```
#define settype(newtype, mode) mode = ((mode) & ~S_IFMT) | (newtype)
```

Il primo termine `((mode) & ~S_IFMT)` azzera i bit di mode che rappresentano il tipo del file, il seguito della macro (`| (newtype)`) imposta il nuovo tipo. Applichiamo la macro al `mode` precedente (`mode = 0b1000000111101101`) per cambiare il tipo in link (`0b1010000000000000`):

```
mode    0b1000000111101101 &
~S_IFMT 0b0000111111111111
        ------------------
        0b0000000111101101 |
        0b1010000000000000 =
        ------------------
        0b1010000111101101
```

riassumendo

Pattern classici per alcune azioni comuni sono:

- accendere il bit n in v

```
v |= 1 << n
```

- invertire il bit n in v

```
v ^= 1 << n
```

- spegnere il bit n in v

```
v &= ~(1 << n)
```

- assegnare il valore del bit n di v alla variable mybit:

```
mybit = (v >> n) & 1
//oppure
mybit = !!(v & (1 << n))
```

- copiare il valore del bit n da v a w

```
w = (w & ~(1<<n)) | (v & (1<<n))
```

- fare qualcosa se il bit n di è acceso:

```
if (v & (1 << n)) ....
```

per fare qualcosa se il bit è spento basta negare la condizione

- accendere in w tutti i bit accesi in v:

```
w |= v
```

- spegnere in w i bit accesi in v:

```
w &= ~v
```

- copiare i bit selezionati dalla maschera m (quelli accesi in m) da v a w:

```
w = (w & ~m) | (v & m)
```

- fare qualcosa se almeno un bit di v è acceso in w:

```
if (v & w) ...
```

- fare qualcosa se i bit selezionati dalla maschera m sono uguali in v e in w:

```
if ((v & m) == (w & m)) ....
```

se i bit accesi di w sono un sottoinsieme di quelli accesi in m (come per esempio `S_IFDIR` rispetto a `S_IFMT`) si può scrivere:

```
if ((v & m) == w) ....
```

- selezionare i singoli byte:

```
unsigned int value;
unsigned int byte[4];
byte[3] = (value >> 24);         // oppure (value & 0xff000000) >> 24;
byte[2] = (value >> 16) & 0xff; // oppure (value & 0x00ff0000) >> 16;
byte[1] = (value >> 8) & 0xff;  // oppure (value & 0x0000ff00) >> 8;
byte[0] = value & 0xff;          // oppure (value & 0x000000ff) >> 0;
```

Se il la variabile byte è un array di interi a 8 bit si può evitare il mascheramento `& 0xff`:

```
unsigned int value;
unsigned char byte[4];
byte[3] = value >> 24;
byte[2] = value >> 16;
byte[1] = value >> 8;
byte[0] = value >> 0;
```

- operazioni veloci di "modulo potenze di 2".

molte strutture dati (buffer, tabelle di hash, ...) hanno dimensioni pari ad una potenza di 2 per poter selezionare velocemente gli elementi usando mascheramenti al posto dell'operatore modulo (%). Infatti `x % (2 ^ n)` equivale a `x & ((2 ^ n) - 1)`.

```
odd = num & 1; // invece di odd = num % 2
elem = index & 255; // invece di elem = index % 256
```

- inverti il valore di v con w:

```
v = v ^ w;
w = v ^ w;
v = v ^ w;
```