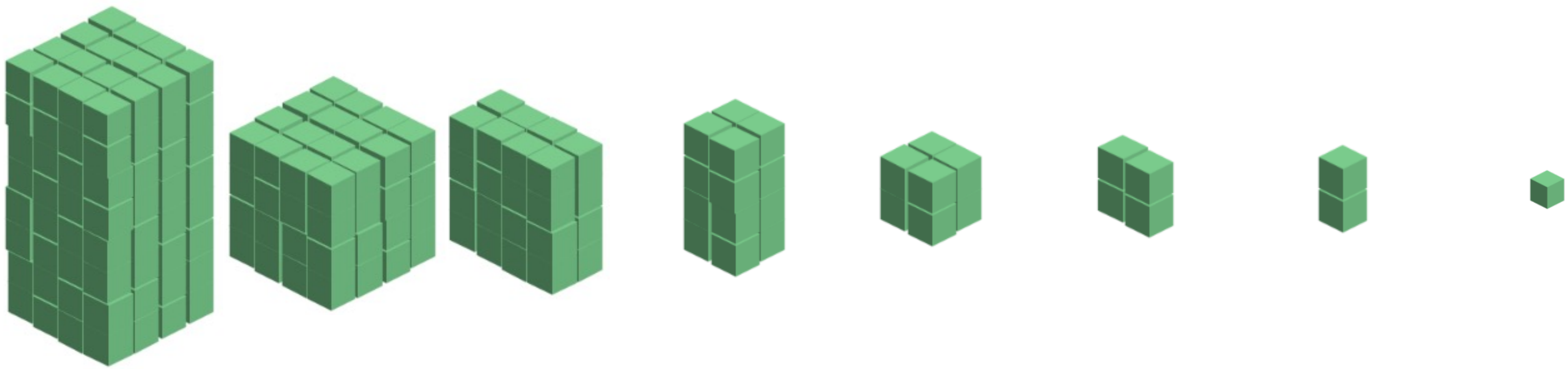
Lecture 3: Representing Signed Integers

CS 105 Fall 2023

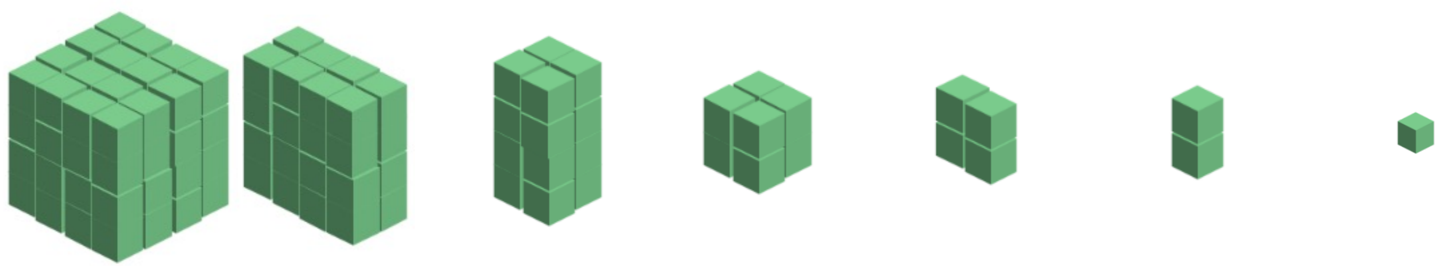
Review: Binary Numbers



128 (2^7)	64 (2^6)	32 (2^5)	16 (2^4)	8 (2^3)	4 (2^2)	2 (2^1)	1 (2^0)
0	0	0	0	0	1	0	1
0	0	1	0	1	1	1	1
1	1	1	1	1	1	1	1

