CS 354 - Lecture 5

Low-level C (1) programming!

int ⟶ 1 byte (8 bits)

max int = ?

1	1	1	1	1	1	1	1
128	64	32	16	8	4	2	1

$$2^8 - 1 = 2^0 + 2^1 + 2^2 + \cdots + 2^7$$

$$= 255$$

---

0 0 0 ⟶ 0

0 0 1 ⟶ 1

⋮

1 1 1 ⟶ 7

Msb	Magnitude		
0	0	0	+0
0	0	1	+1
0	1	0	+2
0	1	1	+3
1	0	0	−0
1	0	1	−1
1	1	0	−2
1	1	1	−3

How to represent negative numbers?

Signed Magnitude

Msb ⟹ Sign bit

0 ⟹ +

1 ⟹ −

Drawback

2 ways to represent

0 → 000 (+0)

0 → 100 (−0)

(3)

000 → +0

001 → +1

010 → +2

011 → +3

100 → -3

101 → -2

110 → -1

111 → -0

Ones' complement

```
  111
- 001   (+1)
-------
  110   (-1)
```

1 → 001 —1's complement→ 110

(+1) (-1)

Two's complement form (4)

000	0
001	1
010	2
011	3
100	−4
101	−3
110	−2
111	−1

$b = 3$ $\quad 2^3 = 1000$

(−3)

$$\begin{array}{r} {}^{1}{}^{1}{}^{2} \\ 1000 \\ -\ 011 \\ \hline 101 \end{array}$$

$$\begin{array}{r} {}^{9} \\ 100 \\ -\ 9 \\ \hline 91 \end{array}$$

1	0	1
-2^2	2^1	2^0

msb

$\Longrightarrow -4+0+1$

$= \boxed{-3}$

$100 \longrightarrow -4$

$110 \longrightarrow -4+2 = \boxed{-2}$

$111 \longrightarrow -4+2+1 = \boxed{-1}$

$-2^2\ 2^1\ 2^0$

int - 1 byte (5) (2's complement)

1	0	1	0	1	0	1	0
-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
-128	64	32	16	8	4	2	1

(or)

$= -128 + (32 + 8 + 2)$

$= -128 + 42 = \boxed{-86}$

1 | 1 1 1 1 1 1 1

-128

$= -128 + (64 + 32 + 16 + 8 + 4 + 2 + 1)$

$= -128 + 127$

$= \boxed{-1}$

n bits

-2^{n-1} $\quad +(2^{n-1} - 1)$

$$[-2^3 \ldots 2^3 - 1] \Rightarrow [-8 \ldots +7]$$

---

int ⟶ 1 byte

+3 ⟶ 0000 0011

−3 ⟶ 1111 1100 (1's complement)

+1

1111 1101

(7)

int (1 byte)

signed

-2^7 to 2^7-1

unsigned

0 to 2^8-1

```
int num;
num = 1;
num = 0xFF
```

```
unsigned int num;
num = 1;
num = 0xFF;
```

Type conversions

char (1 byte) → short (2 bytes)

short → int (4 bytes)

int → long (4 bytes)

long → long long (8 bytes)

