Decimal	Binary
-7	1001
-6	1010
-5	1011
-4	1100
-3	1101
-2	1110
-1	1111
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

The most common method of representing negative numbers in a computer memory location or register is the form known as *two's complement* (see page 328). To obtain the two's complement of a binary number, we invert all the digits (change all the zeros to ones, and vice versa), and add one to the number. Thus, the two's complement of 0101 it 1011.

But how is this used to perform mathematics involving signed numbers? First of all, let's consider the range of numbers that can be represented: an eight-bit register can hold only 256 different bit patterns, which can be used to represent positive numbers in the range 0 to 255 *or* negative and positive numbers in the range -128 to 127. (A 16-bit register can hold values from 0 to 65535 *or* -32768 to 32767.) We give a table in the margin that shows how a four-bit binary representation is made for the decimal values from -7 to 7.

If you look at the table, you will notice that the negative numbers all have a one in the most significant (leftmost) bit position. Similarly, all the positive numbers have a zero in the most significant bit position.

As you can see from inspecting this four-bit table, we can define some basic properties for signed mathematics based on two's complement:

- The two's complement of a negative number gives its positive equivalent, and vice versa.

- The most significant bit is always zero for a positive number, and one for a negative number. This makes recognising whether a number is positive or negative very easy.

- The two's complement of zero is zero (1111 plus 1).

- Addition and subtraction can be carried out in the usual way, and any given answer will have the correct sign.

You might like to try a few simple addition and subtraction sums to verify the legitimacy of the last property. Multiplication, however, is more difficult when using signed numbers. The MUL instruction that we used in the BCD-to-Binary program at the start of this instalment treats the contents of the A and B registers as unsigned numbers. If we want to multiply two signed numbers then we must program it ourselves.

Anybody who has done any programming will realise that we are extremely limited in what we can do using the simple 'linear' programs that we have so far used in this course. We can only begin to do useful things by employing one of the basic forms of control structure:

- Selection: in which we choose between two different courses of action (like the IF statement in BASIC)

- Repetition: in which we repeat a sequence of operations:

1) while a certain condition remains true (the WHILE . . . WEND structure);
2) until a certain condition becomes true (REPEAT . . . UNTIL); or
3) a certain number of times (FOR . . . NEXT).

All of these structures depend on the ability to test a condition to see whether it is true or false, the most common sort of condition being whether a variable has a certain value or not. In Assembly language, we need to use these structures, and will therefore need to be able to test the values in registers. We can usually test directly for only two possibilities (whether a value is zero or not, and whether it is positive or negative). With extra instructions, however, it is possible to carry out other sorts of test.

CONDITION CODE REGISTER

These conditions are made available by using the condition code (CC) register, which we briefly mentioned earlier in the course (see page 537). This is an eight-bit register, but unlike the other 6809 registers, we are not interested in the value stored there. Rather, we are concerned with the state (1 or 0) of each of the eight bits individually. Five of the eight bits are devoted to conditions of the type we have been discussing, the other three are concerned with the handling of interrupts (which we will examine in detail later in the course). One of the five, H (the Half carry flag), is almost solely concerned with BCD arithmetic, and doesn't concern us at present. The remaining four, which are important at this stage, are:

- C: The Carry flag, which holds the carry digit (or borrow in the case of a subtraction) from the most significant bit after an arithmetic operation. It also has a useful function when we want to shift the contents of an accumulator along by one bit; some of the shift operations put the bit that is lost off the end into C. This bit, for example, could be used to test whether a number is odd or even by having the least significant bit shifted into it and tested. This is bit 0 (the least significant bit) in CC.

- V: The oVerflow flag, which is set to one whenever the result of an arithmetic operation is too large for the register that is supposed to contain it. This is bit 1 in CC.

- Z: The Zero flag, which is set to one when the contents of a register are zero. This is bit 2 in CC.

- N: The Negative flag, which is a copy of the most significant bit (the sign bit) of a number in a register; in other words it is set to one if the number is negative. This is bit 3 in CC.

It is one of the most difficult aspects of Assembly language programming to keep track of the state of the flags. Not every instruction will set the flags, and some flags are set depending on the contents of the accumulator while others can depend on other registers as well. The safest procedure is to test only on the values in an accumulator, and to