Bitsel (bitwise) Operatörler

Bilgisayarlarda, tüm veriler bit dizileri olarak temsil edilir. Her bit, 0 veya 1 değerini alabilir. Çoğu sistemde, sekiz bitlik bir dizi (1 byte), char türü bir değişken için depolama birimini oluşturur. Diğer veri türleri daha büyük byte sayılarında depolanır. Bitsel operatörler, hem işaretli hem de işaretsiz işlenenlerin (operand'ların) bitlerini işlemek için kullanılır. İşaretsiz tamsayılar aşağıdaki tabloda özetlenen bitsel operatörlerle kullanılır.

İşaretçisi	Operatör	Tanımı
&	**bitwise AND**	İki işlenenini de bit bit karşılaştırır. İki işlenendeki karşılık gelen bitlerin her ikisi de 1 ise sonuçtaki bitler 1'e ayarlanır.
|	**bitwise inclusive OR**	İki işlenenini bit bit karşılaştırır. İki işlenendeki karşılık gelen bitlerden en az biri 1 ise sonuçtaki bitler 1'e ayarlanır.
^	**bitwise exclusive OR** (also known as **bitwise XOR**)	İki işlenenini (operand'ı) bit bit karşılaştırır. İki işlenendeki karşılık gelen bitler farklıysa sonuçtaki bitler 1'e ayarlanır. AB' +A'B
<<	**left shift**	Birinci işlenenin bitlerini, ikinci işlenen tarafından belirtilen bit sayısı kadar sola kaydırır; sağdan itibaren bitleri 0 ile doldurur.
>>	**right shift**	Birinci işlenenin bitlerini, ikinci işlenen tarafından belirtilen bit sayısı kadar sağa kaydırır;
~	**complement**	0 olan bit'lerin tümü 1'e set edilir ve 1 olan bit'lerin tümü 0 yapılır.

Şekil 10.6 | Bitsel operatörler.

Bit düzeyinde AND, bit düzeyinde XOR ve bit düzeyinde OR operatörleri, iki işlenenini bit bit karşılaştırır. Bit düzeyinde AND operatörü, her iki işlenende karşılık gelen bit 1 ise sonuçtaki her biti 1 olarak ayarlar. Bit düzeyinde OR operatörü, işlenenlerden birinde (veya her ikisinde) karşılık gelen bit 1 ise sonuçtaki her biti 1 yapar. Bit bazında XOR operatörü, her işlenendeki karşılık gelen bitler farklıysa sonuçtaki her bir biti 1 yapar. İşlenendeki bitler aynı ise sonuç bit 0 olur.

Sola kaydırma operatörü, sol işleneninin (operand'ın) bitlerini, sağ işlenenin bit sayısı kadar sola kaydırır. Sağa kaydırma operatörü, sol işlenenindeki bitleri, sağ işleneninde belirtilen bit sayısı kadar sağa kaydırır. Bit düzeyinde tümleyen operatörü, işlenenindeki 0 olan bit'lerin tümünü 1'e ayarlar ve 1 olan bit'lerin tümünü 0'a ayarlar - bu genellikle bitleri değiştirme olarak adlandırılır. Her bir bitsel operatörün ayrıntılı tartışmaları aşağıdaki örneklerde yer almaktadır.

Bu bölümdeki örneklerde bitsel operatör konusu, tamsayı işlenenlerin ikili (binary) temsillerini gösterir. Bitsel operatörleri kullanırken, bu operatörlerin kesin etkilerini göstermek için değerleri ikili (binary) olarak görüntülemek yararlıdır.

Şekil 10.7'deki program, okunabilirlik için her biri sekiz bitlik gruplar (byte) halinde işaretsiz bir int yazdırır. Bu bölümdeki örnekler için, işaretsiz girişlerin 4 bayt (32 bit) bellekte depolandığı varsayılmaktadır.

```c
// Fig. 10.7: fig10_07.c
// Displaying an unsigned int in bits
#include <stdio.h>

void displayBits(unsigned int value); // prototype

int main(void)
{
   unsigned int x; // variable to hold user input

   printf("%s", "Enter a nonnegative int: ");
   scanf("%u", &x);

   displayBits(x);
}

// display bits of an unsigned int value
void displayBits(unsigned int value)
{
   // define displayMask and left shift 31 bits
   unsigned int displayMask = 1 << 31;

   printf("%10u = ", value);

   // loop through bits
   for (unsigned int c = 1; c <= 32; ++c) {
      putchar(value & displayMask ? '1' : '0');
      value <<= 1; // shift value left by 1

      if (c % 8 == 0) { // output space after 8 bits
         putchar(' ');
      }
   }

   putchar('\n');
}
```

```
Enter a nonnegative int: 65000
     65000 = 00000000 00000000 11111101 11101000
```

