**1. Unsigned:**  
1) 0 в двоичной системе: 0→0000000   
2) 13 в двоичной системе: 13→1101.   
Дополняем до 6 бит слева: 001101.  
3) 24 в двоичной системе: 24→11000.  
Дополняем до 6 бит слева: 011000.  
4) 63 в двоичной системе: 63→111111.  
Дополнение не требуется, так как число уже занимает 6 бит.  
**Signed:**  
1. 16 в двоичной системе: 16→10000.  
Дополняем до 6 бит: 010000.  
2. -2 в двоичной системе: 2→000010.  
Инвертируем все биты: 111101.  
Прибавляем 1: 111110.  
3. 31 в двоичной системе: 31→11111.  
Дополняем до 6 бит: 011111.  
4. −32 в 6-битной системе — минимальное значение.  
Результат: 100000.

2. **Unsigned формат:**

Decimal = Binary в десятичной системе

**Two's complement:**

* Если старший бит (MSB) равен 0, то число положительное:   
    
  Decimal = Binary в десятичной системе
* Если MSB = 1, то число отрицательное:   
    
  Decimal = − (2n-1) + остальные биты в десятичной системе системе  
  где n — количество бит.

1) 000101:  
**Unsigned:**  
000101 = 5  
Просто переводим бинарное число в десятичное:  
0 ⋅ 25 + 0 ⋅ 24 + 0 ⋅ 23 + 1 ⋅ 22 + 0 ⋅ 21 + 1 ⋅ 20 = 5  
**Signed:**  
MSB = 0, значит число положительное.  
Переводим как обычное бинарное число:  
000101 = 5.  
2) 101011:  
**Unsigned**:  
101011 = 43  
Переводим бинарное число в десятичное:  
1 ⋅ 25 + 0 ⋅ 24 + 1 ⋅ 23 + 0 ⋅ 22 + 1 ⋅ 21 + 1 ⋅ 20 = 43  
**Signed:**  
MSB = 1 значит число отрицательное.  
Инвертируем биты: 101011 → 010100, добавляем 1: 010100 + 1 = 010101.  
Переводим 010101 = 21, и ставим минус: −21.  
3) 111111:  
**Unsigned**:  
111111 = 63  
Переводим бинарное число в десятичное:  
1 ⋅ 25 + 1 ⋅ 24 + 1 ⋅ 23 + 1 ⋅ 22 + 1 ⋅ 21 + 1 ⋅ 20 = 63  
**Signed**:  
MSB = 1, значит число отрицательное.  
Инвертируем биты: 111111 → 000000, добавляем 1: 000000 + 1 = 000001.  
Переводим 000001 = 1, и ставим минус: −1.  
4) 100000:  
**Unsigned**:  
100000 = 32  
Переводим бинарное число в десятичное:  
1 ⋅ 25 + 0 ⋅ 24 + 0 ⋅ 23 + 0 ⋅ 22 + 0 ⋅ 21 + 0 ⋅ 20 = 32  
**Signed:**  
MSB = 1, значит число отрицательное.  
Инвертируем биты: 100000 → 011111, добавляем 1: 011111 + 1 = 100000.  
Переводим 100000 = 32, и ставим минус: −32.  
  
**3. 1) 7:**

Преобразуем в 8-битное бинарное число:

7 = 000001122.

Переводим в шестнадцатеричную систему:

0000 01112 = 0x07.

**2) 240:**

Преобразуем в 8-битное бинарное число:

240 = 111100002​.

Перевод в шестнадцатеричную систему:  
1111 00002 = 0xF0.

**3) 171:**

Преобразуем в 8-битное бинарное число:

171 = 101010112.

Переводим в шестнадцатеричную систему:

1010 10112 = 0xAB.

**4) 126:**

Преобразуем в 8-битное бинарное число:

126 = 011111102 .

Переводим в шестнадцатеричную систему:

0111 11102 = 0x7E.

**4. 1) 0x3C:**  
Шестнадцатеричное число 0x3C состоит из двух цифр: 3 и C.  
3 в шестнадцатеричной системе = 00112​.  
C в шестнадцатеричной системе = 11002​.  
Объединяем: 3C = 0011 11002.

**2) 0x7E:**  
Шестнадцатеричное число 0x7E состоит из двух цифр: 7 и E.  
7 в шестнадцатеричной системе = 01112​.  
E в шестнадцатеричной системе = 11102​.  
Объединяем: 7E = 0111 11102​.  
  
**3) 0xFF:**  
Шестнадцатеричное число 0xFF состоит из двух одинаковых цифр: F и F.  
F в шестнадцатеричной системе = 11112​.  
Объединяем: FF = 1111 11112.  
  
**4) 0xA5:**  
Шестнадцатеричное число 0xA5 состоит из двух цифр: A и 5.  
A в шестнадцатеричной системе = 10102​.  
5 в шестнадцатеричной системе = 01012​.  
Объединяем: A5=1010 01012​.

