Here's the bit layout of an ARM data processing instruction

Any instruction with bits 27 and 26 as 00 is “data processing”. The four-bit opcode field in bits 24–21 defines exactly which instruction this is: add, subtract, move, compare, and so on. 0100 is ADD.

Bit 25 is the "immediate" bit. If it's 0, then operand 2 is a register. If it's set to 1, then operand 2 is an immediate value.

Note that operand 2 is only 12 bits. That doesn't give a huge range of numbers: 0–4095, or a byte and a half. Not great when you're mostly working with 32-bit numbers and addresses.
Each ARM instruction is encoded into a 32-bit word.

Access to memory is provided only by Load and Store instructions.

ARM data-processing instructions operate on data and produce a new value.

They are not like the branch instructions that control the operation of the processor and sequencing of instructions.
But ARM doesn't use the 12-bit immediate value as a 12-bit number. Instead, it's an 8-bit number with a 4-bit rotation, like this:

![Diagram showing 4-bit rotation and immediate values]

The 4-bit rotation value has 16 possible settings, so it's not possible to rotate the 8-bit value to any position in the 32-bit word. The most useful way to use this rotation value is to multiply it by two. It can then represent all even numbers from zero to 30.

To form the constant for the data processing instruction, the 8-bit immediate value is extended with zeroes to 32 bits, then rotated the specified number of places to the right. For some values of rotation, this can allow splitting the 8-bit value between bytes.
Register, optionally with shift operation
- Shift value can be either be:
  - 5 bit unsigned integer
  - Specified in bottom byte of another register.
- Used for multiplication by constant

Immediate value
- 8 bit number, with a range of 0-255.
  - Rotated right through even number of positions
- Allows increased range of 32-bit constants to be loaded directly into registers
A barrel shifter is a digital circuit that can shift a data word by a specified number of bits in one clock cycle.

It can be implemented as a sequence of multiplexers (mux.), and in such an implementation the output of one mux is connected to the input of the next mux in a way that depends on the shift distance.
The rotated byte encoding allows the 12-bit value to represent a much more useful set of numbers than just 0–4095. It's occasionally even more useful than the MIPS or PowerPC 16-bit immediate value.

ARM immediate values can represent any power of 2 from 0 to 31. So you can set, clear, or toggle any bit with one instruction.

```
ORR r5, r5, #0x8000 ; Set bit 15 of r5
BIC r0, r0, #0x20 ; ASCII lower-case to upper-case
EOR r9, r9, #0x80000000 ; Toggle bit 31 of r9
```

More generally, you can specify a byte value at any of the four locations in the word:

```
AND r0, r0, #0xff000000 ; Only keep the top byte of r0
```
In practice, this encoding gives a lot of values that would not be available otherwise. Large loop termination values, bit selections and masks, and lots of other weird constants are all available.

Faced with the constraint of only having 12 bits to use, the ARM designers had the insight to reuse the idle barrel shifter to allow a wide range of useful numbers.

No other architecture has this feature. It's unique.
0xC0000034

\[\begin{array}{cccccccccccccccccccccccccccc}
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0
\end{array}\]

0x1  \hspace{2cm} 0xD3

0xD3 ROR 2
However, for a large immediate value “mov” does NOT work, it is not ok to use mov r0,#0x55555555.
The reason is because there is no way to fit a 32-bit data into a 32-instruction (an instruction must have the instruction-code-part and the data-part, if the data-part is 32-bit long, there is no room to store the instruction-code-part).

Instead use ldr r0,=0x55555555

Then, the assembler will generate some code to place the constant 0x55555555 in a nearby table in the code area. Then it uses an instruction to load a data from that table pointed by the program counter and an offset to fill up r0.

LDR rn, [pc, #offset to literal pool]
    ; load register n with one word
    ; from the address [pc + offset]