CHAPTER 6

TANBUG
V2

Notes for Users Familiar with TANBUG V1

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

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

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

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

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

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

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

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

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

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 operation SHIFT P yields P on the display. SHIFT SHIFT P yields 8 on the display. Other special purpose keys on the keypad are RST, which issues a reset to the Microtan, and DEL which delete the last character typed. Repeated deletes erase characters back to the beginning of the line.

Keyboard Protocol

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

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

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

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

1 = See Appendix.

All commands are of the form

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

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

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

Starting the Monitor TANBUG:

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

```
TANBUG
█
```

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

Memory Modify/Examine Command M:

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

M <ADDR> <TERM>

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

```
M1ØØ,ØE,█
```

assuming the current contents of the location were ØE.

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

```
M1ØØ,ØE,FF
█
```

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

```
M1ØØ,ØE,FF
MØ1ØØ,ØE,█
```

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

```
M1ØØ,ØE,FF
MØ1Ø1,AB,█
```

and using <ESC>

```
M1ØØ,ØE,
MØØFF,56,█
```

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

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

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

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

This would be entered via TANBUG as

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

List Command L:

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

L <ADDR>, <NUMBER> <TERM>

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

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

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

Go Command G:

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

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

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

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

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

Register Modify/examine Command R:

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

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

```
R
MØØ15,ØØ,█
```

Now proceed as for the M command.

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

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

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

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

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

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

Single Instruction Mode S:

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

Normal Mode Command N:

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

Proceed Command P:

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

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

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

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

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

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

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

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

```
S
G1ØØ
Ø1Ø2 2Ø FF ØØ Ø1
```

TANBUG then responds with the characters shown above.

Ø1Ø2	is the address of the next instruction to be executed.
2Ø	is the processor status word value.
FF	is the low byte value of the stack pointer. The high byte is always set to 1, the stack is therefore pointing at location 1FF.
ØØ	is the value of the index X register.
ØØ	is the value of the index Y register.
Ø1	is the value of the accumulator. It is a 1 as 1 has been added to the accumulator and it is assumed that the user cleared the accumulator before starting the program.

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

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

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

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

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

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

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

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

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

<u>Breakpoint Command B;</u>

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

The format of the breakpoint command is

```
B <ADDR>, <NUMBER>  <CR>
```

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

```
1ØØ  E8              LOOP: INX
1Ø1  C8                    INY
1Ø2  69  Ø1                ADC#1
1Ø4  4C  ØØ  Ø1            JMP  LOOP
```

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

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

The jump instruction was reached and the breakpoint 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>.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
M2Ø,ØØ
MØØ21,Ø1,█
```

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

Offset Command O:

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

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

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

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

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

```
Ø12Ø,1ØØ = DE
█
```

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

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

Copy Command C:

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

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

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

```
CFCØØ,FDØØ,2ØØ
█
```

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

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

Breakpoints and the ESC Key

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

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

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

