# CHAPTER TWO

# THE OPERATING SYSTEM

Assembly code is very difficult to use on its own because it contains no specific input or output commands. This means that, say, printing a character on the screen requires a series of LDA and STA commands to place the relevant bytes in the relevant places in memory. This would prove difficult even in Mode 7 let alone in Mode 2! If, further, we want to use one of the more complicated pieces of the hardware on the machine such as the disc drive then the machine code needed will become ridiculously complicated. The OPERATING SYSTEM ROM in the the computer comprises just under 16K of machine code routines which will handle virtually all the input and output operations you are ever likely to need. Not only does this ROM contain all the *FX and other 'star' commands, but it also contains routines for handling VDU commands and filing commands, to name just a few.

**Useful OS routines**

Some of the operating system routines are rarely used or are too complicated to cover completely in this book. See THE ADVANCED USER GUIDE (Cambridge Microcomputer Centre, 1983). If you are in doubt, the rule is that ANY input or output command that is available from BASIC can be accessed through one of the operating system routines somehow. Here, however, are some of the more useful routines.

**OSBYTE**

(Operating System BYTE routine)
Location: &FFF4
This routine is used to set up any of a large number

of flags that the operating system uses to decide what to do in particular situations. It is the equivalent to the BASIC *FX command. There are over 150 of these commands though most of them are not particularly useful. There is a list of most of the useful commands on page 418-441 of the USER GUIDE. However, for a complete list of all the OSBYTE calls, and what they do, see THE ADVANCED USER GUIDE.

To use the OSBYTE routine from machine code the accumulator must be set to the number of the specific command you wish to use and the X and Y registers must be set to any parameters that the routine needs for that particular command. OSBYTE is at address &FFF4.

**OSWORD** (Operating System WORD operator routine)
Location &FFF1

This routine is similar to OSBYTE but it handles operations that require larger amounts of data. This data is stored in a CONTROL BLOCK. This is a series of bytes which contain the parameters needed for a particular OSWORD call. To use OSWORD you should set aside a block of memory (a parameter block) long enough for the OSWORD command you want to use, and then place the parameters in this block. Then set the accumulator to the number of the particular OSWORD command you want to use and set the X and Y register to the low and high bytes respectively of the address of the first byte of the parameter block. Then call the routine, which starts at &FFF1.

For a complete list of all the OSWORD calls, and what they do, see THE ADVANCED USER GUIDE.

**OSWRCH** (Operating System WRite CHaracter)
Location &FFEE

This is the routine that performs the equivalent of a BASIC VDU command. By loading the accumulator with a number and calling &FFEE the contents of the accumulator will be written to the screen. ALL output to the screen can be directed through this

routine. For example, to clear the text screen the following code should be used:

```
LDA #12
JSR &FFEE
```

To take another example, the BASIC PLOT command is accessed using the sequence VDU25, plot number, low byte of X coordinate, high byte of X coordinate, low byte of Y coordinate, high byte of Y coordinate. Thus to enter graphics Mode 4 and draw a line from the bottom left-hand corner of the screen to the top right-hand corner of the screen, the following piece of code should be used:

```
LDA #22   \ Change mode
JSR &FFEE \ to
LDA #4    \ mode 4
JSR &FFEE \
LDA #25   \
JSR &FFEE \ PLOT
LDA #4    \
JSR &FFEE \ 4,
LDA #0    \
JSR &FFEE \
LDA #0    \ 0,
JSR &FFEE \
LDA #0    \
JSR &FFEE \ 0
LDA #25   \ PLOT
JSR &FFEE \
LDA #5    \
JSR &FFEE \ 5,
LDA #&FC  \
JSR &FFEE \ 1276,
LDA #4    \
JSR &FFEE \
LDA #&FC  \
JSR &FFEE \ 1020
LDA #3    \
JSR &FFEE \
```

However, this would be somewhat tedious. A point to note is that the OSWRCH routine exits with the

A, X and Y registers unchanged which means that we can do this:

```
      LDX #0
.loop LDA table,X
      JSR &FFEE
      INX
      CPX #14
      BNE loop
      RTS
```

We can then place, starting at the address pointed to by TABLE, a table containing the following bytes:

22,4,25,4,0,0,0,0,25,5,&FC,4,&FC,3

Make sure you understand this technique as it is very useful!

For a complete list of what the VDU codes do, see pages 377-389 of the USER GUIDE.

