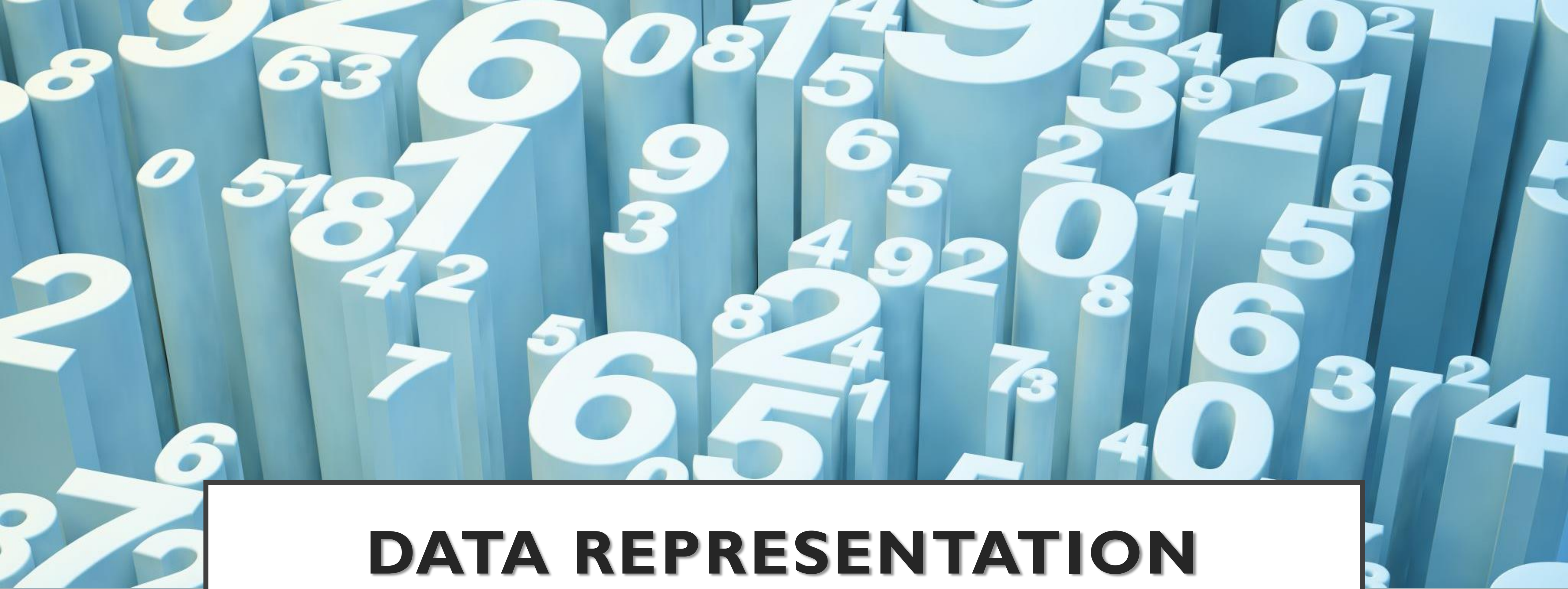


DATA REPRESENTATION PART 1: INTEGERS

CCICOMP

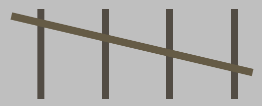
OVERVIEW

- Explain the goals of data representation
- Represent whole numbers in signed and unsigned binary integer format
- Explain the concepts of data range, sign extension, and overflow

