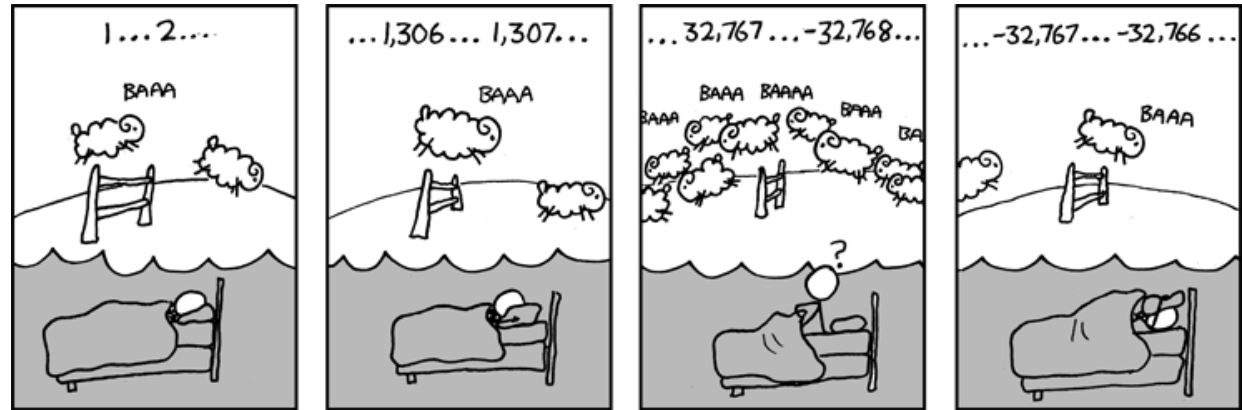


CS 261 Fall 2019

Mike Lam, Professor



<https://xkcd.com/571/>

Integer Encodings

Integers

- Topics
 - C integer data types
 - Unsigned encoding
 - Signed encodings
 - Conversions

Integer data types in C99

C data type	Minimum	Maximum	
[signed] char	-127	127	1 byte
unsigned char	0	255	
short	-32,767	32,767	2 bytes
unsigned short	0	65,535	
int	-32,767	32,767	2 bytes
unsigned	0	65,535	
long	-2,147,483,647	2,147,483,647	4 bytes
unsigned long	0	4,294,967,295	
int32_t	-2,147,483,648	2,147,483,647	4 bytes
uint32_t	0	4,294,967,295	
int64_t	-9,223,372,036,854,775,808	9,223,372,036,854,775,807	8 bytes
uint64_t	0	18,446,744,073,709,551,615	

Figure 2.11 Guaranteed ranges for C integral data types. The C standards require that the data types have at least these ranges of values.

Integer data types on stu

All sizes in bytes; sizes in red are larger than mandated by C99

char	1	int8_t	1
unsigned char	1	uint8_t	1
		bool	1
short	2		
unsigned short	2	int16_t	2
		uint16_t	2
int	4	int32_t	4
unsigned int	4	uint32_t	4
long	8	int64_t	8
unsigned long	8	uint64_t	8
long long	8	size_t	8
unsigned long long	8		

Unsigned integer encoding

- Bit i represents the value 2^i
 - Bits typically written from most to least significant (i.e., 2^3 2^2 2^1 2^0)
 - This is the same encoding we saw on Tuesday!
 - No representation of negative numbers

$$1 = 1 = 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = [0001]$$

$$5 = 4 + 1 = 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = [0101]$$

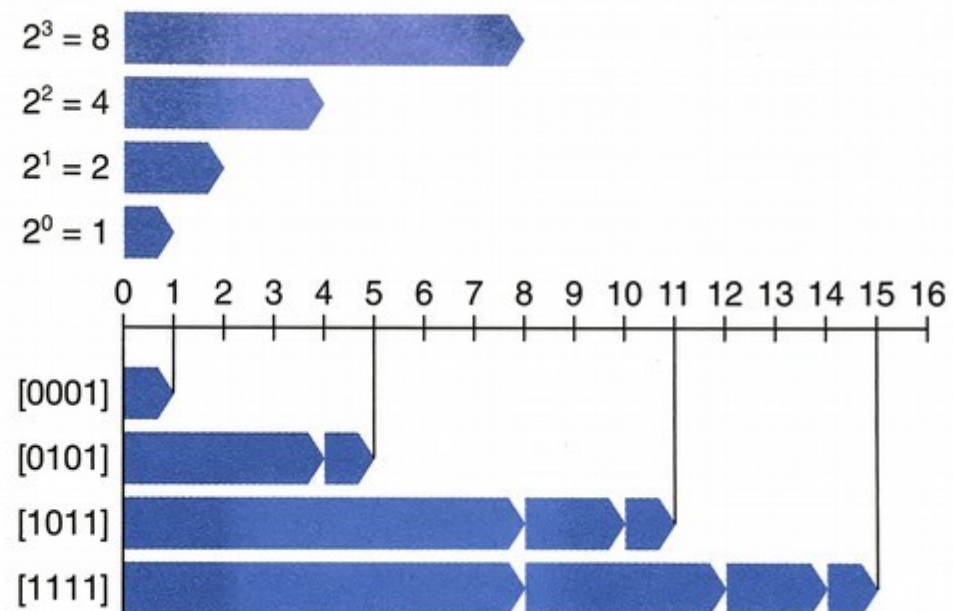
$$11 = 8 + 2 + 1 = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = [1011]$$

