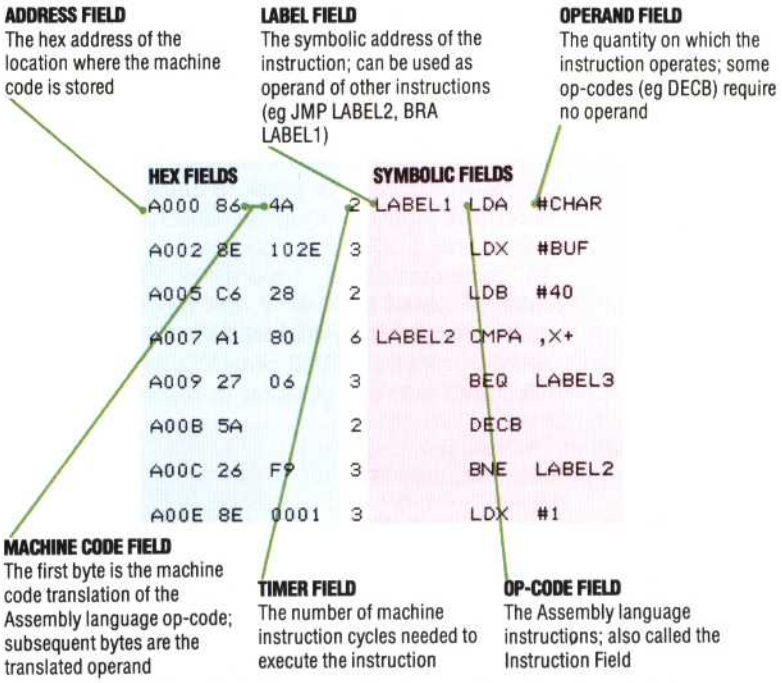
REGISTER TO REGISTER

Our introduction to 6809 machine code began with a general explanation of the role of the registers in the functioning of the microprocessor. Here we look at 6809 registers in greater detail and consider how they are used for storing and moving data.

We have seen that a register is a memory location within the processor chip itself, and have discussed how Assembly language programming involves manipulating the values stored in the registers and transferring these values to and from main memory. We further saw that some registers — mainly the ones that store and process addresses — are 16 bits in length, while others are eight bits in length, and that the various registers perform different functions.

- **Index registers** are used to modify the addresses that we use in our program.
- **Stack pointers** are used by the processor to address workspace memory and can be used in the same way by the programmer for quick storage and retrieval.
- The **program counter** holds the address of the next instruction, and can be altered by the programmer in order to transfer control, giving the Assembly language equivalent of a GOTO statement.
- **Accumulators** are the most frequently used registers, and are used to perform arithmetical functions.
- The **condition code** register contains a number of flags representing the state of the processor (such as whether the last operation gave a zero result); these flags can be tested in order to select or loop, giving the Assembly language equivalent of the IF... THEN structure.

The 6809 processor contains all these registers. As it is a development of the original Motorola 6800 processor (as is the 6502, which is used in the BBC Micro as well as many others), there are many similarities between the Assembly languages used on both processors. They are not, however, compatible — code written for one will not run on the other. Many 6800 programs are source code compatible, however — an Assembly language program written for the 6800 may be reassembled for the 6809 with at least a chance of it running. But even this small degree of compatibility is not available for the 6502 (or its later development, the 6510, which is used in the Commodore 64). However, the similarities between the processors at least mean that the task of translating an Assembly language program into a 6809 version is not too difficult and can be a good introduction to the 6809 for anyone familiar with the 6502. The 6809 contains the following registers:



- Two eight-bit accumulators, known as A and B. There is no functional difference between them when used as eight-bit registers, so either may be used. The fact that there are two of them enables values to be retained in one accumulator while work is done in the other. Alternatively, the two eight-bit accumulators can be treated as a single 16-bit accumulator, a very powerful facility that allows the processor to carry out 16-bit arithmetic directly. Because of this, the 6809 is sometimes referred to as a pseudo-16-bit processor, but this is not really the case and it is better to think of it as being an advanced eight-bit processor. When the two accumulators are used together, they are referred to as the 16-bit D register.
- There are two index registers, referred to as X and Y. Again, there is no functional difference between them: either may be used in any situation. There is one slight operational difference, however, in that some instructions using the Y register will translate into two-byte instructions, as opposed to one byte for the corresponding X register instruction, thus making the program a little longer and slower. Where one index register only is required, it is therefore better to use X.
- The 6809 has two stack pointers, S and U. The processor uses S for all its stack operations. Although the programmer is free to use S if he or she wishes, it is always necessary to ensure that the

Fields Of Study

Assembly language programs look very different from BASIC program lists, and, when printed out by an Assembler program (which translates Assembly language into machine code) can seem completely baffling. The key to understanding them is to concentrate on the columns (or fields) that interest you (usually just the label, op-code and operand fields), and ignore the rest. The specimen program shown should help