BBC MICROBASE SERIES
BBC 6502 Machine Code by Geoff Cox

This series was first published on Micronet
between April and October 1991

The BBC Micro Operating System
Part One: The moving electron writes

In the last series we reviewed the basics of machine code programming using the 6502. You will have noticed that not too many examples of programming were given. This is because there are two levels of programming on any machine, machine level and operating system level.

Operating Systems

An operating system is basically a group of routines that sit between the user and the electronics of the computer. To illustrate what an operating system does we have to turn briefly from the path of the series.

We'll imagine that you want to write the letter A on the screen of your monitor. First we have to work out the shape of the character and slice it horizontally into eight sections, one for each of eight screen scanning lines on the monitor.

Now we need to detect when the frame synchronising pulse for the monitor is sent by the computer. Next we need to count the line synchronising pulses to find the one corresponding to the start of the first line of the character.

Then we must time from this pulse to the start of the first line of the character, turn on the appropriate electron guns in the monitor and turn them off at the correct time for the end of the of the character. Finally we have a few microseconds to do it all again for the next line. That's more or less what happens fifty times a second on your monitor screen.

Mapped Screens

This makes even the easiest task very complex. We can make life easier by storing a picture or map of the screen in memory and writing the character shape to the appropriate map locations.

The map can then be scanned in synchronisation with the electron beam on the monitor. This can either be via a clever piece of electronics or a software routine.

To illustrate the BBC map, type the following program on any 6502-based BBC without screen RAM shadowing.

```
10 MODE 0
20 ?&7D00=65
```

You should see two little white dots towards the bottom right of the screen. Now two dots are not the letter A so if we want to write A we have to designthe character and write it to the map. This program does just that.

```
10 MODE 0
20 FOR A=&7D00 TO &7D07
30 READ B
40 ?A=B
50 NEXT
60 DATA &3C, &66, &66, &7E
70 DATA &66, &66, &66, &00
```

The sharp-eyed among you will have spotted something a little odd here. The screen seems to be arranged in blocks of eight bytes in this mode. Don't worry about this - it simply makes character table design simpler.

Character Tables

We can make life even easier by designing a set of characters and putting them in an area of memory where they can be looked up. In the BBC B this area is in ROM starting at &C000.

A quick diversion here. If we have to have a series of character designs in memory it helps to have a standard method of accessing them.

The standard character ordering is called ASCII. A space has ASCII number 32 and this is the first "printable" character in the set. The last is 127 (Delete). There are eight bytes in each character matrix so the design for any character is at &C000+(8*(ASCII - 32)).

This program examines the ROM character table and shows how characters are arranged.

```
10 MODE 0
20 PRINT "CHARACTER ?"
30 A$=GET$
40 PRINT A$
50 START=(8(ASC(A$)-32))+&C000
60 FIN=START+7
70 FOR BYTE=START TO FIN
80 PEEK=?BYTE
90 PROC_BIN
100 PRINT~BYTE;TAB(15);~?BYTE;TAB(20);
110 NEXT
120 END
130 DEFPROC_BIN
140 A$=""
150 FOR BIT=7 TO 0 STEP-1
160 X=(PEEK AND 2^BIT)
170 IF X>1 THEN A$=A$+"" ELSE A$=A$+"."
180 NEXT
190 ENDPROC
```

As you will often want to write characters on the screen it makes a lot of sense to have a little routine to take a character from a register and print it on the screen.

The routine will look up a character in the table and print it to the appropriate position on the screen. There will also need to be a record of the cursor position on the screen.

Routines for printing and moving a cursor on the screen will also be useful. You could, of course, write all of these routines yourself as part of every program but as the computer manufacturer has to provide them in order to tell you the machine is working he usually leaves an access point for you to use the routines yourself.

Incidentally if you rewrite the first program but change line 10 to read MODE 7 you will see a letter A on the screen.

This is an example of clever electronics. A chip on the BBC board contains a character generator. Instead of the character design being stored in the screen map the character's ASCII value is stored. This is read by the electronics and used to generate characters directly on the screen. This allows a 40 x 80 text and block graphics screen to be

generated in just 1K of RAM.

Using the Operating System

The screen handling routines and several others make up an operating system. In the BBC micro the series of routines that write a character on the screen can be accessed though a point called OSWRCH at &FFFE.

So instead of having to design characters, time lines and do all of the other things, you can output a letter A to the screen simply using:

```
LDA #65
JSR &FFEE ;OSWRCH
```

and leave the operating system to do the rest.

BBC 6502 Machine Code
Part Two: In the beginning

Last week we looked at the reasons for an operating system and how it can simplify the programmer's task. Unfortunately you will not always be able to use the operating system. There may not be a suitable routine or it may be too slow. In these cases you have to write to the device itself. So to continue our look at machine code we need to examine programming with device handling and the operating system.

This series will attempt to kill two birds with one stone by taking avery close look at Acorn's OS 1.20 used on Model Bs.

This changed little for the B+ and Master so while the routines may be in a slightly different place the basic system will be the same. The Master uses 65C02 code which has a few extra instructions so there may be some differences in the length of the code.

What you will need

A disassembler or machine code monitor that can handle 6502 or 65C02 codes. If If you don't have either you can use:

```
PRINT ~?address
```

to get the hexadecimal codes which you can then look up in the back of the BBC User Guide to get the relevant command.

Suitable Monitors are EXMON, BBC Monitor or the SYSTEM monitor. (I use "Maxim" - Ed.) If you have a single- stepping monitor like one of these, you will be able to trace some of the routines for yourself.

For this series we will use standard 6502 mnemonics except that DB and DW will be used to show byte assignments rather than EQUS, EQUW and EQUB. This is because the disassembler I am using produces these codes and it's a lot easier to follow than convoluted BBC-type statements.

First things first

Remember that an operating system is not a program in the usual sense. Normal programs have a defined entry and exit routines. An operating system can have a large number of entry and exit points as well as interlocking routines. So to examine the operating system we need a starting point.

The 6502 regards memory as a series of 256-byte pages 0 to &FF (255). Any address can be considered to be a page number plus an offset within the page. Both figures can be represented by a single byte. So

address &FF01 is on Page &FF offset 01. The concept of offsets is very useful if you ever get involved in 80n86 programming.

The BBC Manual gives a series of system entry points on page FF. Most of these are indirected through Page 2 and as we cannot guarantee what the contents of Page 2 should be (the vectors can be and are changed) these are useless as starting points. This leaves three sensible entry points.

```
6502 Vectors
FFFA
DW &0D00      ;NMI address
FFFC DW &D9CD ;RESET address
FFFE
DW &DC1C      ;IRQ address
```

The NMI address is in RAM so no joy there, but the other two look fine. The best is RESET as this is where the machine starts when it is turned on or BREAK is pressed. In the case of Model B and OS 1.20 that address is &D9CD, so what happens?

In the beginning

Reset can be effected by turning on the computer or pressing BREAK. If it is a power-up then the system VIA and processor are reset electronically.

If this is a power on situation then nothing has been set up. The first thing that happens when power is turned on is that the 6522 VIAs, the processor and the floppy disc controller are reset. This is done by means of one of three printed circuit tracks. The tracks are RSTA, RST and NOTRESET.

RSTA is only connected to the system 6522 Versatile Interface adaptor (VIA). This operates through a little resistor/capacitor circuit that only works when the power is turned on. The effect of this is that the 6522 System VIA Interrupt Enable Register (IER) bits 0 to 6 will be clear (0) only if the reset is caused by a power on condition.

If the Reset is caused by BREAK being pressed then the machine must have been on and therefore one or more of the System VIA IER bits will be set (to 1). If one or more bits are set then bit 7 of the VIA will also be set. This is used to determine the type of Reset. So let's look at the operating system more closely.

```
D9CD
LDA
#&40  ;set NMI first instruction to RTI
D9CF
STA
&0D00 ;NMI RAM start
```

RESET is the ultimate Act of God as far as the machine is concerned. Anything could be happening so the operating system has to clean up the system as its first act.

These first instructions just make sure that if a disc is running no more information will be read or written from or to the disc. This illustrates why you shouldn't press BREAK when a disc is being accessed!

