Back to school

Auntie has a nice surprise – a machine code tutorial in two parts for those of you who still don't know your Z-80s from your Data Bases.

"Oh no," you think. "Not another machine code tutorial . . ." Yes, I am afraid it is. However I am determined that this one will be the definitive one. Wherever men may revel and quaff they will speak of 'Machine Code – The Tutorial' and remember that this is where they learned machine code. Future generations will speak with awe about the verbose examples, the eloquent text and the dreadful puns. Or then again, perhaps they will not.

Let us lay down some ground rules. Firstly, the old 'Buy an Assembler' story. Wait and see if you want to learn machine code before rushing out and spending lots of money. If, though, you ever find yourself spending more than five minutes converting codes yourself then buy one, and make sure it is MAXAM, preferably on ROM.

Secondly, there are the various reasons for machine code. Some will say that the great speed of code is the reason to learn; some will say it is the extra flexibility and control that Z80 code allows. They are wrong. You will only learn machine code if you want to. You must enjoy it. Programming is a hobby simply because it is an intellectual challenge. Some people will program for the same reason that others play chess: it is a test of brainpower. If you do not enjoy coding you will not find it fulfilling, and you will not do it, for all the extra speed and control will mean nothing.

Chapter One: Machine Code – What is it? As you may be aware, the heart of the Amstrad CPC computer is a microprocessor called the Z80. The microprocessor is small (hence the micro bit) and processes (hence the processor bit) information. The Z80 was originally designed by the Zilog Corporation, who based it upon the 8080 processor designed by Intel. (This resulted in various court actions, but everyone's lawyers are one big happy family now.) It has become arguably the most popular processor ever and is used in many different computers, ranging from the wonderfully named Superbrains to the ill-fated Jupiter Ace. Although the newer generation of computers are turning to new and more powerful processors such as the 68000 or 80086 families, the Z80 will live on in folk legend for many years to come.

Any computer using a Z80 and a disc drive also has the option of using a standard operating system called CP/M, which is another reason for its continued existence.

Your Amstrad comes complete with the programming language Basic 'built-in'. Basic was chosen because it is the nearest thing to a standard language and is probably the easiest to learn and use. Part of the Basic system is an 'interpreter' which converts the program into machine code for the Z80 to run.

If you wish, you can bypass Basic and instead talk directly to the Z80, with an immense increase in speed and complexity. To do this you must use the list of commands that the Z80 speaks (the 'instruction set' and none other). Using a Basic instruction such as 'PRINT' or 'PLOT' is not possible using a single Z80 instruction, although, as we shall see later, careful combinations of instructions will enable us to do anything that is possible

from Basic, and much more.

The only difference is that our machine code equivalent will be faster, more flexible and will take up less space in memory. It will also crash quite a lot.

Chapter Two: A Brief Overview of Computer Systems – meet Mr Memory. In order to understand machine code, we must explore the underlying hardware of our computer system.

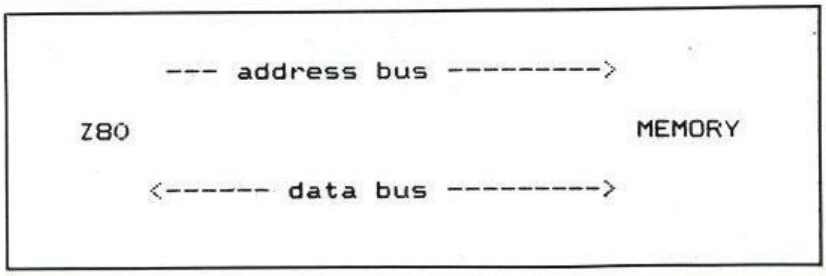
The Z80 microprocessor is linked to the memory by two groups of connections: the Data Bus and the Address Bus. The memory consists of a number of 'cells' each of which contains a number in the range from 0 to 255: a BYTE of memory. To enable each memory cell to be specified uniquely, it is given an 'Address', in the form of another number, this one in the range from 0 to 65535.