```
0100  20  FF  01  01  01
█
```

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

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

2 = See Appendix

Notes:

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

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

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

Input/Output Control

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

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

bit 7							bit Ø
BAS WARM	SCN DIS	SER O/P ON	PAR O/P ON	EXT O/P ON	DC FLAG	SPECIAL PRINT MODE	SER I/P ON

In more detail:

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

BIT 1 SPECIAL PRINT	- set to Ø by monitor on initialisation. When Ø, a line of output to a parallel device is terminated by LF only, while to a serial device CR LF is output. This may be set to 1 by the user so that, if his printers require them, the serial interface terminates with LF only while the parallel interface gives CRLF.
BIT 2 DC FLAG	- used by the monitor to denote output control codes for printers on/off.
BIT 3 EXT OUTPUT ON	- zeroed by the monitor on initialisation. If set to 1 by the user, can be linked to a user output driver subroutine.
BIT 4 PAR O/P ON	- zeroed by the monitor on initialisation. If set to 1 by any of the various control commands, initialises the printer and directs output to it.
BIT 5 SER O/P ON	- as Bit 4, but for serial printer interface on Tanex.
BIT 6 SCN DIS	- set to Ø by initialisation. If set to a 1 by control commands, inhibits output to the screen.
BIT 7 BAS WARM	- Used by the monitor for BASIC warm start.

Important Notes:

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

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

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

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

Parallel Printer

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

Table 1 shows the pin connections for the interface cable.

Fig. 1

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

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

Fig. 2 - Initialisation

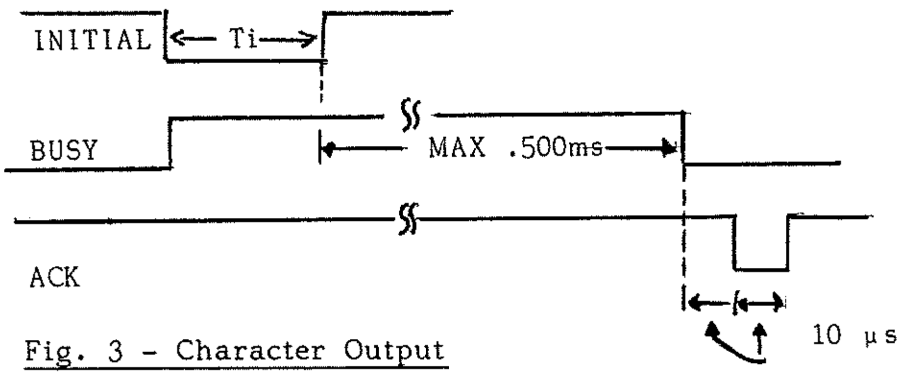
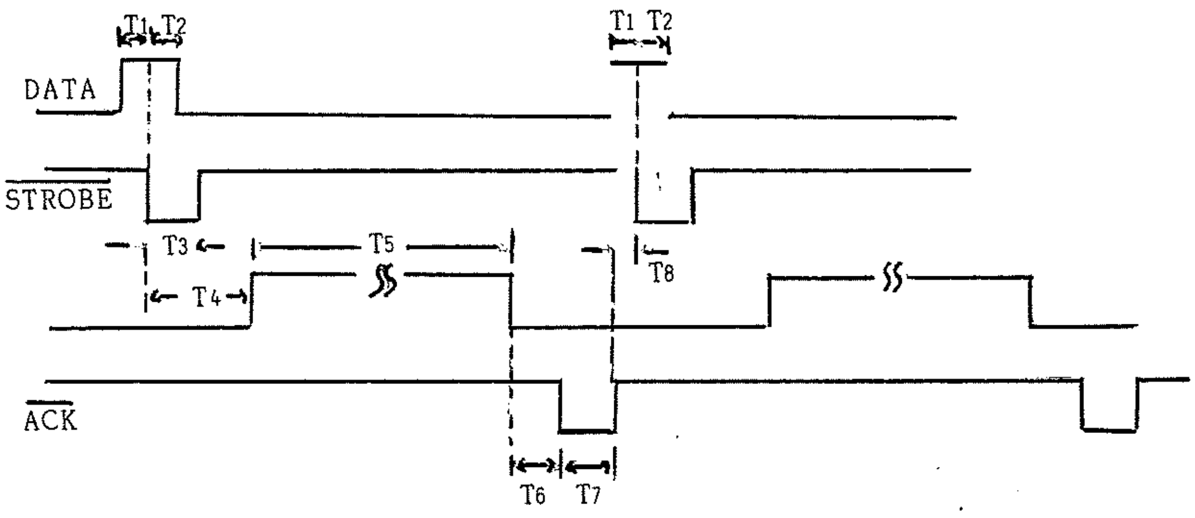


Fig. 3 - Character Output



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

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

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

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

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

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

Fig 4 - Signal Descriptions

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

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

Controlling the Parallel Printer

The parallel printer can be controlled in several ways:

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

Program entry

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

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

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

Example, to turn the printer on:

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

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

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

d) The printer is turned off by a RESET.

Printer Errors

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

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

Using Other Types of Parallel Printers

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

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

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

IMPORTANT NOTE

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

Serial Printer

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

Fig. 5 shows the printer connections.

Fig. 5 - Serial Printer Connections

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

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

Operation

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

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

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

Output can be controlled by the following methods:

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

3 = See Appendix

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

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

See example under parallel printer.

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

d) The printer is turned off by a reset.

Printer Errors

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

IMPORTANT NOTE

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

Using Other Serial Printers

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

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

Screen Output Suppression

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

Screen output is turned off and on as follows:

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

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

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

d) The screen is turned on by a reset.

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

4 = See Appendix

External Output Devices

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

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


          RTS
```