The next section sets up the stack:

```
+
D9D2
SEI  ;disable interrupts just in case
```

```
D9D3
CLD  ;clear decimal flag
D9D4
LDX
#&FF ;reset stack to where it should be
D9D6
TXS  ;(&1FF)
```

Next find out if a power-up reset or a BREAK press by examining the System VIA IER register.

```
D9D7
LDA
&FE4E ;read interrupt enable register of the system VIA
D9DA
ASL   ;shift bit 7 into carry
D9DB
PHA   ;save what's left
D9DC
BEQ
&D9E7 ;if Power up A=0 so go to D9E7 to clear memory
```

That's probably enough for this time. Don't worry! I don't intend to do a complete disassembly of the operating system in this series but we will follow through the power-on sequence to the end because a lot of interesting things happen at this time.

We'll take a look at D9E7 and the next routine in this sequence (D9DE) in the next part.

BBC 6502 Machine Code
Part Three: Cleaning up the mess

In the last part we looked at what happens when you press BREAK or switch on the machine. We'll now continue with a look at an undocumented (at least officially) routine.

The byte at &258 can be used to contain information about what the machine should do if BREAK is pressed. FX200,n is used to set this byte. If n=2 or n=3 then the memory must be cleared. This is often used in program protection.

```
D9DE
LDA
&0258 ;else if BREAK pressed read BREAK Action flags (set by FX200,n)
D9E1
LSR

      ;divide by 2
D9E2
CMP
#&01  ;if &0258 <> 2 or 3
D9E4
BNE
&DA03 ;then Goto &DA03
D9E6
LSR

      ;divide A by 2 again (A=0 if FX200,2/3 else A=n/4
```

Pages 4-&7F are cleared by a simple loop if &258=2 or 3 or it is a power on reset. Look out for the clever way of avoiding problems on 16K machines.

```
D9E7
```

```
LDX
#&04    ;get page to start clearance from (4)
D9E9
STX
&01     ;store it in ZP 01
D9EB
STA &00 ;store A at 00
D9ED
TAY

        ;and in Y to set loop counter

        ;LOOP STARTS
D9EE
STA
(&00),Y ;clear RAM
D9F0
CMP
&01     ;until page address (in &01) =0
D9F2
BEQ
&D9FD
;
D9F4
INY

        ;increment pointer
D9F5
BNE
&D9EE   ;if not zero loop round again
D9F7
INY     ;else increment again (Y=1) this avoids overwriting the RTI
        ;instruction at &D00 D9F8 INX

        ;increment X
D9F9
INC
&01     ;increment &01
D9FB
BPL
&D9EE   ;loop until Page (in 01)=&80 then exit
```

Note that RAM addressing for 16K loops around to &4000=&00 hence the checking of &01 for 00. This avoids overwriting zero page on BREAK which would cause the machine to crash!

```
D9FD
STX
&028E   ;writes marker for available RAM 40 =16K,80=32
DA00
STX
&0284   ;write soft key consistency flag
```

This routine shows the basic structure of a loop. Those of you who program in BASIC will recognise it as a very simple structure:

```
10 A=A+1
20 IF A<20 GOTO 10
```

The loop uses zero page addressing with the target address in 00 and 01 (Page) and the index in Y.

The loop is exited when the value in 01 becomes negative. Remember that all values between 0 and &7F are considered to be positive, so the BPL instruction can be used to exit the loop at page &80, the first negative number. This is the first of the useful loop techniques

we'll see in this series.

Notice that the first byte of each page is left unchanged. This is useful if you want information to survive a BREAK of this type. This clearing of memory is not normally carried out.

Next week we'll have a look at the normal RESET path.

BBC 6502 Machine Code
Part Four: Cleaning up even more mess

As we saw last week, a normal warm reset avoids the memory clearance and proceeds to set up the System VIA.

```
DA03 LDX #&0F  ;set PORT B data direction register to output on bits
               ;0-3 and input bits 4-7
DA05 STX &FE42 ;
```

The next bit is a little more complicated and is intimately bound up with hardware. The function is to set up the addressable latch IC 32 for peripherals via PORT B.

The latch value is written by writing the value to &FE40 bits 0 to 2 and either a 1 or 0 to bit 3.

Writing the value + 8 therefore writes a 1 to the latched address, otherwise a 0 is written.

Value +	Peripheral	Effect 0	8
0	Sound chip Speech Chip	Enabled	Disabled
1	(RS)	Low	High
2	(WS)	Low	High
2	(WS)	Low	High
3	Keyboard Write	Disabled	Enabled
4	C0 address modifier	Low	High
5	C1 address modifier	Low	High
6	Caps LED	On	Off
7	Shift LED	On	Off

C0 and C1 are involved with hardware scroll screen address.

```
                    ;X=&F on entry
DA08   DEX          ;loop start
DA09   STX   &FE40  ;Write latch IC32
DA0C   CPX   #&09   ;Is it 9?
DA0E   BCS   &DA08  ;If not go back and do it again
                    ;X=8 at this point
                    ;Caps Lock On, SHIFT Lock undetermined
                    ;Keyboard Autoscan on
                    ;Sound disabled (may still sound)
```

Next the keyboard is scanned to determine the values of the keyboard links and whether a Ctrl-Break has been performed.

Remember that although we have spent a lot of time reading this, we are probably less than 200 microseconds after BREAK was pressed.

The check for Ctrl-Break is effectively looking for simultaneous keypresses.

```
DA10   INX           ;X=9
DA11   TXA           ;A=X
DA12   JSR   &F02A   ;Interrogate keyboard
DA15   CPX   #&80    ;for keyboard links 9-2 and CTRL key (1)
DA17   ROR   &FC     ;rotate MSB into bit 7 of &FC

DA19   TAX           ;Get back value of X for loop
DA1A   DEX           ;Decrement it
DA1B   BNE   &DA11   ;and if >0 do loop again
                     ;On exit if Carry set link 3 is made
                     ;link 2 = bit 0 of &FC and so on
                     ;If CTRL pressed bit 7 of &FC=1 X=0
DA1D   STX   &028D   ;Clear last BREAK flag
DA20   ROL   &FC     ;CTRL is now in carry &FC is keyboard links
DA22   JSR   &EEEB   ;Set LEDs
                     ;Carry set on entry is in bit 7 of A on exit
DA25   ROR           ;Get carry back into carry flag
```

To review what the operating system has done so far, about 400 microseconds after a BREAK press or about 2 milliseconds from a power on. Memory may have been cleared, NMIs have been short circuited, IRQs disabled. The keyboard has been scanned for made links and for Ctrl being pressed.

We have also located two important and undocumented subroutines: &F02A toscan the keyboard and &EEEB to set the keyboard LEDs.

The F02A routine scans for the key whose code is in X being pressed:

```
F02A   LDY   #&03  ;Stop Auto scan
F02C   STY   &FE40 ;by writing to system VIA
F02F   LDY   #&7F  ;Set bits 0 to 6 of port A to input on bit 7.
                   ;Output on bits 0 to 6
F031   STY   &FE43 ;
F034   STX   &FE4F ;Write X to Port A system VIA (key to check)
F037   LDX   &FE4F ;Read back &80 if key pressed (M set)
F03A   RTS         ;And return
```

The routine at &EEEB switches on the selected keyboard lights.

```
EEEB   PHP           ;Save flags
EEEC   LDA   &025A   ;Read keyboard status
                     ;Bit 7=1 shift enabled
                     ;Bit 6=1 control pressed
                     ;Bit 5 =0 shift lock
                     ;Bit 4 =0 Caps lock
                     ;Bit 3 =1 shift pressed
EEEF   LSR           ;Shift Caps bit into bit 3
EEF0   AND   #&18    ;Mask out all but 4 and 3
EEF2   ORA   #&06    ;Returns 6 if caps lock OFF &E if on.
                     ;Remember add 8 to the value for the addressable
                     ;latch to send a 1.
EEF4   STA   &FE40   ;Turn on or off caps light if required
EEF7   LSR           ;Bring shift bit into bit 3
EEF8   ORA   #&07    ;
EEFA   STA   &FE40   ;Turn on or off shift lock light
EEFD   JSR   &F12E   ;Set keyboard counter
EF00   PLA           ;Get back flags into A
EF01   RTS           ;Return
```

In this part we've had a look at subroutines using JSR and RTS, the machine code equivalent of GOSUB, PROC or FN. Subroutines are often used in machine code to perform such frequently needed functions as scanning a keyboard or turning on and off lights.

