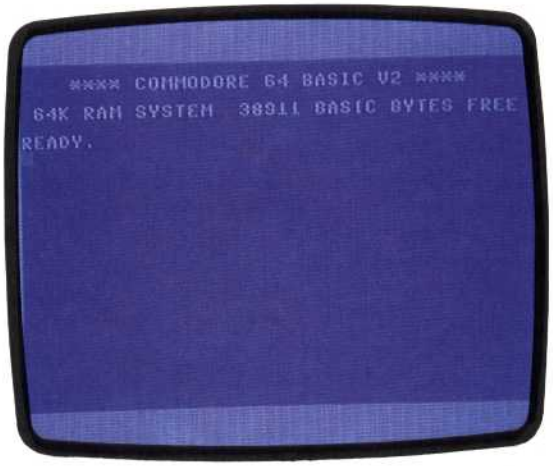
COLD START

A *cold start* is what happens when you switch your computer on, no matter whether it is one second or several days since it was last in use. The individual circuits and chips will take an appreciable time (though still measured in terms of a fraction of a second) to stabilise, during which their behaviour is totally unpredictable. Therefore, a simple circuit (consisting of little more than a resistor and a capacitor) is incorporated, which is sure to produce a pulse perhaps a tenth of a second after the machine is turned on. The output of this circuit is connected to the reset line of the microprocessor, which will have stabilised by the time the pulse is applied.



The display screen of a Commodore 64 after it has been cold started

A signal on this line always causes the micro to abandon whatever it is doing (which will be garbage) and go to a piece of code in a predetermined area of ROM, called the *cold start routine.* This routine usually checks through all areas of RAM to establish how much is available and that it is functioning correctly. It will then jump to the operating system, or boot it in from disk to RAM. A *warm start,* by contrast, is what happens when you press the Reset button on your computer. The warm start routine doesn't perform the memory checks, as this would obliterate the contents. Instead, it merely resets the microprocessor registers and jumps back to the operating system.

COMMAND LANGUAGE

We have already explained the difference between a package that is menu-driven and one that is command-driven: the former presents you with a list of options that can be selected with a single key-press, while the latter requires you to type in a command word at the bottom of the screen for each new action. The menu system has advantages for the newcomer to computing, but a command-driven program may in addition feature a command language.

A *command language* gives the user the ability to combine several individual commands into a single operation. It is common in database applications to find yourself repeating sequences like this: GET the next record, EXTRACT the TOTAL field, MULTIPLY this by the DISCOUNT rate, UPDATE the record with the new value, and STORE the record back in the file. A command language enables such a sequence to be activated by a single command such as MODIFY.

The more sophisticated database packages, of which Ashton-Tate's dBase II is the best example, take this a stage further and allow whole applications to be written in the command language, complete with conditional statements and subroutines. The end result is really a programming language just like BASIC or PASCAL, but using more sophisticated commands. Thus writing an information retrieval application in dBase II language will require considerably less work than writing it in a conventional language.

COMPARATOR

A *comparator* is an electronic circuit featuring two analogue voltage inputs and a single output. Its function is to compare the two inputs so that the output will be in one predetermined state if A is greater than B, and in another state if B is greater than A.

One method of building an analogue-to-digital converter is to use a digital-to-analogue circuit (which is much simpler to construct) in conjunction with a comparator. The D/A is connected between the computer and one of the comparator's inputs, with the analogue signal to be measured connected to the other. The computer then generates a counting sequence in binary, which causes a gradually rising analogue voltage at the output of the D/A. When the computer-generated voltage reaches the level of the signal to be measured, the output of the comparator will change, indicating to the computer that its current binary value is the digital equivalent of the analogue signal.

COMPILER

A computer program that translates a program written in a high-level language (called the source code) into a lower-level language (called the object code) is called a *compiler.* At the end of the compiling operation, the user's program will exist in two forms: a *source* file and an *object* file.

Programs written using a compiler are much faster than programs written with the BASIC that comes with home computers. This is because most versions of BASIC are *interpreters* rather than compilers. Interpreters are languages that convert each program instruction into machine code, one at a time, while the program is running. This means instructions inside a loop will be repeatedly converted into machine code, which wastes time.

Compilers, on the other hand, convert all instructions into machine code before the program is used and store them. Thus, the program wastes none of its time converting program instructions into machine code. Compilers are rarely used on home computers because they are complicated and need lots of memory. The few that are available tend to be limited in what they can do.