Data Representation

Outline

- Introduction
- Numbering Systems
- Binary & Hexadecimal Numbers
- Base Conversions
- Integer Storage Sizes
- Binary and Hexadecimal Addition
- Signed Integers and 2's Complement Notation
- Binary and Hexadecimal subtraction
- Carry and Overflow

Introduction

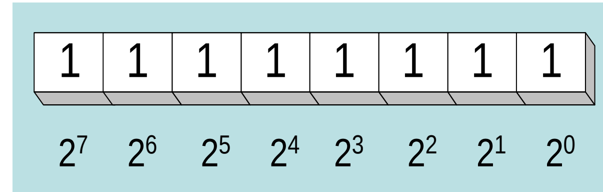
- Computers only deal with binary data (0s and 1s).
- Many different forms of data:
  - **Numbers**: 2 Types
    - Integers: 33, +128, -2827
    - Real numbers: 1.33, +9.55609, -6.76E12, +4.33E-03
  - **Alphanumeric characters** (letters, numbers, signs, control characters): examples: A, a, c, 1 ,3, ", +, Ctrl, Shift, etc.
  - **Images** (still or moving): Usually represented by numbers representing the Red, Green and Blue (RGB) colors of each pixel in an image,
  - **Sounds**: Numbers representing sound amplitudes sampled at a certain rate (usually 20kHz).
- So in general we have two major data types that need to be represented in computers; numbers and characters.

Numbering Systems

Numbering System	Base	Digits Set
Binary	2	1 0
Octal	8	7 6 5 4 3 2 1 0
Decimal	10	9 8 7 6 5 4 3 2 1 0
Hexadecimal	16	F E D C B A 9 8 7 6 5 4 3 2 1 0

Binary Numbers

- Each digit (bit) is either 1 or 0
- Each bit represents a power of 2
- Every binary number is a sum of powers of 2



Same concept about
Decimal and
Hexadecimal numbers

Conversions

Rule 1

- ❖ Decimal → Any other Base (DIVISION Method) BUT
  - ✧ Decimal Fraction (Multiplication way)
- ❖ Any other Base → Decimal (MULTIPLICATION by Weights Method) Also
  - ✧ Other Base Fraction (Multiplication way)

Note: Decimal → Other Base (Single Division Case)

Converting Binary to Decimal

Multiplication

$decimal = (d_{n-1} \times 2^{n-1}) + (d_{n-2} \times 2^{n-2}) + \ldots + (d_1 \times 2^1) + (d_0 \times 2^0)$

d = binary digit

❖ binary 10101001 = decimal 169:

$(1 \times 2^7) + (1 \times 2^5) + (1 \times 2^3) + (1 \times 2^0) = 128+32+8+1=169$

Convert Unsigned Decimal to Binary

❖ Division:

Division	Quotient	Remainder
37 / 2	18	1
18 / 2	9	0
9 / 2	4	1
4 / 2	2	0
2 / 2	1	0
1 / 2	0	1

least significant bit

most significant bit

stop when quotient is zero

37 = 100101

Converting from Decimal fractions to Binary

- **Using the multiplication method to convert the decimal 0.8125 to binary, we multiply by the radix 2.**
  - The first product carries into the units place.

$$\begin{array}{r} .8125 \\ \times \quad 2 \\ \hline 1.6250 \end{array}$$

Converting from Decimal fractions to Binary

- **Converting 0.8125 to binary . . .**
  - You are finished when the product is zero, or until you have reached the desired number of binary places.
  - Our result, reading from top to bottom is:

    $$0.8125_{10} = 0.1101_2$$

  - This method also works with any base. Just use the target radix as the multiplier.

$$\begin{array}{r}
.8125 \\
\times \quad 2 \\
\hline
1.6250
\end{array}$$

$$\begin{array}{r}
.6250 \\
\times \quad 2 \\
\hline
1.2500
\end{array}$$

$$\begin{array}{r}
.2500 \\
\times \quad 2 \\
\hline
0.5000
\end{array}$$

$$\begin{array}{r}
.5000 \\
\times \quad 2 \\
\hline
1.0000
\end{array}$$

Brain storming?

