C Programming

Lecture 10: Bit Operations

Lecturer: *Dr*. Wan-Lei Zhao

Autumn Semester 2022

Email: wlzhao@xmu.edu.cn, copyrights are fully reserved by the author.

Outline

1. Bit operations

What are bit operations?

- Data onside computers are kept in binary form, such as 10101111
- One binary code is a data item, it could be an integer, a float number, or a string
- In some scenarios, we need to operate them bit-wisely

- Given a binary code 10101111
- How could we take out its lower 4 bits

The bitwise operators

- There are 6 bit operators
- bit and &
- bit or $|$
- bit xor ˆ
- bit not $\sim$
- left shift $\ll$
- right shift $\gg$

Truth tables for &, | and ~

c1	c2	c1 & c2
1	1	1
1	0	0
0	1	0
0	0	0

c1	c2	c1 | c2
1	1	1
1	0	1
0	1	1
0	0	0

c1	c2	c1 ^ c2
1	1	0
1	0	1
0	1	1
0	0	0

- Notice that it is applied on one bit ONLY
- If there are multiple bits, the operator is applied on each bit
- The result of one bit operation has NO impact on the other bit

AND & and OR |

- Given two variables $a = 60$ and $b = 13$ of unsigned char
- See what are the result for a & b
- See what are the result for a | b

$$\begin{array}{rr} & 00111100 \\ \& & 00001101 \\ \hline & 00001100 \end{array} = 12_{(10)} \qquad \begin{array}{rr} & 00111100 \\ | & 00001101 \\ \hline & 00111101 \end{array} = 61_{(10)}$$

```c
int main(){
  unsigned char a = 60, b = 13;
  unsigned char c = a & b;
  unsigned char d = a | b;
  printf("c = %d, d = %d\n", c, d);
  return 0;
}
```

OR | and xOR ^

- Given two variables a = 60 and b = 13 of unsigned char
- See what are the result for a | b
- See what are the result for a ^ b

$$\begin{array}{rr} & 00111100 \\ | & 00001101 \\ \hline & 00111101 \end{array} == 61_{(10)}$$

$$\begin{array}{rr} & 00111100 \\ \wedge & 00001101 \\ \hline & 00110001 \end{array} == 49_{(10)}$$

```c
int main(){
  unsigned char a = 60, b = 13;
  unsigned char c = a | b;
  unsigned char d = a ^ b;
  printf("c␣=␣%d,␣d␣=␣%d\n", c, d);
  return 0;
}
```

NOT ~ (1)

c1	~c1
1	0
0	1

- Flip a bit
- 1 → 0, 0 → 1
- The result of one bit operation has NO impact on the other bit

NOT ~ (2)

- Given one variable $a = 60$ of unsigned char
- See what are the result for ~a

$$\begin{array}{r} \sim 00111100 \\ \hline 11000011 \end{array} = 195_{(10)}$$

```c
int main(){
  unsigned char a = 60;
  unsigned char c = ~a;
  unsigned char d = !a;
  printf("c = %d, d = %d\n", c, d);
  return 0;
}
```

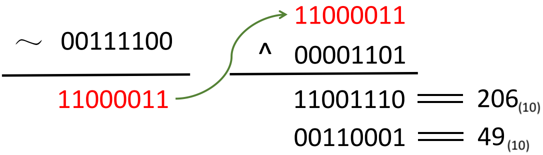
Example-1: implement $\odot$ operation (1)

c1	c2	c1 $\odot$ c2
1	1	1
1	0	0
0	1	0
0	0	1

- In some cases, we need 1 for bits of the same, while 0 for bit of difference
- There is NO such operator in C
- Can we realize it with provided operators?

Think about it in five minutes...

Example-1: implement ⊙ operation (2)



~ 00111100

11000011

11000011

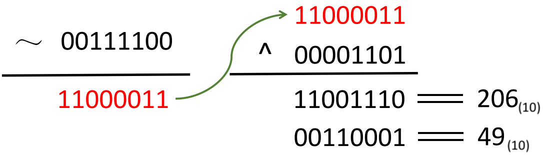
^ 00001101

11001110 == $206_{(10)}$

00110001 == $49_{(10)}$

- We achieve this in two steps
    1. Flip one of the numbers
    2. Apply XOR between the flipped number and another number

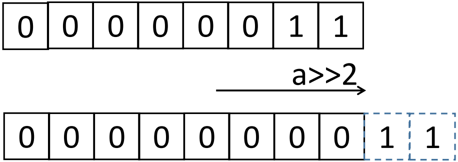
Example-1: implement ⊙ operation (3)



```c
int main(){
  unsigned char a = 60, b = 13;
  unsigned char c = ~a;
  unsigned char d = c ^ b;
  printf("c = %d, d = %d\n", c, d);
  return 0;
}
```

- You will get the same result if you flip b

Left shift $val \ll numb$



0 0 0 0 0 0 1 1

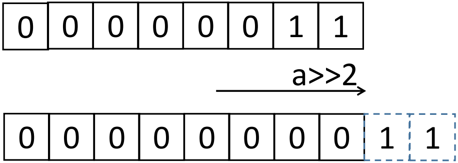
a>>2

0 0 0 0 0 0 0 0 1 1

- Shift the binary code towards the left in numb bits
- For example, a = 3; $a \ll 2$
- The result is 12

```c
int main(){
   unsigned char a = 3, b = 0;
   b = a << 2;
   printf("a = %d, b = %d\n", a, b);
   return 0;
}
```

Right shift $val \gg numb$



0	0	0	0	0	0	1	1

a>>2

0	0	0	0	0	0	0	0	1	1

- Shift the binary code towards the right in numb bits
- For example, a = 3; $a \gg 2$
- The result is 0

```
int main(){
  unsigned char a = 3, b = 10, c = 0;
  b = a >> 2;
  c = a >> 1;
  printf("a␣=␣%d,␣b␣=␣%d,␣c␣=␣%d\n", a, b,
    c);
  return 0;
}
```