Christopher Lum
lum@uw.edu

Lecture
Bit Shifting, Bit Masking, and Bit Manipulation

Lecture is on YouTube

The YouTube video entitled ‘Bit Shifting, Bit Masking, and Bit Manipulation’ that covers this lecture is located at https://youtu.be/4JgtUf5ThqY.

Outline

-References
-Bitwise Operations
    -and, or, xor, not
    -Left Shift
        -Left Shift Unsigned Value
        -Left Shift Signed Value
    -Right Shift
        -Right Shift Unsigned Value
        -Right Shift Signed Negative Value
        -Right Shift Signed Positive Value
-Bit Masks
-Bit Manipulation
    -BitSetTo1
    -BitSetTo0
    -BitSetToValue
    -BitFlip
    -BitIs1
    -BitIs0
-MATLAB and C Implementations

References

-https://www.rapidtables.com/convert/number/binary-to-decimal.html
-https://www.omnicalculator.com/math/twos-complement

-https://en.wikipedia.org/wiki/Arithmetic_shift
-https://en.wikipedia.org/wiki/Logical_shift

Bitwise Operations

The following are the basic bitwise operations

AND	&
OR	|
XOR	^
NOT	~
LEFT SHIFT	<<
RIGHT SHIFT	>>

and, or, xor, not

The truth tables for a single bit are as follows

AND
0 & 0 = 0
1 & 0 = 0
0 & 1 = 0
1 & 1 = 1

OR
0 | 0 = 0
1 | 0 = 1
0 | 1 = 1
1 | 1 = 1

XOR
0 | 0 = 0
1 | 0 = 1
0 | 1 = 1
1 | 1 = 0

NOT
~0 = 1
~1 = 0

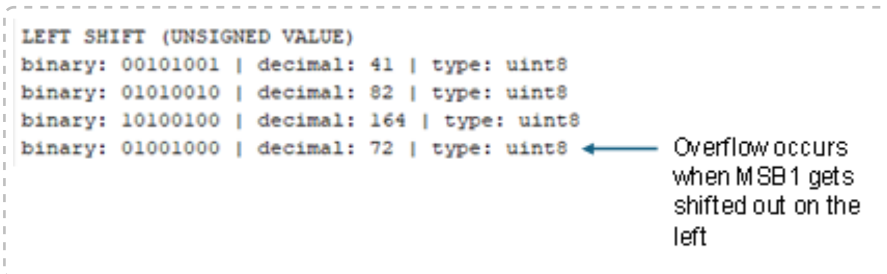
Left Shift

The left and right shift operators require operation on more than 1 bit (in this example we use a full byte)

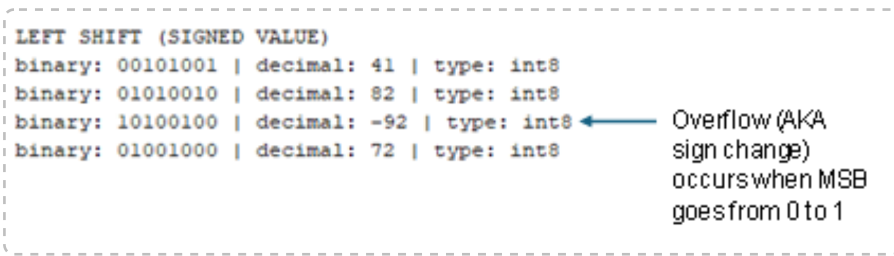
LEFT SHIFT
0010 1001 << 2 = 1010 0100

-This shifts the binary digits to the left by *n* and pads 0's on the right
-Each shift is equivalent to a multiply by 2 (unless there is overflow). Note that overflow can also include a 0 changing to a 1 in the MSB on a signed data type (this changes the sign of the value and is therefore not a multiply by 2)

Left Shift Unsigned Value



Left Shift Signed Value



Right Shift

We need to be a little more careful with the right shift operation, especially w.r.t. signed values (ie the MSB is a 1). This is the difference between an arithmetic shift (https://en.wikipedia.org/wiki/Arithmetic_shift) and logical shift (https://en.wikipedia.org/wiki/Logical_shift)

RIGHT SHIFT
1010 1001 >> 2 = XX10 1010

-This shifts the binary digits to the right by *n* and pads 0's on the left if the type is an unsigned int. If it is signed then it pads with a copy of the MSB.
-Each shift is equivalent to a divide by 2 with a round towards negative infinity (this is true even when the MSB 1 is shifted out on the right).

Right Shift Unsigned Value

Note that MSB = 1 but because data type is unsigned, zeros are shifted in on the left

```
RIGHT SHIFT (UNSIGNED VALUE)
binary: 10101001 | decimal: 169 | type: uint8
binary: 01010100 | decimal: 84 | type: uint8
binary: 00101010 | decimal: 42 | type: uint8
binary: 00010101 | decimal: 21 | type: uint8
binary: 00001010 | decimal: 10 | type: uint8
binary: 00000101 | decimal: 5 | type: uint8
binary: 00000010 | decimal: 2 | type: uint8
binary: 00000001 | decimal: 1 | type: uint8
binary: 00000000 | decimal: 0 | type: uint8
```

