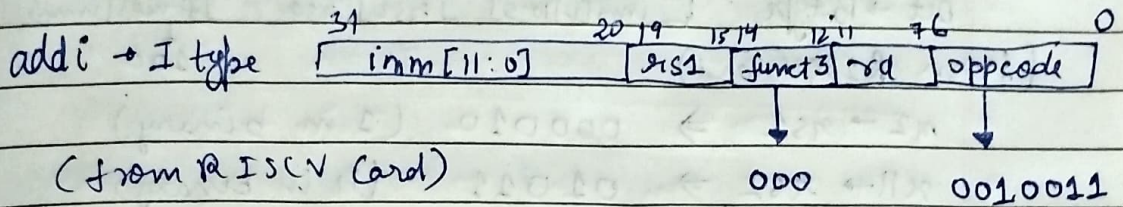


Ques 1 a) addi x15, x22, -45



x15 → rd ⇒ 01111 → (15 in binary)

x22 → rs1 ⇒ 10110 → (22 in binary)

-45 → imm ⇒ 2's complement of 45

45 ⇒ 0000 0010 1101

2's complement of 45 ⇒ 1111 1101 0010 + 1

⇒ 1111 1101 0011 → imm

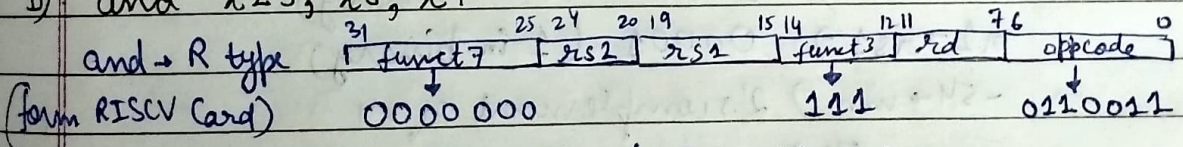
∴ addi x15, x22, -45

Binary ⇒ 1111 1101 0011 1011 0000 0111 1001 0011

imm rs1 funct3 rd opcode

In hex ⇒ 0xf d3 b0 7 9 3

b) and x23, x8, x9



x23 → rd ⇒ 10111 → (23 in binary)

x8 → rs1 ⇒ 01000 → (8 in binary)

x9 → rs2 ⇒ 01001 → (9 in binary)

Assembly ⇒ and x23, x8, x9

Binary ⇒ 0000 0000 1001 0100 0111 1011 0011 0011

funct7 rs2 rs1 funct3 rd opcode

In Hexadecimal ⇒ 0x00947bb3

c) blt x2, x11, 240

blt → B-type
(from RISC-V card)

x2 → rs1 ⇒ 000010 (1 in binary)

x11 → rs2 ⇒ 01011 (11 in binary)

240 → imm ⇒ 000011110000 (240 in binary)

imm[11] imm[10:5] imm[4:1]

Assembly ⇒ blt x2, x11, 240

Binary ⇒ 0000 1110 1011 0001 0100 1000 0110 0011
imm rs2 rs1 funct3 imm opcode

In Hexadecimal ⇒ 0x0eb14863

d) sd x19, -54(x1)

sb → S-type
(from RISC-V card)

x19 → rs2 ⇒ 10011 (19 in binary)

x1 → rs1 ⇒ 00001 (1 in binary)

-54 → imm ⇒ 2's complement of 54

54 → 0000 0011 0110

2's complement of 54 ⇒ 1111 1100 1001 + 1

⇒ 1111 1100 1010
imm[11:5] imm[4:0]

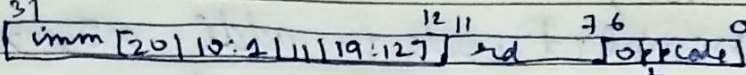
Assembly ⇒ sd x19, -54(x1)

Binary ⇒ 1111 1101 0011 0000 1011 0010 0011
imm opcode

Binary ⇒ 1111 1101 0011 0000 1011 0101 0010 0011
imm rs2 rs1 funct3 imm opcode

In Hexadecimal ⇒ 0xfd30b523

c) jal x3, -10116

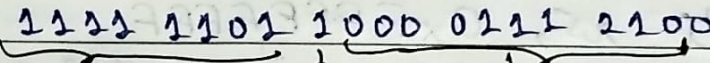
jal → J-Type = 
(from RISC-V card) 2102121

x3 → rd ⇒ 00021 (3 in binary)

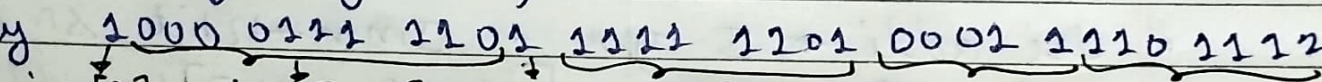
-10116 → imm ⇒ 2's complement of 10116

10116 ⇒ 0000 00100112 1000 0100

2's complement of 10116 ⇒ 1111 1101 1000 0111 1011 + 1

⇒ 
imm[19:12] imm[11] imm[10:1]