We've also discovered that the byte at &25A contains the keyboard

status. Try changing it for yourself. You can therefore use OR and AND to set the shift and Caps lock status of the machine for a particular program.

Next week we'll examine setting up the default vector table in memory.

BBC 6502 Machine Code
Part Five: Vectors Victor

The next stage is to set up the vectors on page 2.

```
DA26   LDX   #&9C  ;
DA28   LDY   #&8D  ;
DA2A   PLA         ;Get back A from &D9DB DA2B
BEQ    &DA36       ;If A=0 power up reset so go to DA36 with X=&9C
                   ;Y=&8D
DA2D   LDY   #&7E  ;else let Y=&7E
DA2F   BCC   &DA42 ;and if not CTRL- BREAK go to DA42 for a WARM RESET
DA31   LDY   #&87  ;else Y=&87 COLD RESET
DA33   INC   &028D ;&28D=1
DA36   INC   &028D ;&28D=&28D+1
DA39   LDA   &FC   ;Get keyboard links set
DA3B   EOR   #&FF  ;Invert
DA3D   STA   &028F ;and store at &28F
DA40   LDX   #&90  ;X=&90
```

What we have done is to set up the high water marks for the reset of vectors.

```
&28D=0 Warm reset, X=&9C, Y=&7E
&28D=1 Power up  , X=&90, Y=&8D
&28D=2 Cold reset, X=&9C, Y=&87
```

```
DA42   LDA   #&00    ;A=0
DA44   CPX   #&CE    ;zero &200+X to &2CD
DA46   BCC   &DA4A   ;
DA48   LDA   #&FF    ;then set &2CE to &2FF to &FF
DA4A   STA   &0200,X ;
DA4D   INX           ;
DA4E   BNE   &DA44   ;
                     ;A=&FF X=0
```

This is another IF-GOTO loop, but in this case it is a double function loop. The test at DA44 to DA46 means that A is 0 only for values of X between the high water mark and &CD. Above this value A is set to &FF by the instruction at &DA48. This saves a few bytes of space, essential when writing a tightly-filled ROM.

The next instructions set up the printer port. The only reason for doing this now is to save two bytes. A must be &FF at this point so it is used to set up the User VIA for outputs as the printer port.

```
DA50   STA   &FE63   ;Set port A of user VIA to all outputs (printer out)
DA53   TXA           ;A=0 DA54   LDX   #&E2    ;X=&E2
```

```
START OF LOOP
DA56   STA   &00,X   ;set zero page addresses &E2 to &FF to zero
DA58   INX           ;
DA59   BNE   &DA56   ;X=0
```

Now set up the vectors in page 2 from the table at &D940:

```
DA5B   LDA   &D93F,Y ;copy data from &D93F+Y
DA5E   STA   &01FF,Y ;to &1FF+Y
DA61   DEY           ;until
DA62   BNE   &DA5B   ;1FF+Y=&200
```

Note that this is a decrementing loop which, for loops ending when an index register reaches zero, is faster and shorter because no compare is needed. More space saved!

Now the RS423 port is set up via a subroutine affecting the ACIA. (Asynchronous Communications Interface Adaptor)

```
DA64   LDA   #&62    ;A=&62
DA66   STA   &ED     ;store in &ED
DA68   JSR   &FB0A   ;set up ACIA ;X=0
```

Now Acorn clears the interrupt and enable registers of both VIAs.

```
DA6B   LDA   #&7F    ;bit 7 is 0!
DA6D   INX           ;
DA6E   STA   &FE4D,X ;
DA71   STA   &FE6D,X ;
DA74   DEX           ;
DA75   BPL   &DA6E   ;
                     ;This loop only has two passes as X=0 on entry.
DA77   CLI           ;Briefly allow interrupts to clear anything
                     ;pending
DA78   SEI           ;Disallow again NB: all VIA IRQs are disabled
DA79   BIT   &FC     ;If bit 6=1 then JSR &F055 as there must be a
                     ;hardware interrupt!
DA7B   BVC   &DA80   ;else DA80
DA7D   JSR   &F055   ;
```

What have we here? Another undocumented routine. If bit 6 of &FC is set there must have been a hardware interrupt when the SEI occurred.

From the circuit diagram the only place that this IRQ could have come from is the 1MHz bus - let's have a look at the routine at &F055.

```
F055   JMP   (&FDFE) ;Jim paged entry vector
```

So we jump to some piece of hardware on the 1MHz bus. This would probably be a ROM which would take over the system at power on and Break. This has some very interesting applications. It was designed by Acorn to provide a crude Econet facility to allow a batch of machines to be functionally tested without the need to install a full Econet kit.

Next week we shall examine the VIA bus.

BBC 6502 Machine Code
Part Six: The VI bus

The next interesting routine we find in the BBC operating system is the one that sets up the system VIA interrupts. It is located at &DA80. Refer to the manual for the meanings of Sheila addresses.

```
DA80   LDX   #&F2  ;Enable interrupts 1,4,5,6 of system VIA
DA82   STX   &FE4E ;
                   ;0 Keyboard enabled as needed
                   ;1 Frame sync pulse
                   ;4 End of A/D conversion
                   ;5 T2 counter (for speech)
                   ;6 T1 counter (10 mSec intervals)

DA85   LDX   #&04  ;set system VIA PCR
DA87   STX   &FE4C ;
                   ;CA1 Interrupt on negative edge (Frame sync)
                   ;CA2 Handshake output for keyboard
                   ;CB1 Interrupt on negative edge (end of conversion)
```

```
                    ;CB2 Negative edge (Light pen strobe)
DA8A   LDA   #&60   ;Set system VIA ACR
DA8C   STA   &FE4B  ;
                    ;Disable latching
                    ;Disable shift register
                    ;T1 counter continuous interrupts
                    ;T2 counter timed interrupt

DA8F   LDA   #&0E   ;Set system VIA T1 counter (low)
DA91   STA   &FE46  ;
                    ;This becomes effective when T1 hi set
DA94   STA   &FE6C  ;Set user VIA PCR
                    ;CA1 interrupt on -ve edge (Printer Acknowledge)
DA80   LDX   #&F2   ;enable interrupts
                    ;CA2 High output (printer strobe)
                    ;CB1 Interrupt on -ve edge (user port)
                    ;CB2 Negative edge (user port)
DA97   STA   &FEC0  ;Set up A/D converter Bits 0 and 1 determine
                    ;channel selected
                    ;If Bit 3=0 it is set for an 8-bit conversion.
                    ;If bit 3=1 12-bit conversion.
```

Now although the machine now knows how much RAM it has it still doesn't know if it's a Model A or Model B, so it does not know if a user VIA is present at &FE60-FE6F.

The next routine tests for the presence of a user VIA. The system timers are then set up to interrupt every 10mSec. Sound channels are cleared and the serial ULA is set up. Then the function keys are reset.

Now we need a catalogue of sideways ROMS. This is not a catalogue in the conventional sense as the ROM title is always at the same place in the ROM itself and can be read from there. It is a catalogue of the ROM types and positions.

There is a ROM latch at &FE30. Writing a number between 0 and 15 to this switches the corresponding ROM into the area between &8000 and &BFFF. A short subroutine does this and maintains a copy of the current ROM in zero page at location &F4.

```
                    ;on entry X=required ROM number
DC16   STX   &F4    ;RAM copy of ROM latch
DC18   STX   &FE30  ;Write to ROM latch
DC1B   RTS          ;and return
```

You should use this subroutine if you want to switch ROMs. Now we can look at the ROM cataloguing routines;

A ROM is considered to be valid if it contains a string identical to astring at location &DF0C in the Operating System ROM.

```
DF0C   DB    ')C('   ;
DF0F   DB    0       ;
```

The location of this string is pointed to by an offset byte located at &8007.

```
;X=0 on entry
DABD   JSR   &DC16   ;Set up ROM latch and RAM copy to X
DAC0   LDX   #&03    ;Set X to point to offset in table
DA80   LDX   #&F2    ;Enable interrupts
DAC2   LDY   &8007   ;Get copyright offset from ROM
DAC5   LDA   &8000,Y ;Get first byte
DAC8   CMP   &DF0C,X ;Compare it with table byte
DACB   BNE   &DAFB   ;If not the same then goto DAFB
```

```
DACD   INY           ;Point to next byte
DACE   DEX ;(s)
DACF   BPL   &DAC5   ;and if still +ve go back to check next byte.
                     ;This point is reached if 4 bytes indicate
                     ;valid ROM
```

Next the first 1K of each ROM is checked against higher priority ROMs to ensure that there are no matches. If a match is found, the lower priority ROM is ignored.

