DigiPen
INSTITUTE OF TECHNOLOGY
SINGAPORE

[CS 225] Advanced C/C++

Lecture 13: Bit manipulation

Sławomir "Swavek" Włodkowski
(Fall 2020)

Agenda

- Bit manipulation
- Bitwise operators review
  - ~ not
  - | or
  - & and
  - ^ xor
  - << shift left
  - >> shift right

Bit manipulation

Motivation

- Not all data is byte-aligned (compression, encryption)
- Not all addresses point to RAM, some may be used by memory-mapped I/O in embedded devices with individual bits for separate devices (single I/O pins).
- Some operations are faster when we shuffle bits.
- Some storage methods are more efficient.
- Makes us look smart…

Bit manipulation

Terminology

Bit – binary digit capable of storing a value `0` or `1`.

- Not directly addressable; this would be inefficient.
- In memory stored always within some byte.

Bit manipulation

Terminology

Byte – addressable unit of data storage large enough to hold any member of the basic character set of the execution environment.

- Represented by `unsigned char` for raw memory.
- Represented by `char` for efficient processing (i.e. text).
- By definition: `sizeof(char) == 1`.
- 8 or more bits (`CHAR_BIT` macro in `<climits>`):
- Windows, POSIX (Linux, Android), consoles mandate 8 bits.

Bit manipulation

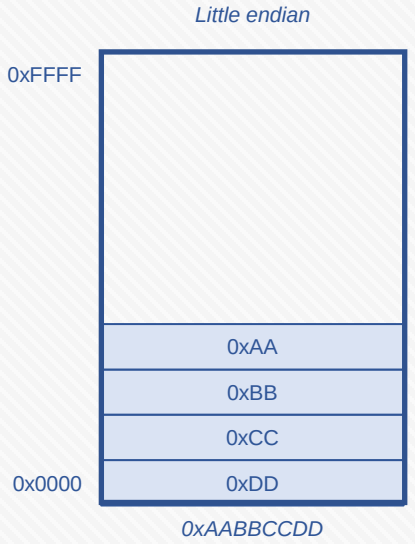
Terminology

Mask – one or more bytes with predefined bits for selecting which bits of input to include or omit in an operation.

Flag – one of bits in a byte where individual bits have distinct business meaning.

Bit manipulation

Terminology



Endianness – sequencing of multibyte data in memory

- ***Little-endian* –** least significant byte at the smallest address (x86, x64, ARM, RISC-V)
- ***Big-endian* –** most significant byte at the smallest address (TCP/IP, ancient Motorola-based Macintosh)

Operators

Truth table for r = ~p:

p	r
0	1
1	0

Bitwise NOT (unary) **r = ~p;**

- One’s complement after promotion.
- Used for unconditional bitwise negation (toggling).

```
#include <iostream>
int main()
{
        unsigned char p = 0b1000'0000;
        std::cout << ~p << std::endl;
        //           0b1...1'0111'1111
        //                        -129
}
```

Operators

Truth table for r = p | q:

p	q	r
0	0	0
0	1	1
1	0	1
1	1	1

Bitwise OR (binary) **r = p | q;**

- Setting a bit to 1 if **any** of input bits is 1; 0 otherwise.
- Used for setting masked or remaining bits to 1.

```
#include <iostream>
int main()
{
        unsigned short p = 0x0105;
        unsigned char  q =      3; // mask
        std::cout << (p | q) << std::endl;
        // 0b1`0000`0101 |
        // 0b0`0000`0011 =
        // 0b1`0000`0111 = 263
}
```

Operators

Truth table for r = p & q:

p	q	r
0	0	0
0	1	0
1	0	0
1	1	1

Bitwise AND (binary) **r = p & q;**

- Setting a bit to 1 if **both** of input bits is 1; 0 otherwise.
- Used for clearing masked or remaining bits to 0.

```
#include <iostream>
int main()
{
        unsigned short p = 0x0105;
        unsigned char  q =      3; // mask
        std::cout << (p & ~q) << std::endl;
        // 0b1`0000`0101 &
        //~0b0`0000`0011 =
        // 0b1`0000`0100 = 260
}
```

Operators

Truth table for r = p ^ q:

p	q	r
0	0	0
0	1	1
1	0	1
1	1	0

Bitwise XOR (binary) **r = p ^ q;**

- Setting a bit to 1 if input bits differ; 0 otherwise.
- Used for toggling masked bits (selective negation).
- Used for comparisons and unary clearing (r = p ^ p).

```
#include <iostream>
int main()
{
        unsigned short p = 0b1111'0000;
        unsigned short q = 0b0001'0001;
        std::cout << (p ^ q) << std::endl;
        // 0b1111`0000 ^
        // 0b0001`0001 =
        // 0b1110`0001 = 225
}
```

Operators

Bitwise << (binary) **r = p << q;**

- Left shift of bits after promotion (MSB is discarded).
- Used for positioning a mask.
- Used for integral multiplication by powers of 2.

```
#include <iostream>
int main()
{
        unsigned short p = 0xF0;
        unsigned short q = 1 << 2; // mask
        std::cout << (p ^ q) << std::endl;
        // 0b1111`0000 ^
        // 0b0000`0100 =
        // 0b1111`0100 = 244
}
```

Operators

Bitwise >> (binary) **r = p >> q;**

- Right shift of bits after promotion (LSB is discarded).
- Used for integral division by powers of 2.
- Used for printing masks and binary values.

```
#include <iostream>
void print_bin(unsigned int p)
{
        if (p) print_bin(p >> 1);   // MSBs
        std::cout << (p & 1);       // LSB
}
int main()
{
        print_bin(0b0011'0101);
}
```