Launch pad for people in a hurry

Auntie John may be one register short of a CPU, but the CPC firmware routines are his recipe for fame and success as this new series starts

GREETINGS to you all. Allow me to introduce myself – I am Auntie John and it is my honour, nay privilege, to have a chat with you about machine code, firmware calls and interesting stuff like that. Make no mistake, machine code is a very interesting subject. Only last week I was talking to Una (a very pretty girl) about return stacks and the like, when she said "You really are boring, aren't you?"

It was clear to me that under this facade of scorn, she was secretly enthralled and almost overwhelmed with admiration for my knowledge of Zilog op-codes.

So, my little nephews and nieces, it is with this in mind that we shall explore the workings of the Z80.

Never again will you sit alone in the corner at parties; never again will people refer to you as "The dull kid with glasses and acne"; never again will you be stuck for things to do in the bath; never again will you be stuck for a witty reply in an argument ("Oh, go and shift your accumulator, you jumped up system bug!") – now you can woo the boy or girl of your dreams with little machine code poems left on their doorsteps.

Yes, it's the answer to life, the universe and how to get published in magazines... it's...

Firmware jumpblocks

As well as being almost phenomenally exciting, machine code is very useful if you want things done in a hurry. Writing a program to move lots of graphics around, or to run very quickly, is almost impossible in Basic, a language which, while begin relatively easy to learn, is extremely slow.

Speed is the main reason people give themselves brain damage learning machine code, but it also allows you to get to grips with all the little intricate fiddly bits of the machine's hardware.

An easy way to get to grips with machine code is to compare it with a high level language like, say, Basic. In this context you can say machine code has eight or so integer variables (A,B,C,D,E,F,H,L) which can hold values from 0 to 255. By linking two of these variables together (HL, DE, BC) you have three new variables that can hold values from 0 to 65535.

Here are some Z80 machine code assembly mnemonics and their high-level (or pseudo-code) equivalents.

```
LD HL,3000    -->   LET HL=3000
LD DE,2000    -->   LET DE=2000
ADD HL,DE     -->   LET HL=HL+DE
INC HL        -->   LET HL=HL+1
DEC HL        -->   LET HL=HL-1
LD A,L        -->   LET A=L
```

When planning a machine code program it is sometimes a good idea to write it out in psuedo-code first, just to get the idea in a form you can easily understand. But you must make sure to remember that you are only allowed to do certain things with these registers, namely:

- Load them with values.
- Add and substract them.
- Perform logical operations on them.
- Compare them.

You might think all this is a bit limiting. How for example could we print a letter A to the screen? Well, this is where the Amstrad operating system firmware specification and a decent assembler package comes in (I didn't say being popular was going to be cheap).

An assembler is a special program that converts assembler mnemonics into the actual machine codes. If you want to drive yourself completely crazy, you can do without an assembler and just play with these code numbers, but it can't be emphasised enough that a good assembler is essential if you want to learn machine code and stay sane.

If you were to have a browse through the wondrous ring-bound tome of knowledge called SOFT 968, you might find something like this:

```
30:TXT OUTPUT                    #BB5A

Output a character or control code to the Text
VDU

(and lots of technical looking stuff)
```

Looking at the entry conditions, you would see that you simply load the A register with the Ascii value of the letter you want to print, and – Hey Tesco – you call the magic number BB5A and the letter appears on the screen.

Listing I does just this. You must type it into an assembler (not the psuedo-code – just the assembler mnemonics), or, if you don't have an assembler yet (subtle hint), you can type in the Basic listing instead, which POKEs the machine code directly into memory and then executes it.

Pretty exciting stuff, I'm sure you'll agree – how many times have you wanted a letter A to appear in the middle of your program? But wait, here comes the breathtaking bit – it doesn't have to be the letter A! No kidding, you could print any letter you want. Isn't that amazing? Now sit down and get your breath back, because there are more shocks to come.

So, what exactly is the firmware jumpblock?

```
LET A=65
CALL PRINT TEXT ROUTINE
RETURN TO BASIC
```

Pseudo-code

```
ORG &8000
.TXT_OUTPUT EQU &BB5A
LD A,65
CALL TXT_OUTPUT
RET
```

Assembler

```
10 MEMORY &7FFF
20 a=&8000
30 READ b
40 IF b=-1 THEN 80
50 POKE a,b
60 a=a+1
70 GOTO 30
80 END
90 '
100 ' Now type CALL &8000 to
110 ' run the machine code.
120 '
130 DATA &3e,&41,&cd,&5a,&bb,&c9
140 DATA -1
```

Basic poker

Listing I

Have a look at Figure I. It's a picture of the Amstrad CPC's memory map. The main sections are the screen memory right at the top, and the memory pool in the middle. Sandwiched between these is some technical stuff and the jumpblock.

The jumpblock starts at about #BB00 and goes on till about #BE00, with another little section just below this at around #B900. These blocks of memory contain nothing but instructions to jump to various places in memory. When we CALL #BB5A in machine code, the instructions at address #BB5A re-route the computer to another routine buried deep in memory somewhere. This

particular routine displays Ascii characters for us.

The beauty of the system is that it always works – if Amstrad changes the internal workings of the computer, as long as the Jumpblock stays in the same place, all the programs written before the change will still run after the change. That is why most CPC464 programs will run on 664s and 6128s.

Another important point about the jumpblock system is that it is in ram. This means you can change – or patch – bits of it to point at your routines instead of the built-in ones. But I diverge...

If you glance through the Firmware Guide you will see lots of calls to do wonderful things like drawing lines or changing pens. In fact, you can do everything that you could from Basic, and more besides. So now nothing can stop you. You can write anything in machine code – you can be popular.

