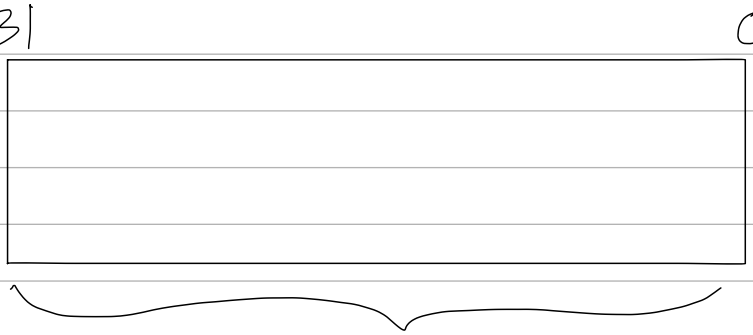


Divide the 32 bits of an instruction into "fields"

↳ Conflict

Desire to keep all the instructions length same

V/s

Desire to have a single instruction format.

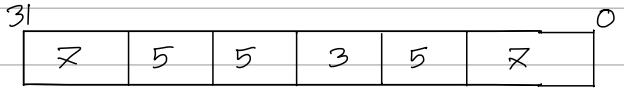
Design Principle 3: Good design demands good compromises.

⇒ RISC-V chooses to keep all the instruction length same; thereby requiring distinct instruction formats for different instructions.

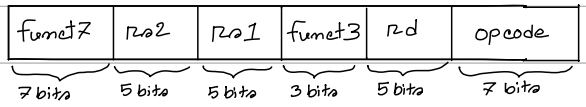
Instruction Formats:

- ↳ R type ⇒ instructions that use 3 registers. (Add, Sub, SLL, XOR, OR, AND, ...)
- ↳ I type ⇒ " " " immediate and 2 registers. (Addi, SLLI, SRLI, ORI, ANDI, Load)
- ↳ S type ⇒ Store instruction

R type instruction



* each field has unique name and size.



(Partially)

Opcode = Denotes the format of an instruction and instruction itself.

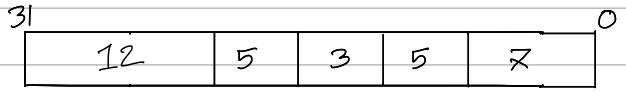
rd = Register destination operand

rs1 = " source 1 "

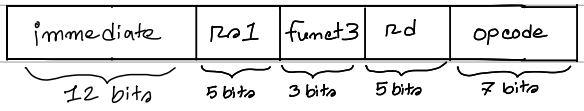
rs2 = " source 2 "

funct3 = } their combination tells
funct7 = } us which instruction
to perform.

I type instruction



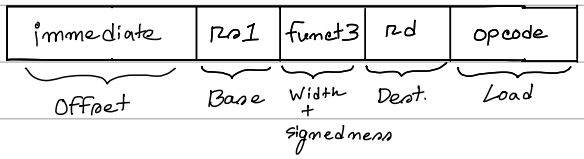
* each field has unique name and size.



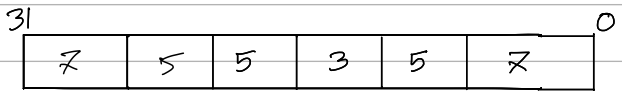
Immediate = Constant / offset [2's comp.]

rs1 = Source / Base Register number

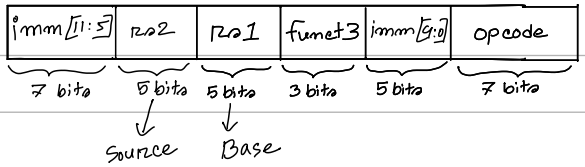
Load



S type instruction



* each field has unique name and size.



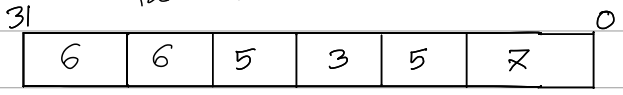
reason for imm. split is they want to keep rs1 and rs2 fields in the same place in all instruction formats.

Shift Operation

↳ follows I-type but modified

slli x11, x10, 4

rd rs imm



Why ?? ⇒ If you shift a 64 bit value more than 63 bits what happens?

Shift Left

↳ shift left and fill the positions with 0

⇒ We can perform multiplication by 2^i using slli.

slli x11, x10, 4

↓

the value that will be stored in x11 is basically, val in x10 * 2^4

Shift Right

↳ shift right and fill the positions with 0

⇒ We can perform division by 2^i using srli.

srli x11, x10, 4

↓

the value that will be stored in x11 is basically, val in x10 / 2^4