will not be used as stack storage in the majority of programs. TANBUG requires to use locations 1F0-1FF as stack storage (only 16 locations). The rest of this area is free for user programs. Locations 40-FF are also available as user RAM, the preceeding locations 0-3F being reserved for use by TANBUG. User programs which do not use the stack may therefore be loaded anywhere i.e. the area 40-1EF. 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 conatins 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.

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

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 operations 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 deletes the last character typed. Repeated deletes erase characters back to the beginning of the line.

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 whatever 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 terminators <CR>, <SP>, <ESC> or <LF>.

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 argument applicable to the command being used. The required 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
```

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.

<u>Memory modify/examine command M:</u>

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 100 type M100<CR>
TANBUG then responds on the display with

```
M100,0E,
```

assuming the current contents of the location were 0E.

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>

```
M100,0E,FF
```

location 100 will now contain FF. If however <SP> is typed, the location is re-opened and unmodified.

```
M100,0E,FF
M0100,0E,
```

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>

```
M100,0E,FF
M0101,AB,
```

and using <ESC>

```
M100,0E,FF
M00FF,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>, the responding to the cursor prompt for subsequent data words.

NOTE that locations 1FE and 1FF should not be modified. These are the stack locations 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.

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 location 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 FC00-FFFF) type LFC00,2<CR>.
The display will then be

```
LFC00,2
FC00  A2  FF  9A  E8  86  17  20  B7
FC08  FF  8D  F3  BF  A2 0E  BD DF
█
```

If zero line are requested (i.e. <NUMBER> = 0) 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 100 type G100<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 micro-processors 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 00 say)

```
R
M0015,00,█
```

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 registers locations are as follows.

Location	Function
15	Low order byte of program counter (PCL)
16	High order byte of program counter (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 (executing 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 microprocessors internal registers and the next instruction in the users 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 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
100	69	ADC #1	add 1 to acc
101	01		
102	4C	JMP 100	
103	00		
104	01		

Set the single instruction mode and start the program. The user may wish to initially set the accumulator to 00 by using the M command.

```
S
G100
0102  20 FF  00 00 01
█
```

TANBUG then responds with the characters shown above.

0102 is the address of the next instruction to be executed.
20 is the processor status word value.
FF is the low byt value of the stack pointer. The high byte is always set to 1, the

stack is therefore pointing at location 1FF.
00 is the value of the index X register.
00 is the value of the index Y register.
01 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 G100<CR> again. If the user wishes to continue then type P<CR>. The resulting display is

```
S
G100
0102  20 FF  00 00 01
P
0100  20 FF  00 00 01
█
```

Since the instruction at location 102 was "Jump to 100", 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
G100
0102  20 FF 00 00 01
P
0100  20 FF 00 00 01
P
0102  20 FF 00 00 02
█
```

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
G100
0102  20 FF 00 00 01
P
0100  20 FF 00 00 01
P
0102  20 FF 00 00 02
P4
0102  20 FF 00 00 04
█
```

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 subroutines, 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 0-7 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  C9 01           ADC #1
104  4C 00 01        JMP LOOP
```

Firstly set index X, index Y and the accumulator to 00 using the R command. To set breakpoint 0 at the jump instruction and start the program type B104,0<CR>. The display will then be

```
B104,0
G100
0104  20 FF 01 01 01
```

The jump instruction was reached and the breakpoint re-directed 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>.

```
B104,0
G100
0104  20 FF 01 01 01
P
0104  20 FF 02 02 02
```

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 0. For example, imagine a breakpoint is set as follows; B102,2, and it is subsequently wished to remove it but leave any others unaltered; type B0,2<CR> to remove it.

Caution. The breakpoint system works by replacing the users instruction with a special instruction (BRK) whose opcode is 00. 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 20-2F and are as follows:

Address	Contents
20	PCL B0
21	PCH B0
22	PCL B1
23	PCH B1
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 M20<CR> followed by <LF> gives

```
M20,00,
M0021,01,█
```

This indicates that breakpoint 0 is set to location 100 by taking the contents of location 20 as PCL and of location 21 as PCH. If the breakpoint is set at location 0 then this particular breakpoint is disabled.

<u>Offset command O:</u>

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:-

```
100 E8    LOOP:   INX
101 C8            INY
102 69            ADC #1
103 01            .
                  .
                  .
120 D0            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 O120,100<CR>. The display would be

```
O120,100 = DE
█
```

DE 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.

<u>Copy command C:</u>

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 FC00-FD00 into a block starting at location 200. This may be achieved by typing CFC00,FD00,200<CR>. The display will be

```
CFC00,FD00,200
█
```

As 200 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.

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 beaing 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 withour using reset and losing the breakpoints that have been set.
Notes:

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

b) Interrupts 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.

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.

Display format

TANBUG uses the bottom line of the display for all text operations. Initially the cursor is at the left hand edge of the screen, and moves gradually to the right as a line is filled. When either a carriage return is output, or the bottom line overflows, the display is scrolled (all lines shift up one row) and the bottom line becomes available for more output. Therefore the cursor always remains on the bottom line. 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 POLLKB

Subroutine POLLKB 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 0001) 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 consider the user code:

```
CLI                            enable interrupts
JSR   POLLKB                   poll the keyboard
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 POLLKB sub-routine until a key is pressed.
3. The program exits from POLLKB with the ASCII code for the key pressed in the location label-led ICHAR. The accumulator is loaded with this value.

Notes: Address of subroutine POLLKB is FDFA. Address of ICHAR is 0001. The registers IX, IY and A are corrupted, therefore the user must save and restore their values if necessary.

Subroutine OUTPCR

This subroutine causes the display to scroll up one line by out-putting a carriage return to it. It also reinstates the cursor, which is switched off when a user program is started. This sub-routine should be called in a user program prior to any display input or output to clear the bottom line.

Notes: Address of subroutine OUTPCR is FE73. Registers IX and IY are unaffected. Register A is corrupted and must be saved if required.

Subroutine OPCHR

This subroutine is called to output a character held in the accumulator, to the display. The cursor, obliterated on a user program start, is re-instated. As an example, consider the code

```
LDA#30
JSR OPCHR
LDA#31
JSR OPCHR
```

Since 30 is the ASCII code for the character “0” and 31 is the ASCII code for the character “1”, the result (assuming this is the first call to this subroutine) on the bottom line of the display is:

```
01█
```

Repetitive calls of OPCHR will fill the bottom line of the display with the appropriate characters. When the end of the line is reached, OPCHR scrous the display up one line and then writes characters in the newly vacated bottom line and so on.

Notes: Address of subroutine OPCHR is FE75. Registers IX and IY are unaltered. Register A is corrupted and must be saved if required.

Subroutine HEXPNT

Subroutine HEXPNT takes a binary value from the accumulator and outputs it as two hexadecimal characters on the display. Consider the code

```
PHA                      ; save A on stack
JSR OUTPCR               ; scroll display
PLA                      ; recover A
JSR HEXPNT               ; output A in hex
JSR OUTPCR               ; 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 HEXPNT is FF0B. Register IY is unaltered. Registers IX and A are corrupted and must be saved if required.

Subroutine HEXPCK

This subroutine reads hex characters from the bottom 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 POLLKB is used in conjunction with OPCHR to enter data to the display, then HEXPCK called when a carriage return is encountered. The following user code could be used to do this

```
                JSR   OUTPCR        ; scroll display
      NXTCHR:   JSR   POLLKB        ; wait for character
                LDA   ICHAR         ; put it in A
                CMP#  $0D           ; carriage return ?
                EE Q  GOPACK        ; yes, p ack it
                JSR   OPCHR         ; else store in display
                JM P  NXTCHR        ; get next character
  1)   GOPACK:  LDY#  FF            ; set IY to first char.
  2)            JSR   HEXPCK        ; 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 bottom line corresponds to setting IY tc) 0. The next location corresponds to IY equal to 1 etc.
2. Call HEXPCK. Characters are packed until a character other that 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 = 0. 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 HEXPCK is FF28. Address of HXPKL is is 0013 and HXPKH is 0014. Registers IX, IY and A are all corrupted and must be saved if necessary.

INTERRUPTS

TANBUG uses both 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 break-point 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 INTSLL 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 break-points 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.

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 it 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 break-points can be set.

e) Addresses of RAM locations are; INTFS1=0004, INTFS2=0005, INTFS3=0006, INTSLI=0010, INTSL2=0011, INTSL3=0012.

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, NMIJPL 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=0007, NMIJP1=0008 and NMIJP2=0009.

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 ROM's 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 fill 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.

Example program listing