Note that the subroutine must be in memory <u>before</u> the external drive is enabled.

The external printer may be turned off by the code

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

It is also turned off by a reset.

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

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

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

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

which will cause one DC1 to be printed.

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

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

Using an External Keyboard

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

Fig. 6 shows the serial input connections.

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

To enable the input, $\overline{\text{DCD}}$ and $\overline{\text{DSR}}$ must be tied to ground. This can be done on the input plug so that serial input is only recognised when a keyboard is plugged in.

Method of Operation

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

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

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

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

Press return key on teletype
Press return key on keypad

Input will now be accepted from the teletype.

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

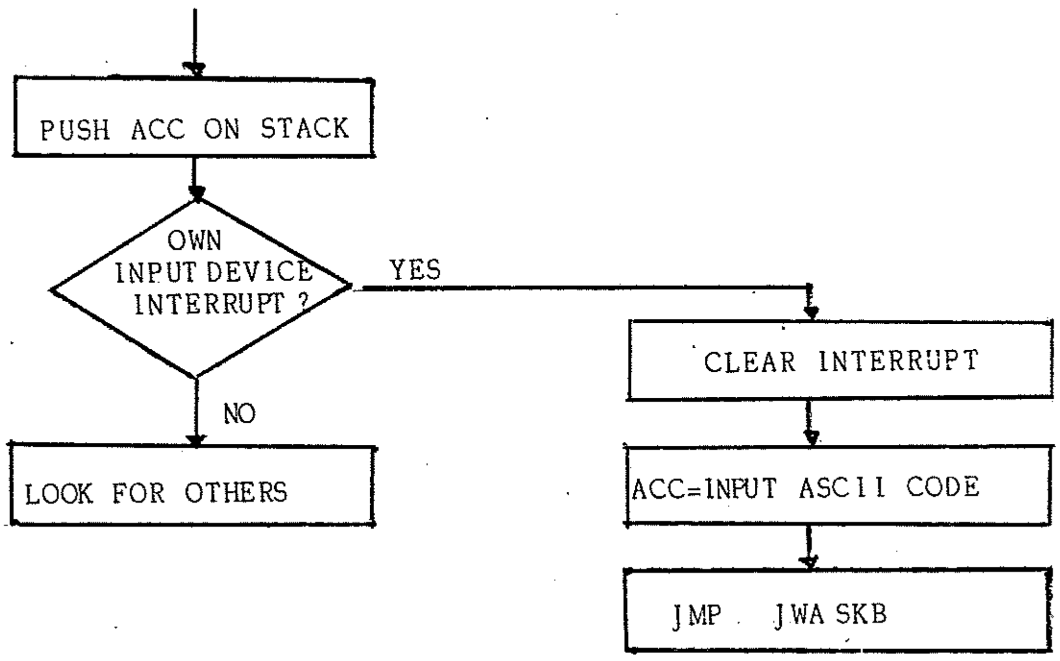
NOTE that if you are using serial input/output with no display, you must use the ↑V command to turn off the printer while reading cassette tapes, otherwise due to the slow printing speed, Filename errors will occur.

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

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

External Input to Monitor

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



User Subroutines

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

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

Display Format

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

Subroutine JPLKB

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

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

The sequence of operations here are

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

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

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

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

Subroutine OUTRET

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

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

Subroutine OUTALL

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

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

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

```
Ø1█
```

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

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

Subroutine JHXPN

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

```
.
.
.
```

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

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

```
        2C
        █
```

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

<u>Subroutine JHXPK</u>

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

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

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

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

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

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

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

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

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

Subroutine JCURSF

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

Subroutine JCURSN

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

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

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

JCURSN is located at F826. No registers are corrupted.