One important thing to note about this routine is that, as with the VDU command, a carriage return (ASCII code 13) returns the cursor to the beginning of the line it is already on − it DOES NOT move the cursor down to the beginning of the next line. To do this you must send a carriage return followed by a line feed (ASCII 10). This can be a nuisance, so the operating system provides another two routines.

The first of these is OSNEWL (Operating System NEW Line) which is at address &FFE7. This routine sends a carriage return and a line feed to OSWRCH.

The second is OSASCI (Operating System ASCII output routine) which is at address &FFE3. This routine is the same as OSWRCH except that, if a carriage return is sent to it, it does both a carriage return and a line feed. It is interesting to see how this is actually done by the operating system. Here is the relevant section:

```
.OSASCI  CMP #13
         BNE OSWRCH
.OSNEWL  LDA #10
         JSR OSWRCH
         LDA #13
```

Notice that OSWRCH jumps via the vector at &20E to the routine that actually prints a character.

**OSRDCH** (Operating System ReaD CHaracter) − the GET routine.
Location &FFE0

This is the routine that the BASIC interpreter uses to get values for the BASIC GET command. After calling the OSRDCH routine the carry flag will be set if an error has occurred; the accumulator will then contain the error number. This will usually only occur if the ESCAPE key has been pressed, in which case the accumulator will contain 27. Here, the program must acknowledge this by calling OSBYTE with the accumulator set to &7E (see page 429 of the USER GUIDE or page 149 of THE ADVANCED USER GUIDE).

If the carry flag is clear then the accumulator will contain a character that has been read from the current input device. This will usually be the keyboard although, in some cases, it could be the RS423 interface, or a disc file if a *EXEC command has been used, for example.

If no key has been pressed the routine will wait until a key has been pressed before it returns a value, so it cannot be used for 'arcade' games. OSBYTE with the accumulator set to &81 should be used) (see page 153 of the ADVANCED USER GUIDE. So, if we want to get a key from the keyboard, the following piece of code should be used:

```
       JSR OSRDCH \ Get a key
       BCC next   \ if no error process key
       CMP #27    \ if it isn't escape
       BNE error  \ goto error routine
       LDA #&7E   \ if escape
       JSR OSBYTE \ acknowledge and
       JMP escape \ goto escape routine
.next  ...        \ process key
```

If we are using the keyboard then the error can only

be an escape, so we can use this:

```
        JSR OSRDCH \ Get a key
        BCC next   \ if no error process key
        LDA #&7E   \ if escape
        JSR OSBYTE \ acknowledge and
        JMP escape \ goto escape routine
.next ...          \ process key
```

If the escape key has been disabled then all that is needed is a call to OSRDCH.

**OSCLI** (Operating System Command Line Interpreter)
Location: &FFF7

This is the routine the operating system uses to process * commands. If in BASIC you use a command preceded by a * then the BASIC interpreter uses this routine.
To use this routine you need to have the command you want to perform stored in memory as an ASCII string. At the end of the string there should be a carriage return (ASCII code 13). Then you should set the X and Y registers to the low and high bytes respectively of the address of the first character of the string in memory. Then you should call the OSCLI routine, which starts at &FFF7.
As an example, let us take a game program which loads into the computer and then loads a Mode 2 graphics screen before running the game. If we assume that the screen has been saved after the main program on the tape or disc using *SAVE SCREEN 3000 8000 then the following code can be used:

```
LDA #22           \ Mode command:
JSR &FFEE         \
LDA #2            \ Change to
JSR &FFEE         \ Mode 2.
LDX #str MOD256   \ set X and Y to
LDY #str DIV256   \ start of string.
JSR &FFF7         \ call OSCLI
...               \ rest of game.
...
```

```
.str EQUS "LOAD SCREEN" \ string with carriage
     EQUB 13            \ return at end.
```

Note that a * is not needed at the start of the string. This routine can be used for all * commands. However, as we have already seen, there is an easier way to perform *FX commands.

