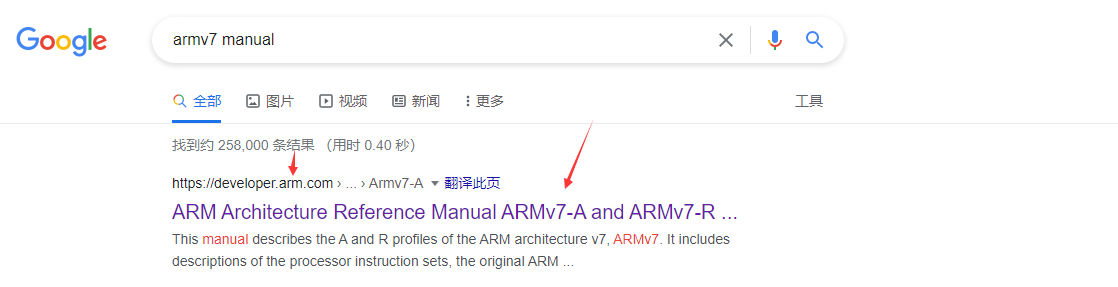
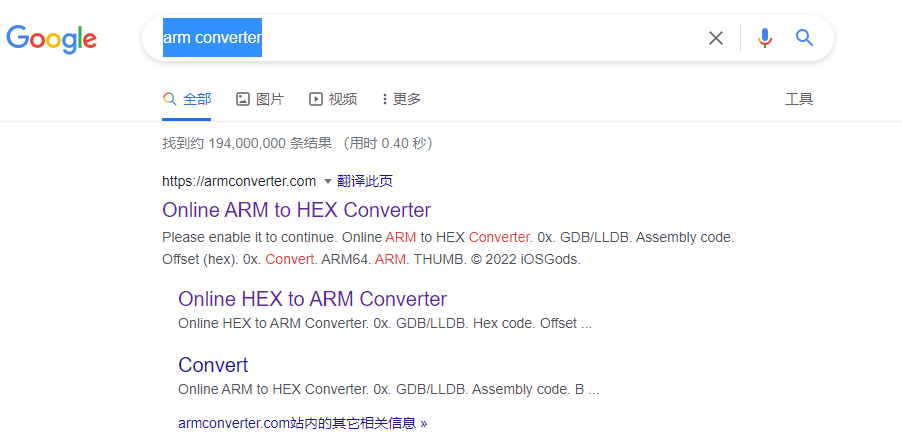
在arm官网可以下载相应的手册。