A ROM type byte is located at &8006. A catalogue of these bytes is held at &2A1-&2B0. If bit 7 of this byte is 0 then the ROM is BASIC. The position of this ROM is stored at &24B.

Now the ROMs are catalogued it is time to set up the speech system and screen. More about that next week.

BBC 6502 Machine Code
Part Seven: Talk to me

The operating system start-up routines next checks the SPEECH system. At this point the X register is set to 16 (&10) by previous routines.

This is one of the reasons why this routine is inserted here. Setting X to the required value would use two more bytes. This is not much space but it can make the difference between all of the OS fitting into a single ROM and a complete hardware or software redesign.

```
DB11  BIT  &FE40  ;If bit 7 low then we have speech system fitted
DB14  BMI  &DB27  ;else goto DB27 for screen set up routine.
DB16  DEC  &027B  ;(027B)=&FF a RAM flag that indicates that a speech
                  ;chip is present.
DB19  LDY  #&FF   ;Y=&FF
DB1B  JSR  &EE7F  ;Initialise speech generator
DB1E  DEX         ;via this
DB1F  BNE  &DB19  ;loop
```

Now X = 0 so:

```
DB21  STX  &FE48  ;Set T2 timer for speech
DB24  STX  &FE49  ;
```

Screen set-up

X=0 on entry to this routine which gets the default screen mode and then goes off to the screen setup routine.

```
DB27  LDA  &028F  ;Get back start up options (mode)
DB2A  JSR  &C300  ;then jump to initialise screen.
```

One of the things that I wondered when I got a BBC was how the RESET key could possibly act as a soft key. As we all know BREAK acts as soft key 10. But the keyboard buffer is cleared by the Reset. Tucked away is the five-byte routine that makes the BREAK key act as soft key 10.

Soft keys work by inserting a byte greater than 127 into the keyboard buffer. &CA is the code for key 10.

```
DB2D  LDY  #&CA   ;Y=&CA
DB2F  JSR  &E4F1  ;to enter this value in the keyboard buffer
```

Simple isn't it? You can use the routine yourself although further investigation will show that E4F1 is part of an OSbyte call. Remember that the keyboard buffer is buffer 0.

```
E4F1  LDX  #&00  ;X=0 keyboard buffer

************************************
*                                  *
*   OSBYTE 153 Put byte in input   *
*    Buffer checking for ESCAPE    *
*                                  *
************************************
```

On entry X = buffer number which is either 0 or 1. If it's 0 then the keyboard buffer is selected. If it's 1 then it is the RS423 buffer.

Notice that the JSR to EF41 ensures that ONLY the keyboard buffer can be selected. Once again we are looking at coding economy, in this case with a specific keyboard buffer entry routine. Y contains the character to be written.

```
E4F3  TXA         ;A=buffer number
E4F4  AND  &0245  ;and with RS423 mode (0 treat as keyboard 1 ignore
                  ;Escapes no events no soft keys)
E4F7 BNE &E4AF    ;so if RS423 buffer AND RS423 in normal mode (1)
E4AF              ;
E4F9 TYA          ;else Y=A character to write
E4FA  EOR  &026C  ;compare with current escape ASCII code (0=match)
E4FD  ORA  &0275  ;or with current ESCAPE status (0=ESC, 1=ASCII)
E500  BNE  &E4A8  ;if ASCII or no match E4A8 to enter byte in buffer
E502  LDA  &0258  ;else get ESCAPE / BREAK action byte
E505  ROR         ;Rotate to get ESCAPE bit into carry
E506  TYA         ;get character back in A
E507  BCS  &E513  ;and if escape disabled exit with carry clear
E509  LDY  #&06   ;else signal EVENT 6 Escape pressed
E50B  JSR  &E494  ;
E50E  BCC  &E513  ;if event handles ESCAPE then exit with carry clear
E510  JSR  &E674  ;else set ESCAPE flag
E513  CLC         ;clear carry
E514  RTS         ;and exit
```

This routine will normally be accessed by assembly language programmers by OSbyte 138 which calls EF43.

BBC 6502 Machine Code
Part Eight: Breaker Break

One of the 'secret' features of the BBC Micro OS 1.20 when it was arrived was the BREAK intercept. This is a useful method of taking over the machine and is sometimes used by ROM software.

There are two entry points, entered with the carry flag reset to 0 and set to 1 respectively. The first call comes before sideways ROM calls.

Enter BREAK intercept with Carry Clear

```
DB32  JSR  &EAD9  ;check to see if BOOT address is set up if so
                  ;JMP to it
```

The address &287 is written by OSbyte 247 and the jump addresses in &288 and &289 by OSbytes 248 and 249. The machine code for JMP is &4C.

```
EAD9  LDA  &0287  ;get BREAK vector code
EADC  EOR  #&4C   ;produces 0 if JP (4C) not in &287
EADE  BNE  &EAF3  ;if not goto EAF3
EAE0  JMP  &0287  ;else jump to use
BREAK code
EAF3  RTS         ;Return
```

The RTS at the end of another routine is used because it saves code.

Frequently you will find machine code routines where a lot of branches go to a single RTS for just this reason. If you are writing your own code remember that the RTS must be within range of the branch. One of the most common assembler errors is a branch out of range that in turn causes more errors when you add an extra RTS.

Obviously at this point the machine could be totally in your control. You can return control to the OS with an RTS or just continue on your merry way.

Remember that the sideways ROMs don't have any workspace yet and you can't really run BASIC or any other language as the workspace will not exist. But, assuming that you don't want to do any of this, let's go back to the OS routines after testing for BREAK intercept.

```
DB35  JSR  &F140  ;set up cassette options
DB38  LDA  #&81   ;test for tube to FIFO buffer 1
DB3A  STA  &FEE0  ;
DB3D  LDA  &FEE0  ;
DB40  ROR         ;put bit 0 into carry
DB41  BCC  &DB4D  ;if no tube then DB4D
DB43  LDX  #&FF   ;else
DB45  JSR  &F168  ;issue ROM service call &FF to initialise TUBE system
DB48  BNE  &DB4D  ;if not 0 on exit (tube not initialised) DB4D
DB4A  DEC  &027A  ;else set tube flag to show its active
```

Now the Tube is flagged as active, or not as the case may be. We continue next week, with the setup routines for the sideways ROMs.

BBC 6502 Machine Code
Part Nine: A ROM with a view

Now we nearly have a working system, we are, perhaps, 400 milliseconds into the Power up routine. Now is the time to set up all of those nice sideways ROMs we catalogued earlier.

First we set up workspace and hence the value of BASIC's PAGE variable. The call to ROMs is made via F168. This is available to the programmer as OSBYTE 143.

A ROM can have a number between 0 and 15 and will have two entry points - a Service entry at &8003 and a Language entry at &8000. If the ROM does not contain language code it will not have a language entry.

ROMs are paged into main memory by writing the ROM number to a latch at &FE30. Hardware could be arranged to allow 256 ROMs although the operating system does not support this.