```
0050  00  00          VDUIND: 0                ;display index
0052  A0  0F           START: LDY# F           ;set Y index
0054  20  73  FE       SCRAG: JSR   OUTPCR     ;carriage return
0057  88                      DEY              ;do E times
0058  10  (arg 1)             BPL   SCRAG
005A  A9  20                  LDA#  20         ;load A ascii space
005C  8D  E0  03              STA   3E0        ;obliterate cursor
005F  A9  00                  LDA#  0          ;set display index
0061  85  50                  STA   VDUIND
0063  A9  02                  LDA#  2
0065  85  51                  STA   VDUIND+1
0067  A0  00            CONT: LDY#  0          ;clear Y index
```

```
0069  A9  2A                 LDA#  2A         ;set ascii
006B  91  50                 STA  (VDUIND),Y
006D  A2  0F                 LDX# F           ; delay loop
006F  A0  FF                 LDY#  FF
0071  88              DECIT: DEY
0072  D0  (arg 2)            BNE   DECIT
0074  CA                     DEX
0075  D0  (arg 3)            BNE   DECIT
0077  18                     CLC              ;inc display index
0078  E6  50                 INC   VDUIND
007A  D0  (arg 4)            BNE   NOMSB
007C  E6  51                 INC   VDUIND+1
007E  A5  51          NOMSB: LDA   VDUIND+1   ;top of display?
0080  C9  03                 CMP#  3
0082  D0  (arg 5)            BNE   CONT       ;no - continue
0084  A5  50                 LDA   VDUIND
0086  C9  FF                 CMP#  FF
0088  D0  (arg 6)            BNE   CONT       ;double prec. cmp.
008A  00                     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 00, 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 0058 and branches to the label SCRAG at location 0054. By typing O58,54<CR> TANBUG prints out the value of arg 1 as FA. This may now be placed in location 0059 using the M command. By repeating the exercise for the other five arguments it will be found that location 0073 should contain FD, 0076 should contain FA, 007B should contain 02, 0083 should contain E3 and 0089 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 0052. 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 L50,8<CR> yielding a display of

```
L50,8
0050   00   00  A0   0F  20   73   FE  88
0058   10   FA  A9   20  8D   E0   03  A9
0060   00   85  50   A9  02   85   51  A0
0068   00   A9  2A   91  50   A2   0F  A0
0070   FF   88  D0   FD  CA   D0   FA  18
0078   E6   50  D0   02  E6   51   A5  51
0080   C9   03  D0   E3  A5   50   C9  FF
0088   D0   DD  00   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,0<CR>.

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

```
006D 31 FF 0F 00 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.

```
006F 21 FF 0F 00 2A
0071 A1 FF 0F FF 2A
0072 A1 FF 0F FE 2A
006F A1 FF 0F 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,D0,
M0073,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> then 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 stlcks 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.

TABLE OF HEX ASCII CODES

00	NUL	
01	Control A	Home
02	Control B	
03	Control C	
04	Control D	
05	Control E	
06	Control F	
07	Control G	Bell
08	Control H	Backspace
09	Control I	Horizontal Tab - Cursor Right
OA	Control J	Line Feed
OB	Control K	
0C	Control L	Page Clear - Form Feed
OD	Control M	Carriage Return
OE	Control N	
OF	Control 0	
10	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 00 - 1F produce special symbols when used in display memory.

20	Space	40	@	60	`
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	9
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
30	0	50	P	70	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

TANBUG LISTING

```
0000                NULL:    00                  ;reserved for bpt use
0001               ICHAR:    00                  ;ascii character
0002               OCHAR:    00                  ;temp char store
0003              VDUIND:    00                  ;display index
0004              INTFS1:    00                  ;fast int link
0005              INTFS2:    00
0006              INTFS3:    00
0007               NMIJP:    00                  ;nmi link
0008              NMIJP1:    00
0009              NMIJP2:    00
000A               ICURS:    00                  ;cursor index
000B              ICURSH:    00                  ;cursor index high
000C              RUNIND:    00                  ; zero if in user
000D              SINGLE:    00                  ; nonzero if using int3
000E              PROCED:    00                  ; proceed count
000F              SIMCOM:    00                  ;simple/complex kb
0010              INTSL1:    00                  ;slow int link
0011              INTSL2:    00
0012              INTSL3:    00
0013               HXPKL:    00                  ;hexpack low byte
0014               HXPKH:    00
0015              PCLBCK:    00                  ;pseudo reg PCL
0016              PCHBCK:    00                  ;PCH
0017              PSWBCK:    00                  ;PSW
0018               SPBCK:    00                  ;SP
0019                XBCK:    00                  ;IX
001A                YBCK:    00                  ;IY
001B                ABCK:    00                  ;A
001C              MODADL:    00                  ;temp store
001D              MODADH:    00
001E                COPL:    00                  ;copy store
001F                COPH:    00
0020               BPTLO:    00                  ;bpt status table
0021                         00
0022                         00
0023               BPTHL:    00
0024                         00
0025                         00
0026                         00
0027                         00
0028                         00
0029                         00
002A                         00
002B                         00
002C                         00
002D                         00
002E                         00
002F                         00
0030              BPTCOD:    00                  ;bpt code store
0031                         00
0032                         00
0033                         00
0034                         00
0035                         00
0036                         00
0037                         00
0038                         00
0039                         00
003A                         00
003B                         00
003C                         00
003D                         00
003E                         00
003F                         00
```

```
0100                  STKBSE:    00                    ;stack base address
0200                  VDUSTT:    00                    ;display scroll labels
0220                  VDUFST:    00
0300                  VDUMID:    00
0320                  VDUTOP:    00
03EO                  LINBOT:    00
BFFO                  SGRAPH:    00                    ;I/O ports
                      KBINCL:    00                    ;alt. tc) SGRAPH
BFF1                    SNMI:    00
BFF2                  KBWRIT:    00
BFF3                  KBREAD:    00
                       STEXT:    00                    ;alt. to KBREAD
F7F7                  INPERR:    00                    ;error exit link
                              ;start of main program init. sequence
                              ;program starts here on a reset
FC00   A2FF            START:    LDX   #FF             ;set stack pointer
FC02   9A                        TXS                   ;to top of the stack
FC03   E8                        INX                   ;enable interrupts in
FC04   8617                      STX   PSWBCK          ;user prog on a go
                              ;clear breakpoint store as their values will be
                              ;indeterminate on power up.
FC06   20B7FF                    JSR   BPTCLR          ;clear breakpoints.
FC09   8DF3BF                    STA   STEXT           ;set text mode.
                              ;now use table to initiate parameters
                              ;note order of table must correspond with the
                              ;order of INTFS1 to ICURSH in ram definitions.
FC0C   A20E                      LDX   #E              ;set index.
FCOE   BDDFFF         SETUP1:    LDA   SETUP, X        ; get value.
FC11   9504                      STA   INTFS1, X       ; store in ram.
FC13   CA                        DEX
FC14   10FO                      BPL   SETUP1          ; do till all done
                              ;
                              ; determine keyboard type and set flag
                              ; note IX=FF.
                              ;
FC16   EO                        INX
FC17   8E90F0          TSFIV:    STX   KBWRIT          ;clear kb write latch.
FCLA   8DFOBF                    STA   KBINCL          ;clear kbint. flag.
FCLD   CA                        DEX
FCLE   8EF2BF                    STX   KBWRIT          ; write to kb lines.
FC21   E8                        INX                   ;reset IX.
FC22   ADF3BF                    LDA   KBREAD          ;read it back.
FC25   1002                      BPL   KPCPLX          ;if plus not set-alpha.
FC27   E60F                      INC   SIMCOM          ;if set then keypad.
FC29   8DFOBF         KPCPLX:    STA   KBINCL          ;clear kbint.
FC2C   BDECFF           TBMS:    LDA   HDR, X          ; print TANBUG.
FC2F   F006                      BEQ   MONTOR
FC31   2075FE                    JSR   OPCHR           ;o/p chars until a 0.
FC34   E8                        INX
FC35   D0F5                      BNE  TBMS
                         ;
                         ; monitor at present part of main program.
                         ;
FC37   D8             MONTOR:    CLD                   ;set binary mode.
FC38   58                        CLI
FC39   20FAE0                    JSR   POLLKB          ; look at kb.
FC3C   A501                      LDA   ICHAR           ;get char.
FC3E   C921                      CMP   #21             ;LT or EQ space-term.
FC40   3006                      BMI   MONCH1          ;else o/p char.
FC42   2075FE         ISTERM:    JSR   OPCHR
FC45   4C37FC                    JMP   MONTOR
FC48   204FFC         MONCH1:    JSR   MONEN2          ;call string process.
FC4B   A90D              RC1:    LDA   #D              ;set up CR.
FC4D   D0F3                      BNE   ISTERM          ;uncond. branch loop.
FC4F   A000           MONEN2:    LDY   #0              ; set IY to zero.
FC51   B10A                      LDA   (ICURS), Y      ;pick up command.
FC53   AA                        TAX                   ; store in X.
```

