MOV Rd, Operand 2 Rd ← Operand2 (similar to Lab 2 processor "mv" instruction)

- Rd is destination register
- Operand 2 is very flexible
- can be a register, immediate & lots of other things
  - → each kind of MOV has a code (32 bits)
  - (online notes for examples)

LDR

- similar to ld in Lab 2
- More capability
- Function: load/read from memory & put into register
- Load a 32 bit number from memory
- Different forms:

		Address Mode
① LDR R1,[R0]	R1 ← data from memory at [R0]	Index
② LDR R1,[R0,#4]	R1 ← ... [R0+4]	Index+offset Mode
③ LDR R1,[R0,#4]!	R1 ← ... [R0+4] then do R0 ← R0+4	Pre-Index Mode
④ LDR R1,[R0],#4	R1 ← ... [R0] then R0 ← R0+4	Post-Index Mode
⑤ LDRNE R1,[R0]	R1 ← ... [R0] iff Z=0	(not gonna use in this course)
⑥ LDR R1,[R0,+R3]	R1 ← ... [R0+R3]	

- In these examples, R0 is called the base register
- Can write "add 4 numbers" more succinctly

```
_start: MOV R1, #LIST
        LDR R2,[R1]      // R2 ← [R1] = 10
        LDR R3,[R1,#4]   // R3 ← [R1+4] = 20
        ADD R2, R3       // R2 ← R2+R3
        LDR R3,[R1,#8]
        ADD R2,R3
        LDR R3,[R1,#12]
        ADD R2,R3
end:    B   END
LIST:   .word 10, 20, 30, 40
```

Lab 3, you'll be manipulating bytes. Each instruction be told to manipulate just bytes in memory.

Redo → add 4 bytes, not 32 bit words.

```
_start: MOV R0,#LIST
        LDRB R2,[R1]     // R2 ← byte @ address R1. Most significant 24 bits of R2 ← 0.
        LDRB R3,[R0,#1]  // want consecutive bytes, not words as before
                         Least ...  8 ... ... R1 ← [R0]
        ADD  R2,R3
        LDRB R3,[R1,#2]
        ADD  R2,R3
        LDRB R3,[R1,#3]
        ADD  R2,R3
END:    B   END
LIST:   .byte 10, 20, 30, 40
```