The Break Intercept code could be used to make drastic hardware modifications like this.

```
************************************
*                                  *
* OSBYTE 143                       *
* Pass service commands            *
* to sideways ROMs                 *
*                                  *
************************************
                  ;on entry X=command number
F168  LDA  &F4    ;get current ROM number
F16A  PHA         ;store it
F16B  TXA         ;command in A
F16C  LDX  #&0F   ;set X=15
```

The next bit of code is a countdown loop to send the command code to each enabled ROM in turn. The Map at &2A1 is used to decide which ROMs are active. Note the use of a countdown loop. This gives code economy and explains why the highest ROM number has priority.

```
F16E  INC  &02A1,X ;read bit 7 on ROM map (no ROM has type 254 &FE)
F171  DEC  &02A1,X ;
F174  BPL  &F183   ;if not set (+ve result)
F176  STX  &F4     ;else store ROM number in &F4
F178  STX  &FE30   ;switch in paged ROM
F17B  JSR  &8003   ;and jump to service entry
F17E  TAX          ;on exit put A in X
F17F  BEQ  &F186   ;if 0 (command recognised by ROM) reset ROMs & exit
F181  LDX  &F4     ;else point to next lower ROM
F183  DEX          ;
F184  BPL  &F16E   ;and go round loop again
F186  PLA          ;get back original ROM number
F187  STA  &F4     ;store it in RAM copy
F189  STA  &FE30   ;select original page
F18C  TXA          ;put X back in A
F18D  RTS          ;and return
```

Couldn't be easier! So we can now return to the main body of the routine.

```
DB4D  LDY  #&0E    ;set current value of PAGE
DB4F  LDX  #&01    ;issue call to claim absolute workspace
DB51  JSR  &F168   ;via F168
DB54  LDX  #&02    ;send private workspace claim call
DB56  JSR  &F168   ;via F168
```

OSHWM is OS High Water Mark. The highest address used by the operating system.

```
DB59  STY  &0243   ;set primary OSHWM DB5C
STY  &0244   ;set current OSHWM
DB5F LDX   #&FE    ;issue call for Tube to explode character set etc.
DB61  LDY  &027A   ;Y=FF if tube present else Y=0
DB64  JSR  &F168   ;and make call via F168
```

We now have the machine set up to enter a language, all the filing systems have been set up and the sideways ROMs activated.

Next week we finally start the screen messages.

BBC 6502 Machine Code
Part Ten: Stringing it along

The next routine shows why the Machine start up message is not always seen on third-party kit.

```
DB67  AND  &0267  ;if A=&FE and bit 7 of 0267 is set then continue
DB6A  BPL &DB87   ;else ignore start up message
DB6C  LDY  #&02   ;output to screen
DB6E  JSR  &DEA9  ;'BBC Computer ' message
```

Looking at the routine in DE9A we find a very useful string printing routine. Remember that Y = 2 on entry.

```
DEA9  LDA  #&C3   ;point to start &C300
DEAB  STA  &FE    ;store it
DEAD  LDA #&00   ;point to lo byte
DEAF  STA  &FD    ;store it and start loop with Y=2
DEB1  INY         ;print character in string
DEB2  LDA (&FD),Y ;pointed to by &FD/E +Y
DEB4  JSR  OSASCI ;print it expanding Carriage returns
```

```
DEB7  TAX ;store A in X
DEB8  BNE  &DEB1  ;and loop again if not =0
DEBA  RTS         ;else exit
```

Here is the string delimited by BRK. The code for BRK is 00. Y is 3 when the first character is read so its address is &C303.

```
C303  DB  13      ;Carriage Return
C304  DB 'BBC Computer '
C311  BRK
```

Notice that the routine uses TAX to set the zero flag which marks the end of the string. This is a useful tip.

The next part of the Operating system deals with printing correct messages on the screen.

```
DB71  LDA  &028D  ;0=warm reset, If a cold reset continue
DB74  BEQ  &DB82  ;
DB76  LDY  #&16   ;by checking length of RAM
DB78  BIT  &028E  ;
DB7B  BMI  &DB7F  ;and either
DB7D  LDY  #&11   ;
DB7F  JSR  &DEA9  ;finishing message with '16K' or '32K'
DB82  LDY  #&1B   ;and two new lines
DB84  JSR  &DEA9  ;
```

Notice that Y is used to pick the appropriate message.

```
C312  DB  '16K'
C315  DB  7       ;Bell
C316  BRK
C317  DB  '32K'
C31A  DB  7       ;Bell
C31B  BRK
C31C  DB  08,0D,0D
```

Notice the BBC Beep at this point indicates that nearly all set up procedures have been finished.

The hum is generated by the Sound channel which is reset as part of the start routine. Hence the HUM-BEEP start up. If the machine does not start properly the sound signals give a strong clue to the nature of the problem. Having got this far the OS gives us another chance to take control.

Enter BREAK INTERCEPT ROUTINE WITH CARRY SET (call 1)

```
DB87  SEC         ;
DB88  JSR  &EAD9  ;look for break intercept jump
                  ;SEE EARLIER PART
```

Next we set up the keyboard lights

```
DB8B  JSR  &E9D9  ;set up LEDs in accordance with keyboard status
```

This is another 'undocumented' OSBYTE call.

```
************************************
*                                  *
* OSBYTE &76 (118)                 *
* SET LEDs to Keyboard Status      *
*                                  *
************************************
;osbyte entry with carry set
E9D9  PHP ;PUSH P
```

```
E9DA  SEI         ;DISABLE INTERRUPTS
E9DB  LDA  #&40   ;switch on CAPS and SHIFT lock lights
E9DD  JSR  &E9EA  ;via subroutine
E9E0  BMI  &E9E7  ;if ESCAPE exists (M set) E9E7
E9E2  CLC         ;else clear V and C
E9E3  CLV         ;before calling main keyboard routine to
E9E4  JSR  &F068  ;switch on lights as required
E9E7  PLP ;get back flags
E9E8  ROL         ;and rotate carry into bit 0
E9E9  RTS ;Return to calling routine
                  ;
* Turn on keyboard lights and
* Test Escape flag
                  ;
E9EA  BCC  &E9F5  ;if carry clear
E9EC  LDY  #&07   ;switch on shift lock light
E9EE  STY  &FE40  ;
E9F1  DEY         ;Y=6
E9F2  STY  &FE40  ;switch on Caps lock light
E9F5  BIT  &FF    ;set minus flag if bit 7 of &00FF is set indicating
E9F7  RTS         ;that ESCAPE condition exists, then return
```

The Keyboard routine continues via the KEYV. This is a little long to include here so we'll leave it until a later part. So back to the Start up routine next week with the cassette system.

BBC 6502 Machine Code
Part Eleven: Language!

Having got the keyboard nicely set up the machine proceeds to initialise a filing system and run a !BOOT file if one exists. The start up options are already read from the keyboard links.

```
DB8E  PHP         ;save flags
DB8F  PLA         ;and get back in A
DB90  LSR ;zero bits 4-7 and bits 0-2 bit 4 which was bit 7
DB91  LSR         ;may be set
DB92  LSR         ;
DB93  LSR         ;
DB94  EOR  &028F  ;EOR with start up options which may or may not
DB97  AND  #&08   ;invert bit 4
DB99  TAY  ;Y=A
DB9A  LDX  #&03   ;make initialisation call if Y=0 on entry
DB9C  JSR  &F168 ;RUN, EXEC or LOAD !BOOT file from a filing system.
DB9F  BEQ  &DBBE  ;if a ROM accepts this call then
DBBE
DBA1  TYA         ;else put Y in A
DBA2  BNE  &DBB8  ;if Y<>0 DBB8
DBA4  LDA  #&8D   ;else set up standard cassette baud rates
DBA6  JSR  &F135  ;via &F135 which is OSBYTE 140.
DBA9  LDX  #&D2   ;
DBAB  LDY  #&EA   ;
DBAD  DEC  &0267  ;decrement ignore start up message flag
DBB0  JSR  OSCLI  ;and execute /!BOOT
DBB3  INC  &0267  ;restore start up message flag
DBB6  BNE  &DBBE  ;if not zero then DBBE
DBB8  LDA  #&00   ;else A=0
DBBA  TAX  ;X=0
DBBB  JSR  &F137  ;set tape speed via OSBYTE 140.
```

We now have an active filing system. The next job is to preserve the current language on soft RESET.

```
DBBE  LDA  &028D  ;get last RESET Type
DBC1  BNE  &DBC8  ;if not soft reset
DBC8
```

```
DBC3  LDX  &028C  ;else get current language ROM address
DBC6  BPL  &DBE6  ;if +ve (language available) then skip search
                  ;routine
```
For a cold break we search for the language with the highest priority.

```
DBC8  LDX  #&0F   ;set pointer to highest available ROM
DBCA  LDA &02A1,X ;get ROM type from map
DBCD  ROL         ;put hi-bit into carry, bit 6 into bit 7
DBCE  BMI  &DBE6  ;if bit 7 set then ROM has a language entry so DBE6
DBD0  DEX         ;else search for language until X=&ff
```

Check for Tube if no language found.

```
DBD1  BPL  &DBCA  ;check if tube present
DBD3  LDA  #&00   ;if bit 7 of tube flag is set BMI succeeds
DBD5  BIT  &027A  ;and TUBE is connected else
DBD8  BMI &DC08  ;make error
```