Memory comes in two flavours: Ramdom Access Memory (RAM) and Read Only Memory (ROM). 'Random Access' simply means any part of the memory can be read at any time, in the same way that any track of a record can be played at any time just by moving the stylus. Compare this with a music cassette: here you must fast forward and rewind through all the other music before you get to the really good track you want, the one with that fabulous guitar intro.

RAM is a misleading name: all modern memory (including ROM) is Randomly Accessed. A better name would be 'Read and Write Memory', but this does not sound as good so it never caught on. RAM can be written to and read from by the Z80 but when the computer is switched off, the contents of the memory is lost. Thus you must make a copy of any data stored in RAM on a more permanent media, such as tape or floppy disc, before switching off.

In order to write to memory, the address of the memory call in question is placed on the Address Bus and the new value of the cell is placed on the Data Bus. Then a special 'WRITE' control signal is sent to the memory and the memory says "Hey! That's my cue to remember something! What's this number on the Data Bus? Okay, got it. Now where am I to store it? Ah, thank you, Mr Address Bus, right here. Okey-doke."

Of course, the above is slightly inaccurate. The memory cannot really speak, because memory circuits are inanimate objects. Perhaps on a different astral plane to the one on which we live, memory circuits do talk to one another and have really exciting times discussing access times and the like, but that sort of discussion is really left to the type of person who likes vegetarian cigarettes and Pink Floyd. If it helps you visualise what is going on, though, fine.



Magic bus.

When reading memory, the address again is put on the Address Bus, but this time a 'READ' control signal is sent to the memory. Once it gets this signal, the memory puts the contents of the memory cell in question on the Data Bus where it is received by the Z80.

Read Only Memory cannot be written to; it is supplied with its contents fixed and permanently in place. Perhaps the name gave this little surprise away! However Read Only Memory does have one thing going for it, and that is that it will retain data even when no power is supplied.

Write Only Memory is the very latest development from a small company called 'Electronic Parts' from Northern Ireland. They claim 100 per cent reliability from these devices, which will operate at practically no current and at very high access times. It remains to be seen if they will be accepted into the computer industry at large. Personally, I have no doubts, although if you wish to try the memories for yourself, the address of the company is given in the appendices.

As mentioned before, memory stores a collection of bytes which are just numbers from 0 to 255. Each instruction supported by the Z80 is given a code number in this range, and so a machine code program consists of nothing more than a list of numbers ranging from 0 to 255.

It is the way in which the Z80 treats each as an instruction that makes the data appear as a program. Of course, we will also want our program to be able to store and retrieve numbers (for ages, scores, dates etc.), so sometimes memory contains a program, and sometimes it contains data. This dual use of memory was a brilliant innovation, and has led to some of the most spectacular computer hang-ups in history.

Chapter Three: Programmers Start Here. The Z80 has a set of internal registers as letters, such as A, B, C, D, E, F, H and L. Then can be thought of as variables that can store byte values: numbers in the range 0 to 255. Their semi-alphabetical order is of no real significance, and is only there because those psychotically deranged people at Zilog wanted to confuse you.

Sometimes we may want to increase this single byte range and to this end some Z80 instructions pair two registers together, but more of this exciting concept later.

To enter a machine code program we must first decide where to place it, i.e. at what address in the computer memory. The address &8000 is as good a place as any, because it points to a nice big section of RAM just waiting to be used. (The use of the '&' symbol means that the number which follows is in base 16, or Hexadecimal. Check to see if your pocket calculator can change between decimal and 'hex'. The use of 'hex' is a universally accepted way of expressing numbers in computer terminology and it is vitally important that you understand it, or at least are willing to give it a go, see another appendix for details. &8000 is decimal in 32768.)

Once the location is decided, we insert our coded instructions into the memory locations. In Basic, the command 'POKE' will do this for us. For example, 'POKE &8000,99' will place the decimal value '99' into hex address &8000; 'PEEK' will do the opposite to 'POKE' and return the value at the supplied address.

● Continued next month. Bye!