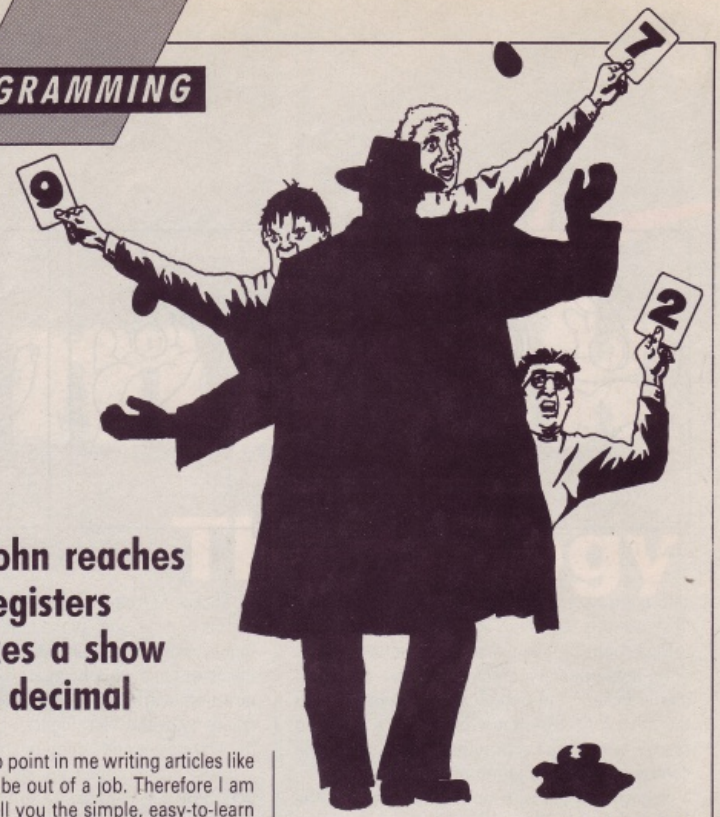
Carry on juggling



AND then a huge albatross swept down from the sky, ripped off his other leg and vanished behind the mountains. Oh, there you are. I was wondering where you lot had got to.

Now then. As the long winter evenings draw in, it is always nice to make a steaming hot mug of coffee and pull up a chair beside your trusty Arnold. I usually have a dozen chocolate digestive biscuits ready as well, just in case I get peckish or have a really long program to save. However, let's cut the cheery chat and get down to some serious coding.

When writing programs in machine code you quickly come to realise that you need to be able to print the contents of the registers to the screen somehow. I'm going to show you a couple methods for doing just this, but you must remember that no routine will cover all circumstances.

The main purpose of using machine code in the first place is to speed things up: Routines must be specially written for the data presented to them. The machine code I've listed is quite general though, and should come in very useful.

The simplest example occurs when you need to print a decimal number from zero to nine – perhaps to indicate the number of lives you have left in a game. Listing I will display the contents of the A register at the current cursor position. Bear in mind that if the A register contains a value less than zero or greater than nine, weird things will happen – Ascii characters will appear and printed control codes will do strange things to the screen.

Man of letters

You know, people often write to me and say, "Why? Tell me why." Sometimes they write other things as well. A case in point is Mr James Brown of Coleraine, Norn Iron, who writes:

Dear Auntie John; Although I am not nearly as clever as you, I am trying to teach myself how to program in machine code. What advice can you give me?

Well James, look at it this way: If everyone could program in machine code as well as I can, there would be no point in me writing articles like this, and I would be out of a job. Therefore I am hardly likely to tell you the simple, easy-to-learn method that I discovered, am I? No, James, you just keep struggling with your assembler, and remember to make the coffee really hot.

Another letter I received this month was from my good friend Chris the Hippy, but as he owns a Spectrum I can see no point in ridiculing him further.