```
FC54   C0                  INY                 ; peek at next char
FC55   B10A                LDA   (ICURS), Y    ;get it in A.
FC57   1036                BPL   MULTI         ;if -ve was cursor.
FC59   A900                LDA   #0            ;else set A to zero.
FC5B   E053       TRYS:    CPX   #53           ; was it S?
FC5D   D003                BNE   TRYM          ; yes-set GT 0 else skip.
FC5F   860D                STX   SINGLE
FC61   60                  RTS
FC62   E04E       TRYN:    CPX   #4E           ; was it N?
FC64   D003                BNE   TRYP          ; no-try P.
FC66   850D                STA   SINGLE        ;else clr single mode.
FC68   60                  RTS
FC69   E050       TRYP:    CPX   #50           ; P with no arg?
FC6B   D004                BNE   TRYR          ; no-try for R.
FC6D   850E                STA   PROCED        ;else clr P count.
FC6F   F056                BEQ   PROC1         ;uncond. branch proc.
FC71   E052       TRYR:    CPX   #52           ;command R?
FC73   D009                BNE   TRYB          ;no-try for no bpts.
                                               ;note A=0
FC75   851D                STA   MODADH
FC77   A915                LDA   #15           ;else set pseudo reg.
FC79   851C                STA   MODADL
FC7B   4CE5FD              JMP   REOPEN        ;jmp mod memory.
FC7E   E042       TRYB:    CPX   #42           ;remove bpts?
FC80   D004                BNE   ERRQ          ;no-error.
FC82   2OB7FF              JSR   BPTCLR        ;else clear.
FC85   60                  RTS
                         ;
                         ;with no TANEX FFF7 will respond to F7F7.
                         ;with TANEX monitor can be expanded.
                         ;FFF7 jumps back to here.
                         ;
FC86   4CF7F7     ERRQ:    JMP   INPERR
FC89   A93F     RETERR:    LDA   #3F           ;load question mark.
FC8B   2075FE              JSR   OPCHR         ;print it.
FC8E   60        PRERC:    RTS
                         ;if we get here command expects data too.
FC8F   88        MULTI:    DEY                 ;readjust IY.
FC90   8A                  TXA                 ;save cmd on stack.
FC91   48                  PHA
FC92   2028FF              JSR   HEXPCK        ; pack its arg.
FC95   D054                BNE   MOREY         ; more data?
FC97   68                  PLA                 ;else is cmd G.
FC98   50EC                BVC   ERRQ          ;error if no arg.
FC9A   C947                CMP   #47           ;is it a G?
FC9C   D021                BNE   TRYPL         ; no-s kip.
FC9E   A200                LDX   #0
FCAO   860E                STX   PROCED        ;clr proceed count.
FCA2   CA                  DEX                 ;set IX to FF.
                         ;if G stack pointer initialised
                         ;and pseudo registers loaded.
FCA3   9A        GOEND:    TXS                 ;reload it.
FCA4   A514                LDA   HXPKH         ;push PC high.
FCA6   48                  PHA
FCA7   A513                LDA   HXPKL         ;push PC low.
FCA9   48                  PHA
FCAA   A517                LDA   PSWBCK
FCAC   48                  PHA
FCAD   C60C                DEC   RUNIND        ;clr run flag.
FCAF   A920                LDA   #20           ;obliterate cursor.
FCB1   A403                LDY   VDUIND
FCB3   910A                STA   (ICURS), Y
FCB5   A619                LDX   XBCK
FCB7   A41A                LDY   YBCK
FCB9   A51B       SRET:    LDA   ABCK          ; set users A.
FCBB   8DF1BF              STA   SNMI          ;set NMI for next.
FCBE   40                  RTI                 ; get to user prog.
```

```
FCBF   C950        TRYPL:   CMP  #50          ;is cmd P with arg?
FCC1   D011                 BNE  TRYM         ; no-try M.
FCC3   A513                 LDA  HXPKL        ; else set P count.
FCC5   850E                 STA  PROCED
FCC7   A516        PROCL:   LDA  PCHBCK       ;restore users PC.
                  PNOARG:
FCC9   8514                 STA  HXPKH
FCCB   A515                 LDA  PCLBCK
FCCD   8513                 STA  HXPKL
FCCF   A618                 LDX  SPBCK        ;set IX tc) users SP.
FCD1   4CA3FC               JMP  GOEND        ;then back ti) user.
FCD4   C94D         TRYM:   CMP  #4D          ; is it M address?
FCD6   DOAE                 BNE  ERRQ         ; no-error popping S.
FCD8   A92C         EQPT:   LDA  #2C
FCDA   2075FE               JSR  OPCHR
FCDD   A000                 LDY  #0           ; pick up value
FCDF   B113                 LDA  (HXPKL), Y
FCE1   20OBFF               JSR  HEXPNT       ;and print it.
FCE4   68                   PLA               ; pop stack return.
FCE5   68                   PLA
FCE6   A92C                 LDA  #2C          ;load comma.
FCE8   4C42FC               JMP  ISTERM       ;back to monitor.
FCEB   E02C        MOREY:   CPX  #2C          ;was term a comma?
FCED   F004                 BEQ  GETPT2       ;yes-continue.
FCEF   68         LINKPH:   PLA               ;else pull command
FCFO   4C86FC      LINKR:   JMP  ERRQ         ;and give error.
FCF3   A513       GETPT2:   LDA  HXPKL        ; no-store previous
FCF5   851C                 STA  MODADL       ;in MODADL&MODADH.
FCF7   A514                 LDA  HXPKH
FCF9   851D                 STA  MODADH
FCFB   2028FF               JSR  HEXPCK       ; pack next value.
FCFE   D07C                 BNE  MOREY1       ; not curs-more yet.
FD00   68                   PLA               ;else pull command.
FD01   50ED                 BVC  LINKR        ; no arg? - error.
FD03   C94C                 CMP  #4C          ;is it L?
FD05   F042                 BEQ  LISTIT       ; yes-list it.
FD07   C94F                 CMP  #4F          ; is it offset O?
FD09   D027                 BNE  TRYBPT       ; no-skip.
FD0B   A513                 LDA  HXPKL        ;else get branch dest.
FD0D   38                   SEC
FD0E   E902                 SBC  #2           ;adj for branch code.
FD10   B002                 BCS  NOTOPO
FD12   C614                 DEC  HXPKH
FD14   38         NOTOPO:   SEC
FD15   E51C                 SBC  MODADL       ; subtract source.
FD17   AA                   TAX               ; hold result in IX.
FD18   A514                 LDA  HXPKH        ; subtract high byte.
FD1A   E51D                 SBC  MODADH
FD1C   A8                   TAY               ; store in IY.
FD1D   8A                   TXA               ; get low byte.
FD1E   3005                 BMI  RNGNG        ; if -ve branch.
FD20   98                   TYA               ; if +ve look at high.
FD21   D0CD                 BNE  LINKR        ;not O-error.
FD23   F003                 BEQ  PNTITO       ;if ok continue.
FD25   C8          RNGNG:   INY               ;if -ve high is FF.
FD26   D0C8                 BNE  LINKR
FD28   A93D       PNTITO:   LDA  #3D          ;ok-print equals
FD2A   2075FE               JSR  OPCHR
FD2D   8A                   TXA
FD2E   200BFF               JSR  HEXPNT       ;and the value.
FD31   60                   RTS
FD32   C942       TRYBPT:   CMP  #42          ;is it B?
FD34   DOBA        LINK1:   BNE  LINKR        ;no-error.
FD36   2898                 LDA  HXPKL        ;max bpt  code is 7.
FD38   30B6                 BMI  LINKR
FD3A   C908                 CMP  #8           ;if greater-error.
FD3C   10B2                 BPL  LINKR
```

