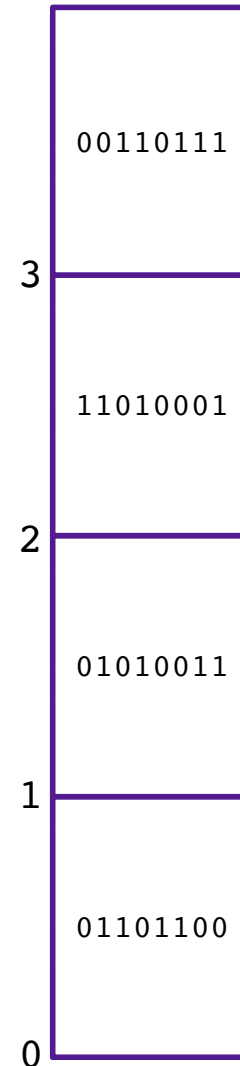


Lecture 3: Representing Signed Integers

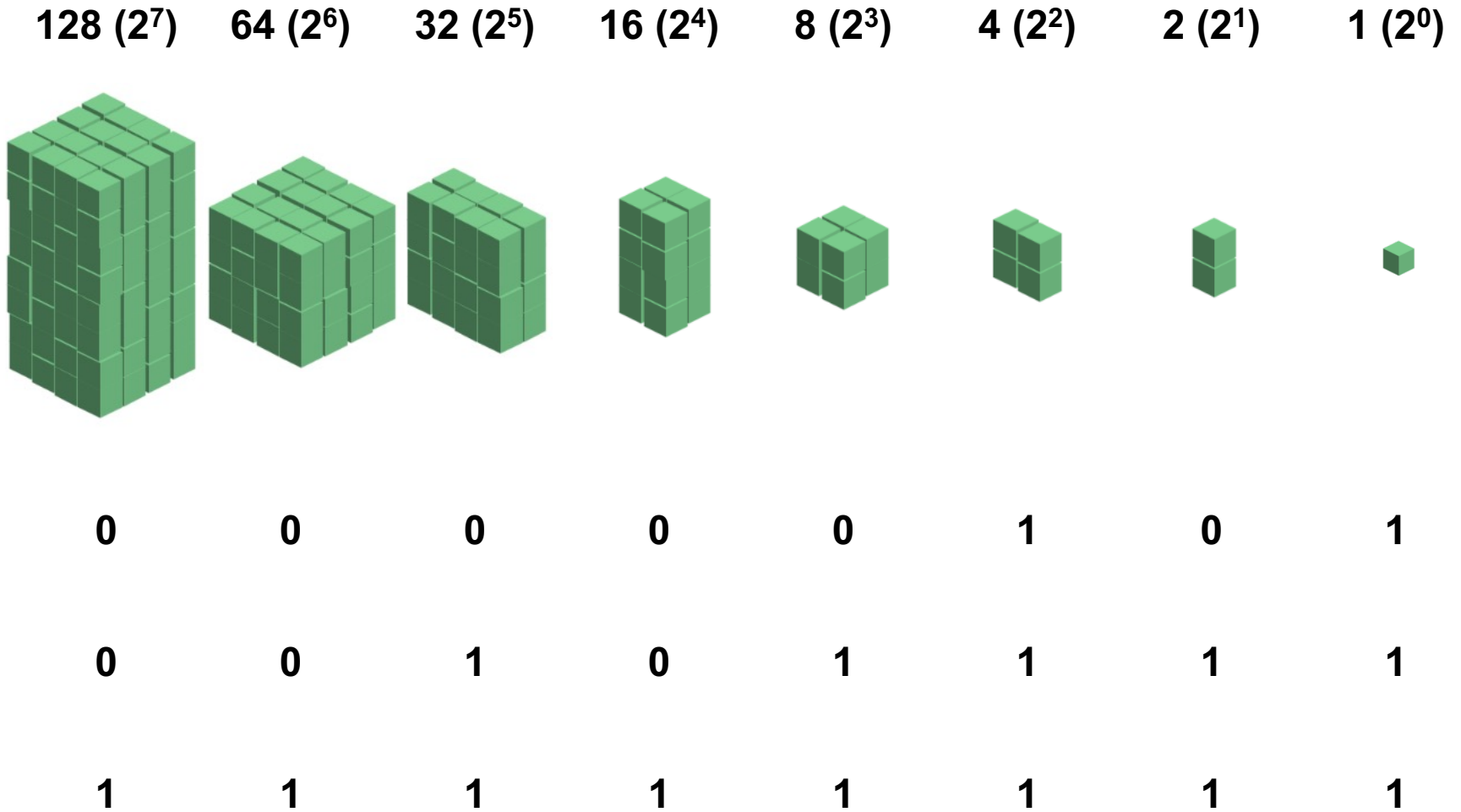
CS 105

Review: Memory

- **Memory** is an array of ~~bits~~^{bytes}
- A **byte** is a unit of eight bits
- An index into the array is an **address**, **location**, or **pointer**
 - Often expressed in hexadecimal
- We speak of the *value* in memory at an address
 - The value may be a single byte ...
 - ... or a multi-byte quantity starting at that address



