Bitwise 연산

소프트웨어학부
박영훈 교수

이 단원의 목표

- Bitwise 연산자

- 이진법 표시에서 특정 자릿수 바꾸기 및 확인하기

개요

- 그 동안 배웠던 연산들은 프로세서나 운영체제와는 관계 없는 High-level 연산이었음.

- 어떤 프로그램들은 Low-level 연산을 요구하는 것이 있음. 그 예는 다음과 같다.

  시스템 프로그래밍 (운영체제나 관련된 것)

  암/복호화 프로그램

  그래픽 프로그램 (영상데이터가 크기 때문에 구조체가 아님)

  최적화를 요하는 프로그래밍

Bitwise 연산자

- C 프로그래밍에서는 여섯 종류의 Bitwise 연산자를 제공한다.

- 이 중 두 가지는 shift 연산자이다.

  << 왼쪽으로
  >> 오른쪽으로 옮겨라

- 세 종류는 Bitwise AND, OR, XOR 연산자이다.

  &
  |
  ^

  &&, || → 논리연산자

- 나머지 하나는 Bitwise NOT 연산자로, Unary operator이다.

  ~

- 이 때, 모든 Bitwise 연산자는 정수 타입 (`char` 포함) 에서만 사용할 수 있다.

Bitwise Shift 연산자 <<

- Bitwise Shift 연산자 중 <<은 정수를 이진법으로 나타내었을 때 자릿수를 왼쪽으로 옮기는 기능을 한다.
- 이 때, 오른쪽은 0으로 채워지고, 왼쪽의 넘어가는 자리수는 무시된다.
- `i << j`는 `i`를 이진법으로 표현하였을 때 자릿수를 `j`만큼 왼쪽으로 옮기고, 오른쪽은 `j`개의 `0`으로 채워지며, 원래 `i`의 왼쪽 `j`개 비트는 없어지게 된다.
- `i << j`는 `i`의 부호가 변하지 않는 한, 2^j배 한 효과가 있다.

cf) 10진법처럼 생각해보면 10^j배가됨

- 예:

```
i = 2020;      // i = 00000000 00000000 00000111 11100100
j = i << 2;    // j = 00000000 00000000 00011111 10010000
```

즉, `j`의 값은 `8080`이 된다.

음수의경우

```
i = -1121;     // i = 11111111 11111111 11111011 10011111
j = i << 3;    // j = 11111111 11111111 11011100 11111000
```

즉, `j`의 값은 `-8968`이 된다.

shift연산에서는 sign bit 무시

```c
#include <stdio.h>
int main(void){
        char *p, *q;
        int i = 2020, j;
        j = i << 2;
        p = (char*)&i;
        q = (char*)&j;
        printf("i = %d, j = %d\n", i, j);
        return 0;
}
~
```

*각각의 비트를 볼 수 있는 방법

* 포인터를 이용하는 방법

i는 4바이트를 차지함

i의 주소를 p가 넣음

p는 char* 이므로 1바이트를 차지함 → p[0], p[1], p[2], p[3] 을 볼 수 있음

∴ i의 공간이 차례로 들어감

```c
#include <stdio.h>
int main(void){
        char *p, *q;
        int i = 2020, j;
        j = i << 2;
        p = (char*)&i;
        q = (char*)&j;
        printf("i = %d, %X, %X, %X, %X\n", i, p[3], p[2], p[1], p[0]);
        printf("j = %d, %X, %X, %X, %X\n", j, q[3], q[2], q[1], q[0]);
        return 0;
}
~
```

음수인 경우 알고 싶으려면 (unsigned char*)로 typecast하면 괜찮음.

```
s1234567@whistle:~/lecture$ ./s
i = 2020, 0, 0, 7, FFFFFFE4
j = 8080, 0, 0, 1F, FFFFFF90
```

Bitwise Shift 연산자 >>

- Bitwise Shift 연산자 중 >>은 정수를 이진법으로 나타내었을 때 자릿수를 오른쪽으로 옮기는 기능을 한다.
- 이 때, 오른쪽으로 넘어가는 자리수는 무시되고, 왼쪽은 sign bit로 채워지게 된다.
- `i >> j`는 `i`를 이진법으로 표현하였을 때 자릿수를 `j`만큼 오른쪽으로 옮기고, 왼쪽은 `j`개의 sign bit로 채워지며, 원래 `i`의 왼쪽 `j`개 비트는 없어지게 된다. 즉, `i`가 양수이면 왼쪽에 `0`이, 음수이면 왼쪽에 `1`이 채워진다.
- `i >> j`는 2^j으로 나눈 효과가 있다.

