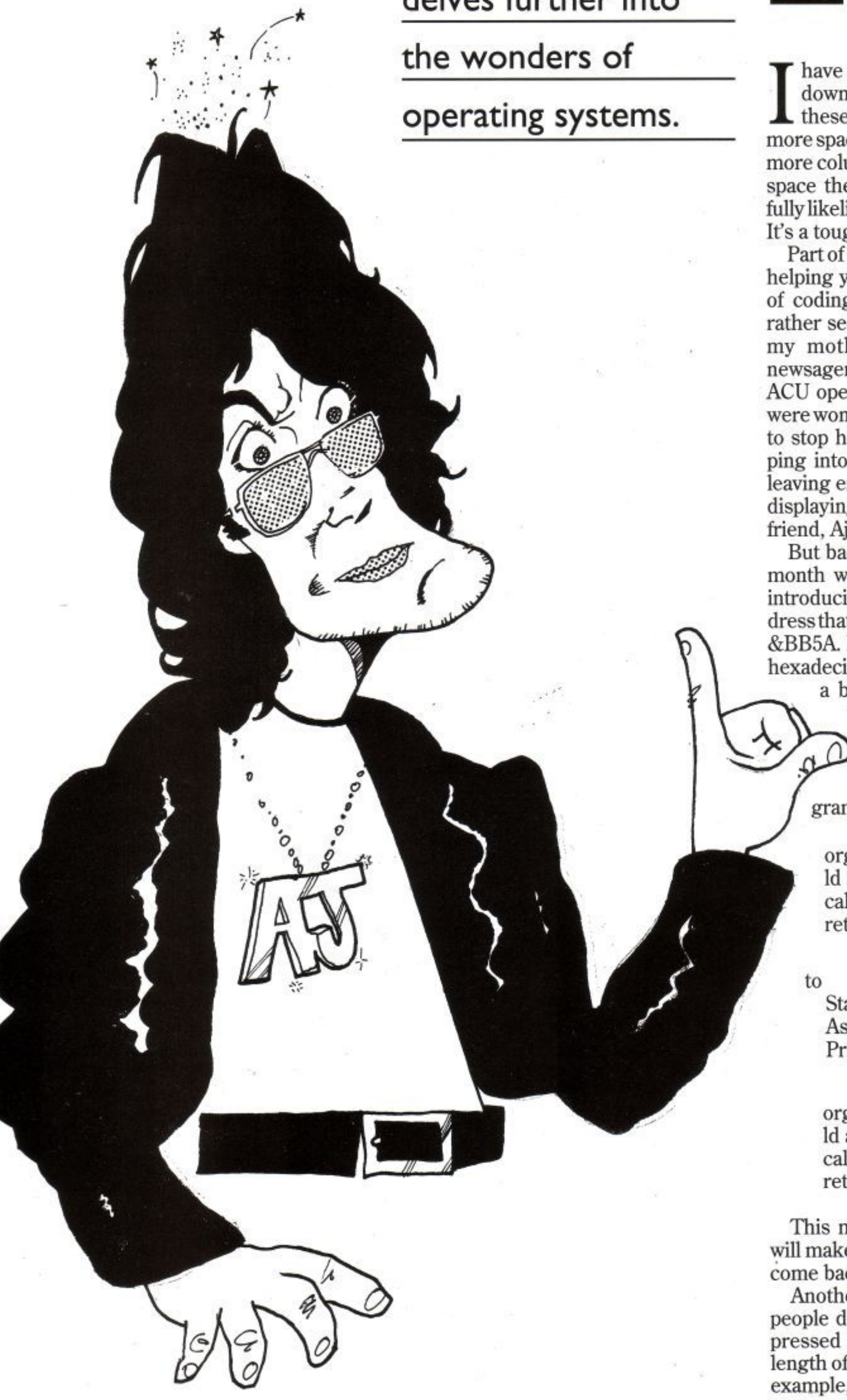
Librari

Build your own library as Auntie John delves further into the wonders of operating systems.



I have a nasty dilemma when I sit down at my keyboard to type all these words. The more I write, the more space it takes up on the page. The more column inches taken up, the less space there is left for those wonderfully likelike pictures of you-know-who. It's a toughie.

Part of me enjoys nothing more than helping you with your monthly doses of coding, but the other part would rather see lots of pictures. It's to help my mother, who sneaks into the newsagents and leaves all the copies of ACU open at this page. Ah, I bet you were wondering who did that. I've tried to stop her, but she's constantly popping into the High Street stores and leaving entire rows of magazine racks displaying nothing more than your old friend, Aj. It's a hard life.

But back to the real world, and last month we finished our little chat by introducing an operating system address that will become a life-long friend, &BB5A. Now to me at least, nasty big hexadecimal numbers like that can be a bit impersonal. Let's use the technique of "labels" we spent so much time on to improve on the situation.