$$15 = 8 + 4 + 2 + 1 = 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = [1111]$$

Unsigned integer encoding

- Textbook's notation
 - Each bar represents a bit
 - Add together bars to represent the contributions of each bit value to the overall value

Figure 2.12
Unsigned number
examples for $w = 4$.
When bit i in the binary
representation has value 1,
it contributes 2^i to the
value.



Signed integer encodings

- Sign magnitude
 - Most natural/intuitive but hardest to implement
- Ones' complement
 - Cleaner arithmetic but less intuitive
- Two's complement
 - Cleanest arithmetic but most complicated
 - Most modern signed integer types use this!

Sign magnitude

- **Sign magnitude**

- Interpret most-significant bit as a **sign bit**
- Interpret remaining bits as unsigned number x (the **magnitude**)
 - If negative, absolute value is x
- To negate: **flip the sign bit**
- Disadvantages:
 - Two zeros: -0 and $+0$ [1000 and 0000]
 - Less useful for arithmetic because the sign bit has no relationship with the magnitude--cannot use unsigned arithmetic logic!

$$0\ 011 = 3$$

$$1\ 011 = -3$$

$$0\ 111 = 7$$

$$0\ 111\ (7)$$

$$\underline{1\ 011\ (-3)}$$

$$?\ 010$$

Ones' complement

- **Ones' complement**

- Interpret most-significant bit as a **sign bit**
- Interpret ALL bits as unsigned integer x
 - If negative, absolute value is $[11111\dots 1] - x$
- To negate: **flip all the bits** (binary NOT)
- Disadvantages:
 - Still have two representations of zero (1111 and 0000)
 - Also, less useful for arithmetic than two's complement
 - Must “end-around carry” to preserve results

$$\begin{array}{l} 0\ 011 = 3 \\ 1\ 100 = -3 \end{array}$$

$$0\ 111 = 7$$

$$\begin{array}{r} 1 \\ 0\ 111\ (7) \\ \hline 1\ 100\ (-3) \end{array}$$

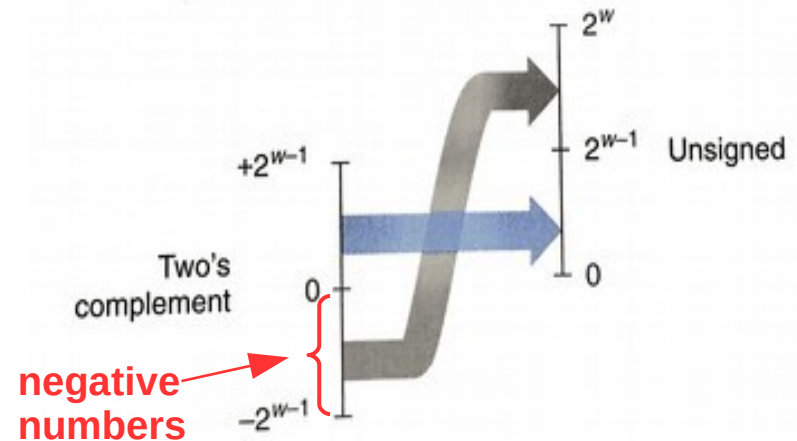
$$\begin{array}{r} 10\ 011 \\ \quad +1\ (end\text{-}around\ carry) \\ \hline 0\ 100 \end{array}$$