No language error

```
DBDA  BRK            ;
DBDB  DB   &F9       ;error number
DBDC  DB 'Language?' ;message
DBE5  BRK            ;
```

This might seem odd as BRK is handled by the current language BRK handler,but we don't have a language! We need to investigate further in another part.

```
DBE6  CLC         ;
```

OSBYTE 142 enter Language ROM at &8000 X=ROM number. Carry is set if this is an OSBYTE call and clear if this is an initialisation routine.

```
DBE7  PHP         ;save flags
DBE8  STX  &028C  ;put X in current ROM page
DBEB  JSR  &DC16  ;select that ROM
DBEE  LDA  #&80   ;A=128
DBF0  LDY  #&08   ;Y=8
DBF2  JSR  &DEAB  ;display text string held in ROM at &8008,Y
DBF5  STY  &FD    ;save Y on exit (end of language string)
DBF7  JSR  OSNEWL ;two line feeds
DBFA  JSR  OSNEWL ;are output
DBFD  PLP         ;then get back flags
DBFE  LDA  #&01   ;A=1 required for language entry
DC00  BIT  &027A  ;check if tube exists
DC03  BMI  &DC08  ;and goto DC08 if it does
DC05  JMP  &8000  ;else enter language at &8000
```

TUBE FOUND enter tube software

```
DC08  JMP  &0400  ;enter tube environment
```

The Tube initialisation would have read the language across to the TUBE usually but it could be loaded by a !BOOT file from the filing system initialisation.

The operating system now stops general control of the system and hands this to the language which looks after command lines etc. The OS however still handles the screen, keyboard and much else.

Notice how every possible eventuality was taken into account during the initialisation routine. This is one of the things that made the Beeb a very powerful machine.

Next week we'll have a look at the Interrupt code.

BBC 6502 Machine Code
Part Twelve: Pardon me!

We finished the last part at the point where the operating systems power up routine handed over control to the language. We'll write our own language later in the series but for now let's dive into another entry point.

When the processor's RQ pin (4) goeslow (0V) the processor finishes off the current instruction and then goes off to run some microcode of its own. This checks that the RDY (2) pin is high and that the interrupt flag in the status register is 0 (not set). If it is set the interrupt is ignored and the processor goes to the next instruction. This continues when the IRQ pin is low.

If the flag is clear then the processor stores the program counter and status register on the stack and sets the interrupt flag. The 6502 then gets the address stored in &FFFE and &FFFF and executes this instruction next.

If a BRK instruction is found in executing code then the processor performs exactly the same actions except that it does not check the status register for the interrupt flag, it does set a flag in the status register, the BRK flag.

The main entry point for IRQ (and BRK) for OS 1.20 is &DC51.

MAIN IRQ Entry point

```
;ON ENTRY STACK contains STATUS REGISTER,PCH,PCL
DC1C  STA  &FC    ;save A
DC1E  PLA         ;get back status (flags)
DC1F  PHA         ;and save again
DC20  AND  #&10   ;check if BRK flag set
DC22  BNE  &DC27  ;if so goto DC27
DC24  JMP (&0204) ;else JUMP through IRQ1V
```

That's pretty straightforward so far. As you can see IRQ1V allows you to put your own hardware at a higher priority than anything else in the machine.

You can also write your own hardware interrupt handler if you wish. This is the flexibility that made the BBC machine so remarkably successful among knowledgeable users.

Let's look at the BRK handler now.

```
* BRK handling routine *
DC27  TXA         ;save X on stack
DC28  PHA         ;
DC29  TSX         ;get status pointer
DC2A  LDA &0103,X ;get Program Counter low byte
DC2D  CLD         ;
DC2E  SEC         ;set carry
DC2F  SBC  #&01 ;subtract 2 (1+carry)
DC31  STA  &FD    ;and store it in &FD
DC33  LDA &0104,X ;get hi byte
DC36  SBC  #&00   ;subtract 1 if necessary
DC38  STA  &FE    ;and store in &FE
DC3A  LDA  &F4    ;get currently active ROM
DC3C  STA  &024A  ;and store it in &24A
DC3F  STX  &F0    ;store stack pointer in &F0
DC41  LDX  #&06   ;and issue ROM service call 6
DC43  JSR  &F168  ;(User BRK) to ROMs
                  ;now &FD/E points to byte after BRK
```

```
                  ;ROMS may use BRK for their own purposes
                  ;and many do!
```

It's interesting to see what happens with the ROM handler. This is also an entry point for OSBYTE 143 so you can use this in your own code.

```
* OSBYTE 143 *
*Pass service commands to sideways ROMs *
                  ;on entry X=command number
F168  LDA &F4     ;get current ROM number
F16A  PHA         ;store it
F16B  TXA         ;command in A
F16C  LDX  #&0F   ;set X=15
                  ;send commands loop
F16E  INC &02A1,X ;read bit 7 on ROM map (no ROM has ;type 2)
4 &FE)
F171  DEC &02A1,X ;
F174  BPL &F183   ;if not set (+ve result)
F176  STX  &F4    ;else store ROM number in &F4
F178  STX  &FE30  ;switch in paged ROM
F17B  JSR  &8003  ;and jump to service entry
F17E  TAX         ;on exit put A in X
F17F  BEQ  &F186  ;if 0 (command recognised by ROM)
                  ;reset ROMs & exit
F181  LDX  &F4    ;else point to next lower ROM
F183  DEX         ;
F184  BPL  &F16E  ;and go round loop again
F186  PLA         ;get back original ROM number
F187  STA  &F4    ;store it in RAM copy
F189  STA  &FE30  ;select original page
F18C  TXA         ;put X back in A
F18D  RTS         ;and return
```

Useful little routine that. So back to the BRK handler.

```
DC46  LDX  &028C  ;get current language
DC49  JSR  &DC16  ;and activate it
DC4C  PLA         ;get back original value of X
DC4D  TAX         ;
DC4E  LDA  &FC    ;get back original value of A
DC50  CLI         ;allow interrupts
DC51  JMP (&0202) ;and JUMP via BRKV (normally into current language)
```

Next week we'll carry on by taking a look at the BRK handler.

BBC 6502 Machine Code
Part Thirteen: Give us a BRK

BRK is usually handled by the default language (or by a Sideways ROM). However, it may be that you are running a machine code program before a current language is set up or perhaps your language doesn't handle BRK (it should but you never know).

That's when a default BRK handler takes over.

```
* DEFAULT BRK HANDLER *

DC54 LDY #&00  ;Y=0 to point to byte after BRK
DC56 JSR &DEB1 ;print message
```

Let's have a look at the print routine. Remember that the error-handling layout is:

```
BRK
Error Number (1 byte)
```

Message
BRK

Y plus the address in &FD &FE points to the error message on entry.

```
DEB1 INY          ;point to first ;character in string
DEB2 LDA (&FD),Y
DEB4 JSR OSASCI   ;print it
                  ;expanding
                  ;Carriage
                  ;returns
DEB7 TAX          ;store A in X to change flags
DEB8 BNE &DEB1    ;and loop again if not =0
DEBA RTS          ;else exit
```

A standard print routine, nothing out of the ordinary but nice and compact.

You can use this in your own print routines by changing the zero page values. Back to the default BRK handler and an interesting bit of code.

```
DC59 LDA &0267 ;if BIT 0 set and DISK EXEC error
DC5C ROR       ;occurs
DC5D BCS &DC5D ;hang up machine!
```

Nasty! But the machine has to be in a pretty unusual configuration for this to happen. Mind you, setting 0267 then doing a JSR to DC59 would confuse the average user.

```
DC5F JSR OSNEWL ;else print two newlines
DC62 JSR OSNEWL ;
DC65 JMP &DBB8  ;and set tape speed before entering the current
                ;language
DBB8 LDA #&00   ;else A=0
DBBA TAX        ;X=0
DBBB JSR &F137  ;set tape speed via OSBYTE 141.
```

There's the end of the BRK handling code. As I said before this is generally handled by the default language but you can arrange for your own code or a Sideways ROM to handle it.

Next week we'll return to the interrupt system with a look at the default entry point for IRQ1.

BBC 6502 Machine Code
Part Fourteen: The story so far...

We left the interrupt-handling routine just after it had gone off to the IRQ1V vector. If you don't change the vector the code continues from DC93.