---

char: 0X1A ⟹ 0001 1010

↓

short: 0X001A ⟹ 0000 0000 0001 1010

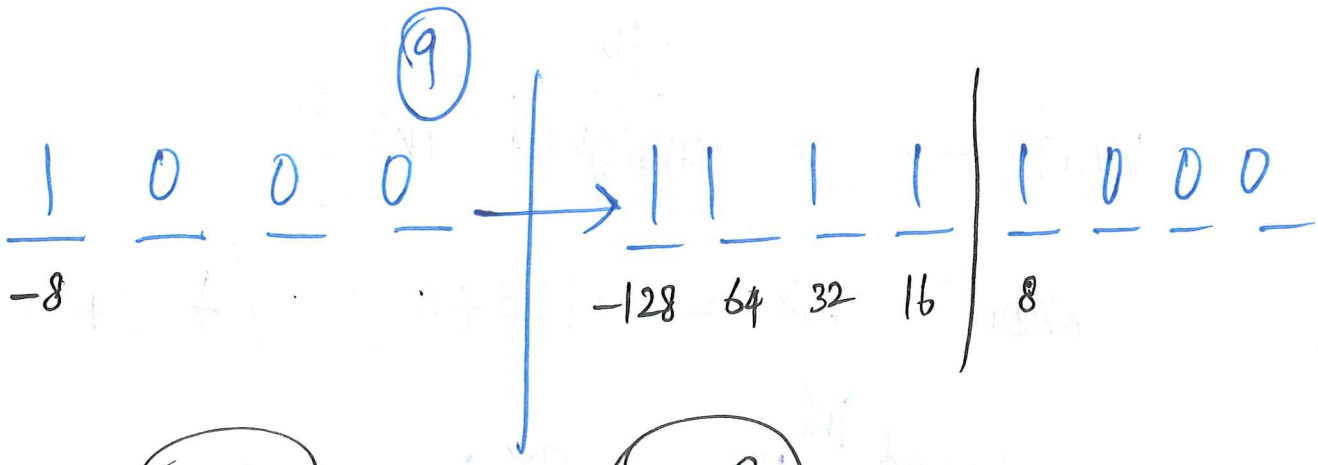
---

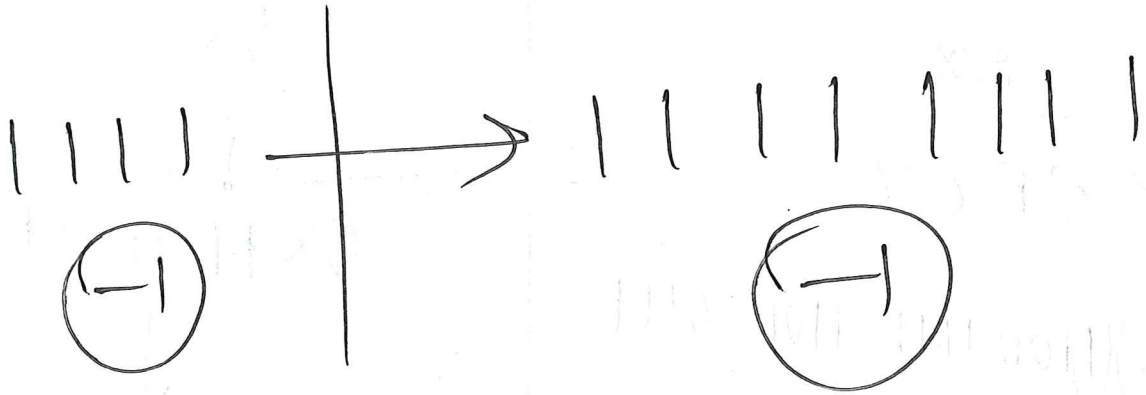
char: 0XF1 ⟹ 1111 0001

↓

short: 1111 1111 1111 0001

sign extension





---

large data type ⟶ small data type.

int ⟶ short

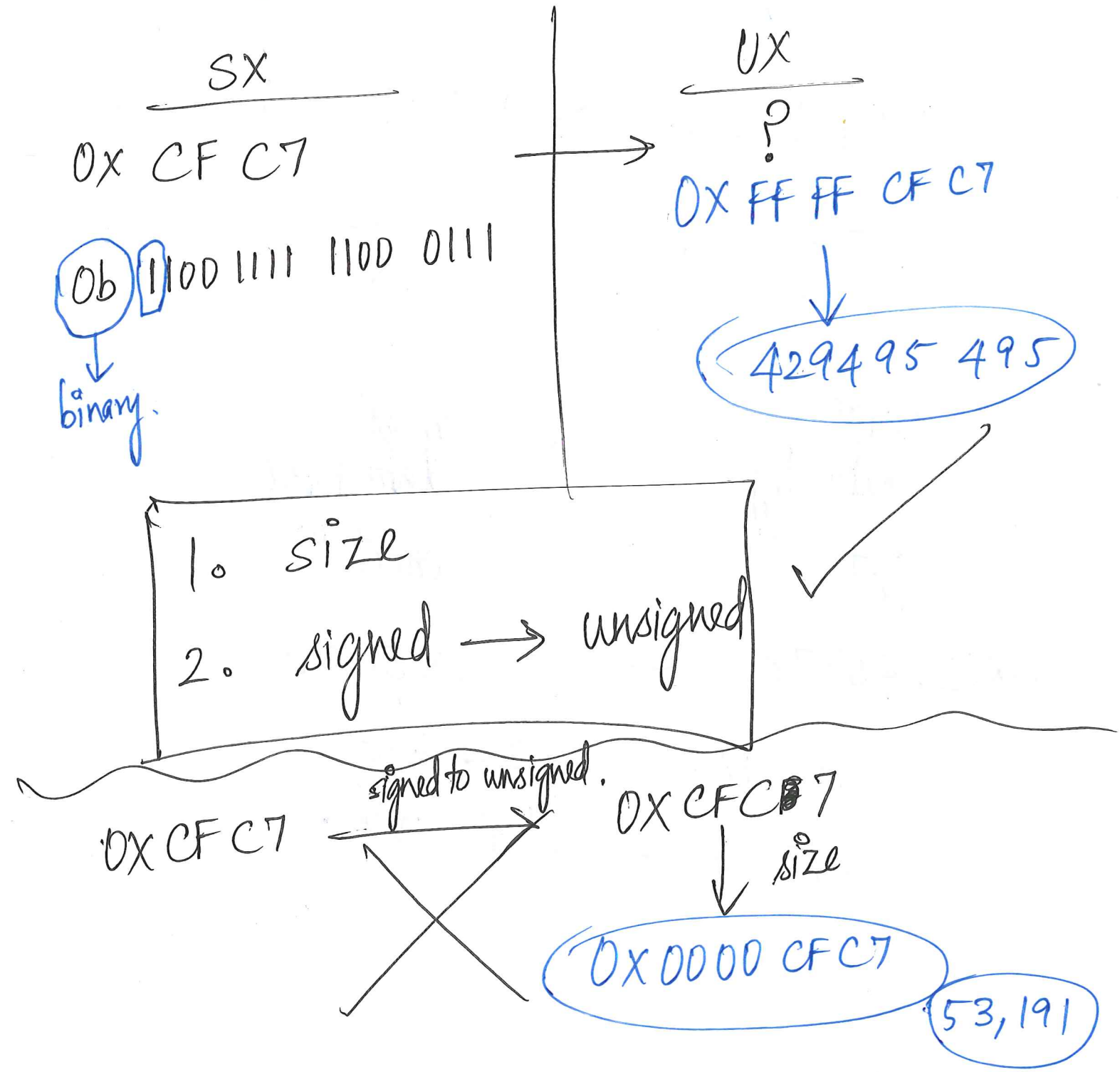
0X12345678 ⟶ 0X5678

```
int num = 0X12345678;
short s = (short) num;
```

