Rukovanje bitima

Brojni sistemi

Cifre brojnog sistema

- Za datu osnovu/bazu `b` cifre brojnog sistema su od `0` do `b-1`
- Češće korišćeni brojni sistemi:
  - binarni (`b=2`): `{0,1}`
  - oktalni (`b=8`): `{0,1,2,3,4,5,6,7}`
  - decimalni (`b=10`): `{0,1,2,3,4,5,6,7,8,9}`
  - heksadecimalni (`b=16`): `{0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}`

Pregled notacije brojnih sistema

```
dec   oct   hex      bin

  0    00   0x0   0b0000
  1    01   0x1   0b0001
  2    02   0x2   0b0010
  3    03   0x3   0b0011
  4    04   0x4   0b0100
  5    05   0x5   0b0101
  6    06   0x6   0b0110
  7    07   0x7   0b0111
  8   010   0x8   0b1000
  9   011   0x9   0b1001
 10   012   0xA   0b1010
 11   013   0xB   0b1011
 12   014   0xC   0b1100
 13   015   0xD   0b1101
 14   016   0xE   0b1110
 15   017   0xF   0b1111
```

`0b` notacija definisana je C++14 standardom i podržana od strane novijih programskih prevodilaca

Heksadecimalni brojni sistem

Vrednosti pojedinačnih bitova u nekom broju možemo zadavati kroz:

```
bin  0b11011110101011011100000011011110
oct  033653340336
dec  3735929054
hex  0xDEADC0DE
```

ali se heksadecimalni brojni sistem pokazao najpogodnijim u praksi, pošto jedna hex cifra u potpunosti opisuje vrednost 4 bita (tzv. *nibble*):

```
  D    E    A    D    C    0    D    E   // hex
1101 1110 1010 1101 1100 0000 1101 1110  // bin
```

Ispis binarnog zapisa broja obično se deli u grupe od po 4 bita, zarad bolje čitljivosti i lakše konverzije u hex brojni sistem.

Bitwise operatori

Bitwise operator ~ (1-komplement)

Ponašanje:

```
a   ~ a

0     1
1     0
```

Primer:

```
0100 0111 0101 0111  //   a

1011 1000 1010 1000  // ~ a   (bitwise operator)
0000 0000 0000 0000  // ! a   (logicki operator)
```

Bitwise operator & (i)

Ponašanje:

```
a  b   a & b

0  0     0
0  1     0
1  0     0
1  1     1
```

Primer:

```
0100 0111 0101 0111  // a
1101 0100 1010 1001  // b

0100 0100 0000 0001  // a &  b   (bitwise operator)
0000 0000 0000 0001  // a && b   (logicki operator)
```

Bitwise operator | (ili)

Ponašanje:

```
a  b   a | b

0  0     0
0  1     1
1  0     1
1  1     1
```

Primer:

```
0100 0111 0101 0111  // a
1101 0100 1010 1001  // b

1101 0111 1111 1111  // a |  b   (bitwise operator)
0000 0000 0000 0001  // a || b   (logicki operator)
```

Bitwise operator ^ (eksluzivno ili)

Ponašanje:

```
a  b   a ^ b

0  0     0
0  1     1
1  0     1
1  1     0
```

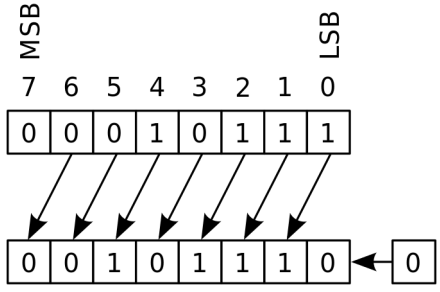
Primer:

```
0100 0111 0101 0111  // a
1101 0100 1010 1001  // b

1001 0011 1111 1110  // a ^ b
```

Bitwise operator << (left shift)

Ponašanje:



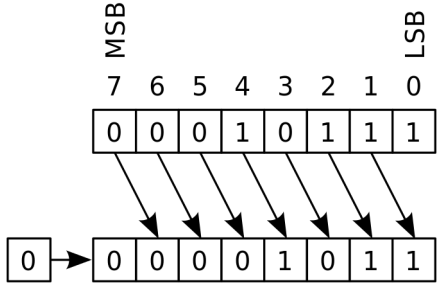
Primer:

```
0100 0111 0101 0111  // a

1000 1110 1010 1110  // a << 1
0001 1101 0101 1100  // a << 2
0011 1010 1011 1000  // a << 3
0111 0101 0111 0000  // a << 4
```

Bitwise operator >> (right shift)

Ponašanje:



Primer:

```
0100 0111 0101 0111  // a

0010 0011 1010 1011  // a >> 1
0001 0001 1101 0101  // a >> 2
0000 1000 1110 1010  // a >> 3
0000 0100 0111 0101  // a >> 4
```

Bitwise operatori sa sporednim efektom

Ponašanje:

```
a &= b;   // a = a & b;
a |= b;   // a = a | b;
a ^= b;   // a = a ^ b;
a <<= b;  // a = a << b;
a >>= b;  // a = a >> b;
```

Bitmaske

Zašto bitmaske?