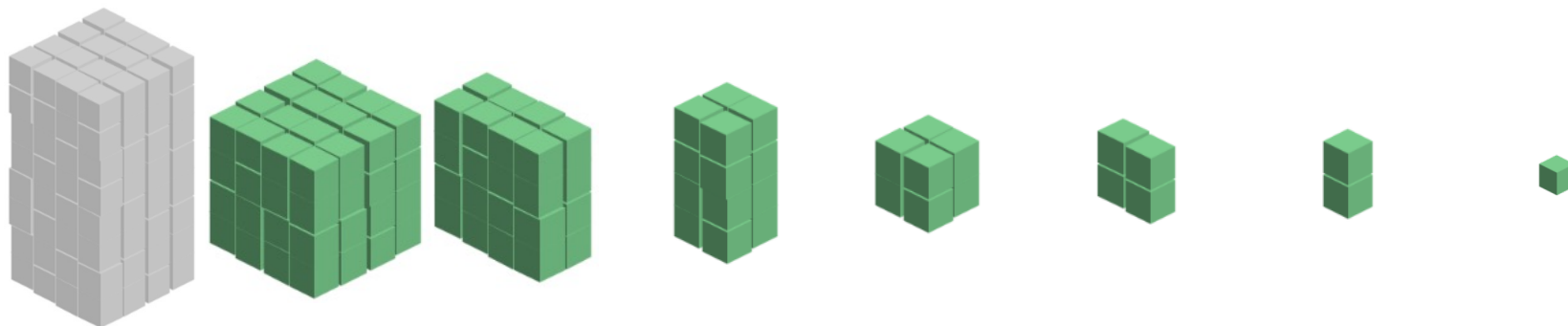
Review: Binary Numbers



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 (-2^6) 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

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

- The high-order bit is the *sign bit*.
- The largest unsigned value is $11 \dots 1$, UMax.
- The signed value for -1 is always $11 \dots 1$.
- Signed values range between TMin and TMax.

This representation of signed values is called *two's complement*.

Important Signed Numbers

	8	16	32	64
TMax	0x7F	0x7FFF	0x7FFFFFFF	0x7FFFFFFFFFFFFFFF
TMin	0x80	0x8000	0x80000000	0x8000000000000000
0	0x00	0x0000	0x00000000	0x0000000000000000
-1	0xFF	0xFFFF	0xFFFFFFFF	0xFFFFFFFFFFFFFFFF

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
<code>x</code>	-17	
<code>sy</code>	-3	
<code>(unsigned int) x</code>		
<code>(int) sy</code>		
<code>(short) x</code>		