ACTIVITY

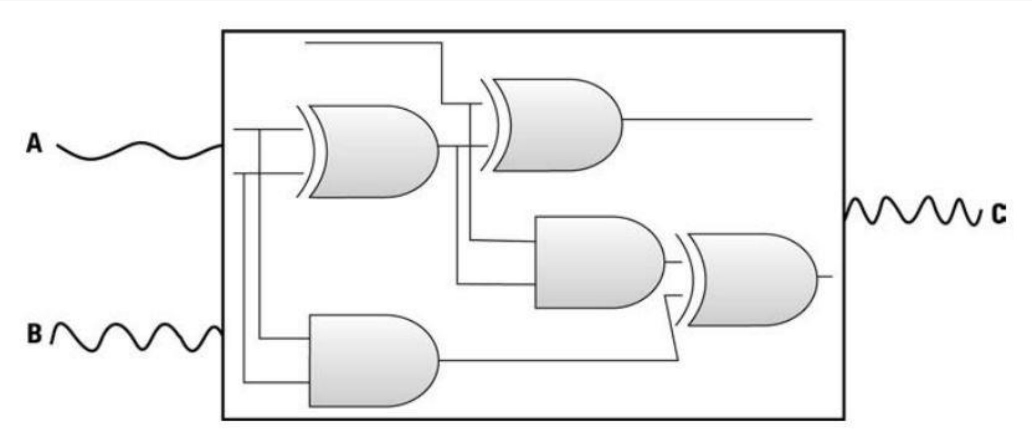
Try to represent the value of 5 in as many ways as you can

5 V 101b

five 五 

RECALL: LANGUAGE OF COMPUTERS

- Computer systems represent data electrically and process it with electrical switches with 2 states (on / off) that express binary data



- Computers deal with all sorts of non-numeric data (e.g. strings, images, video), but they are internally still represented by a set of binary numeric values.

On/Off switches → Processing Circuits → Processing Subsystems → CPUs

RECALL: BINARY

- Positional number system used by computers to represent any form of data
- Uses *radix* or *base* 2 (0 = off, 1 = on)
- Why binary?
  - Binary numbers represented as on/off electrical signals can be transported
    reliably between computer systems and their components
  - Binary numbers represented as electrical signals can be processed by two-state electrical devices that are easy to design and fabricate
  - Correspond directly with Boolean logic – a form of logic that evaluates sequences of statements as 'true' or 'false'

