# CS 261
# Fall 2016

Mike Lam, Professor

# Integer Encodings

# Integers

- Topics
  - C integer data types
  - Unsigned encoding
  - Two’s-complement encoding
  - Conversions
  - Alternative encodings

# 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

| | |
|---|---|
| char | 1 |
| unsigned char | 1 |
| short | 2 |
| unsigned short | 2 |
| int | 4 |
| unsigned int | 4 |
| long | 8 |
| unsigned long | 8 |
| long long | 8 |
| unsigned long long | 8 |

| | |
|---|---|
| int8_t | 1 |
| uint8_t | 1 |
| int16_t | 2 |
| uint16_t | 2 |
| int32_t | 4 |
| uint32_t | 4 |
| int64_t | 8 |
| uint64_t | 8 |

# Unsigned 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 Wednesday!

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

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

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

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

**Binary to decimal:**

Add up all the powers of two (memorize powers of two to make this go faster!)

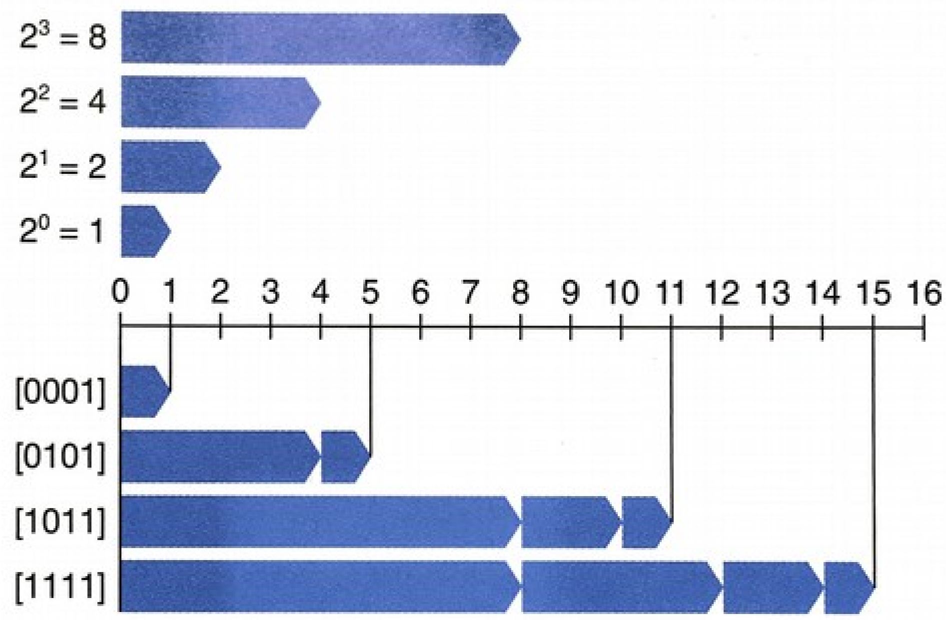
**Decimal to binary:**

Find highest power of two and subtract to find the remainder
Repeat above until the remainder is zero
Every power of two that was used becomes a 1; all other bits are 0

# Unsigned 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.



# Two’s complement encoding

- Value of most significant bit is negated
  - Essentially, this makes half of all representable values negative