```
FD3E   0A                          ASL   A              ; double A.
FD3F   AA                          TAX                  ;set X for mode X.
FD40   A51C                        LDA   MODADL         ;store bpt address.
FD42   9520                        STA   BPTLO,  X
FD44   A51D                        LDA   MODADH
FD46   9521                        STA   BPTH, X
FD48   60              JPRTRN:     RTS
FD49   2073FE          LISTIT:     JSR   OUTPCR         ;o/p carriage return.
                         NXLI:
FD4C   A000                        LDY   #0             ;set IY to zero.
FD4E   A51D                        LDA   MODADH         ; print address.
FD50   200BFF                      JSR   HEXPNT
FD53   A51C                        LDA   MODADL
FD55   200BFF          NXLIST:     JSR   HEXPNT         ;o/p low byte.
FD58   A920                        LDA   #20            ;load a space.
FD5A   2075F0                      JSR   OPDCHR         ;o/p 8 mem locations.
FD5D   B11C                        LDA   (MODADL), Y    ;get character.
FD5F   C8                          INY
FD60   C009                        CPY   #9             ; is IY GT 7?
FD62   30F1                        BMI   NXLIST         ; no-more yet.
FD64   C613                        DEC   HXPKL          ;dec line count.
                                ;
                                ;note that requesting 0 lines gives 256!
                                ;
FD66   F0E0                        BEQ   JPRTRN         ;O? RTS via CR.
FD68   88               DELX1:     DEY
FD69   D0FD                        BNE   DELXL
FD6B   CA                          DEX
FD6C   D0FA                        BNE   DELXL
                                      ;
                                      ;now adjust the address.
                                      ;
FD6E   A51C                        LDA   MODADL         ;get low byte.
FD70   18                          CLC
FD71   6908                        ADC   #8             ;add 8.
FD73   851C                        STA   MODADL         ;store it.
FD75   90D2                        BCC   NXLI           ;more if C set.
FD77   E61D                        INC   MODADH         ;inc high byte.
FD79   4C49FD                      JMP   NXLI
                                ;
                                ; if we get here there  is a third parameter.'
                                ;
FD7C   E02C            MOREYL:     CPX   #2C            ;term a comma?
FD7E   F018                        BEQ   TERMOK         ;no-error..
FD80   4CEFFC          ERJUM2:     JMP   LINKPH
FD83   A513            TERMOK:     LDA   HXPKL          ;else store parameter.
FD85   851E                        STA   COPL
FD87   A514                        LDA   HXPKH
FD89   851F                        STA   COPH           ;then pack new para.
FD8B   2028FF                      JSR   HEXPCK
FD8E   D0F0                        BNE   ERJUM2         ; error not term I d.
FD90   68                          PLA                  ;is command M?
FD91   C94D                        CMP   #4D
FD93   F027                        BEQ   MEM100         ; yes-mod memory.
FD95   C943                        CMP   #43            ; is it C?
FD97   D09B                        BNE   LINK1          ;no-error.
                                ;
                                ;if copy command, start address is held in MODADL.
                                ;end address in COPL.
                                ;destination address in HXPKL.
                                ;
FD99   5099                        BVC LINK1                  ;arg? error if not.
FD9B   A000                        LDY   #0
FD9D   B11C             NXCOP:     LDA   (MODADL), Y    ;get instruction.
FD9F   9113                        STA   (HXPKL), Y     ; store it.
FDAL   A51F                        LDA   COPH           ;is it done?
FDA3   28E8                        CMP   MODADH
```

```
FDA5   D006                    BNE   ICMCOP
FDA7   A51E                    LDA   COPL
FDA9   C51C                    CMP   MODADL
FDAB   F05B                    BEQ   ENDLS
FDAD   E61C          ICMCOP:   INC   MODADL          ;increment source.
FDAF   D002                    BNE   NOHIH1
FDBL   E61D                    INC   MODADH
FDB3   E613          NOHIH1:   INC   HXPKL           ;increment dest.
FDB5   DOE6                    BNE   NXCOP
FDB7   E614                    INC   HXPKH
FDB9   D0E2                    BNE   HXCOP
                            ;destination can not roll over top of memory.
FDBB   60                      RTS                   ;return if end.
FDBC   A601          MEM100:   LDX   ICHAR           ; get term char in IX.
FDBE   E020                    CPX   #20             ;was it a space?
FDCO   F023                    BEQ   REOPEN          ; yes-reopen.
FDC2   5006                    BVC   NOENT           ; branch if nothing.
FDC4   A513                    LDA   HXPKL           ;else enter data.
FDC6   A000                    LDY   #0
FDC8   911C                    STA   (MODADL), Y
FDCA   E00A           NOENT:   CPX   #A              ;was LF typed?
FDCC   F011                    BEQ   WASLF           ; yes-process it.
FDCE   E018                    CPX   #lB,            ; was it ESC?
FDDO   D036                    BNE   ENDLS           ;no-return.
FDD2   C61C           ESCIT:   DEC   MODADL          ; dec mem mod address
FDD4   A51C                    LDA   MODADL
FDD6   C9FF                    CMP   #FF
FDD8   D00B                    BNE   REOPEN
FDDA   C61D                    DEC   MODADH
FDDC   4CE5FD                  JMP   REOPEN
FDDF   E61C           WASLF:   INC   MODADL          ; inc mem address.
FDE1   D002                    BNE   REOPEN          ;also high byte.
FDE3   E61D                    INC   MODADH
FDE5   2073FE        REOPEN:   JSR   OUTPQR          ;o/p carriage return.
FDE8   A94D                    LDA   #4D
FDEA   2075FE                  JSR   OPCHR           ;print M.
FDED   A51D                    LDA   MODADH          ; print address.
FDEF   20OBFF                  JSR   HEXPNT
FDF2   A51C                    LDA   MODADL
FDF4   20OBFF                  JSR   HEXPNT
FDF7   4C4FFC                  JMP   MONEN2          ;go to cmd bit.
                            ;
                            ;POLLKB   subroutine.
                            ; note all addresses are page zero.
                            ;
FDFA   A900          POLLKB:   LDA   #0
FDFC   48                      PHA                   ;push 0-shift ind.
FDFD   8501           PLKB1:   STA   ICHAR           ;set ICHAR to zero.
FDFF   C50F                    CMP   SIMCOM          ;is it,keypad?
FE01   D006                    BNE   SIMPLE
FE03   C501           WAIT1:   CMP   ICHAR           ;else wait for int.
FE05   F0FC                    BEQ   WAIT1
FE07   68            PLKEND:   PLA                   ;pop indicatcor.
FE08   60             ENDLS:   RTS
                            ;simple keypad routine.
FE09   A90F          SIMPLE:   LDA   #F              ;enable all kb lines.
FEOB   8DF2BF                  STA   KBWRIT
FEOE   ADF3BF                  LDA   KBREAD          ;look at kb lines.
FEll   DOF6                    BNE   SIMPLE          ;key down?-wait till up.
FE13   A240                    LDX   #40             ; debounce it.
FE15   88            DEBOUN:   DEY
FE16   D0E8                    BNE DEBOUN
FE18   CA                      DEX
FE19   D0FA                    BNE   DEBOUN
                            ;now poll the keypad properly.
                            ;note IY  is zero.
FELB   AOFF            PLK1:   LDY   #FF            ;set up Y index.
```

```
FELD   68                       PLA                ; peep at shift ind.
FELE   48                       PHA
FELF   F002                     BEQ   NOSHIF       ;if shift set mod IY.
FE21   A013                     LDY   #13
FE23   A208          NOSHIF:    LDX   #8           ;set IX-kb drive.
FE25   8EF2BF          PLK2:    STX   KBWRIT       ;drive kb lines.
FE28   ADF3BF                   LDA   KBREAD       ;get result in A.
FE2B   DOOC                     BNE   ACHAR        ;not 0-a char-skip.
FE2D   C8                       INY                ;else adjust IY.
FE2E   C8                       INY
FE2F   C8                       INY
FE30   C8                       INY
FE31   C8                       INY
FE32   8A                       TXA                ;shift IX right.
FE33   4A                       LSR   A
FE34   AA                       TAX
FE35   F0E4                     BEQ   PLK1         ;if zero repeat.
FE37   D0EC                     BNE   PLK2         ;else next line.
                             ;if we get here a key has been pressed.
FE39   C8             ACHAR:    INY
FE3A   4A                       LSR   A            ;which key of 5?
FE3B   90FC                     BCC   ACHAR        ; C set'? thats key.
FE3D   B94BFE                   LDA   CHRTBL, Y    ; get asc ii e quiv.
FE40   8501                     STA   ICHAR        ; put it in IACHR.
FE42   D0C3                     BNE   PLKEND       ; if zero-shift.
FE44   68                       PLA                ; pull shift.
FE45   49FF                     EOR   #FF          ;change shift state.
FE47   48                       PHA                ; push shift.
FE48   4CO9FE                   JMP   SIMPLE       ; continue.
                             ;character look up table for ascii equivalent.
FE4B   333742        CHRTBL:
FE4E   463F32
FE51   364145
FE54   0D
FE55   313539
FE58   440A
FE5A   303438
FE5D   4300
FE5F   2C524C
FE62   4E3F32
FE65   434953
FE68   20314F
FE6B   lB477F
FE6E   303450
FE71   4D00
                             ;output character routine.
                             ;on entry character is stored in A.
                             ;IX and IY are saved.
FE73   A90D          OUTPCR:    LDA   #D            ;load CR.
FE75   8502           OPCHR:    STA   OCHAR         ;save the character.
FE77   8A                       TXA                 ;save IX and IY.
FE78   48                       PHA
FE79   98                       TYA
FE7A   48                       PHA
FE7B   A403                     LDY   VDUIND        ;get curs position.
FE7D   A920                     LDA   #20           ;obliterate cursor.
FE7F   910A                     STA   (ICURS), Y
FE81   A602                     LDX   OCHAR         ;get char in IX.
FE83   E07F                     CPX   #7F           ; is it delete?
FE85   D010                     BNE   TRYCR         ;no-skip.
FE87   88                       DEY                 ;else dec vdu index.
FE88   1002                     BPL   DODEL         ;if neg set zero.
FE8A   AOM           ZERCUR:    LDY   #0
FE8C   A9FF           DODEL:    LDA   #FF           ; print cursor.
FE8E   910A                     STA   (ICURS), Y
FE90   8403                     STY   VDUIND        ;save index.
FE92   68                       PLA                 ;restore regs and rtn.
```