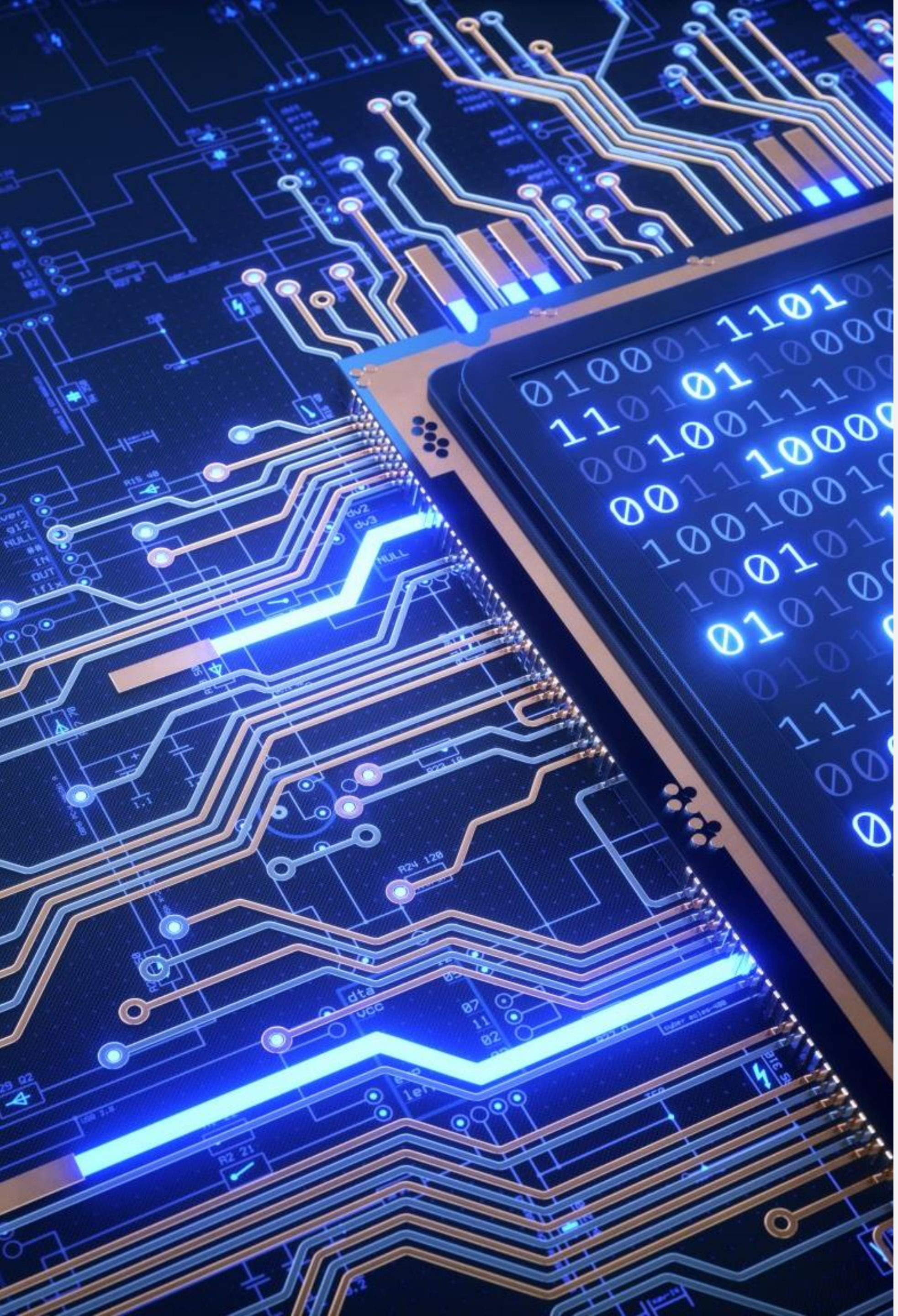
GOALS OF COMPUTER DATA REPRESENTATION

- Although all modern computers represent data internally with binary digits, they don't necessarily represent larger numeric values with positional bit strings.
- Positional numbering systems are convenient for humans to interpret and manipulate but are not suited to be used with the way a computer CPU operates
- Any representation format for numeric data represents a balance among several factors.