~~Binary~~ Assembly ⇒ jal x3, -10116

Binary 
imm[20] imm[10:1] imm[11] imm[19:12] rd opcode

In hexadecimal ⇒ ~~0x08b44863~~ 0x87dfd1ef

Que 2 a) `li x5, -1` $\rightarrow x5 = x0 - 1 = 0 - 1 = -1$

Disassembled code:- `addi x5, x0, -1`

- Here we want to load `x5` with `-1`.
- `addi` \rightarrow I type \rightarrow Imm \rightarrow 12 bits
- So we can load values ~~in~~ ~~range~~ ~~[-2048, 2047]~~ with `x0` in range `[-2048, 2047]` using `addi` by adding `-1` is in this range.

b) `li x5, 0xFFFFFFFF`

Disassembled code:- `addi x5, x0, -1`

$(0xFFFFFFFF)_{hex} \rightarrow (-1)_{decimal}$ [2's complement Representation]

- Load `-1` to `x5`
- Use `addi` \rightarrow I type \rightarrow Imm \rightarrow 12 bit, so can load values in range `[-2048, 2047]` by adding with `x0`.

c) `li x5, 132`

Disassembled code:- `addi x5, x0, 132`

- Here we want to load `132` to `x5`.
- Use `addi` \rightarrow I type \rightarrow Imm \rightarrow 12 bits, so can load values in range `[-2048, 2047]` by adding with `x0`.

d) `li x5, 2134`

Disassembled code:- `lui x5, 0x1`

`addiw x5, x5, -1962`

- $\neq 2134$ not in range `[-2048, 2047]` ^{single} so, `addi` can't be used.
- `lui x5, 0x1` \rightarrow sets first 20 bits
 \rightarrow U type \rightarrow Imm \rightarrow 20 bits \rightarrow (Imm ≤ 12)

Here, Imm = 1 so, $x5 = (1 \ll 12) = 4096$

- Now ~~do~~ `addiw x5, x5, -1962`

add immediate word \checkmark \rightarrow adds sign extended 12-bit to produce sign-extended 32-bit

`addi x5, x5, -1962` $\rightarrow x5 = x5 - 1962 \rightarrow 4096 - 1962 = 2134$

- `2134` is ~~loaded~~ loaded in `x5`

c) li x5, 0x2345abcd

Disassembled code:- lui x5 0x2345b
addiw x5 x5 -1075

- First ~~we load~~ ~~we~~ set first 20 bits with ~~0~~
0x2345b (greater than 0x23459), using lui
- Then use addiw, ~~to~~ subtract 1075, to get 0x2345abcd.
- Similar to que d.

Que 3 (1) lhu x3, 0(x1)

0x0000 0000 0000 3939

- lhu → load half word (unsigned) → ~~zero~~ zero extends
So, 16 bits from 0(x1) & extend 0.
↳ Byte 0

(2) lh x3, 0(x1)

0x0000 0000 0000 3939

- lh → load half word (16 bit) & extend sign-bit
here 3 9 3 9

extend 0 → 0011
for rest 48 bits

(3) lh x3 2(x1)

0xffff ffff ffff 9393

- lh → load half word (16 bit) & extend sign bit
2(x1) → load from Byte 2 of ~~from~~ from(x1)
So, 9393

Here 9393

1001

signbit so extend 1 for rest 48 bits [1111 → f]

(4) `ld x3, 0(x1)` → ~~0x055a a5a5 9393~~ 0x10000
↑

- `ld` → load double word (64 bits) from ~~0x10000~~ `x1` (`0(x1)`)

0x055a a5a5 9393 3939

(5) `lw x3, 12(x1)`

0x0000 0000 3993 3939

- `lw` → load word (32 bits) & extend sign bit
`12(x1)` → ~~Byte~~ 12 from `0(x1)`

Here for sign-bit 3993

signbit = 0011

∴ Extend 0 for 32 bits

(6) `lbu x3, 7(x1)`

0x0000 0000 0000 0a5

- `lbu` → load byte (8 bits) & extend zero
`7(x1)` → first 8 bits from MSB in first word

∴ a5

then extend 0 for rest 56 bits

(7) `lb x3, 7(x1)`

0x ffff ffff ffff fa5

- `lb` → load byte (8 bits) then extend sign bit.
`7(x1)` → first 8 bits from MSB in first word

∴ a5

~~0011~~ 1010

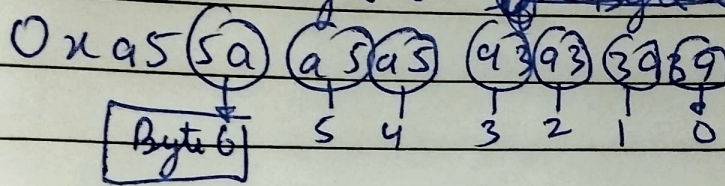
sign bit, so extend 1

for rest 56 bits [1111 → f]

(8) lb x3, 6(x1)

0x0000 0000 0000 005a

- lb → load byte (8bits) then extend sign bit
6(x1) • Byte 6 ~~Byte 0 = 0(x1)~~



5a
① 1 0 1
sign-bit so extend 0 for rest 56-bits.

- Que 4
- .data section in our code maps to address 0×10000000 .
 - It is not fixed address, it is by default in the simulator settings.
 - Yes, it is configurable.

To change it, follow the following steps: (on simulator)

- Click Edit \rightarrow click settings
- Select compiler
- Scroll down.
- In Assembler section, change the address for ".data section start address"
- Click OK.