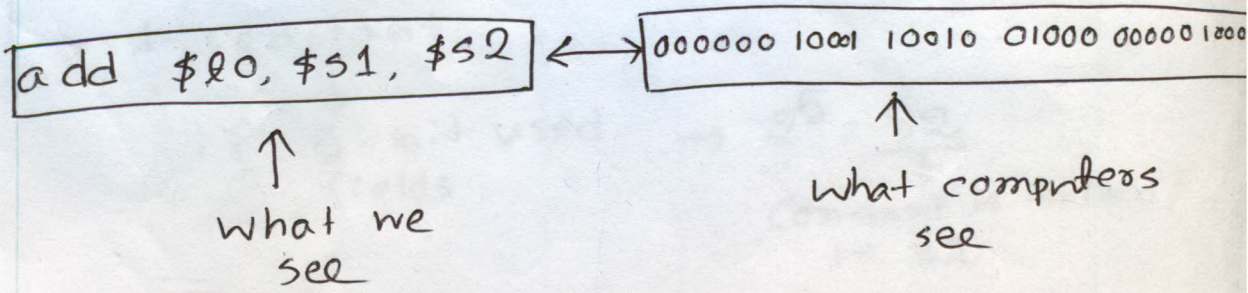
Representing Instructions in the Computer



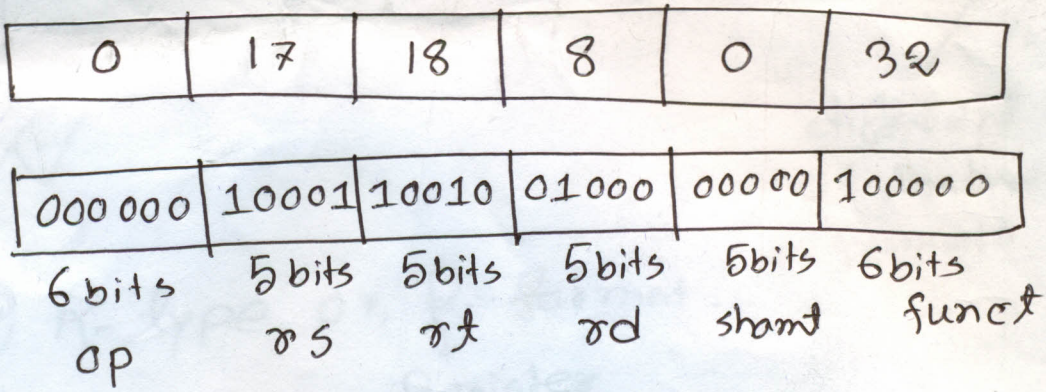
Register names ⟷ numbers

$s0 - $s7 ⟷ 16-23

$t0 - $t7 ⟷ 8-15

Example:

add $t0, $s1, $s2



op + funct ⟶ addition

rs ⟶ first source operand register 17, i.e., $s1,

rt ⟶ second source operand register 18, i.e., $s2

rd ⟶ destination register, 8, i.e. $t0

shamt ⟶ unused

- Instruction format
  - exactly 32 bits } Simplicity favors regularity

What will happen if instruction needs

lw $t0, 32($s3)

