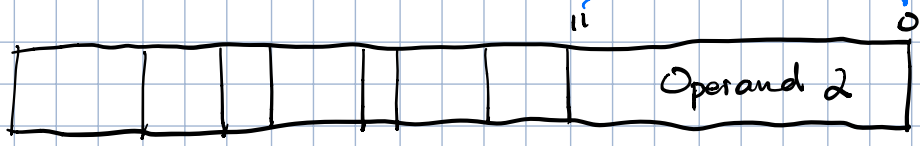
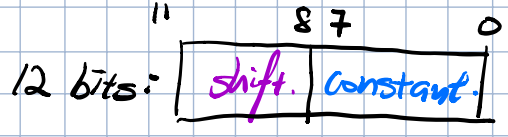
Data processing instructions. (# Immediate operands)

- For labs 1 and 2 your processor's mvi instruction read its 16-bit #D operand from the next address in memory. For ARM, all instructions use consists of one machine code word. A #D argument has to be embedded inside the machine code.

(**) 12 bit #D



(**) If these 12 bits were used directly for #D then we could use constants to a max. value $< 2^{12} = 4096$. Instead, ARM breaks the 12 bits into two fields.



- before using the constant, ARM rotates it to the right an even number of times. (shift ×2 times)

Example

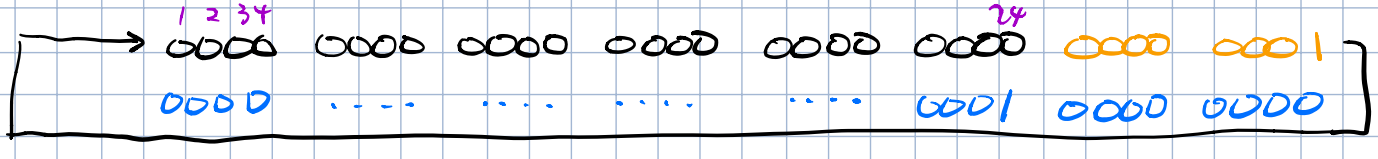
#0 is represented as 0 0 0000 0000 0000

#1 is represented as 0 1 0000 0000 0001

⋮

#255 is represented as 0 255 0000 0000 1111 1111

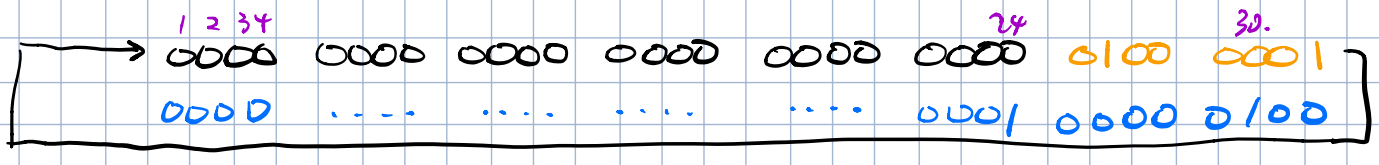
#256 is represented as 12 1 1100 0000 0000 0001



rotate 12×2 = 24 time.

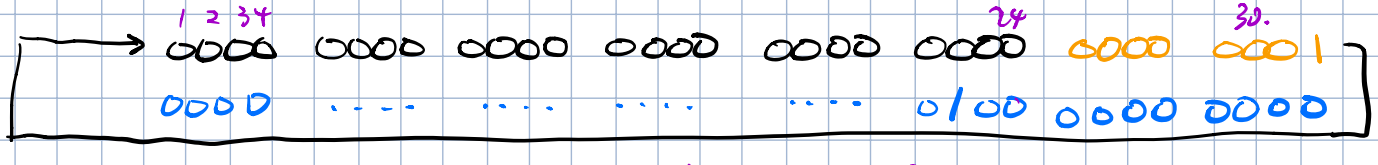
#256-259 couldn't represent in this way.

#260: is represented as 15 65 → 1111 01000001



rotate 15 ×2 = 30 time.

#1024: is represented as 11 1 → 1011 00000001



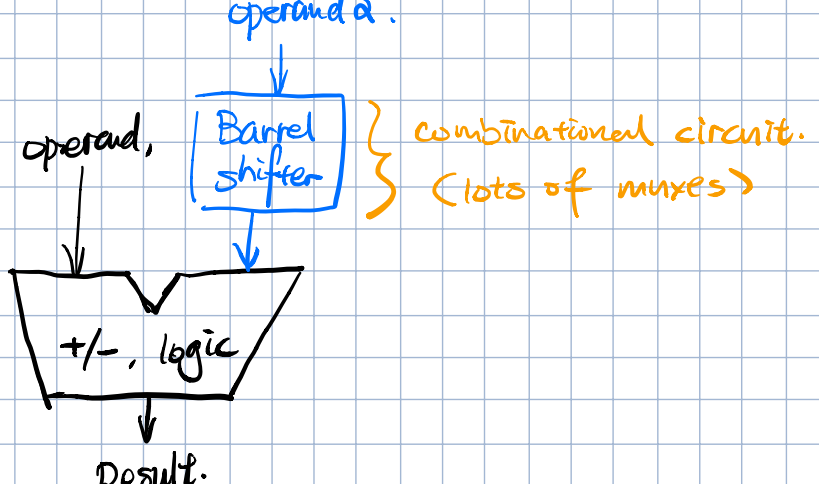
rotate 11 ×2 = 22 times

Summary: some #D constants > 255 can be represented, others cannot.

In general, #D can be used if it can be represented as a byte that will be rotated an even number of times.

Questions: How does the ARM implement the "rotate"?

Answer: there's a fast shifter (called a Barrel shifter) for Operand2.



If operand 2 is a register, R_s, then this register can be shifted.

ADD R0, R1, R2, LSL #2. R0 ← R1 + (R2 × 4)

ADD R0, R1, R2, ASR #4. R0 ← R1 + (R2 ÷ 16)

mov R4, R5, LSL #3 (*). R4 ← R5 × 8

(*) This mov is actually the same as LSL!

LSL R0, R1, #2 ⟷ mov R0, R1, LSL #2

LSL R0, R1, R2 ⟷ mov R0, R1, LSL R2

shift R1 by amt in R2