Chapter 5: Bit Manipulation

Symbol	Operator
&	bitwise AND
|	bitwise inclusive OR
^	bitwise exclusive OR (XOR)
<<	left shift
>>	right shift
~	bitwise NOT (one's complement)

- **Bitwise AND "&"**

a	b	a & b
0	0	0
0	1	0
1	0	0
1	1	1

- **Bitwise OR "|"**

a	b	a | b
0	0	0
0	1	1
1	0	1
1	1	1

- Bitwise XOR "^"

a	b	a ^ b
0	0	0
0	1	1
1	0	1
1	1	0

- Bitwise NOT "~"

a	~ a
0	1
1	0

- Right Shift

  - If the variable 'ch' contains the bit pattern 11100101, then ch >> 1 will produce 01110010, and ch >> 2 will produce 00111001.

- Left Shift

  - If the variable 'ch' contains the bit pattern 11100101, then ch << 1 will produce 11001010.

- For both left and right shifts blank spaces generated are filled up by zeros.

Bit Manipulation By Hand

- Symbol '^' indicates an XOR
- Symbol ~ indicates a NOT (negation)
- Examples

#1
```
  0110 (6)
+ 0010 (2)
  1000 (8)
```

#2
```
  0011 (3)
x 0101 (5)
  0011
 0000
0011
0000
  1111 (15)
```

#3
```
  0110 (6)
+ 0110 (6)
  1100 (12)
```

#4
```
  0011 (3)
+ 0010 (2)
  0101 (5)
```

#5
```
  0011 (3)
x 0011 (3)
  0011
 0011
 0000
  1001
```

#6
```
  0100 (4)
x 0011 (3)
  0100
 0100
 0000
  1100 (12)
```

#7
```
    carrying
     1010
  0110 (6)
- 0011 (3)
  0011 (3)
```

2's Complement
```
  0011
  1100
+    1
  1101
```

```
  0110
  1101
  0011
```

#8)
```
1101 >> 2
= 0011
```

#9)
```
 1101          1101
^(~1101)  →  ^ 0010
              ----
              1111
```

#10)
```
 1000  (8)        0110        2's compliment
-0110  (6)  →     1001
                    +1
                  ----
                  1010
```

```
 1000
 1010
 ----
 0010 (2) ✓
```

#11)
```
 1101
 0101
 ----
 1000
```

#12)
```
 1011          1011
&(~0<<2)  →    1100
               ----
               1000
```

Bit Facts and Tricks

- Note: We use "1s" and "0s" to indicate a sequence of 1s or 0s, respectively.

- XOR
  - $x \wedge 0s = x$
  - $x \wedge 1s = \sim x$
  - $x \wedge x = 0$

- AND
  - $x \,\&\, 0s = 0$
  - $x \,\&\, 1s = x$
  - $x \,\&\, x = x$

- OR
  - $x \mid 0s = x$
  - $x \mid 1s = 1s$
  - $x \mid x = x$

- These operations occur bit-by-bit with what's happening on one bit never impacting the other bits.

Two's Complement and Negative Numbers

- Invert the bits in the positive representation and then add 1.
  - Example: 3 is 011 in binary. Flip the bits to get 100, add 1 to get 101, then prepend the sign bit (1) to get 1101.

Arithmetic vs. Logical Right Shift

- There are two types of right shift operators
  - The arithmetic right shift divides by two.
  - The logical right shift does what we would visually see as shifting the bits.

- Logical right shift indicated with a ">>>"
  - Shift the bits and put a 0 in the most significant bit.

- Arithmetic right shift indicated by a ">>"
  - Shift values to the right but fill in the new bits with the value of the sign bit.
  - Has the effect of (roughly) dividing by 2.

Common Bit Tasks

- Get Bit

```
boolean getBit(int num, int i){
    return ((num & (1 << i)) != 0);
}
```

- Set Bit

```
int setBit(int num, int i){
    return num | (1 << i);
```

Clear Bit

```
int clearBit (int num, int i) {
    int mask = ~(1 << i);
    return num & mask;
}
```

```
int clearBitsMSBthroughI(int num, int i) {
    int mask = (1<<i) - 1;
    return mask & num;
}
```

```
int clearBitsIthrough0(int num, int i) {
    int mask = (-1<<(i+1));
    return num & mask;
}
```

- Update Bit

```
int updateBit(int num, int i, boolean bitIs1) {
    int value = bitIs1 ? 1 : 0;
    int mask = ~(1 << i);
    return (num & mask) | (value << i);
}
```