Using Masks to Manipulate Bits in a Byte
(Just the Basics)

Writing a "1" bit

Bit to be changed

Original byte:

1	0	1	1	**X**	0	1	0

$\downarrow$

Bit-wise *OR* with mask

$\downarrow$

To write a "1", use this mask:

0	0	0	0	**1**	0	0	0

$\downarrow$

Produces this byte:

1	0	1	1	**1**	0	1	0

Writing a "0" bit

Original byte:

1	0	1	1	**X**	0	1	0

$\downarrow$

Bit-wise *AND* with mask

$\downarrow$

To write a "0", use this mask:

1	1	1	1	**0**	1	1	1

$\downarrow$

Produces this byte:

1	0	1	1	**0**	0	1	0

Reading a bit

Bit to be read

Original byte:

1	0	1	1	**X**	0	1	0

$\downarrow$

Bit-wise *AND* with mask

$\downarrow$

To write a “0”, use this mask:

0	0	0	0	**1**	0	0	0

$\downarrow$

Produces this byte:

0	0	0	0	**X**	0	0	0

If the resulting byte is non-zero, the bit was a “1”

Working With Bit Masks

There are many variations on bit masking:

- These examples use bytes as illustration, but bit masks are usually used with *int* variables.
- Bit masks can be saved in arrays, making it easier to manipulate bit states in loops.
- In the “write” example above the bit-wise NOT operator (~) can be applied to the “1” mask to produce the “0” mask.
- A bit-wise XOR mask can be used to “flip” a bit state.
- Bit shift operators can be used with a single mask in a loop to move an individual flag from bit to bit.
- Multiple bits can be set in the same operation by combining masks.