- Binary number with n-bits can represent how many unsigned integers?
- Which method will be used for
  - Hexadecimal to Decimal
  - Decimal to Hexadecimal
  - Decimal fraction to Hexadecimal

  - Hexadecimal to Binary
  - Binary to Hexadecimal

Direct conversion is possible

Hexadecimal Integers

- Binary values are represented in hexadecimal.

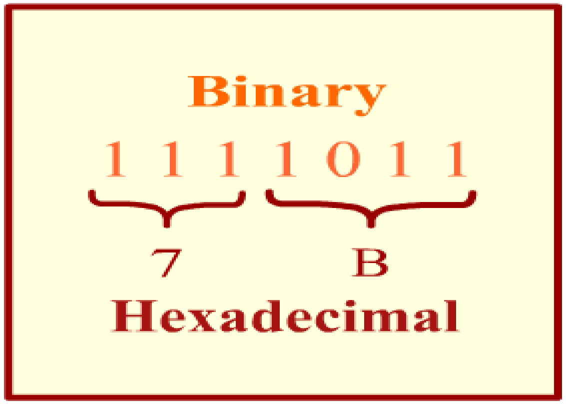
Table 1-5 Binary, Decimal, and Hexadecimal Equivalents.

Binary	Decimal	Hexadecimal	Binary	Decimal	Hexadecimal
0000	0	0	1000	8	8
0001	1	1	1001	9	9
0010	2	2	1010	10	A
0011	3	3	1011	11	B
0100	4	4	1100	12	C
0101	5	5	1101	13	D
0110	6	6	1110	14	E
0111	7	7	1111	15	F

Converting Binary to Hexadecimal

- <u>DIRECT COVERSION.</u>
- Example: Translate the binary integer 000101101010011110010100 to hexadecimal

1	6	A	7	9	4
0001	0110	1010	0111	1001	0100



Converting Binary → Decimal

Bit “0”

101011_2 =>

$$
\begin{array}{rcl}
1 \times 2^0 & = & 1 \\
1 \times 2^1 & = & 2 \\
0 \times 2^2 & = & 0 \\
1 \times 2^3 & = & 8 \\
0 \times 2^4 & = & 0 \\
1 \times 2^5 & = & 32 \\
\hline
& & 43_{10}
\end{array}
$$

Converting Hexadecimal → Decimal

$$
\begin{array}{lrcrcrcr}
ABC_{16} \Rightarrow & C \times 16^0 & = & 12 & \times & 1 & = & 12 \\
 & B \times 16^1 & = & 11 & \times & 16 & = & 176 \\
 & A \times 16^2 & = & 10 & \times & 256 & = & 2560 \\
\hline
 & & & & & & & 2748_{10}
\end{array}
$$

Decimal → Binary

$125_{10} = ?_2$

2	125	
2	62	1
2	31	0
2	15	1
2	7	1
2	3	1
2	1	1
	0	1

$125_{10} = 1111101_2$

Decimal → Hexadecimal

$1234_{10} = ?_{16}$

16	1234	
16	77	2
16	4	13 = D
	0	4

$1234_{10} = 4D2_{16}$

Hexadecimal → Binary

$10AF_{16} = ?_2$

1	0	A	F
0001	0000	1010	1111

$10AF_{16} = 0001000010101111_2$

Binary → Hexadecimal

$1010111011_2 = ?_{16}$

10 → 2

1011 → B

1011 → B

$1010111011_2 = 2BB_{16}$

Exercise – Convert ...

Decimal	Binary	Hexa-decimal
33		
	1110101	
		1AF

Don’t use a calculator!

