Bitwise operations

A value can be thought of as a sequence of bits.

```
#include <stdint.h>
...
uint16_t x = 12;   // x = 0b0000000000001100
uint16_t y = 6;    // y = 0b0000000000000110
```

Bitwise logical operations treat each bit as true (1) or false (0), and compute all in parallel.

```
~x    == 0b1111111111110011   // Logical NOT
x | y == 0b0000000000001110   // Logical OR
x & y == 0b0000000000000100   // Logical AND
x ^ y == 0b0000000000001010   // Logical XOR
```

Bitwise shift

`x >> n` shifts all the bits in `x` right `n` positions. `x << n` shifts them left.

```
uint16_t x = 32;   // x = 0b0000000000100000
x = x << 4;        // x = 0b0000001000000000
x = x >> 9;        // x = 0b0000000000000001
x = x >> 1;        // x = 0b0000000000000000
```

Common uses

Set a particular bit to 1.

```
x = x | (1 << i);    // Bit indices traditionally start on right
```

Example:

```
uint16_t x = 128;  // x == 0b0000000010000000
x = x | (1 << 5);  // x == 0b0000000010000000 | 0b0000000000100000
                   // x == 0b0000000010100000
```

Common uses

Clear a particular bit to 0.

```
x = x & ~(1 << i);
```

Example:

```
uint16_t x = 128;  // x == 0b0000000010000000
x = x & ~(1 << 7); // x == 0b0000000010000000 & 0b1111111101111111
                   // x == 0b0000000000000000
```

Common uses

Get a particular bit.

```
x = 1 & (x >> i);
```

Example:

```
uint16_t x = 128; // x == 0b0000000010000000
x = 1 & (x >> 7); // x == 0b0000000000000001 & 0b0000000000000001
                  // x == 1
```

Common uses

To rotate the bits of x left i positions.
Let x be uint32_t.

```
hi = x << i;       // copy x shifted left i positions
lo = x >> (32-i);  // copy x bits that disappeard to low i positions
x = hi | lo;       // reassemble
```

Example:

```
uint32_t x = 0xFF00F0F0; // x  == 0b11111111000000001111000011110000
uint32_t hi = x << 8;    // hi == 0b00000000111100001111000000000000
uint32_t lo = x >>24;    // lo == 0b00000000000000000000000011111111
x = hi | lo;             // x  == 0b00000000111100001111000011111111
```

Common uses

To reverse the order of the bytes in `uint32_t x`.

```
a = (x & 0x000000FF) << 24;  // copy byte 0 and shift left 24 bits
b = (x & 0x0000FF00) <<  8;  // copy byte 1 and shift left 8 bits
c = (x & 0x00FF0000) >>  8;  // copy byte 2 and shift right 8 bits
d = (x & 0xFF000000) >> 24;  // copy byte 3 and shift right 24 bits
x = a | b | c | d;           // reassemble
```

Example:

```
uint32_t x = 0x12345678;
uint32_t a = (x & 0x000000FF) << 24; // a == 0x78000000
uint32_t b = (x & 0x0000FF00) << 8;  // b == 0x00560000
uint32_t c = (x & 0x00FF0000) >> 8;  // c == 0x00003400
uint32_t d = (x & 0xFF000000) >> 24; // d == 0x00000012
x = a | b | c | d;                   // d == 0x78563412
```

Common uses

Good compiler knows what you are doing.

```
uint32_t bswap(uint32_t x) {
    uint32_t a = (x & 0x000000FF) << 24;  // copy byte 0 and shift left 24 bits
    uint32_t b = (x & 0x0000FF00) <<  8;  // copy byte 1 and shift left 8 bits
    uint32_t c = (x & 0x00FF0000) >>  8;  // copy byte 2 and shift right 8 bits
    uint32_t d = (x & 0xFF000000) >> 24;  // copy byte 3 and shift right 24 bits
    return a | b | c | d;                 // reassemble
}
```

```
_bswap:
    movl    %edi, %eax
    bswap   %eax
    ret
```