```
FE93   40                        TAY
FE94   68                        PLA
FE95   AA                        TAX
FE96   60                        RTS
FE97   E00D         TRYCR:       CPX   #D              ; is it a CR?
FE99   F008                      BEQ   DOCR            ;yes-process it.
FE9B   8A                        TXA                   ;else o/p character.
FE9C   910A                      STA   (ICURS), Y
FEGE   C8                        INY                   ;inc vdu index.
FE9F   C020                      CPY   #20             ; end of line?
FEAL   30E9                      BMI   DODEL           ; no-tidy up-exit.
FEA3   A200          DOCR:       LDX   #0              ;else scroll.
FEA5   BD2002      LOWBLK:       LDA   VDUFST, X       ; do in two blocks.
FEA8   9D0002                    STA   VDUSTT, X
FEAB   E8                        INX
FEAC   D0F7                      BNE   LOWBLK
FEAE   BD2003       HIBLK:       LDA   VDUTOP, X
FEB1   9D0003                    STA   VDUMID, X
FEB4   E8                        INX
FEB5   E0E0                      CPX   #E0             ; see if done.
FEB7   D0F5                      BNE   HIBLK
FEB9   A920                      LDA   #20             ;fill line spaces.
FEBB   A8                        TAY
FEBC   88           MORSP:       DEY                   ; set IX as count.
FEBD   910A                      STA   (ICURS), Y
FEBF   D0FB                      BNE   MORSP           ;do for whole line
FEC1   F0C7                      BEQ   ZERCUR          ;when done tidy up.
                              ;keyboard interrupt subroutine.
                              ;note this is entered via jump instruction
                              ;stored in RAM location labelled INTFS1
                              ;so that user can access interrupts quickly.
                              ;a reset will always initiahse INTFS1.
FEC3   48           KBINT:       PHA                   ;save accumulator.
FEC4   D8                        CLD                   ;set binary mode.
FEC5   8A                        TXA                   ;save IX.
FEC6   48                        PHA
FEC7   BA                        TSX                   ; get SP in IX.
FECB   E8                        INX                   ; IX to add old PSW.
FEC9   E8                        INX
FECA   E8                        INX
FECB   BD0001                    LDA   STKBSE, X       ;get the PSW.
FECE   2910                      AND   #10             ;was it a break?
FED0   D01E                      BNE   BRKP            ;yes-process it.
FED2   68                        PLA                   ;else pull IX.
FED3   AA                        TAX                   ;restore and continue.
FED4   ADF3BF                    LDA   KBREAD          ;read keyboard.
FED7   3004                      BMI   WASKB           ;if -ve w as kb.
FED9   68            USER:       PLA                   ;else restore A.
FEDA   4C1000                    JMP   INTSL1
                              ;note that INTSL1 normally holds an RTI
                              ;unless the user has modified it.
FEDD   297F         WASKB:       AND   #7F             ; mask top bit.
FEDF   8501                      STA   ICHAR           ; put it in ICHAR.
FEE1   8DF0BF                    STA   KBINCL          ; clr kb int flip-flop.
FEE4   C91B                      CMP   #1B             ; was it ESC?
FEE6   D0F1                      BNE   USER            ; no-normal return.
FEE8   A50C                      LDA   RUNIND          ;user prog running?
FEEA   D0ED                      BNE   USER            ; no-normal.
FEEC   850E                      STA   PROCED          ;else clr proc count.
FEEE   F012                      BEQ   ACTBP           ;uncond br to break.

                              ;break processing code-break was encountered.
                              ; note the user should not set a break at own break
                              ;or in this interrupt routine else crashes.
                              ;now adjust PC to return to instr must subtract 3.
FEF0   E8            BRKP:       INX                   ; address PCL.
FEF1   38                        SEC                   ;set C.
```

```
FEF2   BD0001                   LDA   STKBSE, X        ;get PCL.
FEF5   E902                     SBC   #2               ;subtract 3.
FEF7   9D0001                   STA   STKBSE, X        ;put it back.
FEFA   B004                     BCS   NOROLL           ;C set? NOHI byte.
FEFC   E8                       INX                    ;else address PCH.
FEFD   DE0001                   DEC   STKBSE, X        ;dec PCH.
FF00   68             NOROLL:   PLA                    ;pull IX.
FF01   AA                       TAX                    ;restore it.
FF02   68              ACTBP:   PLA                    ;pull accumulator.
FF03   851B                     STA   ABCK             ;back it up.
FF05   20C1FF                   JSR   BPTREM           ;restore user code.
FF08   4C75FF          NOROL:   JMP   NMNT1            ;service break.
                             ;HEXPNT   takes a hex value stored in accumulator
                             ;ie a two character number and displays it.
                             ;registers are not saved.  IX is corrupted.
FF0B   48             HEXPNT:   PHA                    ;save value of char.
FF0C   A201                     LDX   #1               ;load IX with 1.
FF0E   4A                       LSR   A                ; get top part
FF0F   4A                       LSR   A                ;by multiple shifts.
FF10   4A                       ISR   A
FF11   4A                       LSR   A
FF12   18               PNT2:   CLC
FF13   6930                     ADC   #30              ;add hex 30.
FF15   C93A                     CMP   #3A              ;more than 9?
FF17   3003                     BMI   PNT1             ;no- print it.
FF19   18                       CLC
FF1A   6907                     ADC   #7
FF1C   2075FE           PNT1:   JSR   OPCHR            ;print it.
FF1F   CA                       DEX
FF20   1001                     BPL   MORL             ;-ve?-end else low bit.
FF22   60                       RTS
FF23   68               MORL:   PLA                    ;recover low bits.
FF24   29OF                     AND   #F               ;clr unwanted bits.
FF26   10EA                     BPL   PNT2             ;uncond branch.

                             ;HEXPCK uses values in TXTP to store hex values
                             ;of characters indexed by IY     from display RAM.
                             ;on exit, caused by any non-hex character,these
                             ;contain a two byte number.  HEXENT contains
                             ;the number of characters entered.
                             ;registers are not saved.
                             ;on exit IY points tc) the last character.
                             ;the zero flag is clear if the terminating
                             ;character was the cursor, else set.
                             ;the overflow flag is set if this subroutine
                             ;encountered some hex data.
FE28   A900           HEXPCK:   LDA   #0               ;clr accumulator.
FF2A   48                       PHA                    ;push as PSW.
FF2B   8513                     STA   HXPKL            ;clr parameters.
FF2D   8514                     STA   HXPKH
FF2F   CB               NXHX:   INY
FF30   B10A                     LDA   (ICURS), Y       ;get character.
FF32   AA                       TAX                    ; store in IX
FF33   38                       SEC                    ;subtract hex 30.
FF34   E930                     SBC   #30              ; to give 0 to 9.
FF36   3010                     BMI   ENDTS
FF38   C90A              HX1:   CMP   #A               ; is it 0 to 9?
FF3A   3010                     BMI   HX2              ;yes-pack it.
FF3C   38                       SEC                    ;else sub hex 11.
FF3D   E911                     SBC   #11
FF3F   3007                     BMI  ENDTS             ;goto carry setup.
FF41   18                HX3:   CLC                    ;else add 9.
FF42   690A                     ADC   #A
FF44   C910                     CMP   #10              ;more than 15? exit.
FF46   3004                     BMI   HX2
FF48   28              ENDTS:   PLP                    ;deal with V flag.
FF49   E0FF                     CPX   #FF              ; cursor.
```