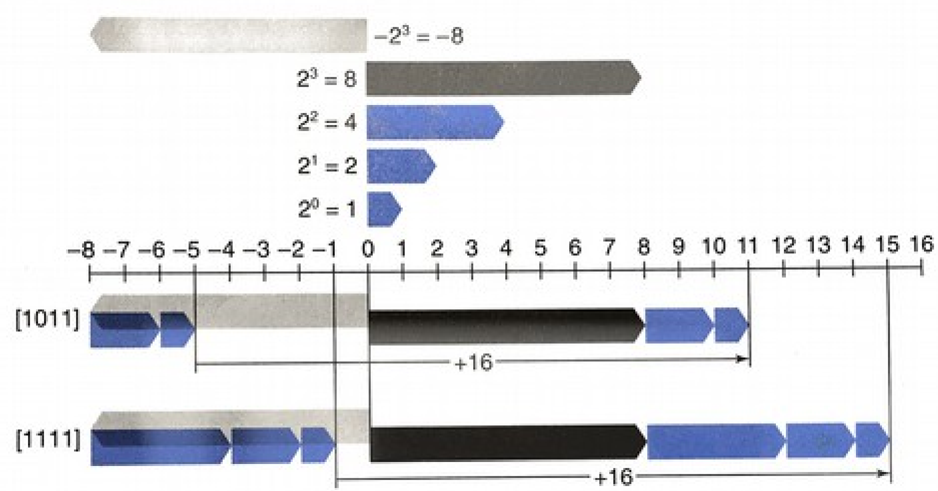



**Figure 2.16**
**Comparing unsigned and two’s-complement representations for $w = 4$.** The weight of the most significant bit is $-8$ for two’s complement and $+8$ for unsigned, yielding a net difference of 16.

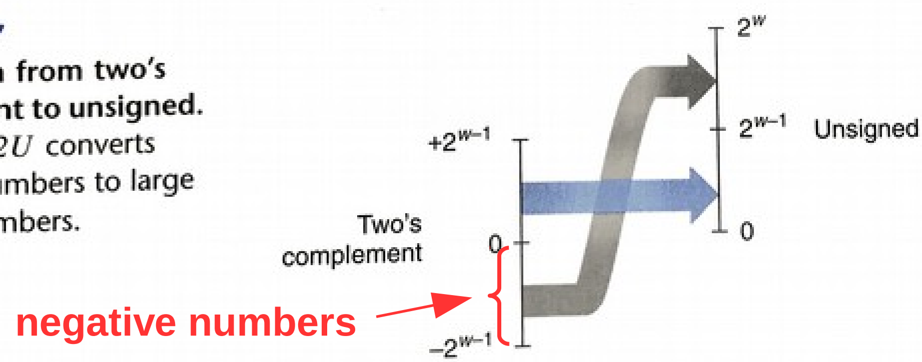



**Figure 2.17**
**Conversion from two’s complement to unsigned.** Function $T2U$ converts negative numbers to large positive numbers.

# Conversions

- Smaller unsigned → larger unsigned
  - Safe; zero-extend to preserve value

`0101 (5) → 0000 0101 (5)`

- Smaller two's comp. → larger two's comp.
  - Safe; sign-extend to preserve value

`1101 (-3) → 1111 1101 (-3)`

- Larger → smaller (unsigned or two's comp.)
  - Overflow if new type isn't large enough to fit (otherwise, truncate)

`0000 0101 (5) → 0101 (5)`
`0011 0101 (53) → 0101 (5)`

- Unsigned → two's comp.
  - Overflow if first bit is non-zero (otherwise, no change)

`0101 (5) → 0101 (5)`
`1101 (13) → 1101 (-2)`

- Two's comp. → unsigned
  - Overflow if value is negative (otherwise, no change)

`0101 (5) → 0101 (5)`
`1101 (-2) → 1101 (13)`

# Two's complement encoding

- Taking the two's complement is equivalent to subtracting the number from $2^N$, where N is the number of bits in the integer
- Advantage: can use arithmetic as usual
  - 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)

# Other encodings

- Sign magnitude
  - Interpret most-significant bit as a sign bit
  - Interpret remaining bits as a normal unsigned int
  - Disadvantages:
    - Two zeros: -0 and +0  [1000 and 0000]
    - Less useful for arithmetic

# Other encodings

- Ones' complement
  - Invert all the bits for negative numbers
  - Less useful for arithmetic than two's complement
  - However, it enables a neat trick: to perform two's complement, just do one's complement then add one

Ex: 5 = 0101 → (one's comp.) → 1010 → (add one) → 1011 = -5 (-8 + 2 + 1)

***Aside***: Why does this work? The sum of a number and it's ones' complement is all ones (or $2^N-1$ where N is the number of bits). Because taking the two's complement of x is equivalent to subtracting x from $2^N$, the results are equal:

$2^N-1 - x + 1 = 2^N-x$