(10)

short → unsigned int?

```
short SX = -12345; /* CF C7 */
unsigned int UX = SX;
```



$n_1 =$ 1 1 1 1 (11)

unsigned

↓

$n_2 =$ 0 0 0 0 1 1 1 1

Sign extension only happens for signed ~~numbers~~ types!

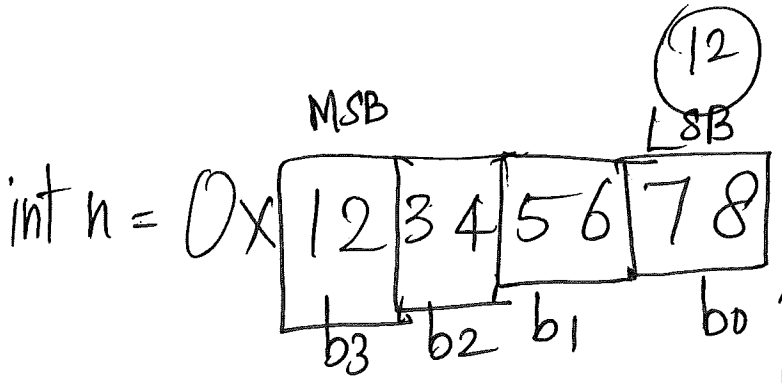
$-1 < 0 \Longrightarrow$ True

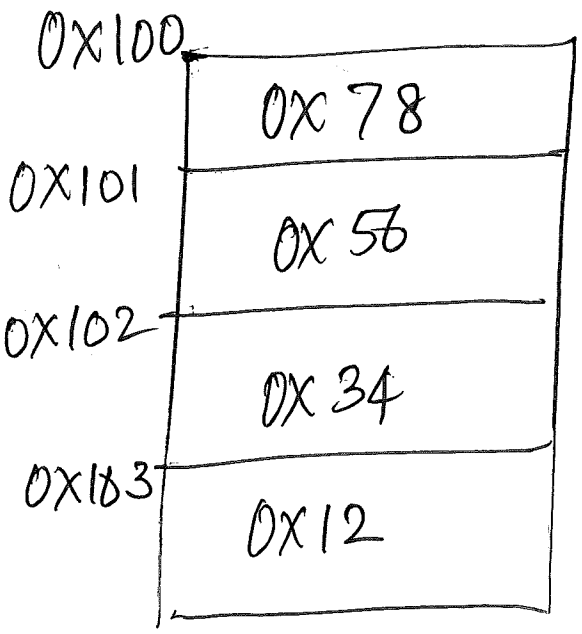
$-1 < 0U \Longrightarrow$ False.