Two's complement

- **Two's complement**

- Interpret most-significant bit as a **sign bit**
- Interpret ALL bits as unsigned integer x
 - If negative, absolute value is $2^N - x$ where N is the number of bits
- To negate: **subtract value from 2^N** where N is the number of bits
- One zero; positive numbers wrap to negative ones halfway through

<u>2's Comp.</u>		<u>Unsigned</u>
-1	1111	15
	...	
-7	1001	9
-8	1000	8
7	0111	7
	...	
1	0001	1
0	0000	0



Two's complement

- **Two's complement** advantage: uses unsigned arithmetic logic
 - (ignore carries out of the sign bit for now)
 - Ex: $5 - 3 = 5 + (-3) = 0101 + 1101 = 0010$ (2)
 - Ex: $1 - 3 = 1 + (-3) = 0001 + 1101 = 1110$ (-2)
 - Ex: $-2 - 3 = (-2) + (-3) = 1110 + 1101 = 1011$ (-5)

$$\begin{array}{l} 0011 = 3 \\ 1100 \\ 1101 = -3 \end{array}$$

$$0111 = 7$$

$$\begin{array}{r} 0111 \ (7) \\ \underline{1101 \ (-3)} \\ 0100 \ (4) \end{array}$$

Two's complement

- Alternate interpretation: value of most significant bit is negated
 - i.e., start at most negative number and build back up towards zero

Figure 2.12
Unsigned number
examples for $w = 4$.
When bit i in the binary
representation has value 1,
it contributes 2^i to the
value.

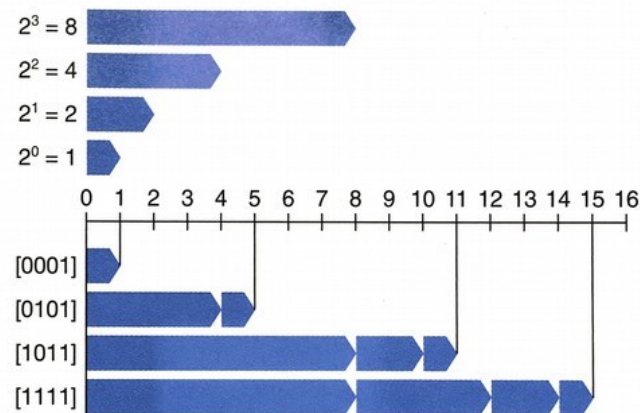
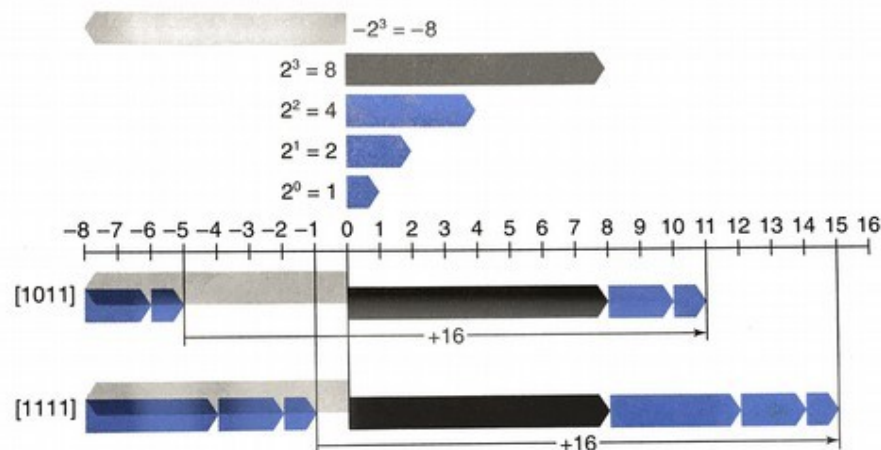


Figure 2.16
Comparing unsigned
and two's-complement
representations for $w = 4$.
The weight of the most
significant bit is -8 for
two's complement and $+8$
for unsigned, yielding a net
difference of 16.



