Firmware Calling

Your Auntie is still waiting for some input. This month he tries asking the firmware to help him.

Greetings fellow machine coders! If you remember our last exciting visit to the wonderful world of mnemonics, we actually managed to write a program which could put something on the screen.

It worked by using the firmware calls embedded in the operating system, or in slightly plainer English, we cheated. We used the routines that the CPC designers thoughtfully left lying around in the ROM. And why not? That's what they're there for.

To recap, when we wanted to output a character to the screen, all we did was load the accumulator (the 'a' register) with its ASCII value and call the routine at &BB5D: The firmware did the rest.

You may guess that getting input for a program from the keyboard is also possible by using the firmware. And you'd be right!

I'm going to show you how you can perform the machine code equivalent of BASIC's INKEY() statement. This routine checks to see if a key is being held down at that precise moment in time. A bit like me, it doesn't wait around if nothing is happening.

To demonstrate the key reading, we'll also make use of two or three other firmware routines. These are GET_INK, SET_INK and another one that's a secret for the moment.

To understand how GET_INK works, you'll need to know a little about how the CPC treats colours. Our beloved Arnold can display twenty seven different colours, but not all at once. Instead it uses "pens" , the number of which depends on the screen mode. In mode 1, there are four pens. One of these pens is the background colour, one the default colour of text and the other two can be anything you want.

The clever thing about these pens is that that can be different colours. Although the text may be white now, you can change it to whatever colour (of the remaining 26) you like. You can even make it the same colour as the background, and so hide it.

To change the colour of a pen, we first find out what colour it is at the moment by calling GET_INK. This routine needs the pen number to be supplied in the accumulator: In the example program, we chose pen 0 – the background colour.

GET_INK returns the two colours in the registers B & C. "Hold on there", you might say, "Did you just say two colours? Why two colours? Surely a pen can hold only one colour of ink?"

"Ah"', says I, "but this is a computer and not constrained by the physical laws of mere mortals". But then, I say that kind of thing a lot. Which explains why I spend a lot of time in the kitchen at parties.

If you remember, the CPC can display a flashing ink on the screen..

Don't believe me? Then try typing this line directly from BASIC:

```
INK 0,20,0
```

You'll find that all the text on screen suddenly starts to flash. It's changing between ink number 20 and ink number 0. You can see that it's really the non-flashing colours which are a special case. If a colour is not flashing, it's just changing between two colours which are the same. See?

Anyway, all this means is that each pen has two ink colours associated with it, and GET_INK puts them into B and C. Our example program will change the pens, by making them cycle both up and down through the ink colours. On a green screen monitor this gives a wonderful impression of fading in and out. In colour, it just looks weird.

We change the ink value by changing B and C, then calling the routine SET_INK. We need to specify which pen again, as the GET_INK routine messed up the first zero we placed in the accumulator. That's an important point: Never assume a routine will preserve the contents of the registers unless told otherwise.

So to increase the pen number we have to increase B and C. You might say, "That's easy: ink bc". You might also cheat and look at the listing. If you do, you'll see that instead of ink bc, we used two instructions: ink b and ink c. Is there a reason for this?

Of course there's a reason. The ink bc instruction assumes you're treating the b and c registers as a pair. That is, a single register that can hold a number between 0 and 65535 (see the last few back issues for details on how and why this works), which of course, we're not. We're treating them as separate registers, one for each colour.

Can you guess what ink bc would do? If you're on the ball, you'll know that it will increase the c register by one, until it reaches 255. Then it will reset it to zero, and increment b. You can see that this would allow the ink numbers to change in a really weird way, and not in step with one another. They would start flashing and all sorts of weirdness would result. Which is why we don't use it.

Let's have a look at the example listing now. It should be typed into your assembler and assembled to run at address &8000 (that's what the ORG statement is all about). Remember to set himem (where BASIC can stop, so as not to overwrite the code) with a MEMORY &7FFf before you begin.

The program starts by assigning our firmware routines and constants to slightly more memorable names. Would you rather remember the number &BB1E or the text TEST_KEY? I rest my case.

Once past three equates, we get into the main loop. The part of the program between .loop and jr loop will repeat indefinitely.

Well, almost, because we check for the space bar being pressed, and if it is, we do a RETURN to BASIC. We check for the space bar using TEST_KEY, which must be supplied with a key number in the accumulator. Now don't confuse the key number with an ASCII value. They have nothing in common except that they can both confuse you.

The key numbers is a way of giving each and every key on the keyboard a unique reference number. The exceptions are the SHIFT keys, which share the same number. You may have come across them when playing with the BASIC keyword INKEY.

These keys numbers are detailed in your manual, or if you have a 6128 they are printed on the disk drive along with the colours. Useful, eh?

TEST_KEY takes this number, looks at the key to see if it's being held down and then changes the Z flag. Remember this flag? It normally detects a zero or a truth in a compare instruction. In this case, if it is SET, then a key is NOT being held down. If it is RESET, then the key IS being held down..

You can see that the three lines after .loop will test for the space bar, and if it's being pressed will do a return. That's how we get back to BASIC.

The next three lines test for the cursor upkey. If it is being pressed, the subroutine COLOUR_UP is called. This one changes the ink number as we did and then returns. In this case, because the subroutine was CALLed, the RET will not go toBASIC, but from where it was called from. The rest of the program you should be able to work out for yourself!

If you have entered and assembled and run this program, you may find that when you press the up and down keys, instead of getting a nice change in colour, you get an instant headache. The reason for this is that the program is too fast. It changes the colours so quickly that you don't have time to notice them. Isn't machine code wonderful? When did you last write a BASIC program that ran too quickly?

To slow our program down, we can use the secret firmware routine FRAME. Put the line CALL FRAME just before JR LOOP. If you now run the program, it will run so much more smoothly.

This is because FRAME causes the program to wait for a bit, and synchronises itself with the TV picture. The result is a much smoother display. In fact, a display which is ideal for moving graphics around. This is quite handy, because moving graphics around is exactly what we'll be doing next month!

See you then!

The Program:

```
TEST_KEY     equ &BB1E
GET_INK      equ &BC35
SET_INK      equ &BC32
FRAME        equ &BD19
SPACE        equ 47
KEY_UP       equ 0
KEY_DOWN     equ 2
ORG &8000 ; Start address of program

.loop ld a,SPACE
call TEST_KEY
ret nz

ld a,KEY_UP
call TEST_KEY
call nz,COLOUR_UP

ld a, key_DOWN
call TEST_KEY
call nz,COLOUR_DOWN

jr loop

.COLOUR_UP ;Cycle colours up
        ld a,0
        call GET_INK
        ld a,0
        inc b:inc c
        call SET_INK
        ret

.COLOUR_DOWN
        ld a0 ;Cycle colours down
        call GET_INK
        ld a,0
        dec b:dec c
        call SET_INK
        ret
```

Well, that's your lot for this month, so keep playing with your firmware until the next exciting instalment.