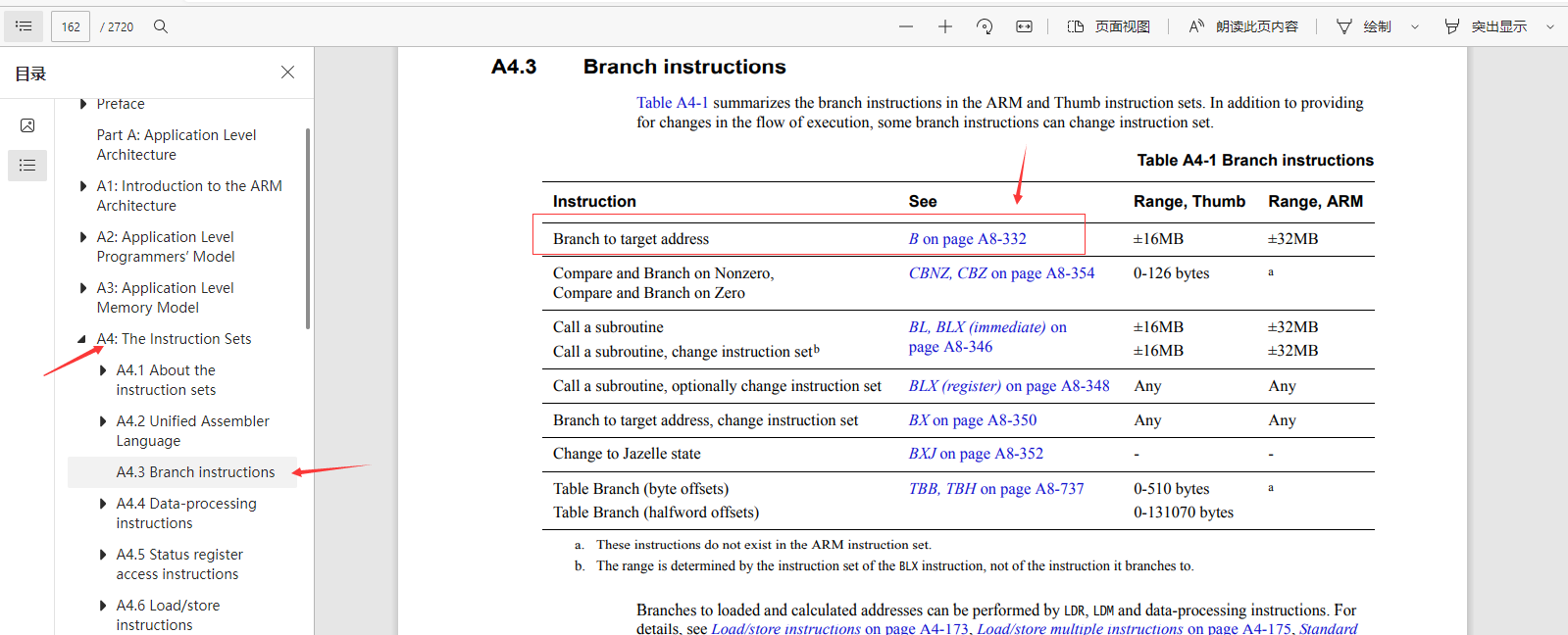
Arm converter可以实现指令和机器码的相互转换



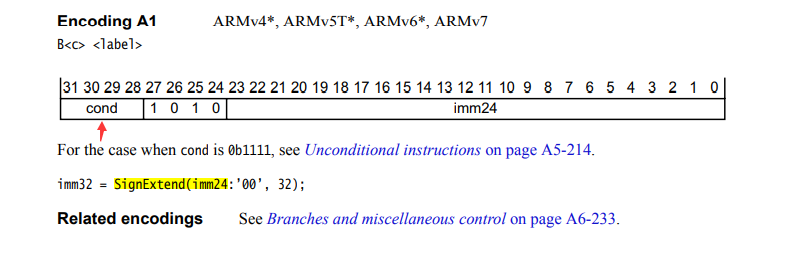
