Bináris műveletek gyakorlati alkalmazása: AND, OR, XOR

A bináris logikai műveletek (**AND**, **OR**, **XOR**) fontos szerepet játszanak a bitek kezelésében, mint például a bitek beállítása, kikapcsolása, illetve megfordítása. Az alábbiakban bemutatjuk, hogyan lehet ezeket a műveleteket felhasználni gyakorlati feladatok megoldására.

1. AND (és) művelet

Az **AND** művelet csak akkor ad 1-es eredményt, ha mindkét bemenet 1. A bitek "kikapcsolására" használhatjuk, mert a 0 hatására minden egyes bitet 0-ra állít.

Igazságtábla:

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

Példa: Adott egy 8 bites számunk: `10101101`. Ha csak az alsó négy bitet (jobbról négyet) akarjuk megtartani, akkor használjuk az AND műveletet:

```
  10101101
& 00001111
  --------
  00001101
```

Ez hasznos, ha egy adott bitcsoportot akarunk *maszkolni*, azaz megtartani a kívánt biteket, és lenullázni a többit.

2. OR (vagy) művelet

Az **OR** művelet akkor ad 1-es eredményt, ha legalább az egyik bemenet 1. Ezt a bitek "bekapcsolására" használhatjuk, mert a 0 nem változtatja meg az eredményt, de a 1-es beállítja az adott bitet 1-re.

Igazságtábla:

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

Példa: Ha egy bitek közül egyet be akarunk állítani (pl. jobbról a negyediket), és a többit érintetlenül hagyni, akkor használjuk az OR-t:

```
  10101101
| 00001000
----------
  10101101
```

Ez hasznos, ha egy konkrét bitet szeretnénk 1-re állítani anélkül, hogy a többi bitet megváltoztatnánk.

3. XOR művelet

Az **XOR** (kizáró vagy) művelet akkor ad 1-et, ha a bemenetek eltérnek. Ezzel biteket kapcsolgathatunk, azaz ha egy bit értéke 1, akkor 0-ra vált, és fordítva.

Igazságtábla:

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Példa: Ha az alsó 4 bit értékét akarjuk megfordítani:

```
  10101101
^ 00001111
----------
  10100010
```

Gyakorlati példa: Bitek beállítása és törlése

Tegyük fel, hogy van egy 8 bites regiszterünk: `R = 10101101`.

Egy bit bekapcsolása (pl. 3. bit):

`R = 10101101`

Maszk legyen: `00000100`

Megjegyzés: a maszkot a gyakorlatban a '«' biteltolás művelettel adják meg, az `1 « 2` művelet az első bitet kettővel balra tolja.

Művelet: `R | Maszk`

Eredmény:

```
  10101001
| 00000100
----------
```

```
  10101101 (a 3. bit 1 lett)
```

Egy bit kikapcsolása (pl. 4. bit):

`R = 10101101`

Maszk legyen: `11110111`, a gyakorlatban ezzel adjuk meg: ~(1 « 3)

Megjegyzés: ~(1 « 3) művelet az első bitet kettővel a negyedik helyre (helyiértékre) tolja, majd a '~'-al negálja, azaz megfordítja biteket.

Művelet: `R & Maszk`

Eredmény:

```
  10101101
& 11110111
----------
  10100101  (a 4. bit 0 lett)
```

Egy bit átkapcsolása (pl. 0. bit):

Maszk legyen: `00000001` (1 « 0)

Művelet: `R ^ Maszk`

Eredmény:

```
  10101101
^ 00000001
----------
  10101100  (a 0. bit megfordult)
```

Újra alkalmazva ugyanezt a műveletet az új eredményen `10101100`:

```
  10101100
^ 00000001
----------
  10101101  (a 0. bit megfordult)
```

Az igazságtáblák és a fenti példák segítségével megérthető, hogyan manipulálhatók a bitek a logikai műveletek segítségével. Ez alapvető fontosságú az alacsony szintű programozásban és a digitális áramkörök tervezésében.

Az XOR fontos tulajdonságai

Az XOR műveletnek több hasznos tulajdonsága van:

```
A XOR 0 = A
```

```
A XOR 1 = NOT A
A XOR A = 0
(A XOR B) XOR B = A
```

Ezek jelentése:

- A 0-val végzett XOR nem változtatja meg a bitet.
- Az 1-gyel végzett XOR megfordítja a bitet.
- Egy bit önmagával végzett XOR művelete mindig 0.
- Ha ugyanazzal az értékkel kétszer végzünk XOR műveletet, visszakapjuk az eredeti értéket.

Az utolsó tulajdonság teszi lehetővé, hogy az XOR műveletet egyszerű titkosításra is használjuk.

Két bináris érték különbségének megkeresése

Az XOR segítségével könnyen megállapítható, hogy két bináris érték mely bitpozíciókban tér el egymástól.

Legyen a két érték:

```
  10101101
^ 10000111
----------
  00101010
```

Az eredményben ott található 1-es, ahol a két eredeti érték bitjei különböznek. A fenti példában a két szám három bitpozícióban tér el egymástól.

Ez például két hardverregiszter, állapotjelző vagy adatcsomag összehasonlításakor lehet hasznos.