Bu programda displayBits fonksiyonu (satır 18–36), değişken değerini değişken displayMask (satır 27) ile birleştirmek için bit düzeyinde AND operatörünü kullanır. Genellikle, bit düzeyinde AND operatörü, maske adı verilen bir işlenenle (operand'la) kullanılır; (maske: belirli bitleri 1'e ayarlanmış bir tamsayı değeri). Maskeler, diğer bitleri seçerken bir değerdeki bazı bitleri gizlemek için kullanılır. displayBits fonksiyonunda, displayMask maske değişkenine şu değer atanır:

1 << 31 (10000000 00000000 00000000 00000000)

Sola kaydırma operatörü, displayMask'ta 1 değerini En Az anlamlı Bit'ten (Least Significant Bit-LSB) (en sağdaki) En Anlamlı bit'e (Most Significant Bit-MSB) (en soldaki) doğru kaydırır ve sağdan itibaren bit'lere 0 doldurur.

Satır 27, değişken değerinin mevcut en soldaki biti için 1 mi yoksa 0 mı yazdırılacağını belirler. value değişkeni ve displayMask, & kullanılarak AND işlemine tabi tutulduğunda, değişken değerindeki yüksek dereceli bit (MSB) dışındaki tüm bitler "maskelenir" (gizlenir), çünkü 0 ile "AND'lenen" herhangi bir bit 0 verir. Eğer, en soldaki bit 1 ise, value & displayMask 1 yazdırılır. Değişken değeri daha sonra value <<= 1 ifadesi ile bir bit sola kaydırılır (bu, value = value << 1'e eşdeğerdir). İşaretsiz değişken değerindeki her bit için bu adımlar tekrarlanır. Şekil 10.8, iki biti bitsel AND operatörüyle işleme tabi tutmanın sonuçlarını özetlemektedir.

Bit 1	Bit 2	Bit 1 & Bit 2
0	0	0
0	1	0
1	0	0
1	1	1

Şekil 10.8 | İki biti bitsel AND operatörü & ile birleştirmenin sonuçları.

Şekil 10.7'nin 21. satırında, 1 değerinin displayMask değişkeninde en soldaki bite kaydırılması gerektiğini belirtmek için 31 (Binary: 0001 1111) tamsayısını kodladık. Benzer şekilde, 26. satırda, döngünün 32 kez yinelenmesi gerektiğini belirtmek için 32 tamsayısını kodladık - değişken değerindeki her bit için bir kez bu işlem tekrarlanmış olur. İşaretsiz girişlerin her zaman 32 bit (dört bayt) bellekte saklandığını varsaydık. Günümüzün bilgisayarları genellikle 32 bit veya 64 bit kelimelik donanım mimarileri kullanır.