# Armv7

## B

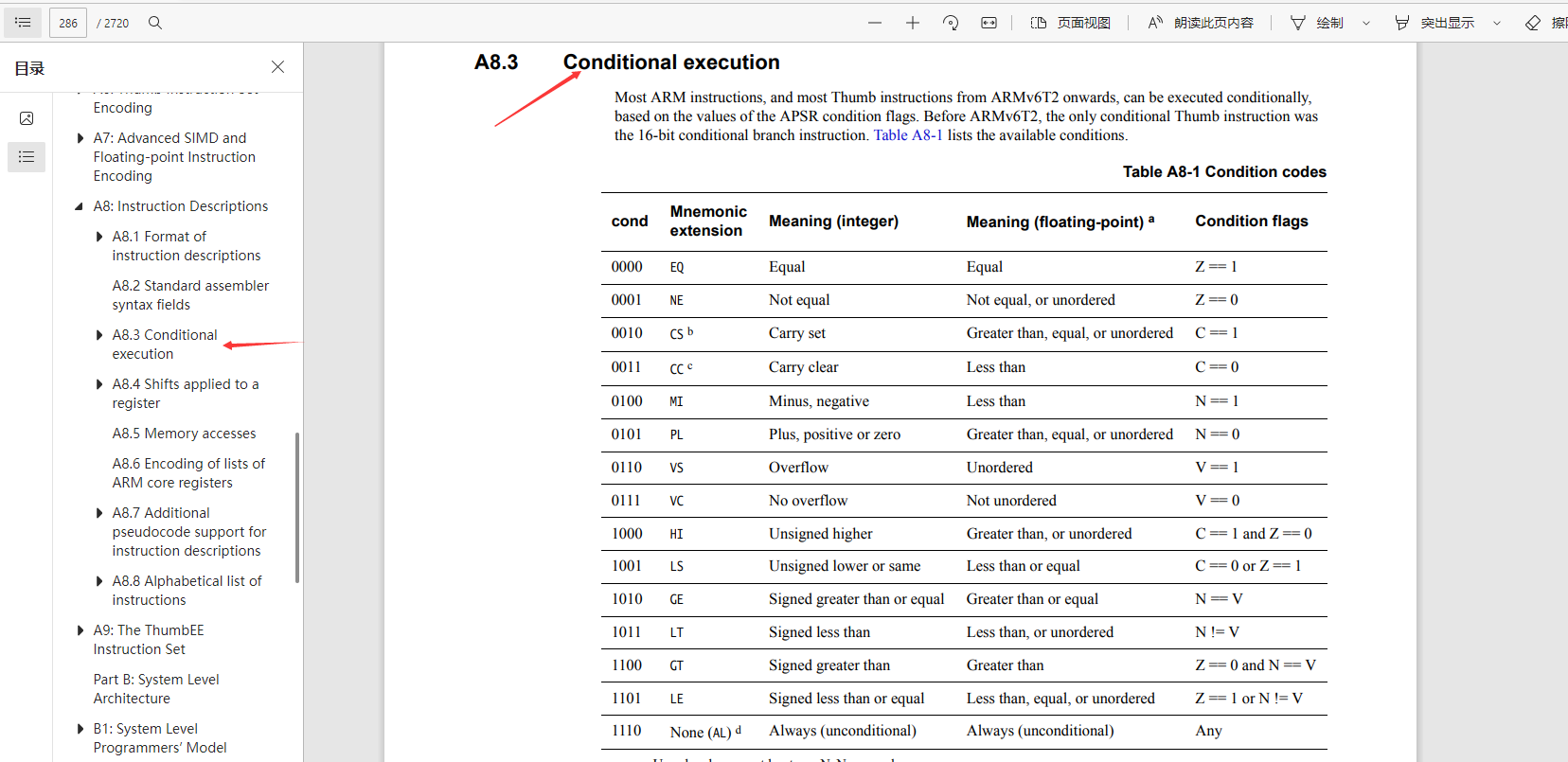
这里示范一个beq #0xc。B指令属于分支跳转指令，在The Instruction Sets-》Branch instructions中找到B