Right Shift Signed Negative Value

Note that MSB = 1 but because data type is signed, ones are shifted in on the left

```
RIGHT SHIFT (SIGNED NEGATIVE VALUE)
binary: 10101001 | decimal: -87 | type: int8
binary: 11010100 | decimal: -44 | type: int8
binary: 11101010 | decimal: -22 | type: int8
binary: 11110101 | decimal: -11 | type: int8
binary: 11111010 | decimal: -6 | type: int8
binary: 11111101 | decimal: -3 | type: int8
binary: 11111110 | decimal: -2 | type: int8
binary: 11111111 | decimal: -1 | type: int8
binary: 11111111 | decimal: -1 | type: int8
```

Right Shift Signed Positive Value

Note that MSB = 0 and because data type is signed, zeros are shifted in on the left

```
RIGHT SHIFT (SIGNED POSITIVE VALUE)
binary: 00010011 | decimal: 19 | type: int8
binary: 00001001 | decimal: 9 | type: int8
binary: 00000100 | decimal: 4 | type: int8
binary: 00000010 | decimal: 2 | type: int8
binary: 00000001 | decimal: 1 | type: int8
binary: 00000000 | decimal: 0 | type: int8
```

Bit Masks

BitMask(position)

mask = 1 << position
return mask

Note that position is a 0-based value (0 = 1st (LSB) bit, 7 = 8th (MSB) bit)

Example

BitMask(2) ⇒ 0b 0000 0100

These are referred to as masks as they allow us to select or manipulate specific bits as we will see in the next section.

Bit Manipulation

BitSetTo1

Also sometimes referred to as ‘Bit Set’.

BitSetTo1(x,position)

mask = BitMask(position)
return x | mask

Example

x = 0b11010101
position = 3

				Bit							
	Decimal	Hex		7	6	5	4	3	2	1	0
x	213	0xD5		1	1	0	1	0	1	0	1
mask	8	0x08		0	0	0	0	1	0	0	0
x | mask	221	0xDD		1	1	0	1	1	1	0	1

BitSetTo0

Also sometime referred to as ‘Bit Clear’.

BitSetTo0(x,position)

mask = BitMask(position)
return x & ~mask

Example

x = 0b11010101
position = 6

				Bit							
	Decimal	Hex		7	6	5	4	3	2	1	0
x	213	0xD5		1	1	0	1	0	1	0	1
mask	64	0x40		0	1	0	0	0	0	0	0
~mask	191	0xBF		1	0	1	1	1	1	1	1
x & ~mask	149	0x95		1	0	0	1	0	1	0	1

BitSetToValue

It is easiest to use BitSetTo1 and BitSetTo0.

Alternatively you can use

BitSetToValue(x,position,state)

mask = BitMask(position)
return (x & ~mask) | (-state & mask)

$$\text{where state} = \begin{cases} 1 & \text{means set the bit to 1} \\ 0 & \text{means set the bit to 0} \end{cases}$$

Example
x = 0b11010101
position = 3
state = 1

Note that

state = 1 (decimal) = 0b00000001

So -state is simply -1 in decimal which is represented using two’s complement

-state = -1 (decimal) = 0b11111111

				Bit							
	Decimal	Hex		7	6	5	4	3	2	1	0
x	213	0xD5		1	1	0	1	0	1	0	1
mask	8	0x08		0	0	0	0	1	0	0	0
~mask	247	0xF7		1	1	1	1	0	1	1	1
x & ~mask	213	0xD5		1	1	0	1	0	1	0	1
state	1	0x01		0	0	0	0	0	0	0	1
-state	255	0xFF		1	1	1	1	1	1	1	1
-state & mask	0	0x08		0	0	0	0	1	0	0	0
(x & ~mask) | (-state & mask)	221	0xDD		1	1	0	1	1	1	0	1

Note that this appears overly complicated but this is because it needs to operate properly in all situations (such as when a bit is already 0 but is requested to be set to 0 again).

BitFlip

BitFlip(x,position)

mask = BitMask(position)
return x ^ mask

Example
x = 0b11010101
position = 2

				Bit							
	Decimal	Hex		7	6	5	4	3	2	1	0
x	213	0xD5		1	1	0	1	0	1	0	1
mask	4	0x04		0	0	0	0	0	1	0	0
x ^ mask	209	0xD1		1	1	0	1	0	0	0	1

BitIs1

IsBit1(x,position)

shifted = x >> position
return shifted & 1

Note that 1 is simply 0b00000001.

Furthermore, note that x can be either a signed or unsigned value. This will change the behavior of the bits that are shifted in on the left but in the end, it will not affect the correctness of the algorithm.

Example

x = 0b11010101

position = 2

				Bit							
	Decimal	Hex		7	6	5	4	3	2	1	0
x	213	0xD5		1	1	0	1	0	1	0	1
shifted	53	0x35		X	X	1	1	0	1	0	1
1	1	0x01		0	0	0	0	0	0	0	1
shifted & 1	1	0x01		0	0	0	0	0	0	0	1

BitIs0

IsBit0(x,position)

return !BitIs1(x,position)   (NOTE: use logical not instead of bitwise not)

MATLAB and C Implementations

See the following GitHub repos for implementations of these functions in MATLAB and C

MATLAB Implementation - https://github.com/clum/MatlabLum

C Implementation - https://github.com/clum/LumCSDK