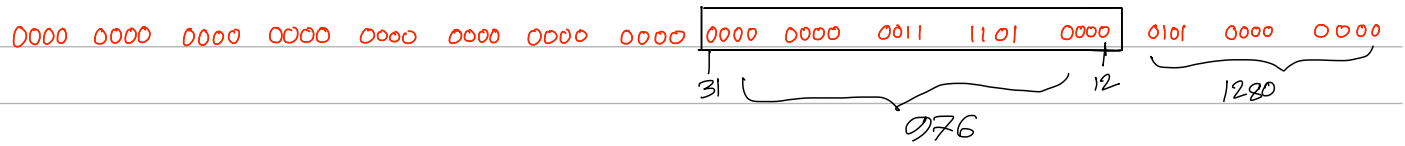
load this 64 bit value into x_{10} ⇒



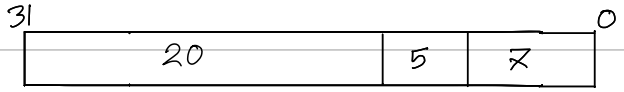
lui x_{10}, 976

x_{10} = 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0011 1101 0000 0000 0000 0000

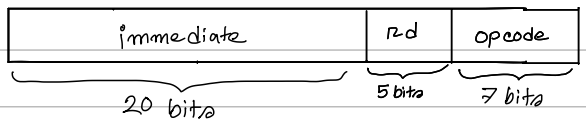
addi x_{10}, x_{10}, 1280

x_{10} = 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0011 1101 0000 0101 0000 0000

U type instruction → LUI

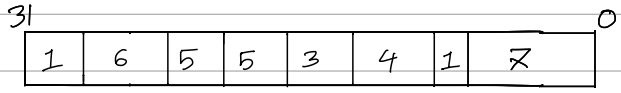


* each field has unique name and size.

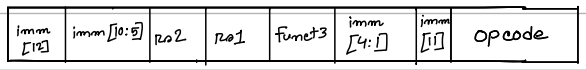


Branching Instructions

SB type instruction — beq, bne, blt, bge



each field has unique name and size.



represents branch addresses in multiples of 2.

forward & backward moving.

the unusual encoding of imm. simplifies datapath design.

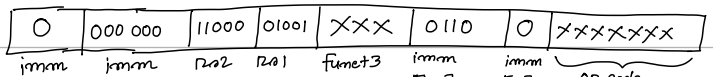
```
        Label-1
#80000  slli  X10, X22, 3      ---- line 1
#80004  add   X10, X10, X25    ---- line 2
#80008  ld    X9,  0(X10)      ---- line 3
#80012  bne   X9,  X24, Exit   ---- line 4
                   rs1  rs2
#80016  addi  X22, X22, 1      ---- line 5
#80020  beq   X0,  X0,  Label-1 --- line 6
Exit:
#80024                         ---- line 7
```

3×4 = 12

0000 0000 1100

0 0000 0000 1100 — Discard it.

12 | 11 | 10:5 | 4:1



5×4 = 20 but its going upward so −20.

= 0000 0001 0100

= 0 0000 0001 0100

= 1 1111 1110 1011

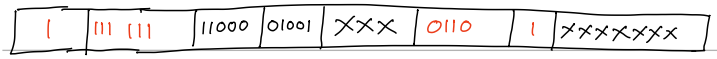
+ 1

1 1111 1110 1100 ⇒ −20 in 2s com.

12 | 11 | 10:5 | 4:1 | 0

PC relative addressing

= PC + immediate×2



0

(i) Form the 12 bit number

11 111 111 0110

(ii) Detect if positive or negative number.

if neg perform 2s comp. again,

1111 1111 0110

0000 0000 1001

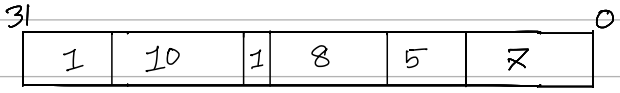
+1

0000 0000 1010 ⇒ 10

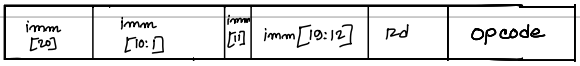
(iii) PC ± imm ×2 → offset

Based on the sign bit

UJ type instruction – Jal

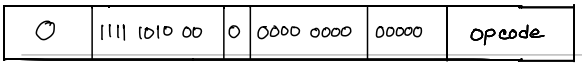


* each field has unique name and size.



* uses 20 bit immediates for larger jumps.
* For more large jump, (32 bit)
lui: load address [31:12] to a temp reg.
jalr: add address [11:0] and jump to target.

Jal X0, 2000



2000 = 0000 0000 0111 1101 0000

= 0 0000 0000 0111 1101 0000

20 | 19:12 | 11 | 10:1