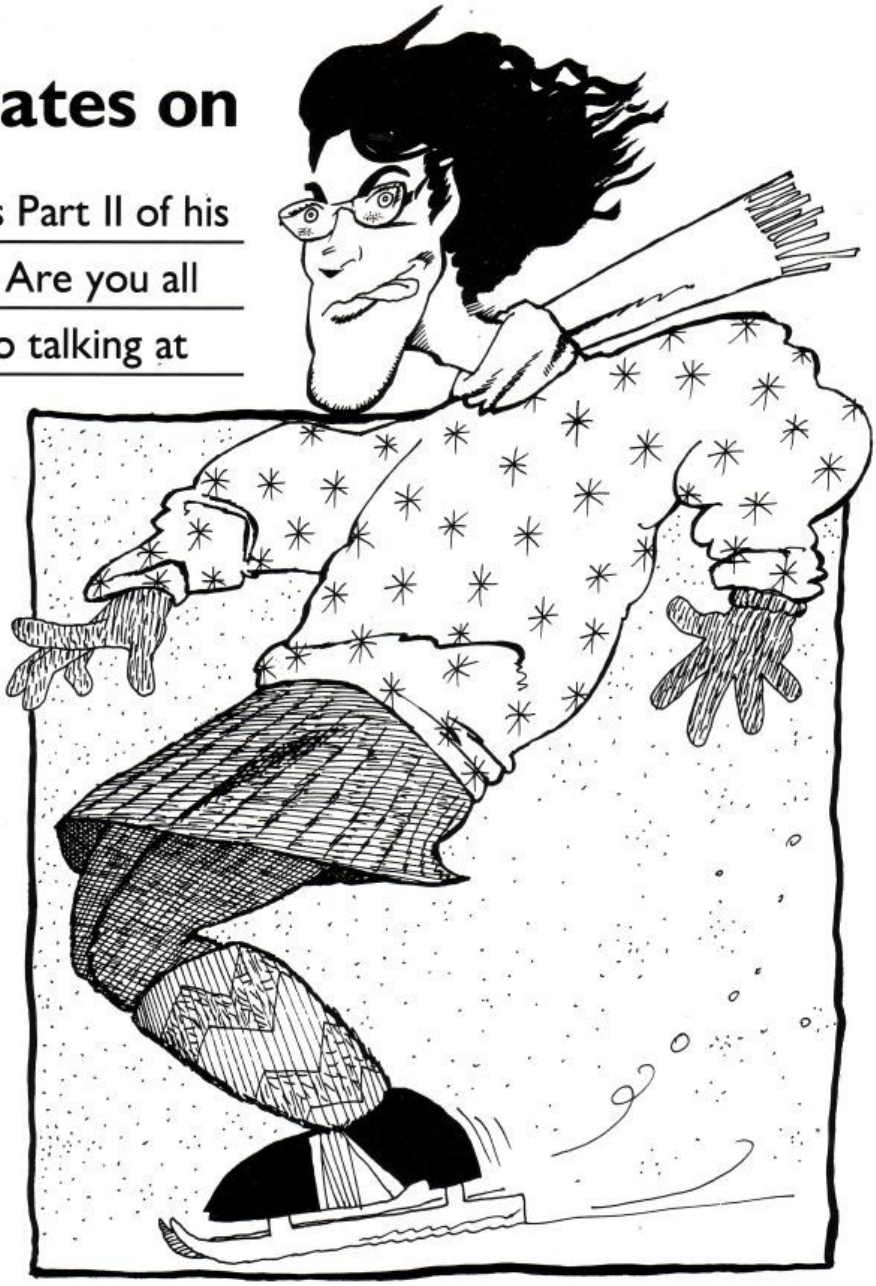
Get your skates on

John Kennedy presents Part II of his machine code tutorial. Are you all sitting comfortably? No talking at the back. Good, then we shall begin.



So here is our first machine code instruction. First we will express it in 'mnemonics', which are a cross between English, Basic and gobbledegook. A mnemonic is simply an expression of the machine code in a form other than a number, in a futile attempt to make things easier to remember. Each mnemonic translates directly to a machine code number. The Z80 could not directly understand a mnemonic any more than it could understand a Basic command, and so we must translate these into their machine code counterparts before we can enter them into memory. We could spend lots of money and buy a program which would do this for us; they are called Assemblers, and cost at least as much as two Pink Floyd albums on CD. I know which I would rather have. And I do not even have a CD player.

The mnemonic LD A,21 places the value into one of the registers, the A register, which is better known as the Accumulator. In Basic, this would be equivalent to LET A=21. The 'LD' is short for 'LOAD' and so the instruction would be read as 'Load A with the value twenty-one'.

The Z80 code for the 'Load A with . . .' instruction is &3E (which is 62 in decimal). The number to be loaded into A must follow the instruction immediately, so if address &8000 contains &3E, then &8001 must contain twenty-one. Remember that the number must be in the range 0 to 255 as only a single byte is involved.

A more advanced instruction allows use of a similar facility to Basic's 'PEEK' command.

The mnemonic uses brackets to mean 'the contents of'. Thus, LD A,(&9000) means 'Load A with THE CONTENTS OF &9000'. In Basic this would be: LET A=PEEK(&9000).