Our machine code program therefore changes from:

```
org &8000
ld a,"A"
call &BB5A
ret
```

to

```
Start_of_code      equ &8000
Ascii_code_A       equ 65
Print_a_char       equ &BB5A


org Start_of_code
ld a,Ascii_code_A
call Print_a_char
ret
```

This may seem a trifly wordy, but will make much more sense when you come back to it in a week or so.

Another enquiry I received (you see, people do write to me you know) expressed concern over the apparent length of machine code programs. For example, compare the example above

ian girl

with the BASIC equivalent.

PRINT "A"

Hmmm. There could be a good point here. The BASIC version certainly seems a lot simpler and quicker. Why is this? What's going on?

The first thing to remember is that your CPC simply cannot understand any program unless it is in Z80 machine code. No matter what language you may think you are writing in, the CPC is using machine code.

BASIC programs must be converted into code for the computer to make any sense of them. This conversion process means that BASIC is an interrupted language.

Normally this process is completely invisible, and happens as soon as you type RUN and press return. At each line of your BASIC program, the CPC takes a deep breath and in a flash does the conversion. It then obeys the machine code which results. (Note strictly those who think they know what is going on: As you might have guessed, the conversion process itself is a program written in machine code. Remember – the CPC knows nothing but Z80, and I mean nothing).

Here is (roughly) what happens when the computer runs the program which consists of our mega-simple PRINT "A" statement.

Step 1

"OK, here I am. Where was I? Can't remember. Better start again. OK, here we go. Clear any variables that may have been left over from last time."

Step 2

"Now, let's take a look at this program. Ah yes, PRINT. I know what that does. And I know the address of the operating system address that does it too. Easy."

Step 3

"Now what am I to PRINT? OK, it's a string of characters. And the length of the string is 1. Fair enough."

Step 4

"Ahoy there Operating System! Please do a PRINT with this string I have here, whose length is one. Thank you."

Step 5

"Hmm. That's the end of the program. Quite a short one. Time for a snooze."

Of course, each step here takes time. Plus the code that the BASIC program is translated into is a good bit more complicated and lengthy than the short program we wrote.

In short, our version will run faster and take less room. We win!

The only drawback is that programming in code is a bit more tricky then programming in BASIC, but I'm sure you will have realised that by now.

We can now deal with the problem of length. You may be thinking that your fingers will have long worn out before you have written any sort of program. The example that wrote a letter "A" to the screen was seven times longer than the BASIC version! Well, let's examine a specific case – our own PRINT routine – and see if we can improve matters somewhat.

Here's the program specifications, or "specs" as we hackers call them.. (An important part of programming is using the right words at the right time. Understanding them is much less vital). We want a routine that prints a string of characters to the screen. The string can be of any length greater than zero characters long, and will end with a 0. Not a "0" character, but an Ascii code 0. If this seems a bit confusing to you, check in the back of your user manual.

The character "0" has a code (48) just like the character "1" or "&". However, the character whose code is 0 is something totally different. In this case we will be using it simply as a way of terminating the string.

OK, so we have the specs. Now for the program.

```
   start_of_code  equ &8000
   Print_a_Char   equ &BB5A

   org Start_of–code

Print_a_string:
;Input: HL contains the address of the string,
;which ends with 0
;Output: A string, plus registers corrupt
loop: la a,(hl)
;Get character to be printed
cp 0
;Is it zero?
ret z
 ;Yes it is – stop
call Print_a_Char
 ;No it isn't so print it
inc hl
 ;Point to next character
jr loop
 ; Continue
```

Nothing too taxing here. All the hard work is done by the operating system, all we have to do is give it the character to print. However, the program has one or two important concepts in there somewhere.

What we have written is not really a fully fledged program. Instead, we have written a reusable function. If you assembled and ran the program as it stands, it would probably crash. Why? Because we haven't fulfilled the input conditions, that's why.

Where is the string? What is HL pointing to? Not a lot. If nothing else, it demonstrates how to put comments in the program to remind yourself what the varoius functions do.

To use the function in our program, we should define some strings, and then call the function, thus:

```
   org Start_of_code
   ld hl, String 1
   call Print_a_string
   ld hl, String 2
   call Print_a_string
   ret
String1:
   db "Hello",0
String2:
   db "World",0
Print_a_string:
   etc, etc.
```

Do you see what is happening? We are using our string printing function twice from within the one program. Now is that value for money or what?

The function is one that we may wish to keep as part of a library of useful routines. Now, whenever you need to print a message in a game you can load up the routine and use it. No need to re-invent the wheel every time we write a new program.

Next month we'll continue our venture into the world of real programming with some more useful routines to incorporate into your own personal libary.

See you then!