点一下跳到B指令，找到下面A1（A->ARM）



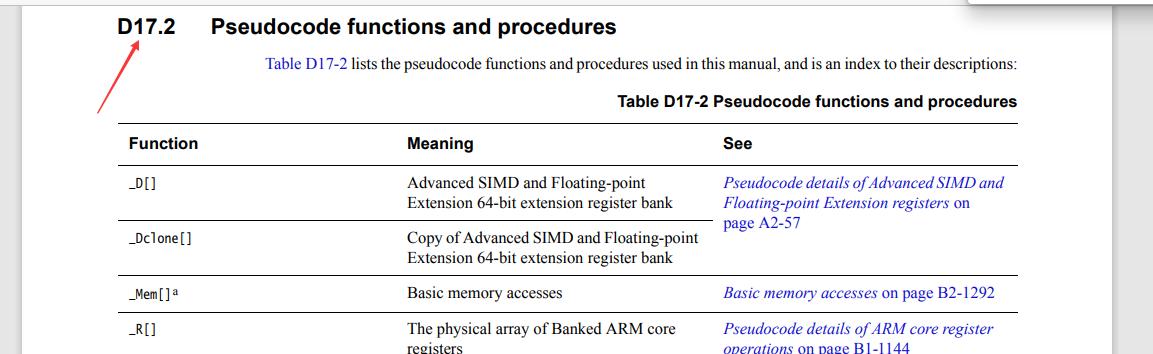
Cond在A8.3可以找到对应的

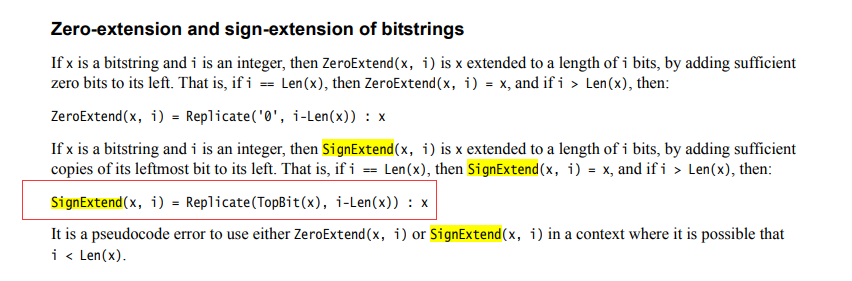


所以我们的beq #0xc指令前8位是b00001010

后面imm32 = SignExtend(imm24:’00’, 32);

SignExtend可以在D17.2 Pseudocode functions and procedures中查看伪代码的定义





所以

cond = 0000

imm24 = 0000 0000 0000 0000 0000 0001

imm24:’00’ -> 0000 0000 0000 0000 0000 000100, 这里不足32bit

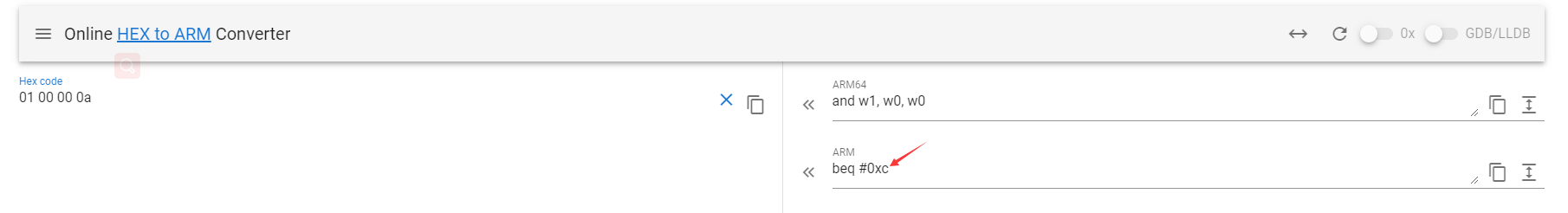
imm32 = SignExtend(imm24:’00’, 32);