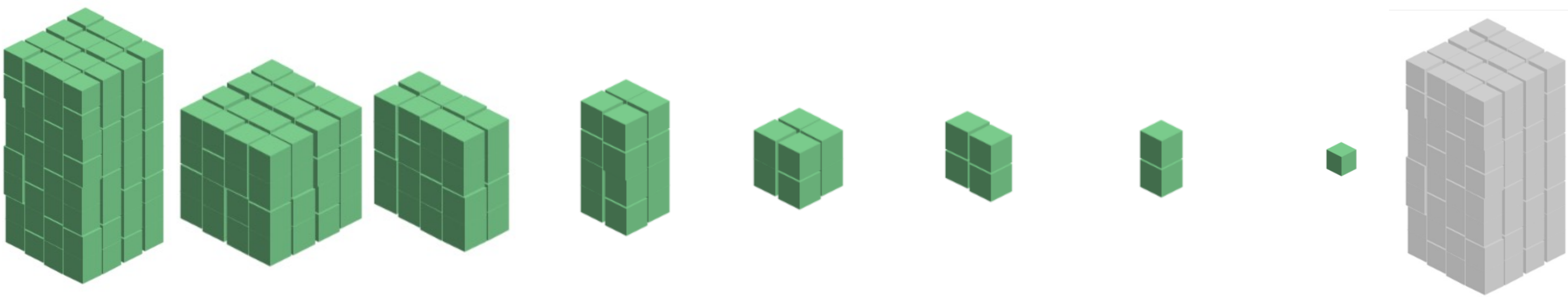
Representing Signed Integers

- Option 1: sign-magnitude
  - One bit for sign; interpret rest as magnitude
  - $Signed(x) = (-1)^{x_{w-1}} \cdot \sum_{i=0}^{w-2} x_i \cdot 2^i$

+/-	64 (2^6)	32 (2^5)	16 (2^4)	8 (2^3)	4 (2^2)	2 (2^1)	1 (2^0)
–							
0	0	0	0	0	1	0	1
1	0	0	0	0	1	0	1
1	1	1	1	1	1	1	1

Representing Signed Integers

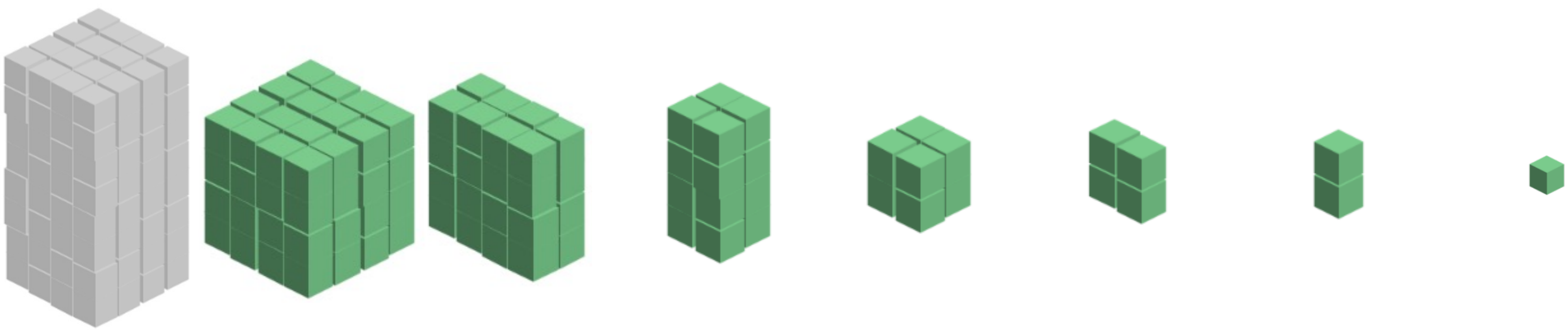
- Option 2: excess-K
  - Choose a positive K in the middle of the unsigned range
  - $Signed(x) = \sum_{i=0}^{w-1} x_i \cdot 2^i \ - 2^{w-1}$



128 (2^7)	64 (2^6)	32 (2^5)	16 (2^4)	8 (2^3)	4 (2^2)	2 (2^1)	1 (2^0)	-128
0	0	0	0	0	1	0	1	
1	0	0	0	0	1	0	1	
1	1	1	1	1	1	1	1	

Representing Signed Integers

- Option 3: two’s complement
  - Like unsigned, except the high-order contribution is *negative*
  - $Signed(x) = -x_{w-1} \cdot 2^{w-1} + \sum_{i=0}^{w-2} x_i \cdot 2^i$



-128 (-2^7)	64 (2^6)	32 (2^5)	16 (2^4)	8 (2^3)	4 (2^2)	2 (2^1)	1 (2^0)
0	0	0	0	0	1	0	1
1	0	0	0	0	1	0	1
1	1	1	1	1	1	1	1

Example: Three-bit integers

Base-10	unsigned	signed
7	111	
6	110	
5	101	
4	100	
3	011	011
2	010	010
1	001	001
0	000	000
-1		111
-2		110
-3		101
-4		100

- For signed ints:
  - high-order bit is 0 for pos values, 1 for neg
  - 000…0 is 0
  - 111…1 is -1
  - same representation as unsigned for numbers that can be represented with both
  - `~x+1 == -1*x`

Exercise 1: Signed Integers

Assume an 8 bit (1 byte) signed integer representation:

- What is the binary representation for 47?
- What is the binary representation for -47?
- What is the number represented by 10000110?
- What is the number represented by 00100101?

