```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    unsigned int a = 60;// 0011 1100
    unsigned int b = 13;
    int result = 0;
    int result2 = 0;
    int result3 = 0;
    int result4 = 0;


    result = ~a;// 0011 1101
    result2 = a << 2;
    result3 = a << 4;
    result4 = a >> 4;

    printf("result is %d\n\n", result);
    printf("result2 is %d\n\n", result2);
    printf("result3 is %d\n\n", result3);
    printf("result4 is %d\n\n", result4);
    return 0;
}
```

- `a = 60` ami binárisan: `0011 1100` (8 bitre vetítve)
- `~a` a bitenkénti negáció (bitwise NOT), azaz minden bitet megfordít (0-ból 1 lesz, 1-ből 0).
  Pl.: `~0011 1100` => `1100 0011`
  (ez az érték a `signed int` esetén negatív szám lesz, mert a legfelső bit a jelző bit).
- `a << 2` azt jelenti, hogy az `a` értékét 2 bittel balra tolod, azaz lényegében szorzod 4-gyel.
- `a << 4` az `a` értékét 4 bittel tolja balra (szorozva 16-tal).
- `a >> 4` az `a` értékét 4 bittel tolja jobbra, vagyis elosztja 16-tal (egész osztás, maradék nélkül).

A `printf`-ben `%d`-vel írod ki az eredményeket, vagyis **signed int**-ként értelmezed őket, még akkor is, ha az `a` és `b` unsigned volt. Ez a `~a` esetén különösen fontos, mert a bitnegáció után az eredmény signed negatív szám lesz.

Összefoglalva:

- A program **számok bitenkénti manipulációját** végzi.
- Nem konvertál vagy fordít át szöveget.
- Bitműveletekkel matematikai és logikai műveleteket hajt végre számokon.

Ha szeretnéd, segítek értelmezni az eredményeket vagy átalakítani a kódot, hogy jobban megértsd. Kérdezz nyugodtan!

*/