imm32 -> 0000 0000 0000 0000 0000 0000 0000 0100 -> 4

由于pc+8，所以实际是0x4+ 0x8 = 0xc

整个opcode：b0000 1010 0000 0000 0000 0000 0000 0001代表beq #0xc

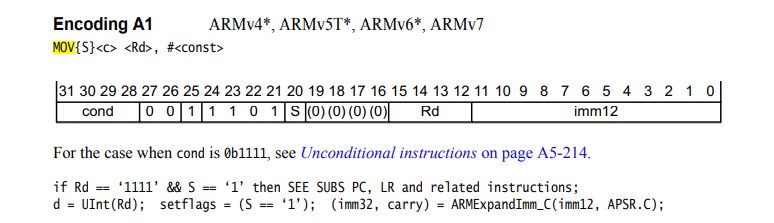
小端序：0x0100000a

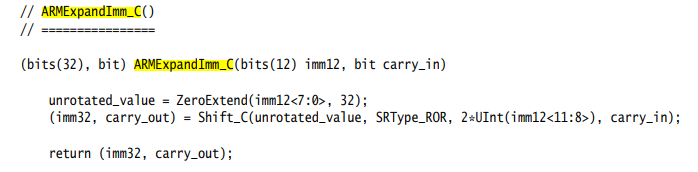


## MOV

写个MOV r1，#1

所有的指令都能在A8.8的按字母表排序的指令中找到

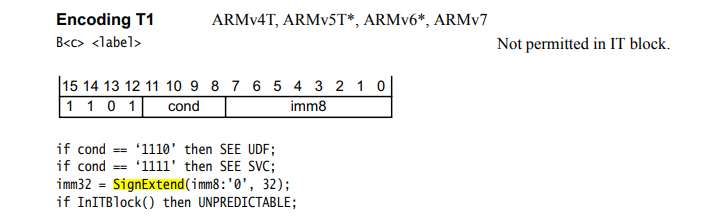




0000 00 1 1101 0 0000 0001（Rd寄存器）imm12: 0000 0000 0001 =>0x0110A0E3