0XFF ↓ 0x00

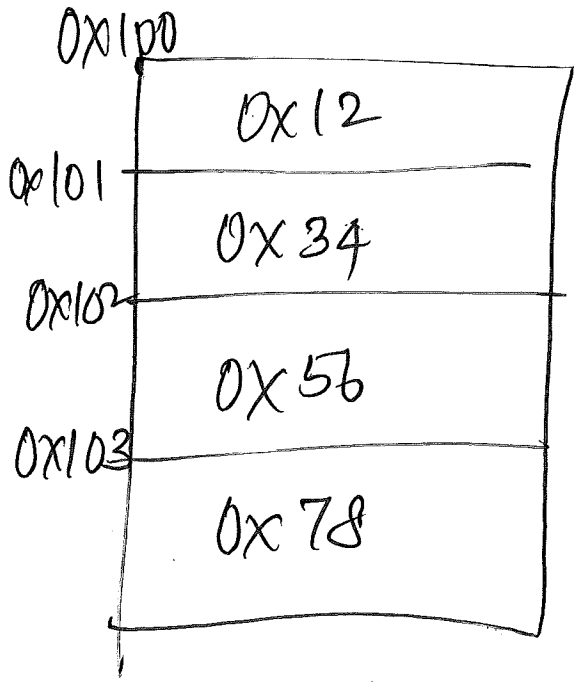
unsigned.

signed ⟶ unsigned.

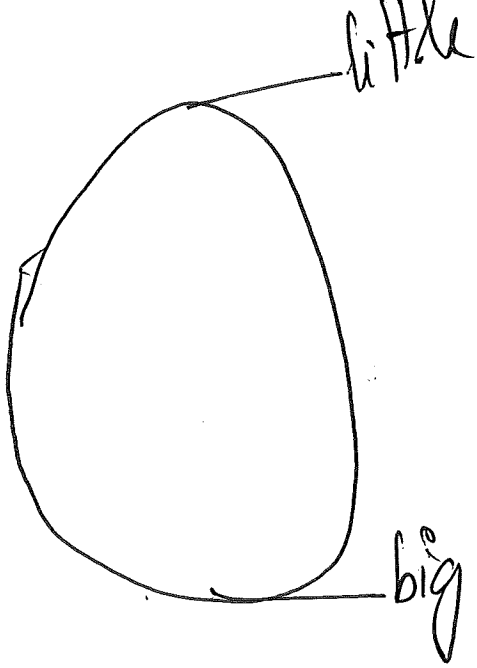




Little-endian

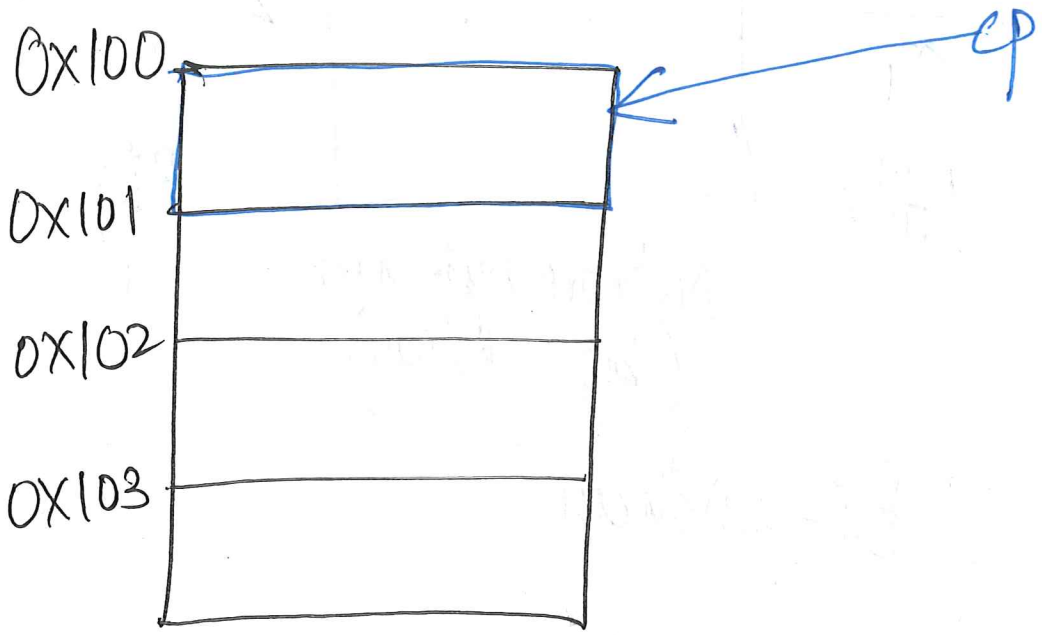


Big-endian



(13)