Stanje fabrike sladoleda možemo opisati sa više različitih promenljiva:

```
int prisutna_cokolada   = 0; // C
int prisutna_jagoda     = 1; // J
int prisutna_vanila     = 0; // V
int prisutan_beli_luk   = 1; // B  :)   https://goo.gl/jUOcBc

int porodicno_pakovanje = 1; // P
int ambalaza_ok         = 1; // A
int temperatura_ok      = 0; // T  :(
int kolicina_smese_ok   = 1; // S
```

ili možemo da spakujemo sve navedene podatke u **svega jedan bajt**, ostvarujući znatnu uštedu memorije (32x) i brži prenos podataka:

```
0101 1101 // CJVB PATS
```

Bitmaske nam omogućavaju da na jednostavan i efikasan način rukujemo podacima na nivou pojedinačnih bita.

Postavljanje vrednosti bita na 1

Kako?

- Bitwise operator |
- Normalna bitmaska (1 na interesantnim bitovima, 0 na ostalim)

Primer:

```
0100 0001  // data          (dec 65, ASCII?)
0010 0000  // mask          (hex 0x20)

0110 0001  // data | mask   (dec 97, ASCII?)
```

Koje su *ASCII* vrednosti?

Slučajne slučajnosti nisu previše slučajne...

Postavljanje vrednosti bita na 0

Kako?

- Bitwise operator &
- *Invertovana* bitmaska (0 na interesantnim bitovima, 1 na ostalim)

Primer:

```
0110 0001  // data          (dec 97, ASCII?)
1101 1111  // mask          (hex 0xDF)

0100 0001  // data & mask   (dec 65, ASCII?)
```

Koje su *ASCII* vrednosti?

Da li bi rezultat bio valjan da nismo koristili invertovanu bitmasku?

Dobavljanje vrednosti bita

Kako?

- Bitwise operator &
- Normalna bitmaska (1 na interesantnim bitovima, 0 na ostalim)

Primer:

```
0100 0111 0101 0111  // data
0000 0000 0000 1111  // mask          (hex 0xF)

0000 0000 0000 0111  // data & mask
```

Invertovanje vrednosti bita

Kako?

- Bitwise operator ^
- Normalna bitmaska (1 na interesantnim bitovima, 0 na ostalim)

Primer:

```
0100 0111 0101 0111  // data
1100 0000 0000 0001  // mask          (hex 0xC001)

1000 0111 0101 0110  // data ^ mask
```

Primeri upotrebe

Da li je broj (ne)paran?

```c
int is_odd(unsigned short num) {
    // A number is odd if its lowest bit is set to 1
    return num & 1;
}

int is_even(unsigned short num) {
    return ! is_odd(num); // https://en.wikipedia.org/wiki/KISS_principle
}
```

Da li je broj (ne)paran?

bin	dec	is_odd	is_even
0000 0000 0010 1010	42	0	1
0000 0000 0010 1011	43	1	0
0000 0000 0010 1100	44	0	1
0000 0000 0010 1101	45	1	0
0000 0000 0010 1110	46	0	1
0000 0000 0010 1111	47	1	0
0000 0000 0011 0000	48	0	1
0000 0000 0011 0001	49	1	0
0000 0000 0011 0010	50	0	1
0000 0000 0011 0011	51	1	0
0000 0000 0011 0100	52	0	1
0000 0000 0011 0101	53	1	0
0000 0000 0011 0110	54	0	1
0000 0000 0011 0111	55	1	0
0000 0000 0011 1000	56	0	1
0000 0000 0011 1001	57	1	0
0000 0000 0011 1010	58	0	1
0000 0000 0011 1011	59	1	0

Da li je broj negativan/pozitivan?

```
int is_negative(short num) {
    // A number is negative if its highest bit is set to 1
    return (num & 0x8000) != 0;
}

int is_positive(short num) {
    return ! is_negative(num); // https://en.wikipedia.org/wiki/KISS_principle
}
```

Da li je broj negativan/pozitivan?

```
bin                     dec        is_positive        is_negative

1111 1111 1111 1011      -5           0                  1
1111 1111 1111 1100      -4           0                  1
1111 1111 1111 1101      -3           0                  1
1111 1111 1111 1110      -2           0                  1
1111 1111 1111 1111      -1           0                  1
0000 0000 0000 0000       0           1                  0
0000 0000 0000 0001       1           1                  0
0000 0000 0000 0010       2           1                  0
0000 0000 0000 0011       3           1                  0
0000 0000 0000 0100       4           1                  0
0000 0000 0000 0101       5           1                  0
```