Şekil 10.9, bit düzeyinde AND operatörünün, XOR operatörünün, OR operatörünün ve değil operatörünün kullanımını göstermektedir. Program, işaretsiz int değerlerini yazdırmak için displayBits (satır 46–64) fonksiyonunu kullanır. Program çıktısı, Şekil 10.10'da gösterilmiştir.

```c
// Fig. 10.9: fig10_09.c
// Using the bitwise AND, bitwise inclusive OR, bitwise
// exclusive OR and bitwise complement operators
#include <stdio.h>

void displayBits(unsigned int value); // prototype

int main(void)
{
   // demonstrate bitwise AND (&)
   unsigned int number1 = 65535;
   unsigned int mask = 1;
   puts("The result of combining the following");
   displayBits(number1);
   displayBits(mask);
   puts("using the bitwise AND operator & is");
   displayBits(number1 & mask);

   // demonstrate bitwise inclusive OR (|)
   number1 = 15;
   unsigned int setBits = 241;
   puts("\nThe result of combining the following");
   displayBits(number1);
   displayBits(setBits);
   puts("using the bitwise inclusive OR operator | is");
   displayBits(number1 | setBits);

   // demonstrate bitwise exclusive OR (^)
   number1 = 139;
   unsigned int number2 = 199;
   puts("\nThe result of combining the following");
   displayBits(number1);
   displayBits(number2);
   puts("using the bitwise exclusive OR operator ^ is");
   displayBits(number1 ^ number2);

   // demonstrate bitwise complement (~)
   number1 = 21845;
   puts("\nThe one's complement of");
   displayBits(number1);
   puts("is");
   displayBits(~number1);
}

// display bits of an unsigned int value
void displayBits(unsigned int value)
{
   // declare displayMask and left shift 31 bits
   unsigned int displayMask = 1 << 31;

   printf("%10u = ", value);

   // loop through bits
   for (unsigned int c = 1; c <= 32; ++c) {
      putchar(value & displayMask ? '1' : '0');
      value <<= 1; // shift value left by 1

      if (c % 8 == 0) { // output a space after 8 bits
         putchar(' ');
      }
   }

   putchar('\n');
}
```

```
The result of combining the following
     65535 = 00000000 00000000 11111111 11111111
         1 = 00000000 00000000 00000000 00000001
using the bitwise AND operator & is
         1 = 00000000 00000000 00000000 00000001

The result of combining the following
        15 = 00000000 00000000 00000000 00001111
       241 = 00000000 00000000 00000000 11110001
using the bitwise inclusive OR operator | is
       255 = 00000000 00000000 00000000 11111111

The result of combining the following
       139 = 00000000 00000000 00000000 10001011
       199 = 00000000 00000000 00000000 11000111
using the bitwise exclusive OR operator ^ is
        76 = 00000000 00000000 00000000 01001100

The one's complement of
     21845 = 00000000 00000000 01010101 01010101
is
4294945450 = 11111111 11111111 10101010 10101010
```

Şekil 10.10 | Şekil 10.9'daki programın çıktısı.

Bit düzeyinde AND Operatörü (&)

Şekil 10.9'da, 11. satırda tamsayı değişkeni olan number1 değişkenine

65535 (00000000 00000000 11111111 11111111)

değeri atanmıştır ve mask değişkenine 12. satırda

1 (0000000000000000 00000000 00000001)

değeri atanmıştır. number1 ve mask değişkenleri bitsel AND operatörü (&) kullanılarak number1 & mask ifadesinde (satır 17) birleştirildiğinde,

sonuç 00000000 00000000 00000000 00000001'dir. Değişken number1'daki düşük sıralı bit (LSB) dışındaki tüm bitler, değişken maskeli “AND işlemi” ile “maskelenir” (gizlenir).

Bitsel OR Operatörü (|)

Bit düzeyinde OR operatörü, bir işlenende (operand’da) belirli bitleri 1'e ayarlamak (set etmek) için kullanılır. Şekil 10.9'da, 20. satırda number1 değişkenine 15 (00000000 00000000 00000000 00001111) ve 21. satırda setBits değişkenine 241 (00000000 00000000 00000000 11110001) atanmıştır. number1 ve setBits değişkenleri, bitsel OR operatörü kullanılarak number1 | setBits (satır 26) ifadesi ile birleştirildiğinde , sonuç 255'tir (00000000 00000000 00000000 11111111). Şekil 10.11, iki biti bitsel OR operatörüyle birleştirmenin sonuçlarını özetlemektedir.

number1 15 (00000000 00000000 00000000 00001111)

setBits 241 (00000000 00000000 00000000 11110001)

sonuç 255 (00000000 00000000 00000000 11111111)

Bit 1	Bit 2	Bit 1 | Bit 2
0	0	0
0	1	1
1	0	1
1	1	1

Şekil 10.11 | İki biti bitsel OR operatörü | ile birleştirmenin sonuçları.