Subroutine RETMS

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

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

Subroutine RETMON

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

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

Subroutine JMNRW

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

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

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

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

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

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

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

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

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

```
    LDA #11             ; set up for read and write
    STA 40              ; to page 1
    LDA #FF
    STA 41              ; data to be written
    LDA #0
    STA 43
    LDA #60             ; set up address
    STA 44
    JSR JMNRW           ; execute
    LDA 42              ; read data loaded to acc.
```

Address of JMNRW is F811. No registers are corrupted.

<u>Subroutine JMNRW1</u>

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

Address of JMNRW1 is F814.

<u>Subroutine PRPUP</u>

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

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

PRPUP is located at F8ØØ. The accumulator is corrupted.

Subroutine OUTPAR

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

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

Subroutine OUTSER

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

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

Subroutine OUTSCR

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

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

Interrupts

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

The Maskable Interrupt

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

a) The program jumps to INTFS1 in RAM.

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

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

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

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

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

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

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

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

A number of things should be noted when using interrupts:

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

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

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

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

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

The Non-maskable Interrupt

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

Error Linking

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

a) The program jumps to F7F7.

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

c) A question mark is printed.

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

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

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

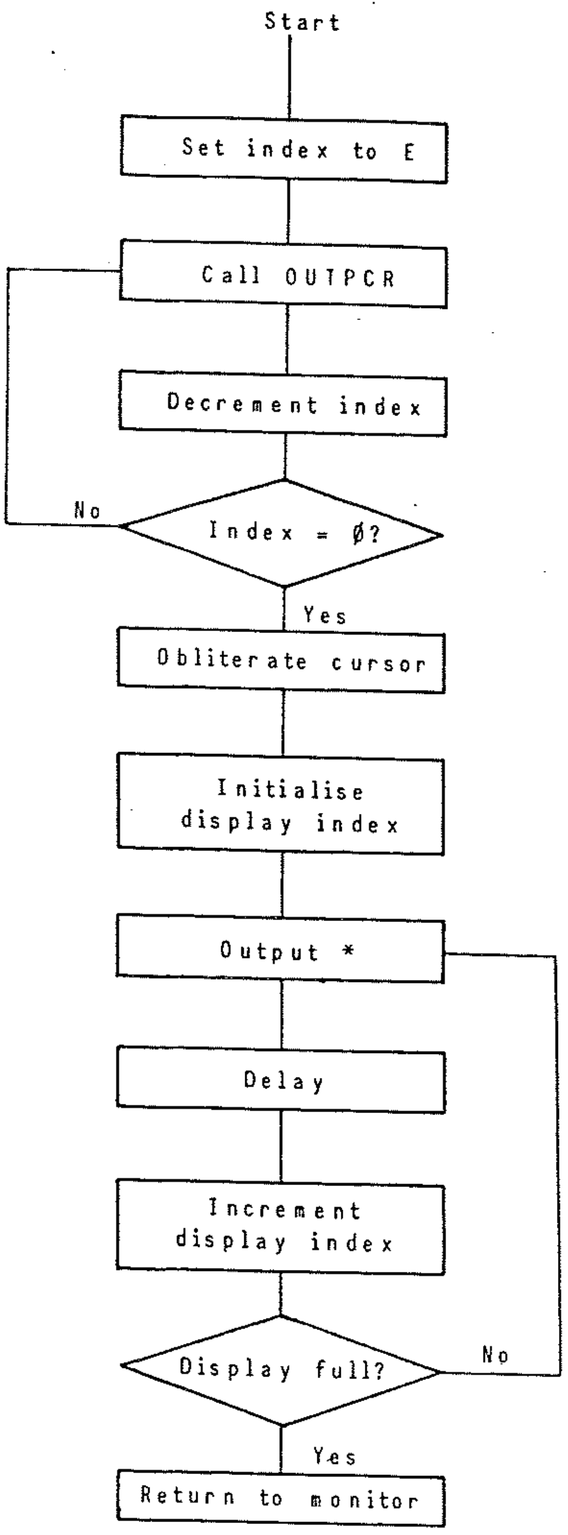
2) The instruction JMP FFF7 returns to the monitor, giving a question mark on the display.