Generisanje jednostavnih bitmaski

```
unsigned short gen_mask(unsigned short pos) { // pos in [0..15]
    return 1 << pos;
}

unsigned short gen_inverted_mask(unsigned short pos) { // pos in [0..15]
    return ~ gen_mask(pos);
}
```

Generisanje jednostavnih bitmaski

```
mask                    pos

0000 0000 0000 0001      0
0000 0000 0000 0010      1
0000 0000 0000 0100      2
0000 0000 0000 1000      3
0000 0000 0001 0000      4
0000 0000 0010 0000      5
0000 0000 0100 0000      6
0000 0000 1000 0000      7
0000 0001 0000 0000      8
0000 0010 0000 0000      9
0000 0100 0000 0000     10
0000 1000 0000 0000     11
0001 0000 0000 0000     12
0010 0000 0000 0000     13
0100 0000 0000 0000     14
1000 0000 0000 0000     15
```

Ispis binarnog zapisa broja

```
void print_bin(unsigned short data) {
    short i;
    for(i=15; i>=0; i--) {
        printf ("%d", (data & gen_mask(i)) != 0);

        if (i % 4 == 0) {
            printf(" "); // Add a space after every 4 bin digits
        }
    }
}
```

Ispis binarnog zapisa broja

```
bin                       dec

0000 0000 0010 1010        42
0000 0000 0010 1011        43
0000 0000 0010 1100        44
0000 0000 0010 1101        45
0000 0000 0010 1110        46
0000 0000 0010 1111        47
0000 0000 0011 0000        48
0000 0000 0011 0001        49
0000 0000 0011 0010        50
0000 0000 0011 0011        51
0000 0000 0011 0100        52
0000 0000 0011 0101        53
0000 0000 0011 0110        54
0000 0000 0011 0111        55
0000 0000 0011 1000        56
0000 0000 0011 1001        57
0000 0000 0011 1010        58
0000 0000 0011 1011        59
```

Broj jedinica u binarnom zapisu broja

```
unsigned short count_ones_v1(unsigned short data) {
    unsigned short i, count = 0;
    for(i=0; i<16; i++) {
        if (data & gen_mask(i)) {
            count++;
        }
    }
    return count;
}

unsigned short count_ones_v2(unsigned short data) {
    unsigned short count = 0, mask = 1;
    while(mask) { // `mask` becomes `0` once all bits have been exhausted
        if (data & mask) {
            count++;
        }

        mask <<= 1; // Move to the next (higher) bit
    }
    return count;
}
```

Broj jedinica u binarnom zapisu broja

bin	dec	count_ones_v1	count_ones_v2
0000 0000 0010 1010	42	3	3
0000 0000 0010 1011	43	4	4
0000 0000 0010 1100	44	3	3
0000 0000 0010 1101	45	4	4
0000 0000 0010 1110	46	4	4
0000 0000 0010 1111	47	5	5
0000 0000 0011 0000	48	2	2
0000 0000 0011 0001	49	3	3
0000 0000 0011 0010	50	3	3
0000 0000 0011 0011	51	4	4
0000 0000 0011 0100	52	3	3
0000 0000 0011 0101	53	4	4
0000 0000 0011 0110	54	4	4
0000 0000 0011 0111	55	5	5
0000 0000 0011 1000	56	3	3
0000 0000 0011 1001	57	4	4
0000 0000 0011 1010	58	4	4
0000 0000 0011 1011	59	5	5

Rukovanje pojedinačnim bitima

```
int get_bit(unsigned short data, unsigned short pos) {
    return (data & gen_mask(pos)) != 0;
}

void set_bit(unsigned short *data, unsigned short pos) {
    *data |= gen_mask(pos);
}

void clear_bit(unsigned short *data, unsigned short pos) {
    *data &= gen_inverted_mask(pos);
}

void invert_bit(unsigned short *data, unsigned short pos) {
    *data ^= gen_mask(pos);
}
```

Rukovanje pojedinačnim bitima

```
1100 0000 1101 1110         data

1100 0010 1101 1110         set_bit:pos=9
       *

0100 0010 1101 1110         clear_bit:pos=15
*

0100 0010 1101 0110         invert_bit:pos=3
               *

0100 0010 1101 0111         invert_bit:pos=0
                  *

0100 0010 1101 0111         get_bit:pos=14 --> 1
 *

0100 0010 1101 0111         get_bit:pos=13 --> 0
  *
```