Bitsel Özel VEYA (XOR) Operatörü (^)

Bit düzeyinde XOR operatörü (^), iki işlenendeki karşılık gelen bitlerden her ikiside aynı ise souç 0 farklı ise sonuç 1 olur. Şekil 10.9'da, number1 ve number2 değişkenlerine sırasıyla 139 (00000000 00000000 00000000 10001011) ve 199 (00000000 00000000 00000000 11000111) değerleri atanmıştır (29–30). Bu değişkenler, number1 ^ number2 (satır 35) ifadesindeki bit düzeyinde XOR operatörü ile birleştirildiğinde, sonuç 00000000 00000000 00000000 01001100'dür. Şekil 10.12, iki biti bit bitsel XOR operatörü ile birleştirmenin sonuçlarını özetlemektedir.

ve 2 değişkenlerine sırasıyla ve değerleri atanmıştır (29–30). Bu değişkenler, number1 ^ number2 (satır 35) ifadesindeki bit düzeyinde XOR operatörü ile birleştirildiğinde, sonuç 'dür

number1: 139 (00000000 00000000 00000000 10001011)

number2: 199 (00000000 00000000 00000000 11000111)

Sonuç: 74 (00000000 00000000 00000000 01001100)

Bit 1	Bit 2	Bit 1 ^ Bit 2
0	0	0
0	1	1
1	0	1
1	1	0

Şekil 10.12 | İki biti bit bazında özel VEYA (XOR) operatörü ^ ile birleştirmenin sonuçları.

Bitsel Tümleyen (Complement) operatörü (~)

Bitsel tümleyen operatörü (~), işlenenindeki bütün 1'leri 0'a 0'ları da 1'e ayarlar (set eder); Şekil 10.9'da, 38. Satırda number1 değişkenine 21845 (00000000 00000000 01010101 01010101) değeri atanmıştır. ~number1 (satır 42) ifadesi işletildiğinde, sonuç 11111111 11111111 10101010 10101010'dur.

number1: 21845 (00000000 00000000 01010101 01010101)

~number1: 4294945450 (11111111 11111111 10101010 10101010)

Şekil 10.13'teki program, sola kaydırma operatörünü (<<) ve sağa kaydırma operatörünü (>>) gösterir. displayBits işlevi, işaretsiz int değerlerini yazdırmak için kullanılmıştır.

```c
// Fig. 10.13: fig10_13.c
// Using the bitwise shift operators
#include <stdio.h>

void displayBits(unsigned int value); // prototype

int main(void)
{
   unsigned int number1 = 960; // initialize number1

   // demonstrate bitwise left shift
   puts("\nThe result of left shifting");
   displayBits(number1);
   puts("8 bit positions using the left shift operator << is");
   displayBits(number1 << 8);

   // demonstrate bitwise right shift
   puts("\nThe result of right shifting");
   displayBits(number1);
   puts("8 bit positions using the right shift operator >> is");
   displayBits(number1 >> 8);
}

// display bits of an unsigned int value
void displayBits(unsigned int value)
{
   // declare displayMask and left shift 31 bits
   unsigned int displayMask = 1 << 31;

   printf("%7u = ", value);

   // loop through bits
   for (unsigned int c = 1; c <= 32; ++c) {
      putchar(value & displayMask ? '1' : '0');
      value <<= 1; // shift value left by 1

      if (c % 8 == 0) { // output a space after 8 bits
         putchar(' ');
      }
   }

   putchar('\n');
}
```

Sola Kaydırma Operatörü (<<)

Sola kaydırma operatörü (<<), soldaki işleneninin (operand'ın) bitlerini sağdaki işlenendeki (operand'daki) belirtilen bit sayısı kadar sola kaydırır. Sağa boşalan bitler 0'larla değiştirilir; sola kaydırılan bitler kaybolur. Şekil 10.13'te, değişken number1'e 9ncu satırında 960 (00000000 00000000 00000011 11000000) değeri atanmıştır. number1 << 8 (satır 15) ifadesindeki 8 bitlik sola kaydırma değişkeninin sonucu 245760'tır (00000000 00000011 1100) 0000 00000000) .

Sağa Kaydırma Operatörü (>>)

Sağa kaydırma operatörü (>>), soldaki işleneninin (operand'ın) bitlerini sağdaki işlenendeki (operand'daki) belirtilen bit sayısı kadar sağa kaydırır. İşaretsiz bir int üzerinde sağa kaydırma yapmak, soldaki boşalan bitlerin 0'larla değiştirilmesine neden olur; sağa kaydırılan bitler kaybolur. Şekil 10.13'te, number1 >> 8 (satır 21) ifadesindeki sağa kaydırma number1'in sonucu 3'tür (00000000 00000000 00000000 00000011).