Egyszerű titkosítás XOR művelettel

Legyen az üzenetet tartalmazó bájt:

```
Üzenet: 01000001
Kulcs:   00110110
```

Az üzenet titkosításához XOR műveletet végzünk:

```
  01000001   üzenet
^ 00110110   kulcs
----------
  01110111   titkosított adat
```

A titkosított adat visszafejtéséhez ugyanazt a kulcsot ismét alkalmazzuk:

```
  01110111   titkosított adat
```

```
^ 00110110   kulcs
----------
  01000001   eredeti üzenet
```

Ez azért működik, mert:

```
(Üzenet XOR Kulcs) XOR Kulcs = Üzenet
```

A kulcs kétszeri alkalmazása megszünteti annak hatását, mivel:

```
Kulcs XOR Kulcs = 0
```

és:

```
Üzenet XOR 0 = Üzenet
```

Szöveg titkosítása XOR művelettel

A számítógép a szöveg karaktereit is számokként tárolja. Például az A karakter ASCII-kódja 65, amely binárisan:

```
A = 65 = 01000001
```

Ha a titkosítási kulcs értéke 54:

```
Kulcs = 54 = 00110110
```

akkor a titkosított bájt:

```
  01000001   A karakter
^ 00110110   kulcs
----------
  01110111   titkosított bájt
```

A visszafejtéskor ugyanezt a kulcsot használjuk:

```
  01110111
^ 00110110
----------
  01000001   A karakter
```

Egy programban az eljárás elvi formája:

```
titkositott_adat = eredeti_adat ^ kulcs;
visszafejtett_adat = titkositott_adat ^ kulcs;
```

Fontos, hogy az XOR művelet önmagában nem garantál biztonságos titkosítást. Ha rövid vagy könnyen kitalálható kulcsot használunk, illetve ugyanazt a kulcsot sokszor megismételjük, a titkosított

üzenet feltörhető lehet.

One-time pad titkosítás

Az XOR-ra épülő különleges titkosítási módszer a *one-time pad*, magyarul egyszer használatos kulcs.

A módszer akkor tekinthető tökéletesen biztonságosnak, ha:

- a kulcs valóban véletlenszerű;
- a kulcs legalább olyan hosszú, mint az üzenet;
- a kulcsot titokban tartják;
- ugyanazt a kulcsot csak egyszer használják fel.

A titkosítás:

```
Titkosított üzenet = Üzenet XOR Kulcs
```

A visszafejtés:

```
Üzenet = Titkosított üzenet XOR Kulcs
```

A gyakorlatban az egyszer használatos kulcs biztonságos továbbítása és tárolása nehéz, ezért a modern titkosítási eljárások általában nem közvetlenül ezt a módszert használják. A folyamtitkosítók azonban hasonló elven működnek: egy kulcsból előállított kulcsfolyamot XOR művelettel kapcsolnak össze az üzenettel.

Egyszerű XOR-titkosítás programban

Az alábbi C program minden karaktert ugyanazzal a kulccsal alakít át:

```c
#include <stdio.h>

int main()
{
    char szoveg[] = "HELLO";
    char kulcs = 42;

    for (int i = 0; szoveg[i] != '\0'; i++)
    {
        szoveg[i] = szoveg[i] ^ kulcs;
    }

    printf("Titkositott adat: ");

    for (int i = 0; szoveg[i] != '\0'; i++)
    {
        printf("%02X ", (unsigned char)szoveg[i]);
```

```
    }

    for (int i = 0; szoveg[i] != '\0'; i++)
    {
        szoveg[i] = szoveg[i] ^ kulcs;
    }

    printf("\nVisszafejtett szoveg: %s\n", szoveg);

    return 0;
}
```

A titkosított adatot hexadecimális számokként jelenítjük meg, mert az XOR művelet eredményeként nem nyomtatható karakterek is keletkezhetnek.

Ez a program csak az XOR működését szemlélteti. Valódi adatok védelmére nem biztonságos, mert minden karakterhez ugyanazt az egybájtos kulcsot használja.

Két változó értékének felcserélése XOR művelettel

Érdekességként két egész szám értéke ideiglenes változó nélkül is felcserélhető:

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

Például legyen:

```
a = 5
b = 3
```

Binárisan:

```
a = 0101
b = 0011
```

A műveletek:

```
a = 0101 XOR 0011 = 0110
b = 0110 XOR 0011 = 0101
a = 0110 XOR 0101 = 0011
```

Az eredmény:

```
a = 3
b = 5
```

A modern programokban azonban általában jobb ideiglenes változót vagy beépített felcserélő műveletet használni, mert az érthetőbb és kevésbé hibalehetőséges.

Ellenőrzőérték és paritás számítása

Az XOR művelet egyszerű hibafelismerésre is használható. Több adatbájt XOR műveletével egy ellenőrzőérték képezhető:

```
ellenorzoertek = adat1 XOR adat2 XOR adat3
```

Ha valamelyik adatbit megváltozik, általában az ellenőrzőérték is megváltozik. Ezért az XOR műveletet adatátviteli és adattárolási eljárásokban is alkalmazzák.

From:
https://edu.iit.uni-miskolc.hu/ - **Institute of Information Science - University of Miskolc**

Permanent link:
https://edu.iit.uni-miskolc.hu/tanszek:oktatas:szamitastechnika:binaris_muveletek

Last update: **2026/09/29 18:00**