Casting between Numeric Types

- Casting from shorter to longer types preserves the value
- Casting from longer to shorter types drops the high-order bits
- Casting between signed/unsigned types preserves the bits (it just changes the interpretation)
- Implicit casting occurs in assignments and parameter lists. In mixed expressions, signed values are implicitly cast to unsigned
  - Source of many errors!

Exercise 2: Casting

- Assume you have a machine with 6-bit integers/3-bit shorts
- Assume variables: `int x = -17; short sy = -3;`
- Complete the following table

Expression	Decimal	Binary
x	-17	
sy	-3	
(unsigned int) x		
(int) sy		
(short) x		

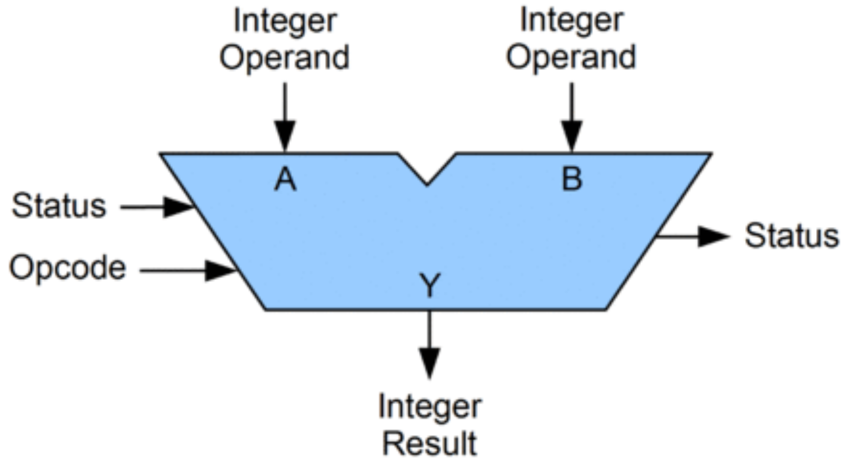
When to Use Unsigned

- Rarely
- When doing multi-precision arithmetic, or when you need an extra bit of range … but be careful!

```
for (unsigned int i = cnt-2; i >= 0; i--){
    a[i] += a[i+1];
}
```

Arithmetic Logic Unit (ALU)

- circuit that performs bitwise operations and arithmetic on integer binary types



Bitwise vs Logical Operations in C

- Bitwise Operators &, |, ~, ^
  - View arguments as bit vectors
  - operations applied bit-wise in parallel

- Logical Operators &&, ||, !
  - View 0 as “False”
  - View anything nonzero as “True”
  - Always return 0 or 1
  - Early termination

- Shift operators <<, >>
  - Left shift fills with zeros
  - For signed integers, right shift is arithmetic (fills with high-order bit)

Exercise 3: Bitwise vs Logical Operations

- What is the binary representation of each of the following expressions? Assume signed char data type (one byte).

1. `~(-30)`
2. `-30 & 23`
3. `-30 && 23`
4. `-30 >> 2`

Addition Example

- Compute 5 + -3 assuming all ints are stored as four-bit signed values

```
  1   1
  0 1 0 1
+ 1 1 0 1
---------
  0 0 1 0     = 2 (Base-10)
```

Exactly the same as unsigned numbers!