Two's complement trick

- Alternate way to negate in **two's complement**
 - **Flip the bits** (binary NOT) then **add one**

Ex: $5 = 0101 \rightarrow$ (binary NOT) $\rightarrow 1010 \rightarrow$ (add one) $\rightarrow 1011 = -5$ ($-8 + 2 + 1$)

Aside: Why does this work? The sum of a number x and $\sim x$ is all ones (or $2^N - 1$ where N is the number of bits), so $\sim x$ can be expressed as $2^N - 1 - x$. Because negating x in two's complement is equivalent to subtracting x from 2^N , if we add one to $\sim x$ the results are equal:

$$\sim x + 1 = (2^N - 1 - x) + 1 = 2^N - x$$

Ones' vs. Two's

- **Ones' complement**
 - Interpret all bits as unsigned integer x
 - Value is $[11111\dots1] - x$
 - I.e., the complement with respect to ones
- **Two's complement**
 - Interpret all bits as unsigned integer x
 - Value is $2^N - x$ where N is the number of bits
 - I.e., the complement with respect to a power of two

Caution: language technicalities

- **Ones' complement** and **two's complement** are both an **operation** and an **encoding**
 - E.g., “perform two's complement” vs “the number is stored in two's complement”
- The operation represents the action necessary to **negate** a number in that encoding.
 - E.g., performing two's complement (ones' complement and add one) negates a number in two's complement encoding
- If you have a value in a particular encoding:
 - If the sign bit is not set, it's a positive number
 - If it is set, perform the operation to recover the positive value

We will avoid using the operation terminology in this course!

Integer encodings

- Information = Bits + Context
 - What does “1011” mean? **It depends!**

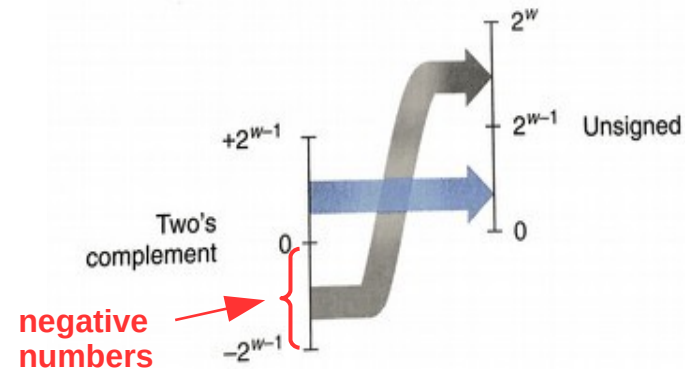
Unsigned:	11
Sign magnitude:	-3
Ones' complement:	-4
Two's complement:	-5

Comparison

- We'll see one more signed integer encoding next week: “offset binary” / “biased” / “excess”
 - For now, here's a comparison (for 1-byte integers):

<u>Binary</u> _____	<u>Unsigned</u>	<u>Sign Mag</u>	<u>Ones' C</u>	<u>Two's C</u>	<u>Offset -127</u>
1111 1111	255	-127	-0	-1	128
1111 1110	254	-126	-1	-2	127
...
1000 0001	129	-1	-126	-127	2
1000 0000	128	-0	-127	-128	1
<hr/>					
0111 1111	127	127	127	127	0
0111 1110	126	126	126	126	-1
...
0000 0001	1	1	1	1	-126
0000 0000	0	0	0	0	-127

Comparison



<u>Binary</u> _____	<u>Unsigned</u>	<u>Sign Mag</u>	<u>Ones' C</u>	<u>Two's C</u>	<u>Offset -127</u>
1111 1111	255	-127	-0	-1	128
1111 1110	254	-126	-1	-2	127
...
1000 0001	129	-1	-126	-127	2
1000 0000	128	-0	-127	-128	1
<hr/>					
0111 1111	127	127	127	127	0
0111 1110	126	126	126	126	-1
...
0000 0001	1	1	1	1	-126
0000 0000	0	0	0	0	-127

Conversions

- Smaller unsigned → larger unsigned
– Safe; **zero-extend** to preserve value
 $0101 (5) \rightarrow 0000 0101 (5)$
- Smaller two's comp. → larger two's comp.
– Safe; **sign-extend** to preserve value
 $\underline{1}101 (-3) \rightarrow \underline{1}111 1101 (-3)$
- Larger → smaller (unsigned or two's comp.)
– **Overflow** if new type isn't large enough to fit (truncate)
 $\begin{array}{l} 0000 0101 (5) \rightarrow 0101 (5) \\ 0011 0101 (53) \rightarrow 0101 (5) \end{array}$
- Unsigned → two's comp.
– **Overflow** if first bit is non-zero (otherwise, no change)
 $\begin{array}{l} 0101 (5) \rightarrow 0101 (5) \\ 1101 (13) \rightarrow \underline{1}101 (-3) \end{array}$
- Two's comp. → unsigned
– **Overflow** if value is negative (otherwise, no change)
 $\begin{array}{l} \underline{0}101 (5) \rightarrow 0101 (5) \\ \underline{1}101 (-2) \rightarrow 1101 (13) \end{array}$