Exercise – Convert …

Answer

Decimal	Binary	Hexa-decimal
33	100001	21
117	1110101	75
451	111000011	1C3
431	110101111	1AF

Fractions

❖ Binary to decimal

$$
10.1011 \Rightarrow \quad
\begin{array}{l}
1 \times 2^{-4} = 0.0625 \\
1 \times 2^{-3} = 0.125 \\
0 \times 2^{-2} = 0.0 \\
1 \times 2^{-1} = 0.5 \\
0 \times 2^{0} = 0.0 \\
1 \times 2^{1} = 2.0 \\
\hline
\qquad\quad 2.6875
\end{array}
$$

Fractions

- Decimal to binary

3.14579

```
  .14579
x      2
--------
0.29158
x      2
--------
0.58316
x      2
--------
1.16632
x      2
--------
0.33264
x      2
--------
0.66528
x      2
--------
1.33056

etc.
```

11.001001...

Exercise – Convert ...

Decimal	Binary	Hexa-decimal
29.8		
	101.1101	
		C.82

Don't use a calculator!

Exercise – Convert …

Answer

Decimal	Binary	Hexa-decimal
29.8	11101.110011…	1D.CC…
5.8125	101.1101	5.D
3.109375	11.000111	3.1C
12.5078125	1100.10000010	C.82

Integer Storage Sizes

Standard sizes:

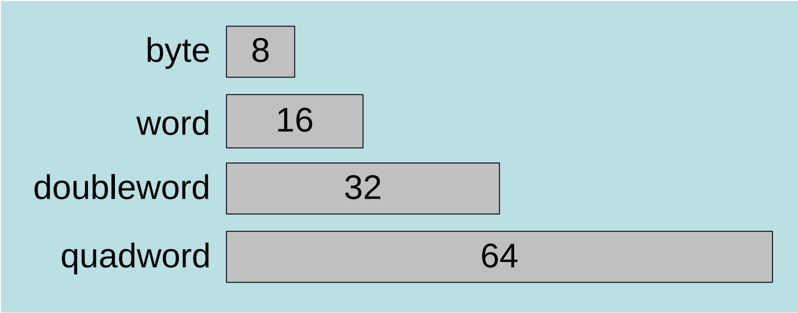
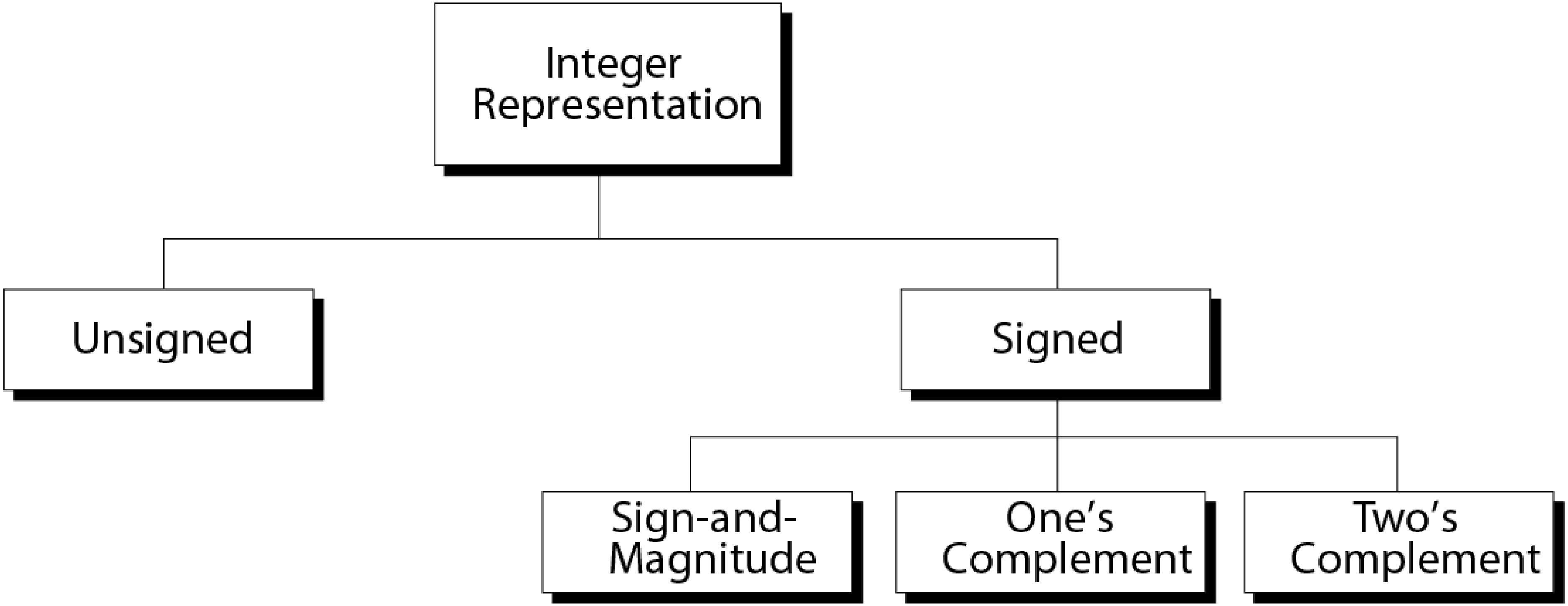