int n = 0x12345678



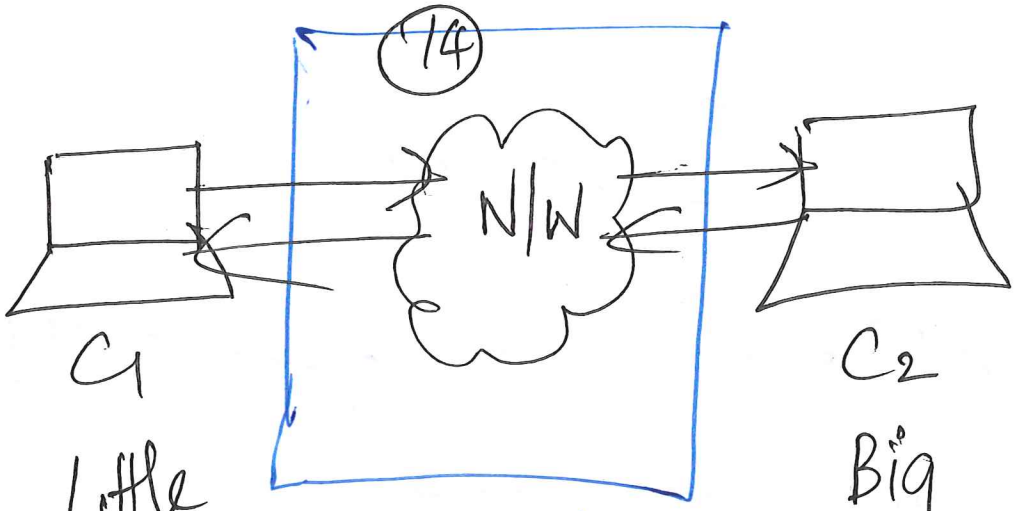
int *pn = &n;

*pn

char *cp = (char*)pn;

*cp → 0x78 ⟹ Little endian

*cp → 0x12 ⟹ big endian.



Network byte order
(big endian)

Bi-endian

Bitwise Operators

1. OR |
2. AND &
3. NOT ~
4. EX-OR ^
5. Left shift <<
6. Right shift >>

$X = 1\ 0\ 1\ 0$ (10)

$Y = 0\ 1\ 1\ 0$ (6)

$X|Y = 1\ 1\ 1\ 0$ (14)

$X\&Y = 0\ 0\ 1\ 0$ (2)

$\sim X = 0\ 1\ 0\ 1$ (5)

$X \wedge Y = 1\ 1\ 0\ 0$ (12)

int - 4 byte

Shifting

X = 1001 0010

X << 4 = 0010 0000

"Left shift"

X >> 4 = 0000 1001

Logical right shift (unsigned)

X >> 4 = 1111 1001

Arithmetic right shift (signed).

(17)

b_7 b_2 b_1 b_0

X = 1010 1001

set to 1

Y = 0000 0110 = 0x06

X | Y = 1010 1111

Z = 0000 1111 (0x0F)

X & Z = 0000 1001 (0x09)

~~K = 1111 0000 (0xF0)~~

X & K = 1010 0000 (0x0A)

add / sub → fast ops

mul / div → slow ops.
(∵ more CPU cycles)

---

14 X 2 = 28

0000 1 1 1 0

0010

$2^3 + 2^2 + 2^1$

n = 0 0 1 0 (2)

n<<1 = 0 1 0 0 (4)

n<<2 = 1 0 0 0 (8)

n = 0110 (6)

n<<1 = 1100 (12)

$n << K \iff n * 2^K$

$14 \times 2 = (19) = (2^3 + 2^2 + 2^1) * 2$

$= 2^{(3)} * (2) + 2^{2} * 2 + 2^{1} * 2$

$= 0\,0\,0\,0\;1\;0\,0\,0\,0$

$0\,0\,0\,0\;1\;0\,0\,0$

$+ \quad 0\,0\,0\,0\;0\;1\,0\,0$

$0\,0\,0\,1\;1\;1\,0\,0 \Longrightarrow 28$

16 8 4 2 1

$00\;1\;0$

$2^{K} * n \Longleftrightarrow n << K$

$14 \times 2 = (2^4 - 2^1) \times 2$

$0\;1\;2$

$0\,0\,1\,0\;0\,0\,0\,0$

$- \quad 0\,0\,0\,0\;0\,1\,0\,0$

$0\,0\,0\,1\;1\,1\,0\,0 \Longrightarrow 28$