```
FF4B   60                        RTS
                              ;note character is in IX on exit
                              ;return does not affect zero flag.
FF4C   A204            HX2:      LDX   #4           ; its hex save it.
FF4E   0614            HX5:      ASL   HXPKH        ; shift left 4 times.
FF50   0613                      ASL   HXPKL        ; add new one.
FF52   9002                      BCC   HX4
FF54   E614                      INC   HXPKH        ; replicate.
FF56   CA              HX4:      DEX                ; dec IX until zero.
FF57   D0F5                      BNE   HXS
FF59   18                        CLC                ; add in char.
FF5A   6513                      ADC   HXPKL
FF5C   8513                      STA   HXPKL        ; store it.
FF5E   68                        PLA                ; get pseudo PSW.
FF5F   A940                      LDA   #40          ; set V bit.
FF61   48                        PHA                ; and push it.
FF62   DOCB                      BNE   NXHX         ; do next.
                              ;non-maskable interrupt routine for single instr.
FF64   851B           NMNT:      STA   ABCK         ;save accumulator.
FF66   D8                        CLD                ;set binary mode.
FF67   A50D                      LDA   SINGLE       ;test single inst.
FF69   D00A                      BNE   NMNT1
FF6B   8A                        TXA                ; if not save IX.
FF6C   48                        PHA
FF6D   20D0FF                    JSR   BPSET        ;set breakpoints.
FF70   68                        PLA                ;restore IX.
FF71   AA                        TAX
FF72   A51B                      LDA   ABCK         ;restore accumulator.
FF74   40                        RTI
FF75   A50E           NMNT1:     LDA   PROCED       ;get proc count.
FF77   F005                      BEQ   ZERBCK       ;if zero break.
FF79   C60E                      DEC   PROCED       ;else dec count.
FF7B   4CB9FC                    JMP   SRET         ;rts via single test.
FF7E   E60C          ZERBCK:     INC   RUNIND       ;set not running.
FF80   68                        PLA                ; pop bpt PSW to acc.
FF81   8517                      STA   PSWBCK       ;back it up.
FF83   68                        PLA
FF84   8515                      STA   PCLBCK       ; ditto PCL.
FF86   68                        PLA
FF87   8516                      STA   PCHBCK       ; ditto PCH.
FF89   8619                      STX   XBCK
FF8B   841A                      STY   YBCK
FF8D   BA                        TSX                ;operators  SP.
FF8E   8618                      STX   SPBCK
FF90   2073FE         PSEUD:     JSR   OUTPCR
FF93   A516                      LDA   PCHBCK
FF95   200BFF                    JSR   HEXPNT
FF98   A515                      LDA   PCLBCK
FF9A   200BFF                    JSR   HEXPNT
FF9D   A000                      LDY   #0           ;clear IY.
FF9F   A920           PSNX:      LDA   #20          ;print two spaces.
FFAL   48                        PHA
FFA2   2075FE                    JSR   OPCHR
FFA5   68                        PLA
FFA6   2075FE                    JSR   OPCHR
FFA9   B91700                    LDA   PSWBCK, Y
FFAC   200BFF                    JSR   HEXPNT
FFAF   C8                        INY                ;do all registers.
FFBO   C005                      CPY   #5
FFB2   30EB                      BMI   PSNX
FFB4   4C4BFC                    JMP   RC1
                              ;breakpoint clearing routine uses IX and A.
FFB7   A21F          BPTCLR:     LDX   #1F          ; set count for 8 bpt3.
FFB9   A900                      LDA   #0
FFBB   9520          BPTCL1:     STA   BPTLO, X     ; clear a location.
FFBD   CA                        DEX                ; dec count.
FFBE   10FB                      BPL   BPTCL1       ; do for all locations.
```

```
FFC0   60                          RTS
                                ;this subroutine removes breakpoints by
                                ;writing back the users original code.
                                ;IX and IY are used.
FFC1   8A              BPTREM:     TXA
FFC2   48                          PHA
FFC3   A20E                        LDX   #E              ; set for 8 bpts.
FFC5   B530            BPTRML:     LDA   BPTCOD, X       ;load old instruction.
FFC7   8120                        STA   (BPTLO, X)      ;write to prog mem.
FFC9   CA                          DEX
FFCA   CA                          DEX                   ;decrement it.
FFCB   10                          BPL   BPTRM1          ;do for all bpts.
FFCD   68                          PLA
FFCE   AA                          TAX
FFCF   60                          RTS

                                ;subroutine BPSET looks at the entered breakpoint
                                ;addresses, stores the present instruction
                                ;and writes brk to program memory.
FFD0   A20E             BPSET:     LDX   #E              ; set for 8 bpts.
FFD2   A120              BPS1:     LDA   (BPTLO, X)      ;get users instr.
FFD4   9530                        STA   BPTCOD, X       ; store it.
FFD6   A900                        LDA   #0              ;set brk..
FFD8   8120                        STA   (BPTLO, X)
FFDA   CA                          DEX
FFDB   CA                          DEX
FFDC   10F4                        BPL   BPS1            ;do till done.
FFDE   60                          RTS

                                ;setup table for initial values.
                                ;order is same as RAM locations.
FFDF   4CC3FE           SETUP:     ;starts at INTFS1, addr 0004
FFE2   4C64FF                   ;NMI JMP
FFE5   E00301
FFE8   000000
FFEB   40                       ;RTI for INTSL1
                                ;header message "TANBUG"
FFEC   0D                 HDR:
FFED   54414E                                            ;TAN
FFF0   425547                                            ;BUG
FFF3   0D00
FFF7   4C89FC                      JMP RETERR
FFFA   070000
FFFD   FC0400
```

CHAPTER 7: SOME GAMES PROGRAMS

LIFE PROGRAM

Start Address: B0

No options required.

Life is a program which simulates a colony of living organisms which can grow, move and die out. "Life" is run in two stages.

1. Seeding the cells. On starting the program, the display is cleared and the cursor is placed in the middle of the screen. Four single letter commands are available to move the cursor about the lower half of the display.

   L moves it to the left.

R moves it to the right.
D moves it down.
M moves it up.

A fifth command, A, whenever typed replaces the cursor with an asterisk, which then remains on the display when the cursor is moved again. By using these five commands any pattern of asterisks can be built up on the lower half of the display.

2. Evolution. Typing "G" causes the program to jump from the seeding mode and into the evolution mode. Every two seconds the display is scanned and a new "generation" of cells produced from the old according to the following rules.

   a) If a cell space is surrounded by exactly three other asterisks, this cell will contain an asterisk in the next generation.
   b) If an asterisk is surrounded by two or three others, it will remain as an asterisk in the next generation.
   c) All- other asterisks will become spaces.

Eventually a stable pattern will occur which will either be an immobile patbern, a blinking pattern or a blank screen. At this point the monitor can be entered by issuing a reset or, if using an alphanumeric keyboard, by depressing ESC and a new game started. Note that asterisks placed on the bottom line of the display do not change; they act as a permanent seeding set.

```
                          ;    life program
                          ;
0040                     LIN1:
0060                     LIN2:
0080                     LIN3:
00A0                    VDULO:                          ;vdu position.
00A1                    VDUHI:
00A2                     YIND:                          ;y inddex.
00B0   A010                       LDY #10               ;clear screen.
00B2   2073FE            NXSR:    JSR   OUTPCR
00B5   88                         DEY
00B6   D0FA                       BNE   NXSR
00B8   A920                       LDA   #20             ;obliterate cursor.
00BA   8DE003                     STA   CUR
00BD   A000                       LDY   #0              ;put curs in mid vdu.
00BF   84A0                       STY   VDULO
00C1   A003                       LDY   #3
00C3   84A1                       STY   VDUHI
00C5   A00F                       LDY   #F
00C7   84A2                NXC:   STY   YIND
00C9   A9FF                       LDA   #FF
00CB   91A0              PLKIT:   STA   (VDULO), Y
00CD   20FAFD                     JSR   POLLKB
00D0   A4A2                       LDY   YIND            ;char=cursor?
00D2   B1A0                       LDA   (VDULO), Y
00D4   C9FF                       CMP   #FF
00D6   D008                       BNE   NODL            ;no-skip.
00D8   A920                       LDA   #20             ;else delete.
00DA   91A0                       STA   (VDULO), Y
00DC   4CE000                     JMP   NODL            ;skip for expansion.
00DF   00
00E0   A601              NODL:    LDX   ICHAR           ;pick up input char.
00E2   E04C                       CPX   #4C             ;L for left?
00E4   D006                       BNE   NOL             ;no-skip.
00E6   88                         DEY
00E7   D0DE                       BNE   NXC             ;look for next command.
00E9   C8                INGO:    INY                   ;else inc it.
```