# Thumb

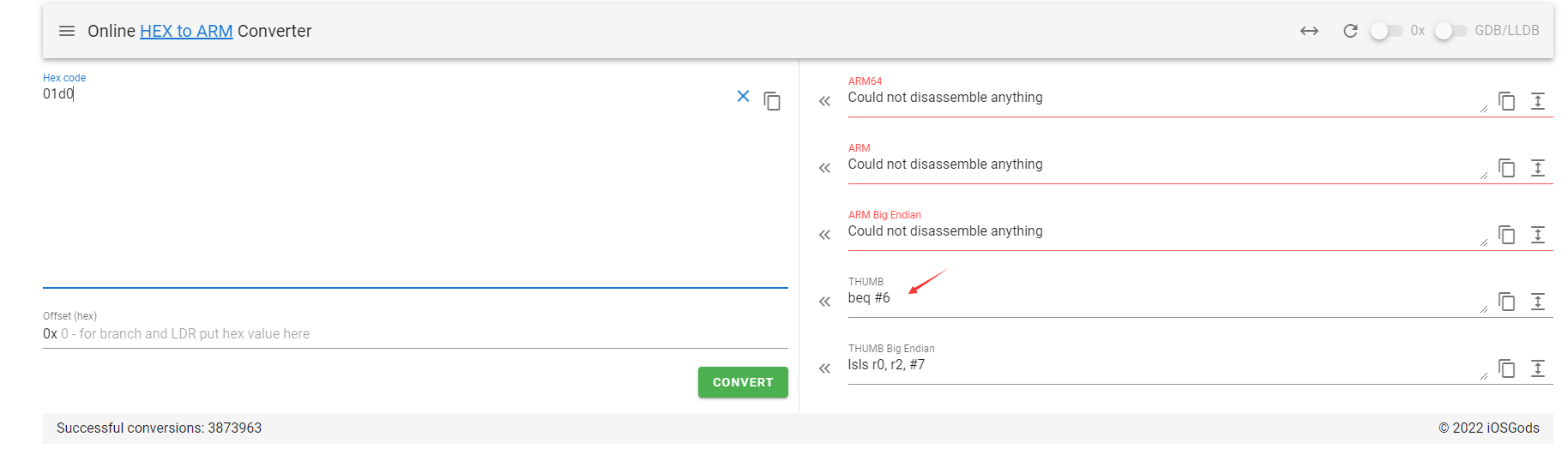
## B



Imm32 => SignExtend(0010,32) => 2 + 4 => 6

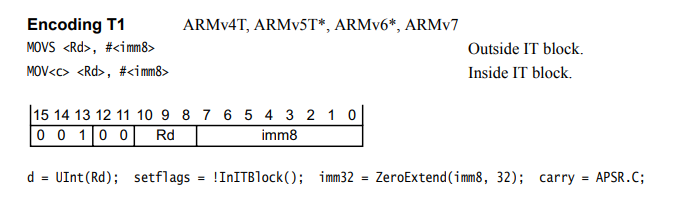
Thumb一条指令16位，所以跳2条指令就是4 byte

所以Beq #6 => b1101 0000 0000 0001 => 0x01d0

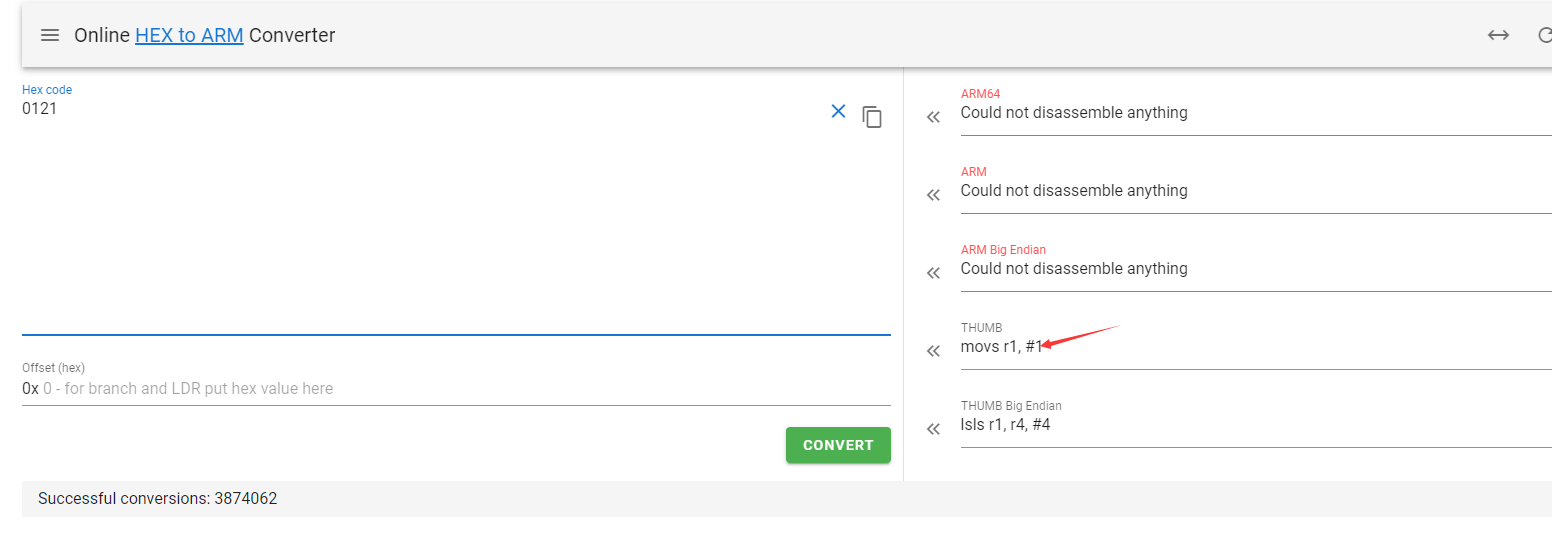


## MOV

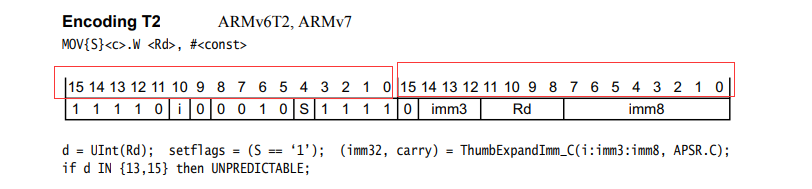
### 16位thumb



0b 001 00 001 0000 0001=0x0121



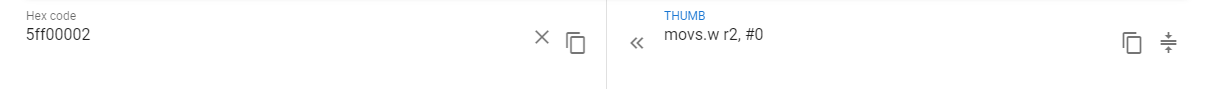
### 32位thumb



这里有一点点区别16位+16位，也就是小端序16位+小端序16位