One very important thing to remember about an interrupt-driven machine like the BBC is that the interrupt flag is not set for too long. If it is the machine could crash. This means that interrupt routines are short and snappy.

* Main IRQ Handling routines, default IRQIV destination *

```
DC93 CLD      ;clear decimal flag
DC94 LDA &FC  ;get original value of A
DC96 PHA      ;save it
DC97 TXA      ;save X
DC98 PHA      ;
DC99 TYA      ;and Y
DC9A PHA      ;on the stack
```

```
              ;note the pre-CMOS code!
DC9B LDA #&DE ;A=&DE
DC9D PHA      ;store it
DC9E LDA #&81 ;save &81
DCA0 PHA      ;store it (a RTS will now jump to DE82)
```

This is quite a useful technique as we will see later. If we now use JMP to go to an OS routine we can ensure that the routine, which ends with an RTS, causes execution to go to a specified point.

This saves a lot of code as it can be arranged that the first device found that called the interrupt will be the only one handled. This, in turn, saves time!

We now poll the hardware looking for who caused it. The first routine deals with the serial/tape system.

```
DCA1 CLV       ;clear V flag
DCA2 LDA &FE08 ;get value of status register of ACIA
DCA5 BVS &DCA9 ;if this was source then DCA9 to process
DCA7 BPL &DD06 ;else if no interrupt requested DD06
DCA9 LDX &EA   ;read RS423 timeout counter
DCAB DEX       ;decrement it
DCAC BMI &DCDE ;and if <0 DCDE
DCAE BVS &DCDD ;else if >&40 DCDD (RTS to DE82)
DCB0 JMP &F588 ;else read ACIA via F588
               ;RTS ends routine!!
DCB3 LDY &FE09 ;read ACIA data
DCB6 ROL       ;
DCB7 ASL       ;
DCB8 TAX       ;X=A
DCB9 TYA       ;A=Y
DCBA LDY #&07  ;Y=07
DCBC JMP &E494 ;check and service EVENT 7 RS423 error
DCBF LDX #&02  ;read RS423 output buffer
DCC1 JSR &E460 ;
DCC4 BCC &DCD6 ;if C=0 buffer is not empty goto DCD6
DCC6 LDA &0285 ;else read printer destination
DCC9 CMP #&02  ;is it serial printer??
DCCB BNE &DC68 ;if not DC68
DCCD INX       ;else X=3
DCCE JSR &E460 ;read printer buffer
DCD1 ROR &02D2 ;rotate to pass carry into bit 7
DCD4 BMI &DC68 ;if set then DC68
DCD6 STA &FE09 ;pass either printer or RS423 data to ACIA
DCD9 LDA #&E7  ;set timeout counter to stored value
DCDB STA &EA   ;
DCDD RTS       ;and exit (to DE82)

               ;A contains ACIA status
DCDE AND &0278 ;AND with ACIA bit mask (normally FF)
DCE1 LSR       ;rotate right to put bit 0 in carry
DCE2 BCC &DCEB ;if carry clear receive register not full so DCEB
DCE4 BVS &DCEB ;if V is set then DCEB
DCE6 LDY &0250 ;else Y=ACIA control setting
DCE9 BMI &DC7D ;if bit 7 set receive interrupt is enabled so DC7D

DCEB LSR       ;put BIT 2 of ACIA status into
DCEC ROR       ;carry if set then Data Carrier Detected applies
DCED BCS &DCB3 ;jump to DCB3

DCEF BMI &DCBF ;if original bit 1 is set TDR is empty so DCBF
DCF1 BVS &DCDD ;if V is set then exit to DE82

DCF3 LDX #&05  ;X=5
DCF5 JSR &F168 ;issue ROM call 5 'unrecognised ;interrupt'
```

We've seen this ROM service routine call before.

```
DCF8 BEQ &DCDD   ;if a ROM recognises it then exit to DE82
DCFA PLA         ;otherwise get back DE82 address from stack
DCFB PLA         ;
DCFC PLA         ;and get back X, Y and A
DCFD TAY         ;
DCFE PLA         ;
DCFF TAX         ;
DD00 PLA         ;
DD01 STA &FC     ;&FC=A
DD03 JMP (&0206) ;and offer to the user via IRQ2V
```

That was a little convoluted, to say the least. Next week we look at how the VIAs are dealt with.

BBC 6502 Machine Code
Part Fifteen: Hardware VIA interrupts

After deciding that it wasn't the ACIA that caused the interrupt, the VIAs are the next port of inquisition.

```
* VIA INTERRUPTS ROUTINES *

DD06 LDA &FE4D ;read system VIA interrupt flag register
DD09 BPL &DD47 ;if bit 7=0 the VIA has not caused interrupt goto DD47

DD0B AND &0279 ;mask with VIA bit mask
DD0E AND &FE4E ;and interrupt enable register
DD11 ROR       ;rotate right twice to ;check for IRQ 1 (frame sync)

DD12 ROR       ;
DD13 BCC &DD69 ;if carry clear then no IRQ 1, else IRQ 1 means
               ;interrupt request 1. This is different from the
               ;vector IRQ1.

DD15 DEC &0240 ;decrement vertical sync counter
DD18 LDA &EA   ;A=RS423 Timeout counter
DD1A BPL &DD1E ;if +ve then DD1E
DD1C INC &EA   ;else increment it
DD1E LDA &0251 ;load flash character counter
DD21 BEQ &DD3D ;if 0 then flash system is not in use, ignore it
DD23 DEC &0251 ;else decrement counter
DD26 BNE &DD3D ;and if not 0 go on past reset routine
```

This routine resets the flashing character system.

```
DD28 LDX &0252 ;get mark period count in X
DD2B LDA &0248 ;current VIDEO ULA control setting in A
DD2E LSR       ;shift bit 0 into C to ;check if first colour
DD2F BCC &DD34 ;is effective if so C=0. Jump to DD34
DD31 LDX &0253 ;else get space period count in X
DD34 ROL       ;restore bit
DD35 EOR #&01  ;and invert it
DD37 JSR &EA00 ;then change colour

DD3A STX &0251 ;&0251=X resetting the counter

DD3D LDY #&04  ;Y=4 and call E494 to check and implement vertical
DD3F JSR &E494 ;sync event (4) if necessary
DD42 LDA #&02  ;A=2
DD44 JMP &DE6E ;clear interrupt 1 and exit
```

Remember the RTS routine last time?

```
* PRINTER INTERRUPT USER VIA 1 *

DD47 LDA &FE6D ;Check USER VIA interrupt flags register
DD4A BPL &DCF3 ;if +ve USER VIA did not call interrupt
DD4C AND &0277 ;else check for USER IRQ 1 printer interrupt.
DD4F AND &FE6E ;
DD52 ROR       ;
DD53 ROR       ;
DD54 BCC &DCF3 ;if bit 1=0 then no ;interrupt 1 so DCF3
DD56 LDY &0285 ;else get printer type
DD59 DEY       ;decrement
DD5A BNE &DCF3 ;if not parallel then :CF3
DD5C LDA #&02  ;reset interrupt 1 flag
DD5E STA &FE6D ;
DD61 STA &FE6E ;disable interrupt 1
DD64 LDX #&03  ;and output data to parallel printer
DD66 JMP &E13A ;and exit via RTS

* SYSTEM INTERRUPT 5 Speech *

DD69 ROL       ;get bit 5 into bit 7
DD6A ROL       ;
DD6B ROL       ;
DD6C ROL       ;
DD6D BPL &DDCA ;if not set this is not ;a speech interrupt so DDCA
DD6F LDA #&20  ;
DD71 LDX #&00  ;
DD73 STA &FE4D ;
DD76 STX &FE49 ;and zero high byte of Timer t2
DD79 LDX #&08  ;&FB=8
DD7B STX &FB   ;
DD7D JSR &E45B ;and examine buffer 8
DD80 ROR &02D7 ;shift carry into bit 7
DD83 BMI &DDC9 ;and if set buffer is empty so exit
DD85 TAY       ;else Y=A
DD86 BEQ &DD8D ;
DD88 JSR &EE6D ;control speech chip
DD8B BMI &DDC9 ;if negative exit
DD8D JSR &E460 ;else get a byte from buffer
DD90 STA &F5   ;store it to indicate speech or file ROM
DD92 JSR &E460 ;get another byte
DD95 STA &F7   ;store it
DD97 JSR &E460 ;and another
DD9A STA &F6   ;giving address to be accessed in paged ROM
DD9C LDY &F5   ;Y=&F5
DD9E BEQ &DDBB ;and if =0 then DDBB
DDA0 BPL &DDB8 ;else if +ve DDB8
DDA2 BIT &F5   ;if bit 6 of F5 =1 (&F5)>&40
DDA4 BVS &DDAB ;then DDAB
DDA6 JSR &EEBB ;else continue for more speech processing
DDA9 BVC &DDB2 ;if bit 6 clear then DDB2
DDAB ASL &F6   ;else double address in &F6/7
DDAD ROL &F7   ;
DDAF JSR &EE3B ;and call EE3B
DDB2 LDY &0261 ;get speech enable/disable flag into Y
DDB5 JMP &EE7F ;and JMP to EE7F

DDB8 JSR &EE7F ;Call EE7F
DDBB LDY &F6   ;get address pointer in Y
DDBD JSR &EE7F ;
DDC0 LDY &F7   ;get address pointer high in Y
DDC2 JSR &EE7F ;
DDC5 LSR &FB   ;
DDC7 BNE &DD7D ;
DDC9 RTS       ;and exit
```