Example of TANBUG's Use

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

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

The flowchart and program listing now follows.


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

Example program listing

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

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

Once the program is entered the branch offsets are calculated. The first is arg 1 which has an opcode address of ØØ58 and branches to the label SCRAG at location ØØ54. By typing 058,54 CR TANBUG prints out the value of arg 1 as FA. This may now be placed in location ØØ59 using the M command. By repeating the exercise for the other five arguments, it will be found that location ØØ73 should contain FD, ØØ76 should contain FA, ØØ7B should contain Ø2, ØØ83 should contain E3 and ØØ89 should contain DD.

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

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

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

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

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

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

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

```
ØØ6D  31  FF  ØF  ØØ  2A
█
```

is displayed. Control is now back in the monitor.

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

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

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

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

```
M72,DØ,
MØØ73,FB,█
```

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

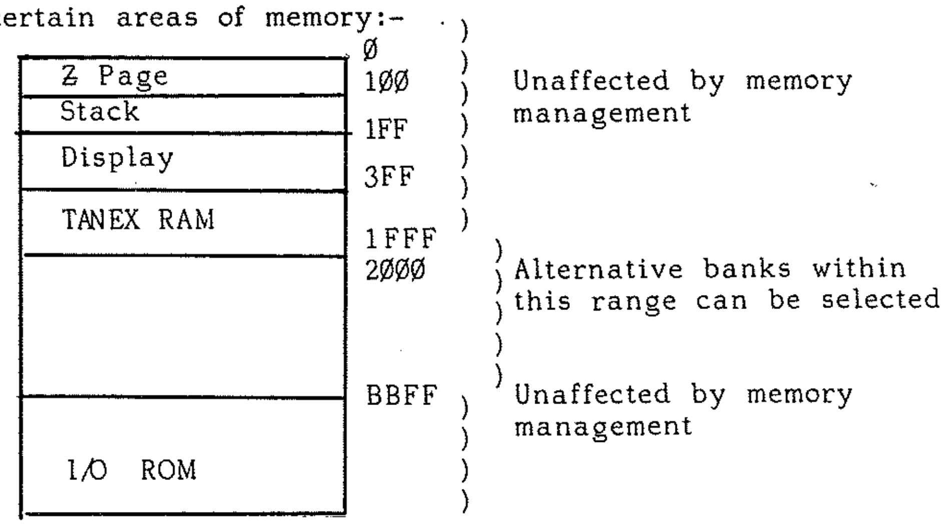
f) Remove single instruction mode and breakpoints by typing N<CR> the B<CR>.

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

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

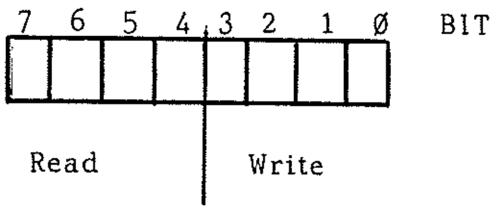
Memory Management Control

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


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

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

The control register is allocated as follows:



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

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

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

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

Tanbug V2 and Microsoft Basic

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

Basic Initialisation and Warm Start

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

BAS <CR>

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

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

WAR <CR>

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

Your can NOT use WAR when:

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

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

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

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

Printer Control from Microsoft BASIC

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

In direct mode, printers can be controlled as follows:

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

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

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

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

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

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

30    PRINT A$

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

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

<u>Clear Screen</u>

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

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

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

<u>Memory Management Control</u>

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

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

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

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

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

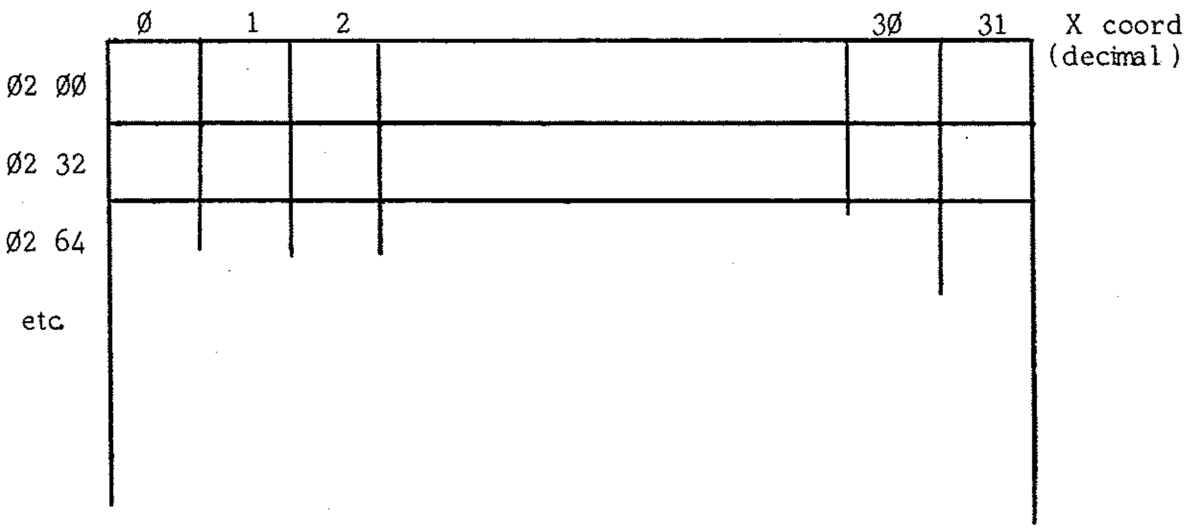
```
4080    POKE 65, 2        ; new data
4090    X = USR(I)        ; write
```

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

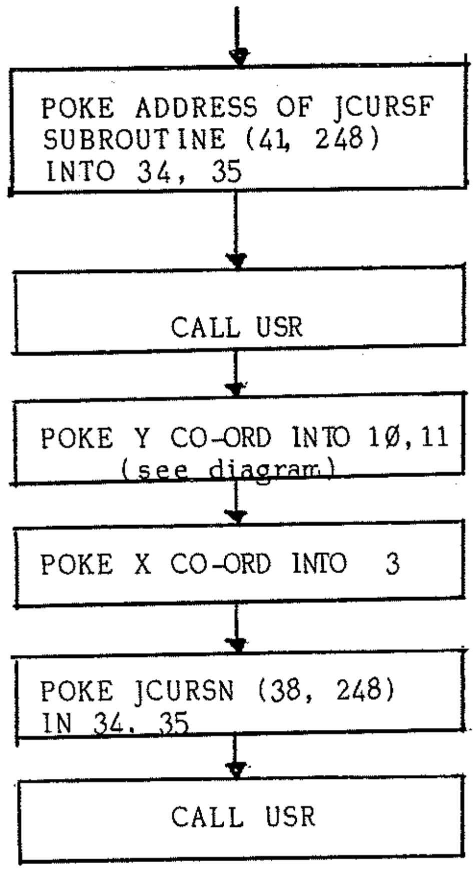
Note that decimal locations 64 - 68 are corrupted by use of the input command, so if this is used between memory management operations, these locations must be reset by POKES.

Cursor Control

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



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


POKE ADDRESS OF JCURSF SUBROUTINE (41, 248) INTO 34, 35
CALL USR
POKE Y CO-ORD INTO 1Ø,11 (see diagram)
POKE X CO-ORD INTO 3
POKE JCURSN (38, 248) IN 34, 35
CALL USR

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

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

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

TABLE OF HEX ASCII CODES

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

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

TABLE OF HEX ASCII CODES (CONTINUED)

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