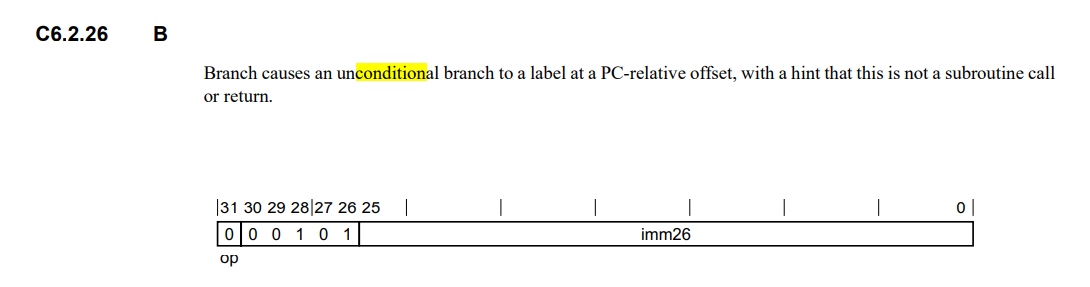
0b1111 0000 0101 1111 0000 0010 0000 0000 = 0x5ff0 0002

MOVS.w r2,#0



# ARMv8

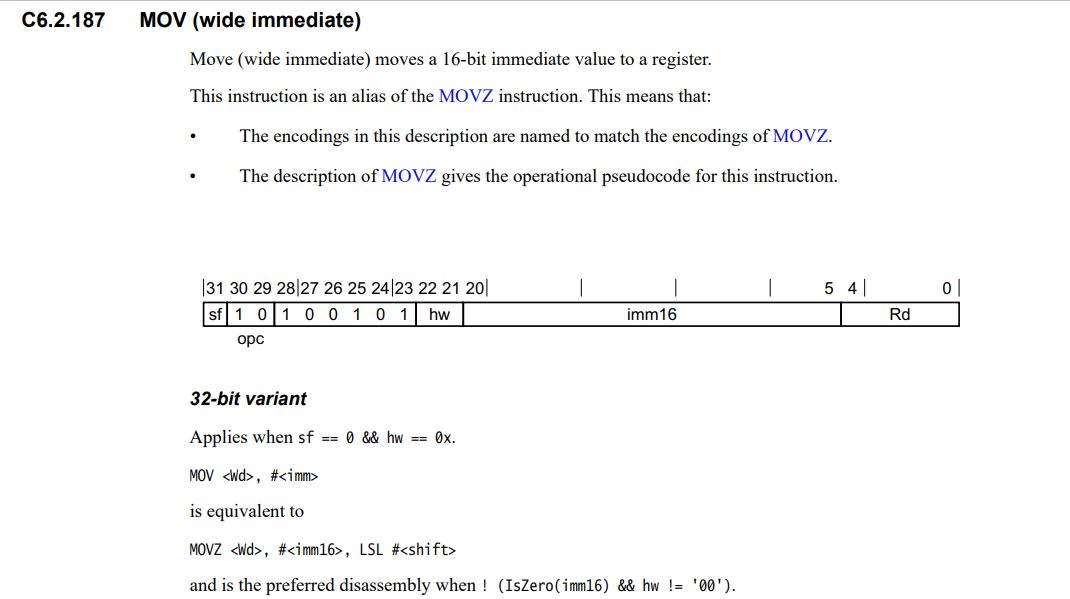
## B



写个B #8，

0b0001 0100 0000 0000 0000 000 0 0000 => 0x00 00 00 14

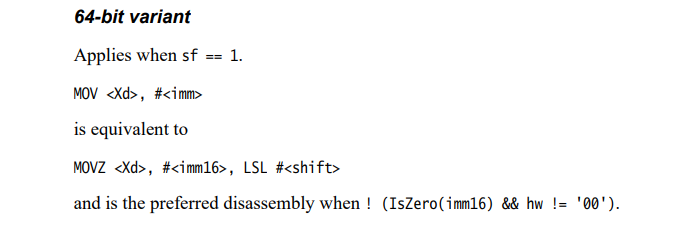
## MOV



来个mov w0, #1

0b0101 0010 1000 0000 0000 0000 0010 0000 = 0x20 00 80 52

来个mov x0, #1



0b1101 0010 1000 0000 0000 0000 0010 0000 = 0x20 00 80 d2