Most data-processing instructions can also update the condition codes according to their result. Placing an "S" after the mnemonic will cause the flags to be updated. For example there are two versions of the MOV instruction:

MOV	R0, #0	Will move the value 0 into the register R*0* without setting the flags.
MOVS	R0, #0	Will do the same, move the value 0 into the register R*0*, but it will also set the condition code flags accordingly, the Zero flag will be set, the Negative flag will be reset and the Carry and oVerflow flags will not be effected.

If an instruction has this ability we denote it using $\langle S \rangle$ in our description of the instruction. The $\langle S \rangle$ always comes after the $\langle cc \rangle$ (conditional execution) modification if it is given. Thus the full description of the move instruction would be:

MOV$\langle cc \rangle \langle S \rangle$ R*d*, $\langle op1 \rangle$

With all this in mind what does the following code fragment do?

```
MOVS     R0, R1
MOVEQS   R0, R2
MOVEQ    R0, R3
```

The first instruction will move R*1* into R*0* unconditionally, but it will also set the N and Z flags accordingly. Thus the second instruction is only executed if the Z flag is set, i.e., the value of R*1* was zero. If the value of R*1* was not zero the instruction is skipped. If the second instruction is executed it will copy the value of R*2* into R*0* and it will also set the N and Z flags according to the value of R*2*. Thus the third instruction is only executed if both R*1* and R*2* are both zero.

3.5.2 Data Processing Operands: $\langle op1 \rangle$

The majority of the instructions relate to data processing of some form. One of the operands to these instructions is routed through the Barrel Shifter. This means that the operand can be modified before it is used. This can be very useful when dealing with lists, tables and other complex data structures. We denote instructions of this type as taking one of its arguments from $\langle op1 \rangle$.

An $\langle op1 \rangle$ argument may come from one of two sources, a constant value or a register, and be modified in five different ways. See Chapter **??** for more detailed information.

Unmodified Value

You can use a value or a register unmodified by simply giving the value or the register name. For example the following instructions will demonstrate the two methods:

MOV	R0, #1234	Will move the immediate constant value 1234_{10} into the register R*0*
MOV	R0, R1	Will move the value in the register R*1* into the register R*0*

Logical Shift Left

This will take the value of a register and shift the value up, towards the most significant bit, by n bits. The number of bits to shift is specified by either a constant value or another register. The lower bits of the value are replaced with a zero. This is a simple way of performing a multiply by a power of 2 ($\times 2^n$).