맨 왼쪽이 0이었으면 모두 0으로 채워짐
맨 왼쪽이 1이었으면 모두 1로 채워짐
→ 부호가 유지될 수 있도록.

- 예:

```
i = 2020;      // i = 00000000 00000000 00000111 11100100
j = i >> 2;    // j = 00000000 00000000 00000001 11111001
```

- 즉, `j`의 값은 505가 된다.

```
i = -1121;     // i = 11111111 11111111 11111011 10011111
j = i >> 3;    // j = 11111111 11111111 11111111 01110011
```

즉, `j`의 값은 `-141`이 된다.

* union을 이용하는방법

```c
#include <stdio.h>
union sh{
        int i;
        unsigned char c[4];
};
int main(void){
        int i = 2020, j;
        union sh p, q;
        j = i >> 2;
        p.i = i;
        q.i = j;
        printf("i = %d, %X, %X, %X, %X\n", p.i, p.c[3], p.c[2], p.c[1], p.c[0]);
        printf("j = %d, %X, %X, %X, %X\n", q.i, q.c[3], q.c[2], q.c[1], q.c[0]);
        return 0;
}
```

ir라CL4]

→ 즉 CC[redacted]

각각의 바이트에 저장들어있는지 확인가능

```
s1234567@whistle:~/lecture$ ./s
i = 2020, 0, 0, 7, E4
j = 505, 0, 0, 1, F9
```

i = -1121, 찍히는경우

```
i = -1121, FF, FF, FB, 9F
j = -141, FF, FF, FF, 73
```

↑ 이용해

Bitwise Shift 연산자 사용시 주의점

- Bitwise shift 연산자는 다른 산술 연산자보다 우선순위가 낮다. 예를 들어:

```
i << 2 + 1;
```

  은 `i << (2 + 1);`, 즉 `i << 3;`과 같은 뜻이다.

- 만일 `i`를 `4`배 하고 `1`을 더하고 싶으면 `(i << 2) + 1;`이라고 해야 한다.

- 정수 연산에서 2의 거듭제곱배를 하거나, 2의 거듭제곱으로 나눈 몫을 구하고 싶다면 `*`나 `/` 연산자 보다 `<<`나 `>>`를 사용하는 것이 연산 속도 측면에서 좋다.

8배하고싶다 : <<3

Bitwise 연산자 &, |, ^, ~

- Bitwise 연산자 중 &, |, ^, ~는 각각 AND, OR, XOR, NOT을 뜻한다. 즉,

  a & b : a와 b의 이진수 표현에서, 각 자리가 둘 다 1이면 1, 적어도 하나가 0이면 0.

  a | b : a와 b의 이진수 표현에서, 각 자리가 둘 다 0이면 0, 적어도 하나가 1이면 1.

  a ^ b : a와 b의 이진수 표현에서, 각 자리가 같으면 0, 다르면 1.

  ~a : a의 이진법 표현에서 0과 1을 서로 바꾼 값이 된다.

- 예:

```
/* 2020 = 00000000 00000000 00000111 11100100
   1121 = 00000000 00000000 00000100 01100001 */

int i = 2020, j = 1121, k;
k = ~i;       // k = 11111111 11111111 11111000 00011011
k = i & j;    // k = 00000000 00000000 00000100 01100000
k = i | j;    // k = 00000000 00000000 00000111 11100101
k = i ^ j;    // k = 00000000 00000000 00000011 10000101
```

Bitwise 복합 연산자

- ~를 제외한 Bitwise 연산자에 =을 붙임으로써 복합 연산자를 만들 수 있다.

  ↗ shift된 값을 I에 넣는?

  `i <<= j;` i가 j비트만큼 왼쪽으로 shift됨. 즉, i의 값이 `i << j`로 바뀜.

  `i >>= j;` i가 j비트만큼 오른쪽으로 shift됨. 즉, i의 값이 `i >> j`로 바뀜.