Table 1-4 Ranges of Unsigned Integers.

Storage Type	Range (low–high)	Powers of 2
Unsigned byte	0 to 255	0 to $(2^8 - 1)$
Unsigned word	0 to 65,535	0 to $(2^{16} - 1)$
Unsigned doubleword	0 to 4,294,967,295	0 to $(2^{32} - 1)$
Unsigned quadword	0 to 18,446,744,073,709,551,615	0 to $(2^{64} - 1)$

Taxonomy of integers



Signed Integer Representation

- There are three ways in which signed binary numbers may be expressed:
  - Signed magnitude,
  - One’s complement and
  - Two’s complement.
- In an 8-bit word, signed magnitude representation places the absolute value of the number in the 7 bits to the right of the sign bit.

Signed Magnitude

- For example, in 8-bit signed magnitude,
- positive 3 is: `00000011`
- Negative 3 is: `10000011`
- `Arithmetic human-like`

Signed Magnitude

❖ Example:

✧ 75 + 46 = ?

```
0     1 0 0 1 0 1 1
0  +  0 1 0 1 1 1 0
      -------------
```

Signed Magnitude

- Example:
  - Using signed magnitude binary arithmetic, find the sum of 75 and 46.

```
          1 1 1
0     1 0 0 1 0 1 1
0  +  0 1 0 1 1 1 0
   ----------------
0     1 1 1 1 0 0 1
```

In this example, we were careful careful to pick two values whose sum would fit into seven bits. If that is not the case, we have a problem→Overflow

Overflow→ occurs when sign is not correct

(what it means. is it correct sayings?)

Signed Magnitude

- Example:
  - 107 +46 =?

- What u get?

Signed Magnitude

- Example:
  - 107 +46 =?

$$\begin{array}{cccccccccc}
1 & & & & & & & & & \\
 & & & 1 & & 1 & 1 & 1 & & \\
 & 0 & & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\
 & 0 & + & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\
\hline
 & 0 & & 0 & 0 & 1 & 1 & 0 & 0 & 1
\end{array}$$

- We see that the carry from the seventh bit *overflows* and is discarded, giving us the erroneous result: 107 + 46 = 25.

Signed Magnitude

❖ -46 - 25

```
       1 1
1    0101110
1  + 0011001
   ---------
1    1000111
```

- Because the signs are the same, all we do is add the numbers and supply the negative sign when we are done.