**Memory Usage**

Normally on the BBC Micro the user is allowed to use the memory stretching from PAGE to HIMEM. This can be used for BASIC programs, machine code programs, variables, data, etcetera. However, it is sometimes necessary to place a piece of machine code somewhere where the BASIC cannot affect it. For example, you might have a machine code routine that was used by several different BASIC programs that chain each other. This routine must be kept clear of the BASIC or it will become corrupted. There are several methods of doing this. The most obvious method is to set HIMEM lower, leaving room for the machine code to be placed just above it. However, if the screen mode is changed, this resets HIMEM according to the new mode and may clear the machine code. It is usually better instead to set PAGE higher to leave room for the machine code just below it, as PAGE is only reset on BREAK. However, this means you need a loader program that sets the value of PAGE, and some programs may reset PAGE for other reasons anyway.

However, there is a large amount of memory set aside for the operating system and the BASIC. On a tape machine PAGE is normally set to &E00 leaving 3.5K for the ROMs to use. Now, since not all of this memory is likely to be in use at any one time, it is usually possible to use some of it to place machine code routines in. This 3.5K is described below, a page (256 bytes) at a time.

Zero Page

This page should be used for machine code variables only, unless you are very short of memory. Locations &00-&8F are reserved for the current language. However, BASIC itself does not use

locations &50-&8F of this so these locations are safe to use. Locations &90-&9F are allocated to the ECONET system, so, unless you are using ECONET, these are safe to use. Locations &A0-&A7 are used byte the NMI interrupt which is used by the disc system. However, on tape machines this is not used and is safe. On disc machines this MUST NOT be used. The rest of zero page is used by the operating system and should not normally be used.

Page 1

This is the processor's hardware stack. However, the processor will not normally use more than the top quarter of the stack so it is reasonably safe to use &100-&1BF though I would recommend that you only use &100-&17F and then only for TEMPORARY storage of strings, etcetra.

Page 2

This is the operating system's main work area and as such should not be used.

Page 3

This contains some more operating system workspace. Memory locations &300-&37F contain the VDU command workspace; &380-&3DF are used by the cassette system and &3E0-&3FF make up the keyboard input buffer. None of this is particularly safe to use.

Page 4

This is used by BASIC for variable storage. Locations &400-&46B contain the values of the integer variables @%-Z%. These are stored in order, using four bytes for each. Each of these is stored low byte to high byte, as a four-byte two's compliment number. The integer variables can be useful for passing variables between BASIC and machine code. The rest of this page (&46C to &4FF) is used for pointers which indicate where the other variables are stored. If BASIC is going to be used, this page cannot be used for machine code. If, however, you are going to write a program − such as a game − which will not use BASIC, and you are short of memory, this page can be used.

Page 5

This is used by BASIC as a stack for FOR, REPEAT and GOSUB return addresses. Again, this can only

be used if BASIC is not needed.

Page 6

This is used by BASIC for working on strings. So long as BASIC is not working on strings at a particular time, this could be used as a temporary work space. However, this page must be clear before returning to BASIC from your machine code routine. As before, if BASIC is not needed, this page is safe to use for machine code.

Page 7

This is the BASIC line input buffer. Again it can be used safely if BASIC is not used.

Page 8

This is laid out as follows:

&800-&83F Sound workspace
&840-&87F Sound buffers
&880-&8BF Printer Buffer
&8C0-&8FF Envelope storage

If any of these sections are not in use then they are safe to use for machine code.

Page 9

This is used in three DIFFERENT ways.

1)
&900-&9BF Extra sound envelopes
&9C0-&9FF Speech buffer

2)
&900-&9BF RS423 output buffer
&9C0-&9FF Speech buffer

3)
&900-&9FF Cassette output buffer.

If none of these is in use then this page can be used.

Page 10

This is either the cassette or the RS423 input buffer. As before, this page can be used if the cassette and RS423 systems are inactive.

Page 11

This is used for soft key definitions. If you use this to store your own code, the soft keys will produce

rubbish if pressed. This can be countered by disabling the soft keys using *FX225

Page 12

This is used for the user-defined characters. This page can be used so long as the user-defined characters are not printed.

Page 13

Memory block &D00-&D9E is used by the NMI system. Cassette users may use this but disc users must use *TAPE first. Memory block &D9F-&DEF is the expanded vector set. This is used by some paged ROMs and by the disc system to vector useful calls. With care, tape users can use it. &DF0-&DFF is used by the ROMs for workspace allocation and should not be used except with extreme caution.

All this means is that, if you are writing a program that uses none of the system's buffers and does not use BASIC, you have available over 2K more than usual for machine code commands. With great care, even short BASIC programs can be placed in pages 8-12 by setting PAGE accordingly.