Section 2: Integer & Floating Point Numbers

- Representation of integers: unsigned and signed
- Unsigned and signed integers in C
- <u>Arithmetic and shifting</u>
- <u>Sign extension</u>

- Background: fractional binary numbers
- IEEE floating-point standard
- Floating-point operations and rounding
- Floating-point in C

Shift Operations for unsigned integers

- **Left shift: x << y**
  - Shift bit-vector x left by y positions
    - Throw away extra bits on left
    - Fill with 0s on right
- **Right shift: x >> y**
  - Shift bit-vector x right by y positions
    - Throw away extra bits on right
    - Fill with 0s on left

x	00000110
<< 3	00110*000*
>> 2	*00*000001

x	11110010
<< 3	10010*000*
>> 2	*00*111100

Shift Operations for signed integers

- **Left shift: x << y**
  - Equivalent to multiplying by 2^y
  - (if resulting value fits, no 1s are lost)
- **Right shift: x >> y**
  - Logical shift (for unsigned values)
    - Fill with 0s on left
  - Arithmetic shift (for signed values)
    - Replicate most significant bit on left
    - Maintains sign of x
  - Equivalent to dividing by 2^y
    - Correct rounding (towards 0) requires some care with signed numbers

x	01100010
<< 3	00010*000*
Logical >> 2	*00*011000
Arithmetic >> 2	*00*011000

x	10100010
<< 3	00010*000*
Logical >> 2	*00*101000
Arithmetic >> 2	*11*101000

Undefined behavior when y < 0 or y ≥ word_size

Using Shifts and Masks

- **Extract the 2nd most significant byte of an integer:**
  - First shift, then mask: (x >> 16) & 0xFF

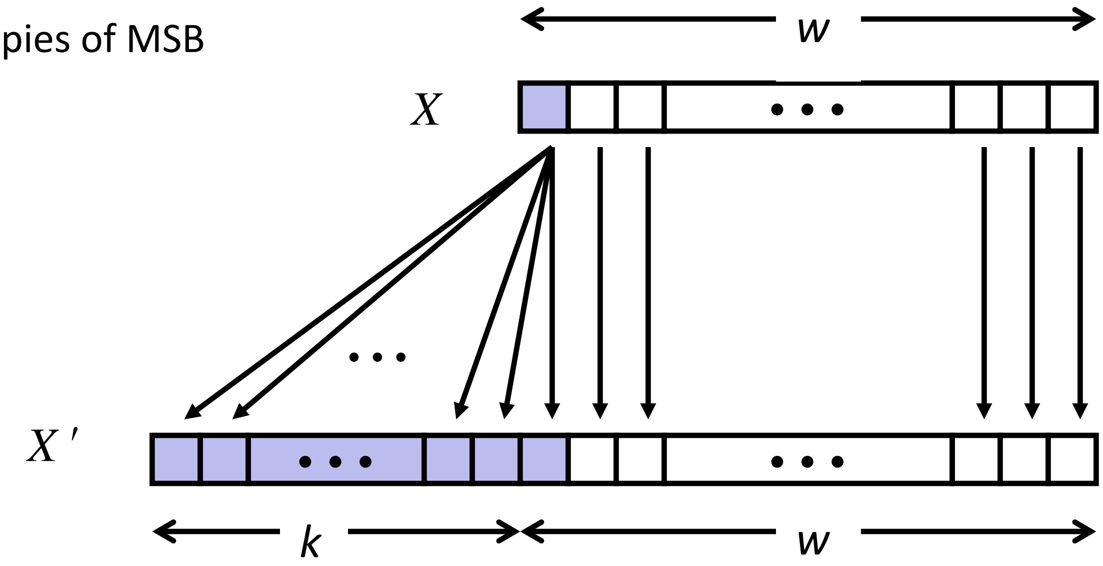
x	01100001 01100010 01100011 01100100
x >> 16	00000000 00000000 01100001 01100010
(x >> 16) & 0xFF	*00000000 00000000 00000000 11111111* 00000000 00000000 00000000 01100010

- **Extract the sign bit of a signed integer:**
  - (x >> 31) & 1 - need the "& 1" to clear out all other bits except LSB
- **Conditionals as Boolean expressions (assuming x is 0 or 1)**
  - if (x) a=y else a=z; which is the same as a = x ? y : z;
  - Can be re-written (assuming arithmetic right shift) as:

    a = ((x << 31) >> 31) & y + ((!x) << 31) >> 31) & z;

Sign Extension

- **Task:**
  - Given w-bit signed integer x
  - Convert it to w+k-bit integer *with same value*
- **Rule:**
  - Make k copies of sign bit:
  - $X' = \underbrace{x_{w-1},\ldots, x_{w-1}}_{k \text{ copies of MSB}}, x_{w-1}, x_{w-2},\ldots, x_0$



Sign Extension Example

- Converting from smaller to larger integer data type
- C automatically performs sign extension

```
short int x =  12345;
int      ix = (int) x;
short int y = -12345;
int      iy = (int) y;
```

	Decimal	Hex	Binary
x	12345	30 39	00110000 01101101
ix	12345	00 00 30 39	00000000 00000000 00110000 01101101
y	-12345	CF C7	11001111 11000111
iy	-12345	FF FF CF C7	11111111 11111111 11001111 11000111