```
00EA   D0DB                BNE   NXC               ;unconditional branch.
00EC   E052         NOL:   CPX   #52               ;R for right?
00EE   D006                BNE   NOLL              ;no-skip.
00F0   C01E                CPY   #lE               ;else shift right.
00F2   F0D3                BEQ   NXC                 ;next char.
00F4   D0F3                BNE   INGO
00F6   E041        NOLL:   CPX   #41               ;A for asterisk?
00F8   D004                BNE   NOL2              ;no-skip.
00FA   A92A                LDA   #2A               ;else output a *
00FC   D0CD                BNE   PLKIT             ;unconditional branch.
00FE   E044        NOL2:   CPX   #44               ;D for down?
0100   D00A                BNE   NOL3              ;no-skip.
0102   18                  CLC                     ;else inc line pointer.
0103   A5A0                LDA   VDULO
0105   6920                ADC   #20
0107   85A0                STA   VDULO
0109   4CC700              JMP   NXC
010C   E04D        NOL3:   CPX   #4D               ;M for up?
010E   D00A                BNE   NOL4              ;no-skip.
0110   38                  SEC                     ;else sub a line.
0111   A5A0                LDA   VDULO
0113   E920                SBC   #20
0115   85A0                STA   VDULO
0117   4CC700      LKAG:   JMP   NXC
011A   E047        NOLA:   CPX   #47               ;G for go?
011C   D0F9                BNE   LKAG              ;no-skip.
011E   A000          GO:   LDY   #0
0120   84A0                STY   VDULO
0122   A002                LDY   #2
0124   84A1                STY   VDUHI
0126   A900       GOTW0:   LDA   #0                ;determine number
0128   85A2                STA   YIND              ;of cell neighbours.
012A   A920                LDA   #20
012C   A207                LDX   #7
012E   BC3C01        NXX:   LDY   INDEX, X
0131   D1A0                CMP   (VDULO), Y
0133   F002                BEQ   DCT
0135   E6A2                INC   YIND
0137   CA           DCT:   DEX
0138   10F4                BPL   NXX               ;YIND holds number.
013A   3008                BMI   CNT
013C   000102     INDEX:
013F   202240
0142   4142
0144   18           CNT:   CLC                     ;clear carry.
0145   A5A2                LDA   YIND
0147   C903                CMP   #3                ;if 3 go set carry.
0149   F00C                BEQ   CST
014B   C902                CMP   #2                ;if not 2 clear carry.
014D   D058                BNE   CNS1
014F   A021                LDY   #21               ;if 2 look at this.
0151   B1A0                LDA   (VDULO), Y
0153   C920                CMP   #20
0155   F050                BEQ   CNS1
0157   38           CST:   SEC                     ;carry set if in next.
0158   A250         CNS:   LDX   #50               ;store it shift prev.
015A   3640        NXPL:   ROL   LIN1, X
015C   CA                  DEX
015D   10FB                BPL   NXPL
015F   E6A0                INC   VDULO             ;do for whole page.
0161   D002                BNE   NHI
0163   E6A1                INC   VDUHI
0165   A5A1         NHI:   LDA   VDUHI
0167   C903                CMP   #3
0169   D006                BNE   TPJP
016B   A5A0                LDA   VDULO
016D   C9BE                CMP   #BE
```

```
016F   F03A                      BEQ   PATCH1
0171   4C2601          TPJP:     JMP   GOTWO
0174   38             DNUPD:     SEC                     ;empty store to vdu.
0175   E901                      SBC   #1
0177   85A0                      STA   VDULO
0179   A5A1                      LDA   VDUHI
017B   E900                      SBC   #0
017D   85A1                      STA   VDUHI
017F   4CB001                    JMP   PATCH2
0182   EA                        NOP
0183   A220           DLYIT:     LDX   #20                ;delay.
0185   88              DLDY:     DEY
0186   D0FD                      BNE   DLDY
0188   CA                        DEX
0189   D0FA                      BNE   DLDY
018B   4C1E01                    JMP   GO
018E   A200           MOROP:     LDX   #0
0190   A050                      LDY   #50
0192   7640           NXROR:     ROR   LIN1, X
0194   E8                        INX
0195   88                        DEY
0196   10FA                      BPL   NXROR
0198   A920                      LDA   #20
019A   9002                      BCC   CHROP
019C   A92A                      LDA   #2A
019E   A000           CHROP:     LDY   #0
01A0   91A0                      STA   (VDULO), Y
01A2   A5A0                      LDA   VDULO
01A4   4C7401                    JMP   DNUPD
01A7   18              CNS1:     CLC
01A8   4C5801                    JMP   CNS
01AB   A9DF          PATCHL:     LDA   #DF
01AD   4C7401                    JMP   DNUPD
01B0   C902          PATCH2:     CMP   #2
01B2   D0DA                      BNE   MOROP
01B4   A5A0                      LDA   VDULO
01B6   C920                      CMP   #20
01B8   D0D4                      BNE   MOROP
01BA   F0C7                      BEQ   DLYIT
```

SHOOTDOWN

Start Address: 0045

No options required.

A row of "aircraft" fly across the screen from left to right. A "gun" is positioned at the bottom of the display. Typing any key causes the gun to shoot at the aircraft. Number of shots and number of hits are recorded. The object of the game is to shoot all the aircraft down with the minimum number of shots. When all aircraft are shot down, the game is automatically re-started after a delay.

```
                        ;    shootdown
                        ;
0040                  LINDL:                      ;aircraft line delay.
0041                  GUNDL:                      ;gun delay.
0042                  SHOOT:                      ;shoot flag.
0043                 GNPOSL:                      ;gun position.
0044                 GNPOSH:
0045   A00F           START:     LDY   #F         ;clear screen.
0047   2073FE        INITCR:     JSR   OUTCR
004A   88                        DEY
004B   IOFA                      BPL   INITCR
004D   A920                      LDA   #20
```

```
004F   8DE003               STA   SCRTP
0052   A01A                 LDA   #lA            ;write aircraft to line 2.
0054   A9FF                 LDA   #FF
0056   994002       NXAC:   STA   AC,  Y
0059   88                   DEY
005A   88                   DEY
005B   88                   DEY
005C   10F8                 BPL   NXAC           ;set up delay.
005E   8540                 STA   LINDL
0060   A930                 LDA   #30            ;zero the scale.
0062   8D2002               STA   VDUT
0065   8D2102               STA   VDUT1
0068   A001       RESGUN:   LDY   #1             ;reset gun delay.
006A   8441                 STY   GUNDL
006C   88                   DEY                  ;clear ICHAR.
006D   8401                 STY   ICHAR
006F   A50F                 LDA   SIMCOM         ;if full kb skip.
0071   F00A                 BEQ  FULLK
0073   A90F                 LDA   #F             ;if keypad wait no key.
0075   8DF2BF               STA   KBWRIT
0078   ADF3BF         WT:   LDA   KBREAD
007B   D0FB                 BNE   WT
007D   8442        FULLK:   STY   SHOOT          ;clear shoot flag.
007F   AOEF                 LDY   #EF            ;set gun position.
0081   8443                 STY   GNPOSL
0083   A003                 LDY   #3
0085   8444                 STY   GNPOSH
0087   A000        ACDLY:   LDY   #0             ;actually put it out.
0089   A95E                 LDA   #5E
008B   9143                 STA   (GNPOSL), Y
008D   A0FF        MSDLY:   LDY   #FF            ; master delay.
008F   88           GMDL:   DEY
0090   D0FD                 BNE   GMDL
0092   C640                 DEC   LINDL          ;dec aircraft delay.
0094   D015                 BNE   ROTEND         ;not O go test shoot.
0096   A960                 LDA   #60            ;else reset delay.
0098   4C3101               JMP   PATCH1         ;rotate ne.
009B   48           BACK:   PHA
009C   A01E                 LDY   #1E
009E   B94002      ROTRT:   LDA   SRTLIN, Y
00A1   994102               STA   SRPLUS, Y
00A4   88                   DEY
00A5   1OF7                 BPL   ROTRT
00A7   68                   PLA
00A8   8D4002               STA   LINBT
00AB   A542       ROTEND:   LDA   SHOOT          ;shoot in progress?
00AD   D023                 BNE   GOSHOT         ;yes-deal with it.
00AF   A50F                 LDA   SIMCOM         ;else keypad?
00B1   D006                 BNE   SIMKBL         ;yes-branch.
00B3   A501                 LDA   ICHAR          ;else full kb hit?
00B5   F0D8                 BEQ   GMDL           ;no-wait.
00B7   D005                 BNE   INSHT          ;else go and shoot.
00B9   ADF3BF     SIMKB1:   LDA   KBREAD         ;look at kb.
00BC   F0D1                 BEQ   GMDL
00BE   E642        INSHT:   INC   SHOOT          ;hit-inc indicator.
00C0   EE2102               INC   VDUT1          ;adjust shots fired.
00C3   AD2102               LDA   VDUT1
00C6   C93A                 CMP   #3A
00C8   3008                 BMI   GOSHOT
00CA   A930                 LDA   #30
00CC   8D2102               STA   VDUT1
00CF   EE2002               INC   VDUT
00D2   C641       GOSHOT:   DEC   GUNDL          ; dec gun delay.
00D4   D0B7                 BNE   MSDLY          ; not O then more delay.
00D6   A91F                 LDA   #lF            ;else reset gun delay.
00D8   8541                 STA   GUNDL
00DA   A000                 LDY   #0             ;obliterate gun.
```