Size	Range	Accuracy	Ease of Manipulation	Standardization

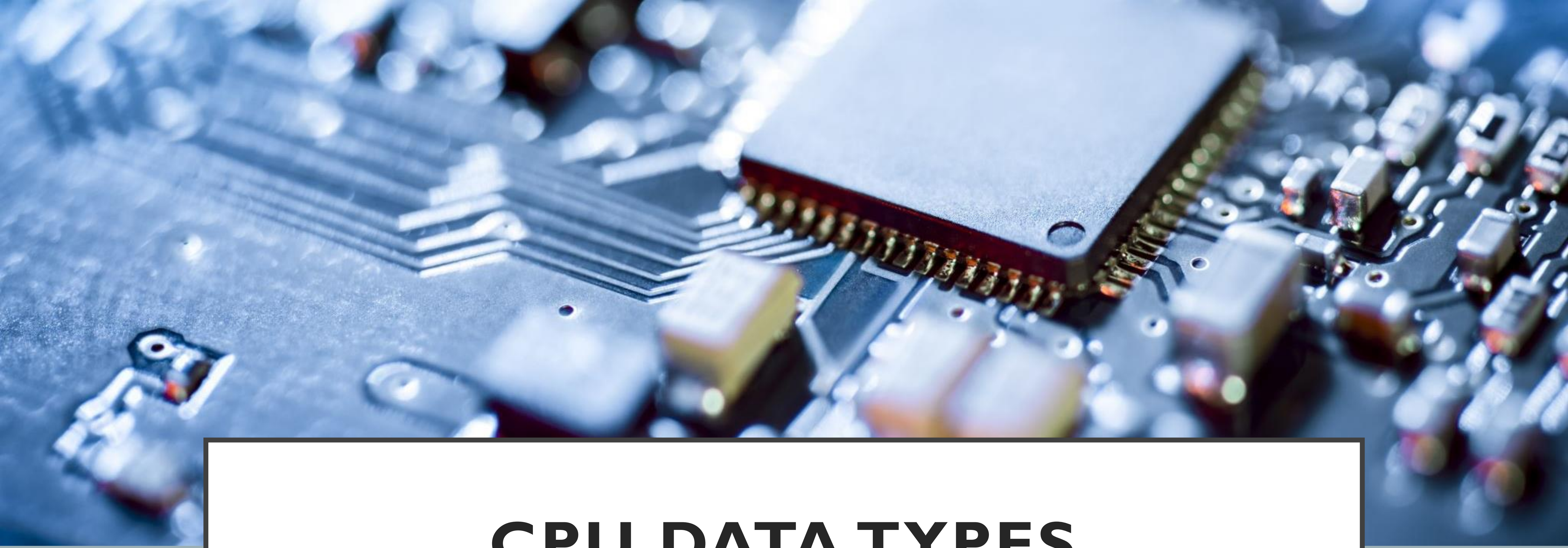
FACTORS FOR REPRESENTATION

- **Data size and Range**
  - Data size describes the number of bits used to represent a numeric value which directly affects the range of values that can be represented.
  - Smaller size = smaller range
- **Accuracy**
  - *Accuracy* or *precision* of representation increases with the number of data bits used.
  - Some calculations can generate quantities too large or too small to be contained in a machine's finite circuitry (e.x. 1 / 3 =1.3333…) – How to store as an approximate finite value?
  - More bits = less error = more space consumed



FACTORS FOR REPRESENTATION

- **Ease of Manipulation**
  - Refers to executing processor operations (e.g. addition, subtraction, comparison, etc)
  - Need internal circuitry to perform the operation.
  - How you represent values has an effect on the complexity of the circuit needed to do operations
- **Standardization**
  - Data must be communicated between devices in a single computer and to other computers via networks
  - Data formats are designed follow known standards in order to be suitable for use with a wide variety of devices and promote compatibility



CPU DATA TYPES

CPU DATA TYPES

- The CPUs of most modern computers can represent and process at least the following primitive data types:

Integer | Real number | Character | Boolean | Memory Address

- The arrangement and interpretation of bits are usually different for each data type.
- Format for each data type balances compactness, range, accuracy, ease of manipulation, and standardization.
- CPUs can also implement multiple versions of each type to support different types of processing operations.

INTEGERS

- An integer is a whole number
- Representation
  - Unsigned Integer
    - All digits serve as part of the numeric value and is always treated as a positive number
  - Signed integer
    - uses one bit to represent whether the value is positive or negative.

UNSIGNED INTEGER

- Unsigned integers are always interpreted as positive numbers only
- <u>All bits</u> are considered to determine the magnitude of a number
  - Given n-bit vector $b_{n-1} \ldots b_2\ b_1\ b_0$
  - Decimal value $= b_{n-1}2^{n-1} + \ldots + b_2 2^2 + b_1 2^2 + b_0 2^0$

Example: Given an 8-bit value 10110110_2

$$= 1x2^7 + 0x2^6 + 1x2^5 + 1x2^4 + 0x2^3 + 1x2^2 + 1x2^1 + 0x2^0$$

$$= 128 + 0 + 32 + 16 + 0 + 4 + 2 + 0$$

$$= \mathbf{182_{10}}$$

UNSIGNED INTEGERS - ZERO EXTENSION

- To extend an unsigned integer data to a given length, we can simply pad 0s on the left while still preserving its numeric value.
- Examples:
  - To represent 25_{10} as an 8-bit unsigned integer:

    25_{10} = 1 1001 $\longrightarrow$ 0001 1001

  - To represent 500_{10} as a 16-bit unsigned integer:

    500_{10} = 1 1111 0100 $\longrightarrow$ 0000 0001 1111 0100

UNSIGNED INTEGERS - RANGE

- Unsigned integers have a range of 0 to 2^n-1 where n = number of bits
  - What is the range of values for 8-bit unsigned integer?