Signed Magnitude

- ❖ 46-25

$$
\begin{array}{ccccccccc}
 & & & 0 & 2 & & & 0 & 2 \\
0 & & 0 & \cancel{1} & 0 & 1 & 1 & \cancel{1} & 0 \\
1 & + & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\
\hline
0 & & 0 & 0 & 1 & 0 & 1 & 0 & 1
\end{array}
$$

- The sign of the result gets the sign of the number that is larger.
  - Note the “borrows” from the second and sixth bits.

Signed Magnitude

- Easy for Human

Complicated Computer Hardware

- Two different representations for zero: positive zero and negative zero. (Does K+(-K) works for both representations?)

- There is discontinuity in the number system, adding 1 to the number 7 should give 8 but it gives the value -0. 7 + 1 should give 8 but it gives -0

0	1	1	1
			1
1	0	0	0

→ Other system (Complements)

One’s complement Representation

- For example, in 8-bit **one’s complement,**

positive 3 is: `00000011`

Negative 3 is: `11111100`

`Just FLIP ALL BITS`.

One’s complement Representation

- With **one’s complement**
- 48 -19

```
1   1 1
    00110000
    11101100
    --------
    00011100
         + 1
    --------
    00011101
```

One’s complement Representation

- one’s complement is <u>simpler to implement</u> than signed magnitude.
- But it still has the disadvantage of having <u>two different representations for zero</u>: positive zero and negative zero.
- <u>Two’s complement solves this problem.</u>

Two's Complement representation

starting value	**00100100 = +36**
step1: reverse the bits (1's complement)	**11011011**
step 2: add 1 to the value from step 1	**+ 1**
sum = 2's complement representation	**11011100 = -36**

Sum of an integer and its 2's complement must be zero:

00100100 + 11011100 = 00000000 (8-bit sum) $\Rightarrow$ Ignore Carry

The easiest way to obtain the 2's complement of a binary number is by starting at the LSB, leaving all the 0s unchanged, look for the first occurrence of a 1. Leave this 1 unchanged and complement all the bits after it.

Two's complement Representation

- To express a value in two's complement:
  - If the number is negative, find the one's complement of the number and then add 1.
- Example:
  - In 8-bit one's complement, positive 3 is: `00000011`
  - Negative 3 in one's complement is: `11111100`
  - Adding 1 gives us -3 in two's complement form: `11111101`.

Two's complement Representation

- With two's complement arithmetic, all we do is add our two binary numbers. Just discard any carries emitting from the high order bit.

48 - 19

```
1        1 1
          0 0 1 1 0 0 0 0
        + 1 1 1 0 1 1 0 1
        -----------------
          0 0 0 1 1 1 0 1
```

We note that 19 in one's complement is:

`00010011`, so -19 in one's complement is:

`11101100`,

and -19 in two's complement is: `11101101`.

Overflow for Unsigned integer Addition

- Unsigned overflow – result is “out of range”
  - Occurs if carryout of MSB is 1

	Binary	Check in decimal
	1111	15
	+ 0100	+4
(cout=1)	0011	3(?)

(unsigned overflow occurs)

- 25 represent in three ways
- -25
- Sm-----1’s CMP-----2’comp

Detection of overflows in (unsigned and signed) additions and subtractions:

	unsigned	signed
Addition	Cout of MSB ==1	
Subtraction		

Overflow for signed integer addition? (a)

❖ (a) Previous example

	1111	-1
	+ 0100	+4
(cout=1)	0011	3(?)
(Cin =1)		(NO signed overflow occurs)

Overflow for signed integer addition (b)

❖ (b) New example

	0110	+6
	+0110	+6
(cout=0)	1100	- 4(?)
(cin = 1)		(Signed overflow occurs)

Overflow for signed integer addition (c)

- ❖ (c) New example

	1010	-6
	+1001	+-7
(cout=1)	0011	3?
(cin = 0)		(signed overflow occurs)

How do we detect overflow in signed integer addition?

