

Masking & Bitwise Operations

In computer science, a mask is data used for bitwise operations.

*Masking can be used to protect bit(s) from change while writing a variable or register. This is done by AND'ing or OR'ing a bit pattern with another bit pattern to select some bits.
(JPUG1)*

Suppose we have two LEDs connected to LATA. One is on bit 2, the other is on bit 5. To turn on each LED we make the associated LATA bit an output using the TRISA register and then make that output high.

Suppose we want to turn on both LEDs. We could do so by writing 0x24 to LATA but that would write 0's to the bits not used by our LEDs. Writing 0xFF would write 1's to the bits not used by our LEDs. But we only want to change bit 2 and 5. We can use masking to do so.

The mask for the bits we need to change is 0b00100100 or 0x24. Conversely we can represent the bits we want to keep from changing as ~mask .

The first step is to extract the values of the bits we want to keep from LATA.

`(LATA & ~mask)`

Next we extract the value of the bits we want to change from the new_bits.

`(new_bits & mask)`

The new value of LATA will be the bitwise or of these.

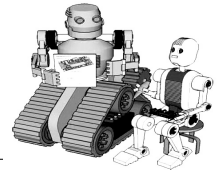
`LATA = (kept bits) | (replaced bits)`
or
`LATA = ((LATA & ~mask) | (new_bits & mask));`

The code to turn the LEDs on is

`LATA = ((LATA & ~mask) | (0xFF & mask));`

The code to turn the LEDs off

`LATA = ((LATA & ~mask) | (0x00 & mask));`



We can write a general purpose masking function that returns a new value.

```
// Returns a byte where bits in oldData have been replaced by newData bits.
#define uByte unsigned char

uByte maskByte(uByte mask, uByte oldData, uByte newData)
{
    uByte result;

    result = ( oldData & ~mask) | (newData & mask) );
    return result;
}
```

Using this function we could turn our LEDs on and off using

```
LATA=maskByte(0x24,LATA,0xFF);
LATA=maskByte(0x24,LATA,0x00);
```

To turn only the led on 2 on we could write.

```
LATA=maskByte(0x24,LATA,0x04);
    or
LATA=maskByte(0x24,LATA,0xDF);
```

Both work because they each have a 0 in the bit 5 position and a 1 in the bit 2 position. The remainder of the bits in the newData are ignored due to masking.

Summary

We have seen that masking is useful because it allows us to change some bits while leaving other bits unchanged. Masking also illustrates the bitwise operators & | and ~.