Auntie John reaches for his registers and makes a show of going decimal

Juggling the figures

Printing single numbers is all fine and dandy, but the time will come when you need to display a really large number – I mean a full 16-bitter in the range zero to 65535.

We have problems here because we need the number printed out in base 10 (decimal) and computers would much rather work in base two (binary). The solution, as supplied in Listing II, is to continually subtract powers of 10 from the number we want to print. First we count how many 10,000s are in the number, then how many 1,000s, then how many 100s, then how many 10s and finally how many ones.

Which brings me back to juggling, the age-old art first practised by the American Indians as they pulled their baked potatoes from the fire. Now although using baked potatoes is the fastest possible way to learn juggling, the scars can take a long time to heal. For this reason, most people prefer the "small object" method.

As the name suggests, the "small object" method involves using small objects instead of red hot potatoes. The best small objects are bean bags, raw eggs, lumps of raspberry jelly and pieces of damp kitchen roll – none of these will roll away when you drop them. And believe me, you are going to drop them a lot.

Green taught me to juggle several years ago, primarily as a way to look really cool and get lots of girlfriends, but it is amazing just how useful juggling really is in everyday life. For example, imagine you are asking your bank manager for a large loan. What for? Probably to help you buy the latest disc-based game for your computer. Cynical? Who, me?

Anyway, chances are your bank manager will have lots of little executive toys and paperweights scattered around the desk. Well, just by grabbing three of them and performing a five-minute juggling routine you will impress him so much that he will probably give you half an hour in the safe with a shopping trolley. That's juggling for you.

Off the tracks

Those of you lucky enough to own a 6128 or a 464/664 with an extra 64k of ram might be interested in how the memory is actually inserted into the Z80 microprocessor, bearing in mind that the Z80 is designed to deal with a maximum of 64k because when it was first sold nobody could imagine ever needing, or being able to afford, more memory.

Those clever chaps who designed Arnold decided to use special moving tracks on the PCB (printed circuit board), which could swing left and right and so connect different ram chips into the memory map when needed. A tiny mechanical switch lifts a small part on the circuit board and rotates it about 90 degrees in a fraction of a second. On some of the older CPCs you can actually hear the PCB tracks as they swing into place.

The firmware call at &BC6E (kl swing track) will switch the second bank of ram into the memory map, and the call at &BC71 (kl swing back) will

```
org &4000

txt_output equ &bb5a

ld a,5          ;LET A = 5
call printn_1   ;PRINT A
ret             ;Return to Basic.

.printn_1
;
; Display the contents of the A register.
; A must be between zero and nine.
;
add a,48        ;Add 48 to convert the number
                ;to its Ascii charcater.
call txt_output ;The firmware to print it.
ret             ;Return.
```

Listing I

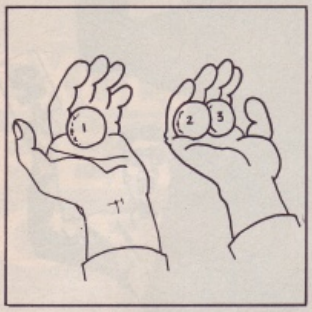


Figure I

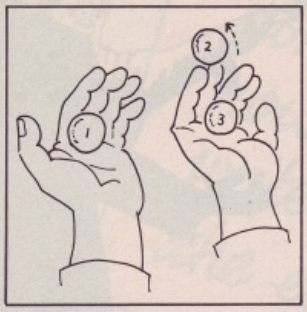


Figure II

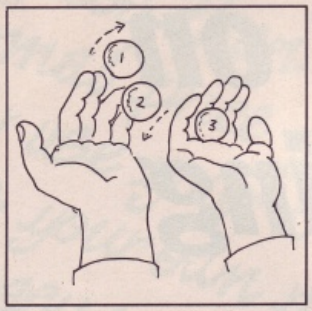


Figure III

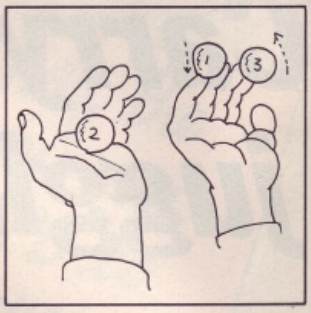


Figure IV

return the tracks to their original position. Listing III shows this routine in action.

