Comments: - Each ARM instruction is assembled into a 32-bit world of machine code.

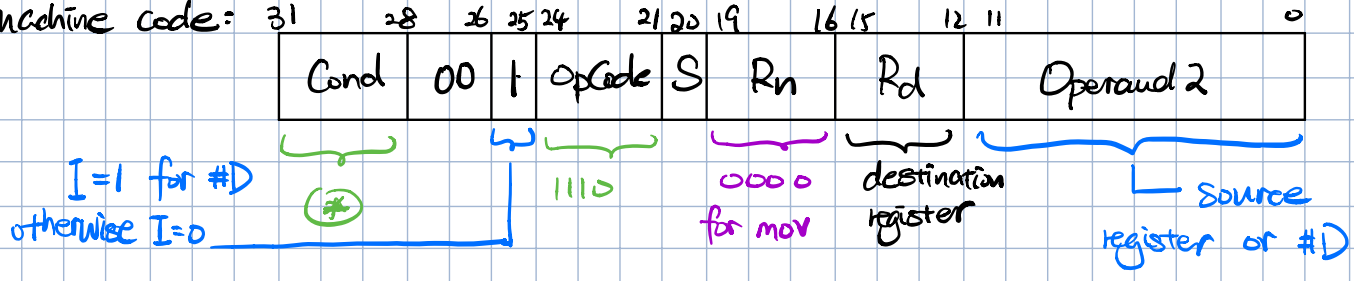
- mov is like both mv and mvi in our previous processor. The 2nd argument for mov is quite flexible when this argument is #D the size of D is quite "limited."

General form

mov Rd , Operand 2

- Operand 2 can be a register , Rd , constant #D,

Machine code:



* In our processor mvnz & mvnc execute based on a condition

For ARM, almost all instructions can be based on a condition

The "special" condition Always is the default, which has cond = 1110

Examples

mov R1, R2 — 1110 00 0 1101 0 0000 0001 000....010

mov R3, #5 — 1110 00 1 1101 0 0000 0011 00....0101

movNE R5, R6 — 0001 00 0 1101 0 0000 0101 00....0110

└ condition NE means not equal to. So R5 ← R6 iff Z=0

Cond = 0001

- The .word directive supports multiple arguments, separated by commas. Each argument produced a 32-bit word in memory.

- LDR is like ld for our previous processor. Again, the 2nd argument is flexible.

Examples

LDR R2, [R1] ① // R2 ← contents of memory at addr. in R1

LDR R2, [R1, #4] ② // R2 ← contents of memory at addr. given by R1 + 4

LDR R2, [R1, #4]! ③ // R2 ← contents of memory at addr. given by R1 + 4. Then R1 ← R1 + 4

LDR R2, [R1], #4 ④ // R2 ← contents of memory at addr. in R1, Then R1 ← R1 + 4

LDRNE R2, [R1] // R2 ← contents of memory at addr. in R1, if Z = 0.

①-④ are called Addressing Modes.

①-②: Index mode

③: pre-index mode.

④: post-index mode.

In all of the above examples, R_1 is called the base register. You can also involve an index register:

```
LDR   R2 , [R1 , +R3]   // R2 ← contents of memory @ Addr. R1+R3

LDR   R2 , [R1 , -R3]   // R2 ← contents of memory @ Addr. R1-R3
```

- plus there are a few other varients of addressing modes. When we write assembly code, we will usually use index mode. The more complex modes are generally used in code generated by a compiler (for a language like C)