```
00DC   4C0000                JMP   NXLL            ;skip for expansion.
00DF   00
00E0   A920          NXLL:   LDA   #20
00E2   9143                  STA   (GNPOSL), Y
00E4   38                    SEC
00E5   A543                  LDA   GNPOSL          ;adjust gun position.
00E7   E920                  SBC   #20
00E9   8543                  STA   GNPOSL
00EB   B002                  BCS   NOHIH
00ED   C644                  DEC   GNPOSH
00EF   A902         NOHIH:   LDA   #2              ;gun in aircraft rank?
00F1   C544                  CMP   GNPOSH
00F3   D092                  BNE   ACDLY           ;no-continue.
00F5   A94F                  LDA   #4F
00F7   C543                  CMP   GNPOSL
00F9   D08C                  BNE   ACDLY           ;else we are.
00FB   AD4F02                LDA   GNLN            ;have we scored?
00FE   C9FF                  CMP   #FF
0100   F003                  BEQ   AHIT            ;a hit-deal with it.
0102   4C6800                JMP   RESGUN          ;else reset gun.
0105   A920          AHIT:   LDA   #20             ;obliterate aircraft.
0107   8D4FO2                STA   GNLN
010A   AD2502                LDA   ACSCO           ;setup the score.
010D   C920                  CMP   #20
010F   D008                  BNE   INCSC
0111   A931                  LDA   #31
0113   8D2502                STA   ACSCO
0116   4C6800        RSJP:   JMP   RESGUN
0119   EE2502       INCSC:   INC   ACSCO
011C   C938                  CMP   #38             ; game over test.
011E   D0F6                  BNE   RSJP
0120   A909                  LDA   #9              ; delay, then restart.
0122   8543                  STA   GNPOSL
0124   CA             DLY:   DEX
0125   D0FD                  BNE   DLY
0127   88                    DEY
0128   D0FA                  BNE   DLY
012A   C643                  DEC   GNPOSL
012C   D0F6                  BNE   DLY
012E   4C4500                JMP   START
0131   8540        PATCH1:   STA   LINDL
0133   AD5F02                LDA   ENDLN
0136   4C9B00                JMP   BACK
```

GRAPHICS SUBROUTINE

Calling Address:	0044 (not relocatable)
Variables:	XCOORD 0040
	YCOORD 0041
Requirements:	Graphics option

The graphics subroutine uses the VDU as a 64 x 64 pixel graphics display. When called it picks up the values of X and Y co-ordinates (set up by the calling program) and turns the relevant pixel on. The bottom left hand side of the screen is point (0,0).

Note that this subroutine does not turn any pixel off - it is therefore necessary to zero the screen before using this subroutine to output a graph.

Notes on use of graphics subroutine.

1. Initially with the monitor in "TANBUG", all unused areas of the screen are filled with spaces (HEX 20). Before attempting to draw a graph using the subroutine shown, the screen must be filled with hex 0's with graphics mode set.
2. If using TANBUG and graphics mode, test mode is not reset on entering TANBUG. Care must therefore be taken that the program resets text mode before exit.
3. The OPCHR subroutine should not be used if graphics mode is in use anywhere on the screen.
4. It is possible to display both alphanumerics and graphics characters simultaneously, e.g. an annotated graph. The procedure is as follows:-
   a) set graphics mode,
   b) fill screen with hexadecimal O's,
   c) draw axes and graph,
   d) set text mode,
   e) write text.

Note that <u>any</u> use of the OPCHR subroutine, e.g. by the monitor, corrupts the graphics display.

To demonstrate this subroutine this example program also contains a main program which utilises it. The program repeatedly draws X and Y axes and the function y = ½ x.
Start address: FD

```
                          ;    Graphics Subroutine
                          ;          Listing.
0040   00              XCOORD:
0041   00              YCOORD:
0042   00               VDULO:
0043   00               VDUHI:
0044   ADF0BF           GRAPH:    LDA   GON              ;set graphics mode.
0047   A93F                       LDA   #3F              ;check coord size.
0049   C540                       CMP   XCOORD
004B   302E                       BMI   EXIT
004D   EA                         NOP
004E   38                         SEC
004F   E541                       SBC   YCOORD
0051   EA                         NOP
0052   48                         PHA
0053   2903                       AND   #3               ;mask to 2 lsbs.
0055   AA                         TAX                    ;store in IX.
0056   68                         PLA
0057   EA                         NOP
0058   293C                       AND   #3C              ;mask two lsbs.
005A   0A                         ASL   A                ;shift 1 3 times.
005B   0A                         ASL   A
005C   0A                         ASL   A
005D   8542                       STA   VDULO
005F   A902                       LDA   92               ;set up VDULO and
0061   8543                       STA   VDUHI            ;VDUHI with y axis
0063   9002                       BCC   NOINC            ;address.
0065   E643                       INC   VDUHI
0067   A540             NOINC:    LDA   XCOORD           ;get xcoordinate.
0069   4A                         LSR   A
006A   A8                         TAY                    ;adjust, store in IY
006B   A901                       LDA   #1               ;determine which
006D   9001                       BCC   NOSF2            ;bit to set in the
006F   0A                         ASL   A                ;graphics word
0070   CA               NOSF2:    DEX                    ;held in acc.
0071   3004                       BMI   ENDOP
```

```
0073   0A                        ASL   A
0074   0A                        ASL   A
0075   D0F9                      BNE   NOSF2            ;jmp to repeat.
0077   1142             ENDOP:   ORA   (VDULO), Y       ;or with present.
0079   9142                      STA   (VDULO), Y       ;write it back out.
007B   60                EXIT:   RTS
                          ;
                          ;this program repeatedly draws X and Y axes
                          ;and the function y=1/2x to demonstrate the
                          ;operation of the graphics subroutine and the
                          ;graphics hardware.
                          ;first save regs, o/p graph subroutine and delay.
                          ;
0084   48                SAVOP:   PHA
0085   98                         TYA
0086   48                         PHA
0087   8A                         TXA
0088   48                         PHA                   ;save registers.
0089   A220                       LDX   #20             ;set delays.
008B   A000                       LDY   #0
008D   88                 DECM:   DEY
008E   D0FD                       BNE   DECM
0090   CA                         DEX
0091   D0FA                       BNE   DECM
0093   204400                     JSR   GRAPH           ;output char.
0096   68                         PLA                   ;restore.
0097   AA                         TAX
0098   68                         PLA
0099   A8                         TAY
009A   68                         PLA
009B   60                         RTS
                          ;
                          ;main program.
                          ;
OOFD   ADFOBF                     LDA   GON             ;set graphics mode.
0100   A900              GTEST:   LDA   #0              ;clear display.
0102   AA                         TAX
0103   9D0002             NXSTO:   STA   VDUL, X
0106   9D0003                      STA   VDUM, X
0109   CA                         DEX
010A   D0F7                       BNE   NXSTO
010C   EA                         NOP
010D   A03F                       LDY   #3F             ;set IY decimal 63.
010F   A900                       LDA   #0              ;clear x co-ord.
0111   8540                       STA   XCOORD
0113   8441                NEX:   STY   YCOORD          ;load IY to y co-ord.
0115   208400                     JSR   SAVOP           ;o/p char with delay.
0118   88                         DEY                   ;do for y axis    ne.
0119   10F8                       BPL   NEX
011B   A03F                       LDY   #3F             ;now do for x axis.
011D   8541                       STA   YCOORD
011F   8440               NEX1:   STY   XCOORD
0121   208400                     JSR   SAVOP
0124   88                         DEY
0125   10F8                       BPL   NEX1
0127   A200                       LDX   #0              ;now draw graph.
0129   8640                NXT:   STX   XCOORD
012B   8A                         TXA
012C   4A                         LSR   A
012D   8541                       STA   YCOORD
012F   208400                     JSR   SAVOP
0132   E8                         INX
0133   E040                       CPX   #40
0135   D0F2                       BNE   NXT
0137   4C0001                     JMP   GTEST
0200                      VDUL:
```

```
0300                    VDUM:
```

01feb2026 ncrook. Proof-read, untested.