

**why do I write LDR r0, =0x00000007. is it same as MOV r0, #7**

**Yes, in this specific case LDR r0, =0x00000007 and MOV r0, #7 will produce the same result, but they work differently.**

**Key Differences**

**LDR r0, =0x00000007 (Pseudo-instruction)**

This is a **pseudo-instruction** that tells the assembler to load the value 7 into r0 using the most efficient method. The assembler will analyze the value and choose the optimal instruction:[[1]](#fn1)[[2]](#fn2)

* **For small values like 7**: The assembler will optimize this to MOV r0, #7 because 7 fits within ARM's 12-bit immediate encoding limitations[[2]](#fn2)[[1]](#fn1)
* **For larger values**: The assembler would place the value in a literal pool and generate a PC-relative LDR instruction[[1]](#fn1)[[2]](#fn2)

**MOV r0, #7 (Direct instruction)**

This directly moves the immediate value 7 into register r0. ARM's MOV instruction can handle immediate values using a 12-bit encoding scheme (8 bits rotated by multiples of 2).[[1]](#fn1)

**Why Use LDR r0, =value?**

The LDR r0, =value pseudo-instruction is preferred for several reasons :[[2]](#fn2)[[1]](#fn1)

**Flexibility**

* Can handle **any 32-bit constant**, regardless of ARM's immediate encoding limitations
* You don't need to worry about whether a value can be encoded in a MOV instruction

**Optimization**

* The assembler automatically chooses the most efficient instruction:
  + Uses MOV when the value fits the 12-bit immediate format
  + Uses literal pool loading for larger values that can't be encoded in MOV[[2]](#fn2)[[1]](#fn1)

**Convenience**

* Provides a universal way to load constants without checking encoding limitations
* Simplifies programming by letting the assembler handle optimization decisions[[1]](#fn1)

**What Actually Happens**

For LDR r0, =0x00000007:

1. Assembler sees the value 7 can be encoded in a MOV instruction
2. Generates: MOV r0, #7
3. Result: Same machine code as writing MOV r0, #7 directly[[2]](#fn2)

The LDR r0, =value syntax is essentially a "smart" way to load constants that works for any value, while MOV r0, #value only works for values that fit ARM's immediate encoding constraints.[[3]](#fn3)[[1]](#fn1)

⁂

1. <https://stackoverflow.com/questions/14046686/why-use-ldr-over-mov-or-vice-versa-for-constants-in-arm-assembly>

1. <https://duetorun.com/blog/20230606/arm-ldr-demo/>

1. <https://en.eeworld.com.cn/news/mcu/eic299559.html>

1. <https://developer.arm.com/documentation/dui0041/latest/Babbfdih>

1. <https://developer.arm.com/documentation/dui0489/c/arm-and-thumb-instructions/pseudo-instructions/ldr-pseudo-instruction>

1. <https://www.reddit.com/r/asm/comments/irynwk/whats_the_pro_in_using_mov_if_it_can_only_move/>

1. <https://comp.sys.arm.narkive.com/fm2axqD4/diference-between-ldr-and-mov-instruction>

1. <https://azeria-labs.com/memory-instructions-load-and-store-part-4/>