Back to the juggling, and it is important to pick your position very carefully. Good places to juggle include the bottom of steep valleys and the fresh fruit department of supermarkets. Bad places include the top of steep mountains and the central reservations of motorways.

You should make sure you are holding the objects correctly: Two in one hand and one in the other (Figure I). If you still have a free hand then you have not counted the objects correctly. Or you could be a Martian. Either of which means you probably will not understand what follows.

Now throw an object from the hand that has two in it in a gentle arc just passing your eyes and in the general direction of the other hand (Figure II). Stick your tongue out as this usually helps you to concentrate. Just as the object has passed the highest point in its trajectory, throw the object in the other hand in a similar arc, but in the opposite direction and aimed slightly lower so it won't collide with the first (Figure III). Now just catch the first object and congratulate yourself.

Oops. You dropped the second object, didn't you. Silly me, I should have explained how you catch that one. Basically it's just the same technique as for the first – get rid of whatever is in the hand you are going to catch with by throwing it up in the air to the other hand (Figure IV).

That is all there is to it. After about a week doing this you will have either gone mad and taken up machine code programming, broken all the small objects in the house or learned to do it right. Same goes for displaying the contents of the registers in decimal. Good luck!

```
;Swing in extra ram into memory map

swing_track equ &bc6e
swing_back equ &bc71

org &4000

call swing_track  ;Extra ram banked in.
call swing_back   ;Normal ram.

ret

end
```

Listing III

Set the DE register pair to required powers of 10, then call the digit printing routine at *.prdigit*. Notice that the last line in this block does not need the *call prdigit* statement because the *prdigit* routine follows directly on from it. This sneaky piece of code keeps speed to the maximum. It also ensures that when the routine is finished it returns directly to the routine that called *.printn_2*, but we'll get to that in a minute.

Remembering that HL contains the original number, these three lines subtract the power of 10 from it. If the power of 10 was in the number – for example 10,000 is in 61,280 six whole times then the routine jumps back to *.loop* and increments the counter. The original number is reduced by the power of 10. Do you see now why A was set to 255 and not zero? If the power of 10 wasn't in the original number, the counter is set to zero, which is the way it should be.

The number is set back to its postive value by adding DE to HL. The counter has 48 added to it to convert it to the Ascii value of the correct decimal digit.

```
org &4000

txt_output equ &bb5a

ld hl,61280     ;HL = 61280
call printn_2   ;PRINT HL
ret             ;Return to Basic.

.printn_2  ld de,10000
           call prdigit
           ld de,1000
           call prdigit
           ld de,100
           call prdigit
           ld de,10
           call prdigit
           ld de,1

.prdigit   ld a,255

.loop      inc a

           scf
           ccf
           sbc hl,de

           jp nc,loop

           add hl,de
           add 48

           jp txt_output

           end
```

Sets up the A register to hold the value 255. We are using this register to count the number of times the power of 10 occurs in the main number, and A gets incremented each time it does so. Why 255 and not 0? Good question. The answer is that the first thing this loop does is increment the counter. This effectively zeros the counter for us. The register is like a car speedometer – after 255 it clocks back to zero.

Assuming the original number is now less than the power of 10, the subtraction produces a negative result. This is detected by the carry flag in the conditional jump.

Instead of a call instruction, a jump is used to access *txt_output*. As the *.prdigit* routine was originally *call*ed, the return address is still stored on the stack. The firmware will return to that address when it has finished its work. However, because a call was not used at the end of the first block of code in *.printn_2*, the routine will return when it has finished to whatever address the *.printn_2* code was originally called from.

Listing II