**5. 1) 0x3C → 001111002:**Инвертируем все биты: 11000011₂  
Прибавляем 1: 11000011₂ + 1 = 11000100₂  
**2) 0x7E → 01111110₂:**Инвертируем все биты: 10000001₂  
Прибавляем 1: 10000001₂ + 1 = 10000010₂  
**3) 0xFF → 11111111₂:**  
Инвертируем все биты: 00000000₂  
Прибавляем 1: 00000000₂ + 1 = 00000001₂  **4) 0xA5 → 10100101₂:**  
Инвертируем все биты: 01011010₂  
Прибавляем 1: 01011010₂ + 1 = 01011011₂  
  
**6. Big-Endian:**  
Здесь порядок такой, что сначала записывается старший байт (0xDE), потом идут 0xAD, 0xBE, и младший байт 0xEF:  
[DE][AD][BE][EF]  
**Little-Endian:**  
Тут всё наоборот: сначала пишется младший байт (0xEF), потом 0xBE, 0xAD, и в конце 0xDE:  
[EF][BE][AD][DE]  
 **7. 1) 7:**5-битное представление (unsigned и signed одинаковы):  
710​=001112**Zero-extension:**  
001112 → 000001112  
(добавляем три нуля слева)  
**Sign-extension:**  
001112 → 0000011122  
(старший бит 0, значит, добавляем три 0)  
**2) 15:**5-битное представление (unsigned и signed одинаковы):  
1510=011112​**Zero-extension:**  
011112 → 000011112​  
**Sign-extension:**  
011112→000011112​  
**3) -16:**  
5-битное представление (signed, two's complement):  
−1610​=100002​  
**Zero-extension:**  
100002 → 000100002​  
(добавляем три нуля слева, интерпретируется как unsigned)  
**Sign-extension:**  
100002​→111100002​  
(старший бит 1, значит, добавляем три 1)  
**4) -5:**5-битное представление (signed, two's complement):  
−510 = 110112​  
**Zero-extension:**  
110112​ → 000110112​  
**Sign-extension:**  
110112 → 111110112​  
(старший бит 1, значит, добавляем три 1)  
  
**8. Unsigned (7,9):**Преобразование в 4-битные числа:  
710 ​= 01112​  
910 ​= 10012​01112 + 10012​ = 100002  
  
Так как это 5 бит, происходит переполнение, и 4-битный результат равен:  
00002​ (младшие 4 бита, переполнение игнорируем).  
  
**Signed (4,−5):**Преобразование в 4-битные числа (two's complement):  
410 ​= 01002​  
-510 ​= 10112​(для −5: берём двоичное представление 510=01012​, инвертируем 10102​, прибавляем 1: 10112).01002 + 10112​ = 11112  
  
Проверка: 11112​ в two's complement = -110  
11112​ (равно −1)

**12. (\*)**x = x ^ y;   
y = x ^ y;   
x = x ^ y;  
**Объяснение:**  
1. x = x ⊕ y.  
Теперь в x лежит результат XOR старых x и y. Допустим, x = a, y = b. Тогда x = a ⊕ b.  
2. y = x ⊕ y.  
Подставляем новое x: y = (a ⊕ b) ⊕ b.  
По свойству XOR: b ⊕ b = 0, остаётся y = a. y теперь стало старым x.  
3. x = x ⊕ y.  
Подставляем новое y: x = (a ⊕ b) ⊕ a.  
По тому же свойству: a ⊕ a = 0, остаётся x = b. x теперь старое y.  
  
**Пример:**  
Допустим, x = 10 (10102​), y = 12 (11002​).  
- x = 1010 ⊕ 1100 = 0110.  
- y = 0110 ⊕ 1100 = 1010.  
- x = 0110 ⊕ 1010 = 1100.  
Результат: x = 12, y = 10. Всё работает.  
  
**16.(\*)**  
1) x&(x − 1) — убирает младший единичный бит.  
Как работает: вычитание 1 инвертирует все биты справа от первого 1. Побитовый AND обнуляет этот единичный бит.  
  
**Пример:**  
x=01011000.  
  
- x – 1 = 01010111.  
- x&(x − 1) = 01010000.  
  
2) x∥(x+1) — ставит младший нулевой бит в 1.  
Как работает: прибавление 1 инвертирует все биты справа от первого 0. Побитовый OR превращает этот ноль в единицу.  
  
**Пример:**  
x = 10100111.  
  
- x + 1 = 10101000.  
- x∥(x + 1) = 10101111.  
  
3) x∥(x−1) — все биты справа от старшего 1 становятся 1.  
Как работает: вычитание 1 инвертирует биты справа от старшего 1. OR заполняет эти позиции единицами.  
  
**Пример:**  
x = 10101000.  
  
- x – 1 = 10100111.  
- x∥(x − 1) = 10101111.