… but with different error cases

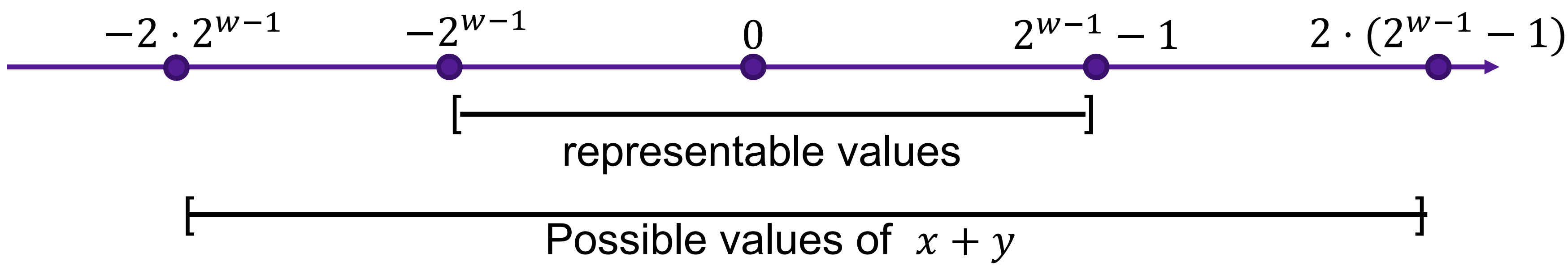
Addition/Subtraction with Overflow

- Compute 5 + 6 assuming all ints are stored as four-bit signed values

$$
\begin{array}{r}
{}^{1} \\
0101 \\
+\,0110 \\
\hline
1011
\end{array}
\quad = -5 \text{ (Base-10)}
$$

Error Cases

- Assume w-bit signed values



- $x +_w^t y = \begin{cases} x + y - 2^w & \text{(positive overflow)} \\ x + y & \text{(normal)} \\ x + y + 2^w & \text{(negative overflow)} \end{cases}$

- overflow has occurred iff $x > 0$ and $y > 0$ and $x +_w^t y < 0$
  or $x < 0$ and $y < 0$ and $x +_w^t y > 0$

Exercise 4: Binary Addition

- Given the following 5-bit signed values, compute their sum and indicate whether or not an overflow occurred

x	y	x+y	overflow?
00010	00101		
01100	00100		
10100	10001		

Multiplication Example

- Compute 3 x 2 assuming all ints are stored as four-bit signed values

$$
\begin{array}{r}
0\,0\,1\,1 \\
\times\; 0\,0\,1\,0 \\ \hline
0\,0\,0\,0 \\
+\; 0\,0\,1\,1\,0 \\ \hline
0\,1\,1\,0
\end{array}
\quad = 6 \text{ (Base-10)}
$$

Exactly like unsigned multiplication!

… except with different error cases

Multiplication Example

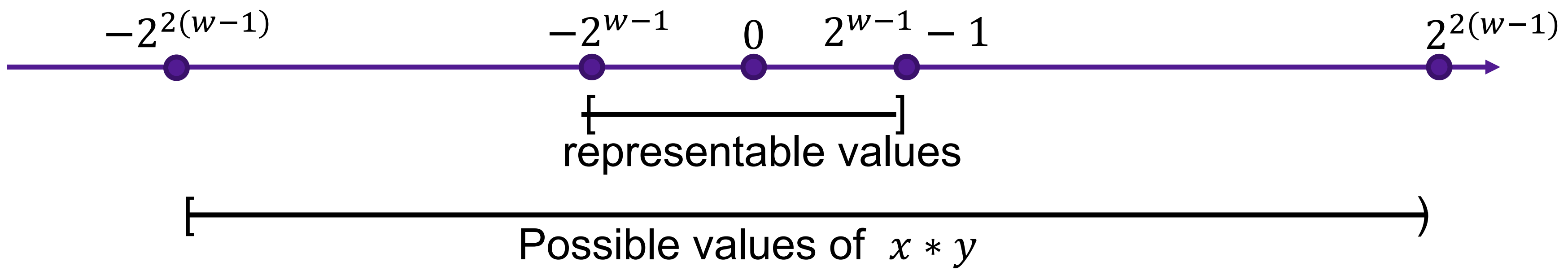
- Compute 5 x 2 assuming all ints are stored as four-bit signed values

$$
\begin{array}{r}
0101 \\
\times\ 0010 \\
\hline
0000 \\
+\ 01010 \\
\hline
1010
\end{array}
$$

= -6 (Base-10)

Error Cases

- Assume w-bit unsigned values



- $x *_w^t y = U2T((x \cdot y) \bmod 2^w)$

Exercise 5: Binary Multiplication

- Given the following 3-bit signed values, compute their product and indicate whether or not an overflow occurred

x	y	x*y	overflow?
100	101		
010	011		
111	010		