- Whenever 2 positive values are added and result is negative
- Whenever 2 negative values are added and the result is positive
- What if one operand is positive and the other is negative in addition?
- Cout ≠ Cin

Detection of overflows in (unsigned and signed) additions and subtractions:

	unsigned	signed
Addition	Cout of MSB ==1	Cout of MSB ≠ Cin of MSB
Subtraction		

Subtraction in computer hardware

To compute x-y:

- ❖ Perform two’s-complement operation on y
- ❖ Add x and result of two’s-complement operation.

- ❖ Works for unsigned representation
- ❖ Works for two’s-complement representation

How does subtraction work for unsigned integers?

- The same adder for unsigned addition is used for unsigned subtraction
- A – B == A + (-B)

	0101	5
	- 1001	-9
	0101	5
	+0111	+(-9)
(cout=0)	1100	12? (unsigned overflow)

Note no carryout and the result is wrong.

Overflow in unsigned integer subtraction

	1001	9
	- 0101	-5
	1001	9
	+1011	+-5
(cout=1)	0100	4 (correct answer)

There is carryout and the result is correct!

Carryout of MSB == 1 means no overflow in unsigned subtraction

Carryout of MSB == 0 means overflow in unsigned subtraction

Overflow in signed integer subtraction

- No need to implement a separate subtractor
- A – B = A + (-B)
- Overflow may occur
- Overflow detection same as overflow addition of signed integers
  - Whenever 2 positive values are added and result is negative
  - Whenever 2 negative values are added and the result is positive
- Or Carryout of MSB $\neq$ Carryin of MSB

Detection of overflows in (unsigned and signed) additions and subtractions:

	unsigned	signed
Addition	Cout of MSB ==1	Cout of MSB $\neq$ Cin of MSB
Subtraction	Cout of MSB ==0	Cout of MSB $\neq$ Cin of MSB

Overflow detection Example

- Example:
- **107 + 46** (in 2’s Complement)
- We see that the nonzero carry from the seventh bit *overflows* into the sign bit, giving us the erroneous result: 107 + 46 = -103.

```
    1 1   1 1 1
    0 1 1 0 1 0 1 1
+   0 0 1 0 1 1 1 0
-------------------
    1 0 0 1 1 0 0 1
```

<u>Rule for detecting two’s complement overflow:</u> When the “carry in” and the “carry out” of the sign bit differ, overflow has occurred.

Negative Numbers Range

32 bits can only represent 2^{32} numbers – if we wish to also represent negative numbers, we can represent 2^{31} positive numbers (incl zero) and 2^{31} negative numbers

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{two} = 0_{ten}$
$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = 1_{ten}$
…
$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two} = 2^{31}-1$

$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{two} = -2^{31}$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = -(2^{31} – 1)$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two} = -(2^{31} – 2)$
…
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two} = -2$
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two} = -1$

Two's Complement Representation

- Positive numbers
  - Signed value = Unsigned value
- Negative numbers
  - Signed value = Unsigned value - 2^n
  - n = number of bits
- Negative weight for MSB
  - Another way to obtain the signed value is to assign a negative weight to most-significant bit