The code number for this instruction is &3A. The address to be 'peeked' must follow directly afterwards. If only a single byte value from 0 to 255 was allowed for the address, the possible number of memory locations capable of being 'peeked' would be very small indeed. Instead we use two byte values linked together to provide a much larger range. The address has been split into two byte values, this means into two groups of two digits. But there is a twist: the two bytes are listed in reverse order. This may seem completely confusing, stupid, silly and pointless. And indeed, it may be so, but humour me. Therefore, it comes to pass that the code for the above mnenonic is &3A &00 &90.

Doing a 'poke' is easy: LD (&9000),A
or 'Load the contents of address &9000 with the value of A', and which in Basic is: POKE &9010,A
The code number is &32, and the above is therefore: &32 &10 &90. Remember that addresses are listed sdrawkcab!
Note: The use of hex address &9000 for our poking and peeking examples is totally arbitrary. However, if you are

experimenting please keep to addresses around this area. Poking an address vastly different could change a vital area of memory, and as a result the computer would hang-up or crash. No damage will be done, except that you will have to switch off and start again. Worse than an immediate crash

is one where you poke a strange value and nothing happens. For the moment, that is.

Our First Program. Here is a short program to transfer a byte from one part of memory to another:

Mnemonics	Basic Equivilent	Code
LD A,(&9000)	10 LET A=PEEK(&9000)	&3A &00 &90
LD (&9001),A	20 POKE &9001,A	&32 &01 &90

It takes the byte held in address &9000 and copies it into the accumulator. Next the memory at address &9001 is changed to contain a copy of this value. At the end of the program, both memory addresses and the accumulator all contain the value that was on the first memory address. But do not take my word for it, run the program!

A Basic program to POKE all the data into memory is given below. You can type it in directly and run it.

Listing 1 – Excitement and really wild things.

```
5 MEMORY &7FFF: REM First
reserve some space for
the code
10 LET ADD=&8000
20 READ X$:X=VAL("&"+X$)
30 IF X="-1" THEN STOP
40 POKE ADD,X
50 LET ADD=ADD+1
60 GOTO 20
70 DATA &3A,&00,&90,&32,
&01,&90,&C9,-1
```

The more observant ones among you will notice an extra instruction included in the data list. It is the code &C9 which has the mnemonic 'RET', short for 'RETURN'. When your machine code program reaches this instruction, it will 'RETurn' to Basic. If you leave it out, your program will carry on trying to decode and execute memory as though it were a program, and as a result something nasty may occur, such as a crash. If this happens, your only recourse is to switch off and start again. Moral: remember to put a RET statement in.

So once the program has been RUN, what happens? Well, nothing. Although the program is now in memory, the Z80 has not been told to start executing it. To do that, we use the Basic word 'CALL', like this: CALL &8000. And once this is typed, the program will run, do its business and return to Basic. The net result is that, once again, nothing will have seemed

to have happened. So what was all the excitement about? Is machine code a giant con? Well, let us give the program something to work at.

Put the number 42 into memory location &9000, with a POKE: POKE &9000,42. Now type PRINT PEEK (&9000). The number displayed is 42, is it not? Good, because we just put it there. However, we did not put it at address &9001, which you can check with a PRINT PEEK (&9001).

Now type CALL &8000. Okay, so now type PRINT PEEK (&901). And hey! Your value of 42 has been moved! Bravo! Your first machine code program – and it worked!

More Advanced Programs. Here is a new instruction: INC A. It stands for 'Increment the A register' and has the code value &3C. It increases the value of the A register by one.

In Basic this would be let A=A=1.

If a program consists of the following,

LD A,10
INC A
RET

by the time the RET statement is reached, A will contain the value 11.

Another instruction DEC A, or 'decrement A', will subtract one from A, like this: LET A=A−1.

So if a program read like this:

LD A,10
INC A
DEC A
DEC A
RET

. . . then at the end of the program, A would contain the value 9. It has been increased to 11, then decreased twice to 9. The code for 'DEC A' is &3D.

The Second Program. Here is a program to read a value from a memory location, increase it by one and return it, *to the same location.*

LD A,(&9000)
INC A
LD (&9000),A
RET

In Basic this would be:

10 LET A−PEEK (&9000)
20 LET A−A+1
30 POKE &9000,A
40 STOP

The complete Basic program to enter the code is given below. You can see that with the exception of the data the program is exactly the same, so that you can simply retype line 70 of listing 1. That is what I did.

Listing 2 – More excitement and really wild things.

```
5 MEMORY &7FFF
10 LET ADD=&8000
20 READ X$:X=VAL("&"+X$)
30 IF X="-1" THEN STOP
40 POKE ADD,X
50 LET ADD=ADD+1
60 GOTO 20
70 DATA &3A,&00,&90,&3C,&32,
&00,&90,&c9,-1
```

Once you have entered and RUN this Basic program, try it out with the following: POKE &9000,10 CALL &8000 PRINT PEEK (&9000). You should get the number 11 printed on the screen. Now try the following:

POKE &9000,255
CALL &8000
PRINT PEEK (&9000)

What do you think will be printed, and why?

Summary. Machine is faster but slightly more tricky than Basic. If you think it will help you can think of the memory of a nice jovial chap with a beard and a penchant for numbers in the range 0 to 255. The Z80 has a specific set of instructions, and a set of registers which also have a penchant for numbers in the range 0 to 255.

I lied about the CD, player, I do have one.

(RET is a bit like RETURN from a GOSUB, but we will look at it in detail later – just make sure you put it in for the moment!).