0000 0000 ⟶ 1111 1111

0 ⟶ **255**

  - What about a 16-bit unsigned integer?

0000 0000 0000 0000 ⟶ 1111 1111 1111 1111

0 ⟶ **65535**

SIGNED INTEGER

- Signed integers encompass both positive and negative whole numbers
- 3 common representation schemes:

Sign and Magnitude	One's Complement	Two's Complement

- Representation schemes all assign the most significant bit (*MSB*) as a ***sign bit*** (positive value if sign bit is 0; negative value if sign bit is 1).
- Ex:

Sign bit = 0
(positive number) ← 0 | 1111 0100 → Magnitude bits
(value depends on
representation scheme)

SIGN-AND-MAGNITUDE

The **Sign-and-Magnitude** scheme simply uses the binary equivalent of the absolute value of a number then appends a sign bit as the MSB

- Convert the integer to its unsigned form
- If the number of bits is less than the required length, perform zero extension until 1 bit less than the required data length
- Append the MSB according to the sign (0 if positive, 1 if negative)

Example 1 : $+25_{10}$ as an 8-bit integer

0 00 11001

- 11001 → 25 in binary
- 00 → zero extension
- 0 → positive sign bit

Example 2 : -25_{10} as a 16-bit integer

1 0000000000 11001

- 11001 → 25 in binary
- 0000000000 → zero extension
- 1 → negative sign bit

ONE'S COMPLEMENT

The **1's Complement** scheme uses the following steps to represent an integer value:

1. Convert the integer to its unsigned form
2. If value is a negative number, invert all bits
3. Append the MSB according to the sign (0 if positive, 1 if negative)
4. Extend to the required length by replicating the sign bit (sign extension)

Example 1 : $+25_{10}$ as an 8-bit integer

00 0 11001

- 11001 → 25 in binary
- 0 → positive sign bit
- 00 → sign extension

Example 2 : -25_{10} as an 16-bit integer

11001 25 in binary

1111111111 1 00110

- 00110 → invert bits (negative)
- 1 → negative sign bit
- 1111111111 → sign extension

ONE'S COMPLEMENT

To convert a signed integer in 1's complement format back to decimal:

1. Determine if value is positive or negative based on its MSB
2. If negative (MSB = 1), invert all bits
3. Compute for the decimal equivalent of the binary value

Example 1: $00011001_2 = (?)_{10}$

$00011001 = 0x2^7 + 0x2^6 + 0x2^5 + 1x2^4 + 1x2^3 + 0x2^2 + 0x2^1 + 1x2^0$

$\rightarrow +25$

Example 2 : $11100110_2 = (?)_{10}$

$11100110 \rightarrow 00011001$ invert bits (negative)

$= 0x2^7 + 0x2^6 + 0x2^5 + 1x2^4 + 1x2^3 + 0x2^2 + 0x2^1 + 1x2^0$

$\rightarrow -25$

TWO'S COMPLEMENT

The **2's Complement** scheme uses the following steps to represent an integer value:

1. Convert the integer to its unsigned form
2. If value is a negative number, invert all bits then add 1
3. Append the MSB according to the sign (0 if positive, 1 if negative)
4. Extend to the required length by replicating the sign bit (sign extension)

Example 1 : $+25_{10}$ as an 8-bit integer

00 0 11001

25 in binary

positive sign bit

sign extension

Example 2 : -25_{10} as an 16-bit integer

11001	25 in binary
00110	invert bits (negative)
+ 1	add 1

1111111111 1 00111

negative sign bit

sign extension

TWO'S COMPLEMENT

To convert a signed integer in 2's complement format back to decimal:

1. Determine if value is positive or negative based on its MSB
2. If negative (1 MSB), invert all bits then add 1
3. Compute for the decimal equivalent of the binary value

Example 1: $00011001_2 = (?)_{10}$

$00011001 = 0x2^7 + 0x2^6 + 0x2^5 + 1x2^4 + 1x2^3 + 0x2^2 + 0x2^1 + 1x2^0$

+25

Example 2 : $11100111_2 = (?)_{10}$

11100111 □ 00011000 invert bits (negative)

00011001 add 1

$= 0x2^7 + 0x2^6 + 0x2^5 + 1x2^4 + 1x2^3 + 0x2^2 + 0x2^1 + 1x2^0$

- 25

TWO'S COMPLEMENT AND THE MODERN CPU

- 2's-complement is commonly used by modern CPUs to represent signed integers because it uses only one representation of 0
- Has a range of –(2n-1) to +(2n-1 - 1) where n = number of bits
- Some specific numbers:
  - 0: 0000 0000 … 0000
  - –1: 1111 1111 … 1111
  - Most-negative: 1000 0000 … 0000
  - Most-positive: 0111 1111 … 1111

OVERFLOW CONDITIONS

- Most modern CPUs use 64 bits to represent a 2’s complement value and support 32-bit formats for backward compatibility with older software.
- The same data size is used regardless of the value - e.g. 1 is still represented using 64 bits even if it can be represented using 2 bits (sign bit + value)
- Fixed width is needed because computer circuitry is not infinite!
- An operation that produces a result exceeding the data width leads to an overflow condition
- Overflow conditions can produce mathematically illogical results

OVERFLOW CONDITIONS

Example 1: Adding 8-bit **unsigned** values

102		0	1	1	0	0	1	1	0
+ 230		1	1	1	0	0	1	1	0
	1	0	1	0	0	1	1	0	0

Extra bit overflows out of data size (Disregarded)

Interpreted as 76_{10}

Example 2: Adding 8-bit **signed** values

90	0	1	0	1	1	0	1	0
+ 40	0	0	1	0	1	0	0	0
	1	0	0	0	0	0	1	0

Magnitude value overflows into the sign bit

Interpreted as 2's complement -126_{10}

RANGE AND OVERFLOW

- Data format length affects overflow
  - Longer format – less chance of overflow condition because of capability to represent larger range of values, but more space possibly wasted
  - Shorter format - higher chance of overflow condition because of capability to represent smaller range of values, but more compact
- CPU designers and programmers often take these into consideration
  - To avoid overflow, some programming languages have additional data types that use 2 adjacent fixed-length data items (double precision)
  - For integers, this data type is often referred to as a *long integer*

SUMMARY

- Data representation aims to strike a balance among size, range, accuracy, ease of manipulation, and standardization
- Unsigned integers can represent positive whole numbers only.
  - All bits are used to signify the magnitude of its numerical value
  - Zero extension is used to fill bit positions to the required data width
- Signed integers can represent both negative and positive whole numbers
  - The MSB is the sign bit and represents whether the numerical value is positive (MSB = 0) or negative (MSB = 1)
- Sign-and-magnitude scheme
  - Appends the sign bit to the absolute value of the number in binary
  - Performs zero extension on the magnitude bits to fill bit positions to the required data width

SUMMARY

- 1's complement scheme
  - Reverses all magnitude bits if a numeric value is negative
  - Appends the sign bit then performs sign extension to fill bit positions to the required data width
- 2's complement scheme
  - Reverses all magnitude bits and adds 1 if a numeric value is negative
  - Appends the sign bit then performs sign extension to fill bit positions to the required data width
  - Is the representation scheme for signed integers used by modern CPUs
- The width of data affects it range of valid values
- Overflow conditions occur when an operation results in a value exceeding width of data and produce illogical mathematical results