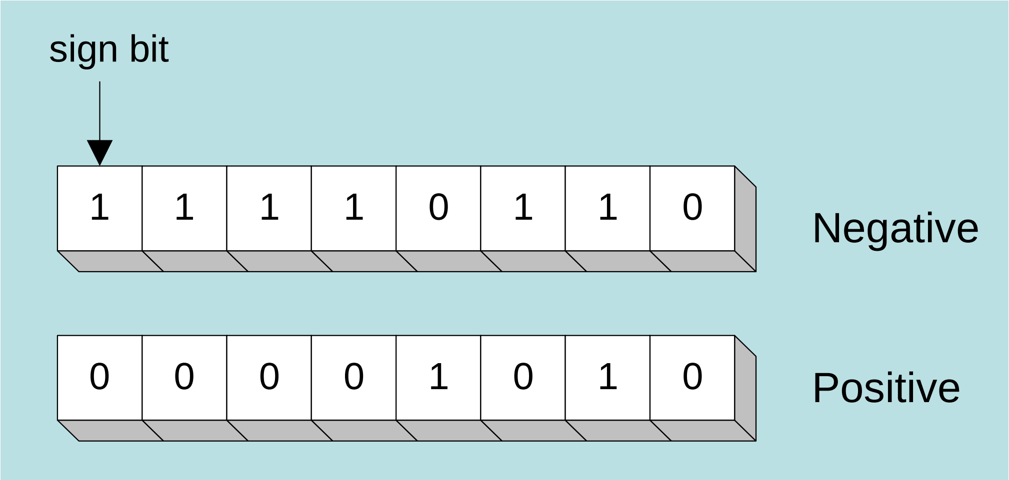
1	0	1	1	0	1	0	0
-128	64	32	16	8	4	2	1

= -128 + 32 + 16 + 4 = -76

8-bit Binary value	Unsigned value	Signed value
00000000	0	0
00000001	1	+1
00000010	2	+2
. . .	. . .	. . .
01111110	126	+126
01111111	127	+127
10000000	128	-128
10000001	129	-127
. . .	. . .	. . .
11111110	254	-2
11111111	255	-1

Sign Bit

Highest bit indicates the sign. 1 = negative, 0 = positive



If highest digit of a hexadecimal is > 7, the value is negative

Examples: 8A and C5 are negative bytes

A21F and 9D03 are negative words

B1C42A00 is a negative double-word

Mapping Signed ↔ Unsigned

Bits	Signed	Unsigned
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	-8	8
1001	-7	9
1010	-6	10
1011	-5	11
1100	-4	12
1101	-3	13
1110	-2	14
1111	-1	15