Of course, there are other ways to be popular. There is the classic "I've got lots of money, would you like some?" approach, which can be very effective, especially if you do happen to have lots of money. Also guaranteed to get you talked about is the wonderful Exploding Cooker Trick.

Simply sneak into the kitchen and when no one is looking put all the tins of catfood you can find into the oven and switch it on to full power. Then retire gracefully to the safety of the living room, happy in the knowledge that within 30 minutes everyone will want to know where you are. (NOTE: this can be dangerous – meaty chunks travelling at 90 miles per hour can seriously damage your decor).

Undoubtedly the most effective way to look Really Cool and make lots of friends is to take up juggling, but I have neither the time nor the energy to explain how to start doing that here. Another time, maybe...

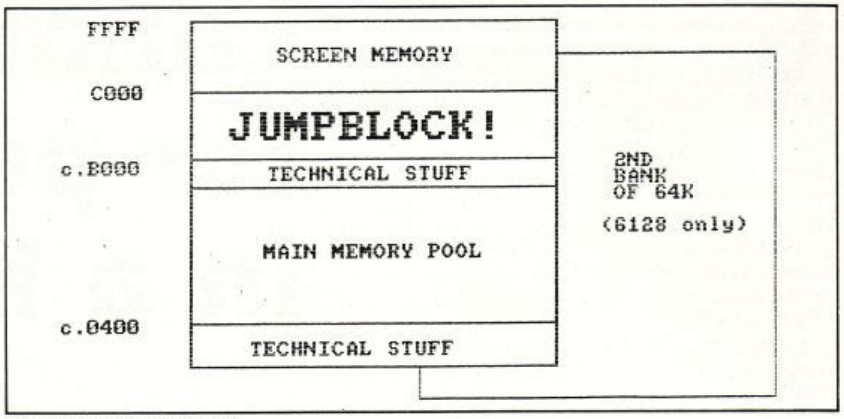

Figure I: The CPC's memory map

```
HL=ADDRESS OF THE START OF THE STRING
loop
A=CONTENTS OF MEMORY POINTED TO BY HL
IF A=END OF STRING MARKER
  THEN RETURN TO BASIC
CALL PRINT TXT ROUTINE
HL=HL+1
GOTO loop
```

Pseudo-code

```
ORG &8000
.TXT_OUTPUT EQU &BB5A
LD HL,string    ;Point HL at the text
.loop
LD A,(HL)       ;Load a char into A
CP 0            ;Is it the EOTM
RET Z           ;Yes, return to Basic
CALL TXT_OUTPUT ;No, print it
INC HL          ;Point to next char
JR loop         ;Round the loop again
.string
DB "HELLO TO YOU FROM THE DEPTHS OF MA
CHINE CODE!",0
```

Assembler

```
10 MEMORY &7FFF
20 a=&8000
30 READ b
40 IF b=-1 THEN 80
50 POKE a,b
60 a=a+1
70 GOTO 30
80 END
90 '
110 ' Now type CALL &8000 to
120 ' run the machine code.
130 '
140 DATA &21,&0d,&80,&7e,&fe,&00,&c8,&
    cd,&5a,&bb,&23
150 DATA &18,&f6,&48,&45,&4c,&4c,&4f,&
    20,&54,&4f,&20
160 DATA &59,&4f,&55,&20,&41,&4c,&4c,&
    21,&00
170 DATA -1
```

Basic poker

Listing II

Beyond the letter A

Anyway, let's put our letter printing routine to good use by writing a short program that will print a string on the screen. (A string is simply a collection of Ascii characters that usually represent words). Look at the pseudo-code program in Listing II. (For those of you STILL without an assembler, the Basic program will achieve the same result).

Now that looks easy enough, doesn't it? The end of text marker (EOTM) is just a nule character that we have defined to say "OK, there is no more stuff to print". Notice that it is never printed because it is checked for before the call to the print routine.

It's at this point that I think I should mention the concept of modular programming. You can see that the string printing routine is a complete little program in itself – to get it to print different strings you just load HL with the suitable address and then call the string printing routine.

It's worth putting a title on this routine – just a name and something to tell you what it does – and saving it away on tape or disc for future use. Then, whenever you are writing a program in machine code and you suddenly need a routine to print a string, you'll have one all ready and waiting.

If you build up a library of these modules of code, you will find that writing programs becomes a lot faster since you won't have to re-invent the wheel every time. Also, because each module has a little story saying what it does, your programs will be much easier to understand.

There, don't say I never tell you anything interesting.

If you want the string printing routine to do something really exciting, you can try printing non-Ascii characters with it. A quick glance towards the rear of your User Guide will reveal a table called "Basic control characters". Funnily enough, that's exactly what these characters do – control things. Who said computer programmers have limited imaginations?

Let's start with a simple one – character 7 – which is given the TLA (Three Letter Abbreviation) of BEL. Every time you print the BEL character, the computer will make that hideous little bleep that gets right up your nose. So, to change the program to print "URGH!" and then go beep! we just replace the final statement to be:

```
DB "URGH!",7,0
```

and that will do the trick. Try it if you don't believe me.

Using these control characters you can affect the way your text is printed. What do you think the following will do when printed?

```
DB 24
DB "Shumething different shurely?"
DB 24,0
```

Looking at the control codes, 24 will swap the pen and paper inks and so print the text in "inverse". Remember that we still need the end of text marker.

A slightly more useful code is 31 which will place the cursor for us. In other words, it sets where the next piece of text is to be printed. Combining this with the code to clear the screen (12), the following will perform a CLS and then print the word "Hello" halfway down the screen and 2 spaces to the right.

```
DB 12          ;cls
DB 31,2,12     ;locate 2,12
DB "Hello!",0 ;the text and EOTM
```

Isn't it amazing what you can do with only one firmware call? Just think what you will be able to do when we look at the remaining two hundred and twenty...