  `i &= j;` i의 값이 `i & j`로 바뀜

  `i |= j;` i의 값이 `i | j`로 바뀜

  `i ^= j;` i의 값이 `i ^ j`로 바뀜

- 예:

```
int i = 21, j = 56;     // 21 = 10101(2), 56 = 111000(2)
i &= j;    // i becomes 16 (10000(2))
i ^= j;    // i becomes 40 (101000(2))
i |= j;    // i becomes 56 (111000(2))
```

Bitwise 연산자의 활용

- Bitwise 연산자를 활용하여 정수의 이진법 구성을 바꾸거나 확인할 수 있다.

- Bitwise 연산자로 보통 다음과 같은 작업들을 수행한다

  각 자리 정하기

  각 자리에 저장된 값 확인하기

Bitwise 연산자를 이용한 각 자리 정하기

- 어떤 자리를 1로 정할 때는 `|=` 연산자를, 0으로 정할 때는 `&=` 연산자를 사용하면 된다.

- 예를 들어, `i`의 값이 `2020` (= $11111100100_{(2)}$)인 상황에서 오른쪽에서 4번째 비트만 `1`로 바꾸고 싶으면 다음과 같이 하면 된다.

```
i |= 8;      또는      i |= 0x8;
```

그러면, `i`의 값은 `2028` (= $11111101100_{(2)}$)이 된다.

- 또한, `i`의 오른쪽에서 8번째 비트만 `0`으로 바꾸고 싶으면 다음과 같이 하면 된다.

```
i &= 0xFFFFFF7F; 또는 i &= ~0x80;
```

그러면, `i`의 값은 `1900` (= $11101101100_{(2)}$)이 된다.

- Shift연산자를 활용하여 좀더 간편하게 할 수 있다. `i`의 오른쪽에서 `j`번째 비트만을 `1` 또는 `0`으로 바꾸려면 다음과 같이 하면 된다.

1로 바꿀 때: `i |= (1 << (j-1));`

0으로 바꿀 때: `i &= ~(1 << (j-1));`

Bitwise 연산자를 이용한 각 자리 정하기

- Bitwise 연산자를 이용하여 연속된 여러 자리를 한꺼번에 정할 수 있다.

- i의 오른쪽에서 연속 j개 bit만을 1로 정하기:

```
i |= ((1 << j) - 1);   또는   i |= ~(-1 << j);
```

- i의 오른쪽에서 연속 j개 bit만을 0으로 정하기:

```
i &= ~((1 << j) - 1);   또는   i &= (-1 << j);
```

- i의 왼쪽에서 연속 j개 bit만을 1로 정하기:

```
i |= (-1 << (32 - j));
```

(단, char형이면 32 대신 8을 쓴다)

- i의 왼쪽에서 연속 j개 bit만을 0으로 정하기:

```
i &= ~(-1 << (32 - j));
```

(단, char형이면 32 대신 8을 쓴다)

```c
#include <stdio.h>
int main(void){
        int i = 0xABCDEF98;
        // 10101011 11001101 11101111 10011000
        // 00000000 00000000 00000000 00111111
        i |= 63;
        printf("%X\n", i);
        return 0;
}
~
```

→ 1011 1111 이되므로
B F

```
s1234567@whis
ABCDEFBF
s1234567@whis
```

이렇게 해버린 와우서 쓰기 어려움

①

```c
#include <stdio.h>
int main(void){
        int i = 0xABCDEF98;
        // 10101011 11001101 11101111 10011000
        // 00000000 00000000 00000000 00111111
        // 00000000 00000000 00000000 01000000 = 2^6 - 1 = (1 << 6) - 1
        i |= ((1<<6) - 1);
        printf("%X\n", i);
        return 0;
}
~
```

6개 1을 저하면 이렇게됨
(오른쪽에서 6번째1, 즉 2^6)

$\therefore 2^6 - 1$

$= (1<<6) - 1$

②

```c
#include <stdio.h>
int main(void){
        int i = 0xABCDEF98;
        // 10101011 11001101 11101111 10011000
        // 00000000 00000000 00000000 00111111
        // 00000000 00000000 00000000 01000000 = 2^6 - 1 = (1 << 6) - 1
        // 11111111 11111111 11111111 11000000 = (-1 << 6)
        // 11111111 11111111 11111111 11111111 = -1
//      i |= ((1<<6) - 1);
        i |= ~(-1 << 6);
        printf("%X\n", i);
        return 0;
}
~
```

반전

활용

Bitwise 연산자를 이용한 각 자릿수 확인하기

- Bitwise 연산자를 이용하여 각 자리가 0 또는 1인지 확인할 수 있다.