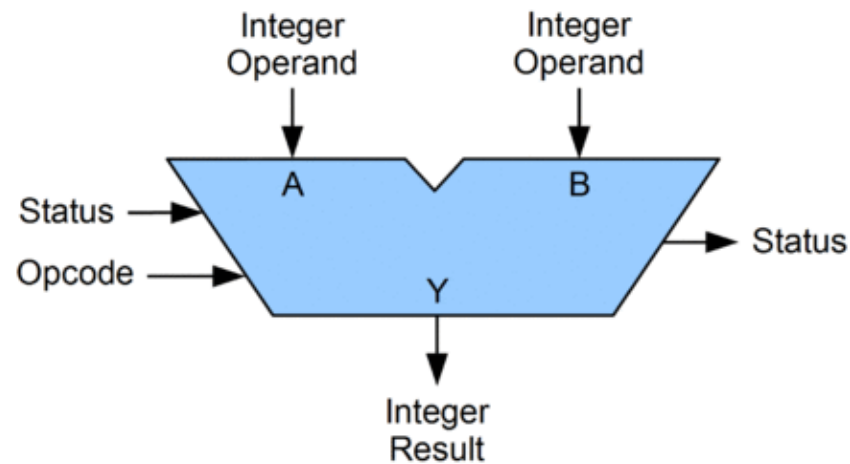
When to Use Unsigned

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

```
unsigned i;  
for (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 $\&$, $|$, \sim , \wedge
 - 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 \ll , \gg
 - Left shift fills with zeros
 - For signed integers, right shift is arithmetic (fills with high-order bit)

Exercise 3: Bitwise vs Logical Operations

- Assume signed char data type (one byte)
 - $\sim(-30)$
 - $!(-30)$

 - $-30 \ \& \ 21$
 - $-30 \ \&\& \ 21$

 - $-30 \ \ll \ 2$
 - $-30 \ \gg \ 2$
 - $21 \ \gg \ 2$

Addition Example

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

$$\begin{array}{r} 1 1 \\ 0 1 0 1 \\ + 1 1 0 1 \\ \hline 0 0 1 0 \end{array} = 2 \text{ (Base-10)}$$

Exactly the same as unsigned numbers!

... but with different error cases

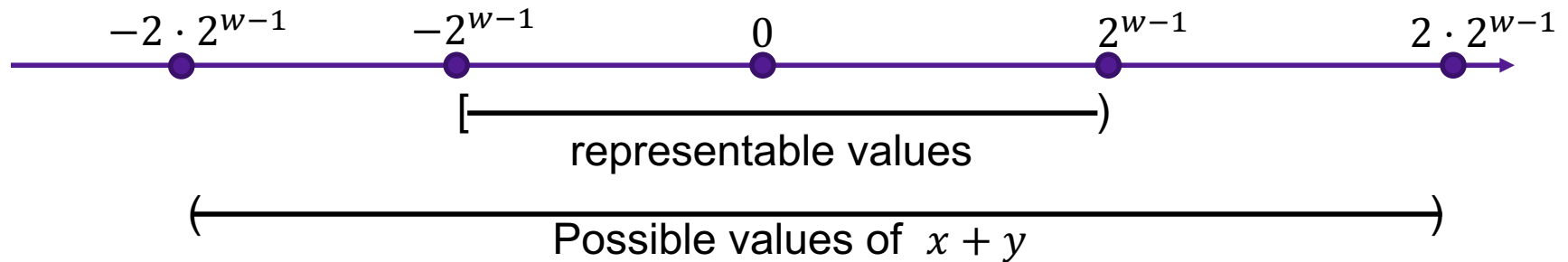
Addition/Subtraction with Overflow

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

$$\begin{array}{r} 111 \\ 0101 \\ + 0011 \\ \hline 1000 \end{array} = -8 \text{ (Base-10)}$$

Error Cases

- Assume w -bit signed values



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

- overflow has occurred iff $x > 0$ and $y > 0$ and $x + \overset{t}{w}y < 0$
or $x < 0$ and $y < 0$ and $x + \overset{t}{w}y > 0$

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

$$\begin{array}{r} 0011 \\ \times 0010 \\ \hline 0000 \\ + 00110 \\ \hline 0110 = 6 \text{ (Base-10)} \end{array}$$

Exactly like unsigned multiplication!
... except with different error cases

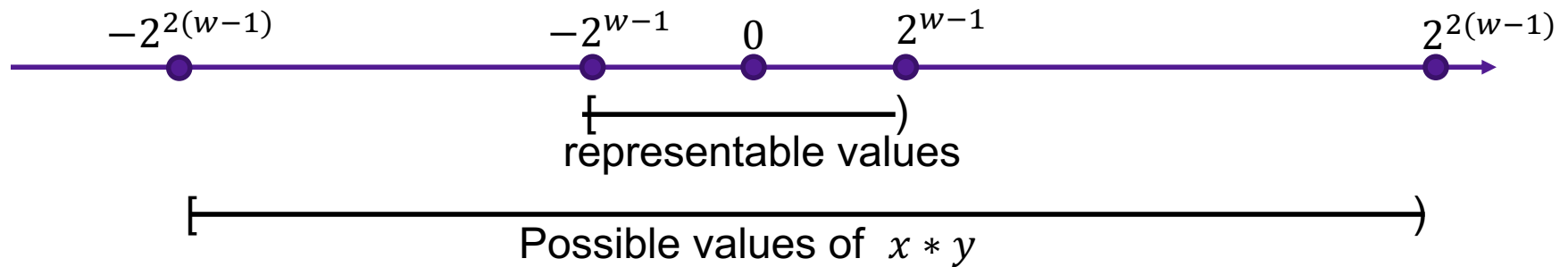
Multiplication Example

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

$$\begin{array}{r} 0101 \\ \times 0010 \\ \hline 0000 \\ + 01010 \\ \hline 1010 \end{array} = -6 \text{ (Base-10)}$$

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		