Next week we continue with a look at the remaining System Interrupts.

BBC 6502 Machine Code
Part Sixteen: Timers and Keyboard Interrupts

The last part showed how the BBC Micro handles some of the system interrupt calls. Most of these are pretty routine so we won't continue with an interminable list.

The next interesting routines concern how the timers and keyboard interrupts are handled.

```
* SYSTEM INTERRUPT 6 10mS Clock *

DDCA  BCC  &DE47  ;bit 6 is in carry so if clear there is no 6 so go
                  ;on to DE47
DDCC  LDA  #&40   ;Clear interrupt 6
DDCE  STA  &FE4D  ;
```

This is the start of the update timers routine, This is interesting because of the way that the timer information is stored. It's very clever. There are two timer stores, &292-6 and &297-B. These are updated by adding 1 to the current timer and storing the result in the other, the direction of transfer being changed each time of update.

This ensures that at least one timer is valid at any call as the current timer only is read. Other methods would cause inaccuracies if a timer was read while being updated.

```
DDD1  LDA  &0283   ;get current system clock store pointer (5,or 10)
DDD4  TAX          ;put A in X
DDD5  EOR  #&0F    ;and invert lo nybble (5becomes 10 and vv)
DDD7  PHA          ;store A
DDD8  TAY          ;put A in Y. Carry is always set at this point
DDD9  LDA  &0291,X ;get timer value
DDDC  ADC   #&00   ;update it
DDDE  STA  &0291,Y ;store result in alternate
DDE1  DEX          ;decrement X
DDE2  BEQ  &DDE7   ;if 0 exit
DDE4  DEY          ;else decrement Y
DDE5  BNE  &DDD9   ;and go back and do next byte
DDE7  PLA          ;get back A
DDE8  STA  &0283   ;and store back in clock pointer (ie. inverse
                   ;previous contents)
DDEB  LDX  #&05    ;set loop pointer for countdown timer
DDED  INC  &029B,X ;increment byte and
DDF0  BNE  &DDFA   ;if not 0 then DDFA
DDF2  DEX          ;else decrement pointer
DDF3  BNE  &DDED   ;and if not 0 do it again
DDF5  LDY  #&05    ;process EVENT 5 interrupt timer
DDF7  JSR  &E494   ;
DDFA  LDA  &02B1   ;get byte of inkey countdown timer
DDFD  BNE  &DE07   ;if not 0 then DE07
DDFF  LDA  &02B2   ;else get next byte
DE02  BEQ  &DE0A   ;if 0 DE0A
DE04  DEC  &02B2   ;decrement 2B2
DE07  DEC  &02B1   ;and 2B1
DE0A  BIT  &02CE   ;read bit 7 of envelope processing byte
DE0D  BPL  &DE1A   ;if 0 then DE1A
DE0F  INC  &02CE   ;else increment to 0
DE12  CLI          ;allow interrupts
DE13  JSR  &EB47   ;and do routine sound processes
DE16  SEI          ;bar interrupts
DE17  DEC  &02CE   ;DEC envelope processing byte back to 0
DE1A  BIT  &02D7   ;read speech buffer busy flag
DE1D  BMI  &DE2B   ;if set speech buffer is empty, skip routine
```

```
DE1F  JSR  &EE6D   ;update speech system variables
DE22  EOR  #&A0    ;
DE24  CMP  #&60    ;
DE26  BCC  &DE2B   ;if result >=&60 DE2B
DE28  JSR  &DD79   ;else more speech work
DE2B  BIT  &D9B7   ;set V and C
DE2E  JSR  &DCA2   ;check if ACIA needs attention
DE31  LDA  &EC     ;check if key has been pressed
DE33  ORA  &ED     ;
DE35  AND  &0242   ;(this is 0 if keyboard is to be ignored, else
                   ;&FF)
DE38  BEQ  &DE3E   ;if 0 ignore keyboard
DE3A  SEC          ;else set carry
DE3B  JSR  &F065   ;and call keyboard
DE3E  JSR  &E19B   ;check for data in use defined printer channel
DE41  BIT  &FEC0   ;if ADC bit 6 is set ADC is not busy
DE44  BVS  &DE4A   ;so DE4A
DE46  RTS          ;else return

* SYSTEM INTERRUPT 4 ADC end of conversion *

DE47  ROL          ;put original bit 4 from FE4D into bit 7 of A
DE48  BPL  &DE72   ;if not set DE72
DE4A  LDX  &024C   ;else get current ADC channel
DE4D  BEQ  &DE6C   ;if 0 DE6C
DE4F  LDA  &FEC2   ;read low data byte
DE52  STA  &02B5,X ;store it in &2B6,7,8 or 9
DE55  LDA  &FEC1   ;get high data byte
DE58  STA  &02B9,X ;and store it in hi byte
DE5B  STX  &02BE   ;store in Analogue system flag marking last channel
DE5E  LDY  #&03    ;handle event 3 conversion complete
DE60  JSR  &E494   ;
DE63  DEX          ;decrement X
DE64  BNE  &DE69   ;if X=0
DE66  LDX  &024D   ;get highest ADC channel present
DE69  JSR  &DE8F   ;and start new conversion
DE6C  LDA  #&10    ;reset interrupt 4
DE6E  STA  &FE4D   ;
DE71  RTS          ;and return

* SYSTEM INTERRUPT 0 Keyboard *

DE72  ROL         ;get original bit 0 in bit 7 position
DE73  ROL         ;
DE74  ROL         ;
DE75  ROL         ;
DE76  BPL  &DE7F  ;if bit 7 clear not a keyboard interrupt
DE78  JSR  &F065  ;else scan keyboard
DE7B  LDA  #&01   ;A=1
DE7D  BNE  &DE6E  ;and off to reset interrupt and exit
DE7F  JMP  &DCF3  ;and again a subroutine to exit.
```

Now we come to the point you've all been waiting for. This mystery RTSreturns all subroutines to &DE82.

```
************** exit routine
DE82  PLA         ;restore registers
DE83  TAY         ;
DE84  PLA         ;
DE85  TAX         ;
DE86  PLA         ;
DE87  STA   &FC   ;store A

* IRQ2V default entry *

DE89  LDA   &FC   ;get back original value of A
```

```
DE8B  RTI         ;and return to calling routine.
```

NEXT WEEK: OSBYTE entry.

BBC 6502 Machine Code
Part Seventeen: The BBC Operating System

We've been examining the BBC operating system in some detail over the last few weeks. Unfortunately the demise of Micronet means that we cannot finish completely, as we hoped. So we've put together the next twenty weeks' articles in the form of a completely commented disassembly of OS 1.20.

This is an excellent example of BBC programming and is full of tips.

Just to remind you of the main points of the software. Entry points are pointed to by a jump table in the last six bytes of the ROM.

The font characters are located from &C000 to &C2FF.

OK, so here it is all commented and ready for you to peruse.

Ed says: I have uploaded the series of disassembly articles as ten short TSW files. Look on Micronet on 700100239 (before it's too late!)

ÿ

********** THE END **********

▯