→ 2- registers

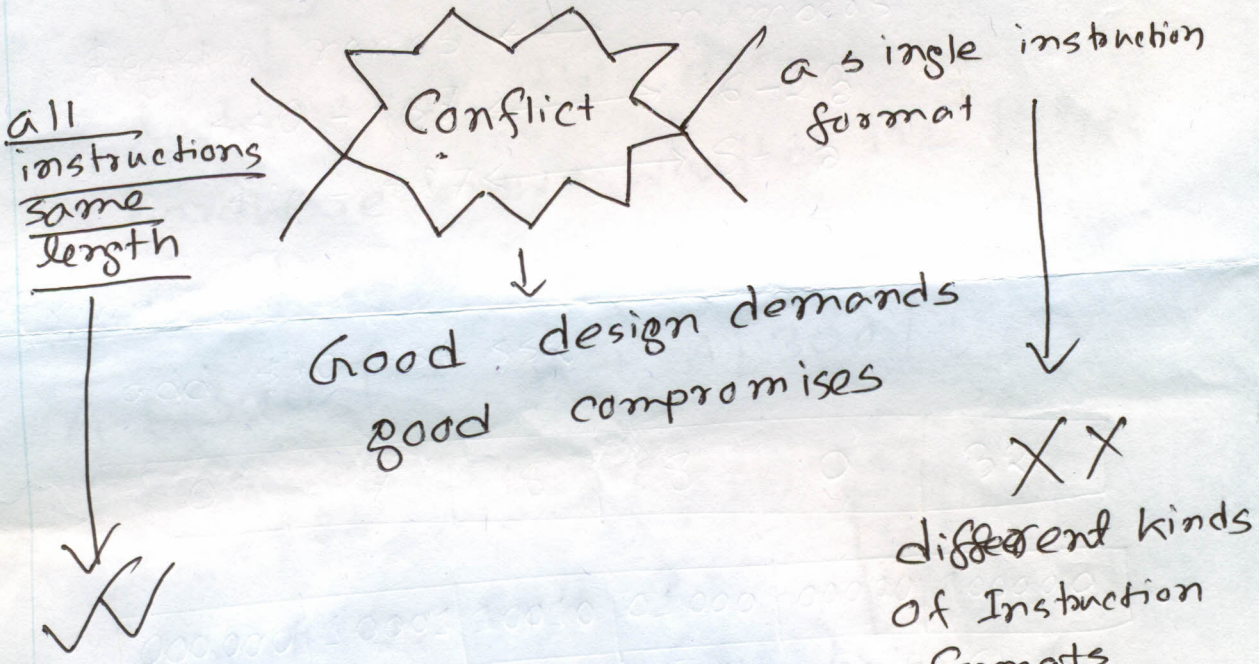
→ 1- constant

↓

if 5-bit used → $2^5 = 32$

fields ↓

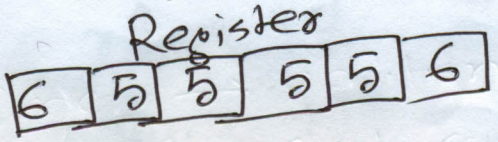
constant is limited to 32



* R-type or, R- format.

↓

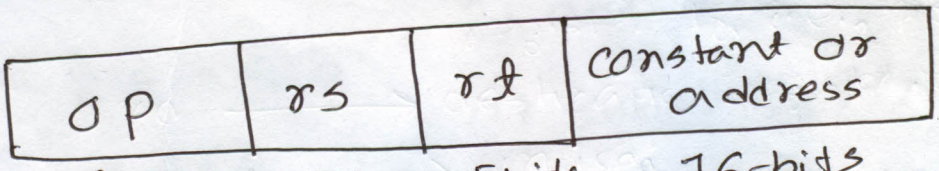
Register



* I-type or, I- format,

↓

immediate

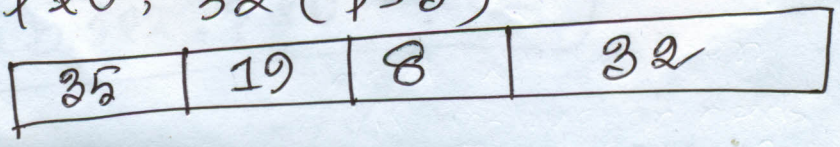


6 bits 5 bits 5 bits 16-bits

rs — source

rt — destination

lw $t0, 32($s3)



Instruction	Format	op	rs	rt	rd	shamt	funct	address
add	R	0	reg	reg	reg	0	32	n.a.
sub	R	0	reg	reg	reg	0	34	n.a.
addi	I	8	reg	reg	n.a.	n.a.	n.a.	constant
lw	I	35	reg	reg	n.a.	n.a.	n.a.	address
sw	I	43	reg	reg	n.a.	n.a.	n.a.	address

looking at this opcode hardware knows whether it's R-type or I-type

(*) A[300] = h + A[300];

```
lw  $t0, 1200($t1)
add $t0, $s2, $t0
sw  $t0, 1200($t1)
```

35	9	8	1200		
0	18	8	8	0	32
43	9	8	1200		

(*)

op	rs	rt	rd	shamt	funct
0	8	9	10	0	34

```
00000001000 01001
010 10000000100010
```

0 - 34 → sub

8 → $t0

9 → $t1

10 → $t2

sub $t2, $t0, $t1

1) sub $t0, $t1, $t2

2) add $t2, $t0, $t1

3) sub $t2, $t1, $t0

4) sub $t2, $t0, $t1