해당 비트만 1인 수랑 &시킴

- `i`의 오른쪽에서 `j`번째 비트를 확인하려면 다음과 같이 하면 된다.

```
i & (1 << (j-1))
```

만일 `i`의 오른쪽에서 `j`번째 비트가 1이면 위의 식이 0이 아니고, 0이면 위의 식이 0이 된다.

- `i`의 오른쪽 또는 왼쪽에서 연속된 `j`개의 비트를 확인하려면 다음과 같이 하면 된다.

오른쪽 `j`개 비트 확인: `i & ~(-1 << j)` 또는 `i & ((1 << j) - 1)`

왼쪽 `j`개 비트 확인: `i & (-1 << (32 - j))` (단, `char`형이면 `32`대신 `8`)

특정자리수만 바꾸는방법 (활용).

```c
#include <stdio.h>
int main(void){
        int i = 0xABCDEF98, j;
        // 10101011 11001101 11101111 10011000
        // 00000000 00011111 11110000 00000000
        // 00000000 00000000 00000001 11111111
        j = i | (((1 << 9) - 1) << 12);
        printf("i = %X, j = %X\n", i, j);
        return 0;
}
~
```

shift

자리복사

```c
#include <stdio.h>
int main(void){
        int i = 0xABCDEF98, j;
        // 10101011 11001101 11101111 10011000
        // 00001111 10000000 00000000 00000000
        // 00001011 10000000 00000000 00000000
        // 00000000 00000000 00000000 00101110
        // 10101011 11001101 11101111 10000000
        // 10101011 11001101 11101111 10101110

        j = i & (~(-1 << 5) << 23);
        j = (j >> 22) & ~(-1 << 6);
        j = i & ((-1 << 6) | 1) | j;
        printf("i = %X, j = %X\n", i, j);
        return 0;
}
~
```

1 2 3 4 5 6

① 원래

② 추출

③ 추출결과

④ 어떻게붙여야함

⑤ 중간과나머지부분 0으로만들기 → or

12 → and

⑥ 최종결과

Bitwise 연산자를 이용한 각 자릿수 확인하기

- Bitwise 연산자를 이용하여 2의 거듭제곱으로 나눈 나머지를 판별할 수 있다.

- 기우성 판별:

```
if(i & 1){
     // case for odd number
}
else{
     // case for even number
}
```

- 4로 나눈 나머지로 switch-case문 작성하기:

```
switch(i & 3){
case 0:
     // case for i = 4k
case 1:
     // case for i = 4k + 1
case 2:
     // case for i = 4k + 2
case 3:
     // case for i = 4k + 3
}
```

기타 유용한 기능

- 정수형 변수 **a**와 **b**의 값을 서로 바꾸고 싶을 때:

```
int tmp = a;
a = b;
b = tmp;
```

와 같이 해도 되지만, XOR의 성질을 이용하여 다음과 같이 해도 된다.

```
a ^= b;
b ^= a;
a ^= b;
```

위와 같이 XOR을 이용하여 프로그래밍 하면 추가 메모리 공간 없이 두 변숫값을 바꿀 수 있다.

```c
#include <stdio.h>
int main(void){
        int a = 10, b = 23;
        a ^= b; // a = 10 ^ 23 , b = 23
        b ^= a; // a = 10 ^ 23 , b = 23 ^ (10 ^ 23) = 23 ^ (23 ^ 10) = (23 ^ 23)
 ^ 10 = 0 ^ 10 = 10
        a ^= b; // a = (10 ^ 23) ^ 10 = (23 ^ 10) ^ 10 = 23 ^ (10 ^ 10) = 23 ^ 0
 = 23
        printf("a = %d, b = %d\n", a, b);
        return 0;
}
~
```

결과.