→ T2U →

← U2T ←

Mapping Signed ↔ Unsigned

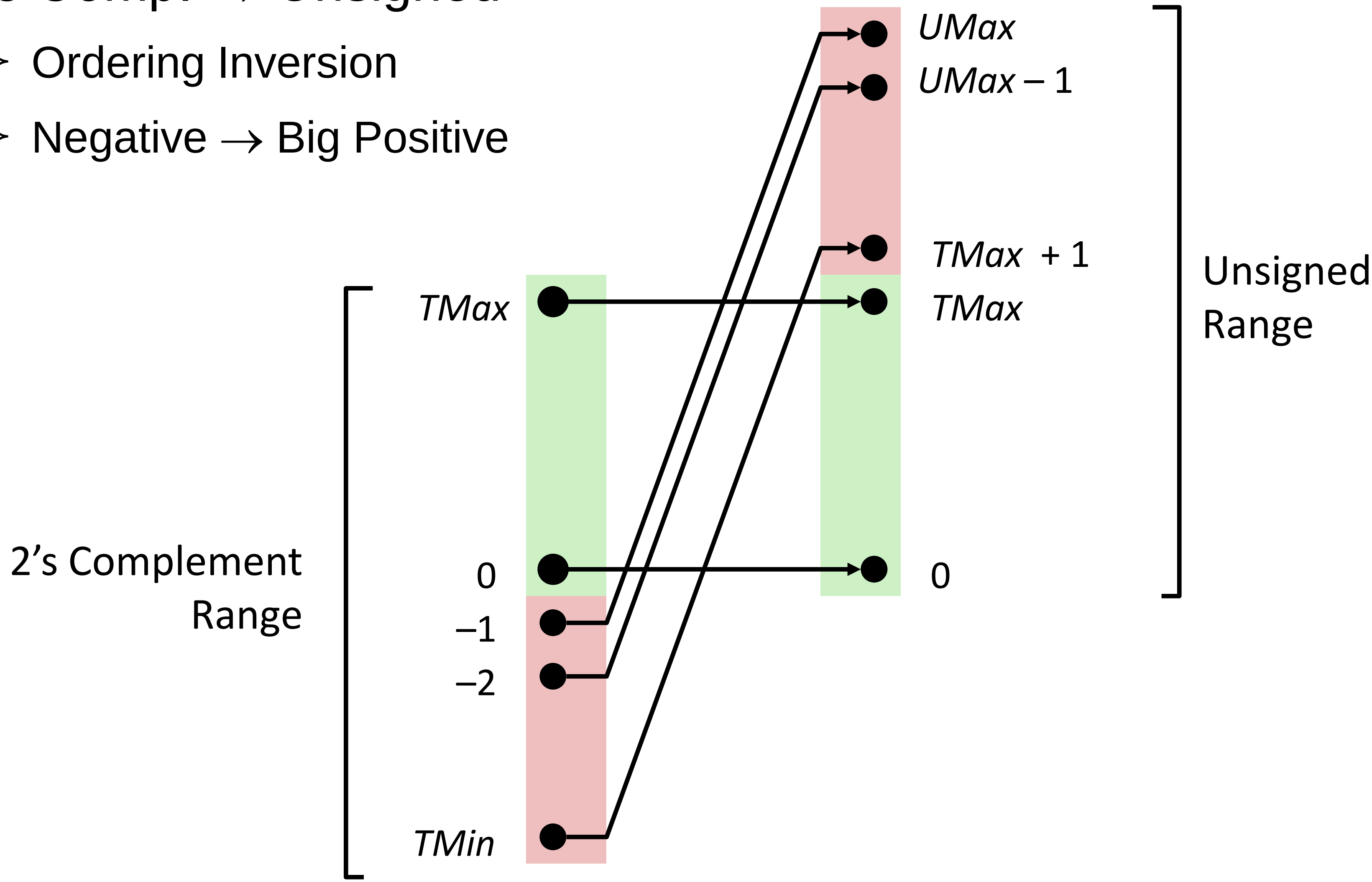
Bits	Signed	Unsigned
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	-8	8
1001	-7	9
1010	-6	10
1011	-5	11
1100	-4	12
1101	-3	13
1110	-2	14
1111	-1	15

Signed ↔ Unsigned (rows 0000–0111): =

Signed ↔ Unsigned (rows 1000–1111): +/- 16

Conversion Visualized

- 2's Comp. → Unsigned
  - Ordering Inversion
  - Negative → Big Positive



Summary of integer representation

Contents of Memory	Unsigned	Sign-and-Magnitude	One's Complement	Two's Complement
0000	0	+0	+0	+0
0001	1	+1	+1	+1
0010	2	+2	+2	+2
0011	3	+3	+3	+3
0100	4	+4	+4	+4
0101	5	+5	+5	+5
0110	6	+6	+6	+6
0111	7	+7	+7	+7
1000	8	–0	–7	–8
1001	9	–1	–6	–7
1010	10	–2	–5	–6
1011	11	–3	–4	–5
1100	12	–4	–3	–4
1101	13	–5	–2	–3
1110	14	–6	–1	–2
1111	15	–7	–0	–1

Ranges of Signed Integers

The unsigned range is divided into two signed ranges for positive and negative numbers

Storage Type	Range (low–high)	Powers of 2
Signed byte	−128 to +127	-2^7 to $(2^7 - 1)$
Signed word	−32,768 to +32,767	-2^{15} to $(2^{15} - 1)$
Signed doubleword	−2,147,483,648 to 2,147,483,647	-2^{31} to $(2^{31} - 1)$
Signed quadword	−9,223,372,036,854,775,808 to +9,223,372,036,854,775,807	-2^{63} to $(2^{63} - 1)$

Practice: What is the range of signed values that may be stored in 20 bits?

Thanks!