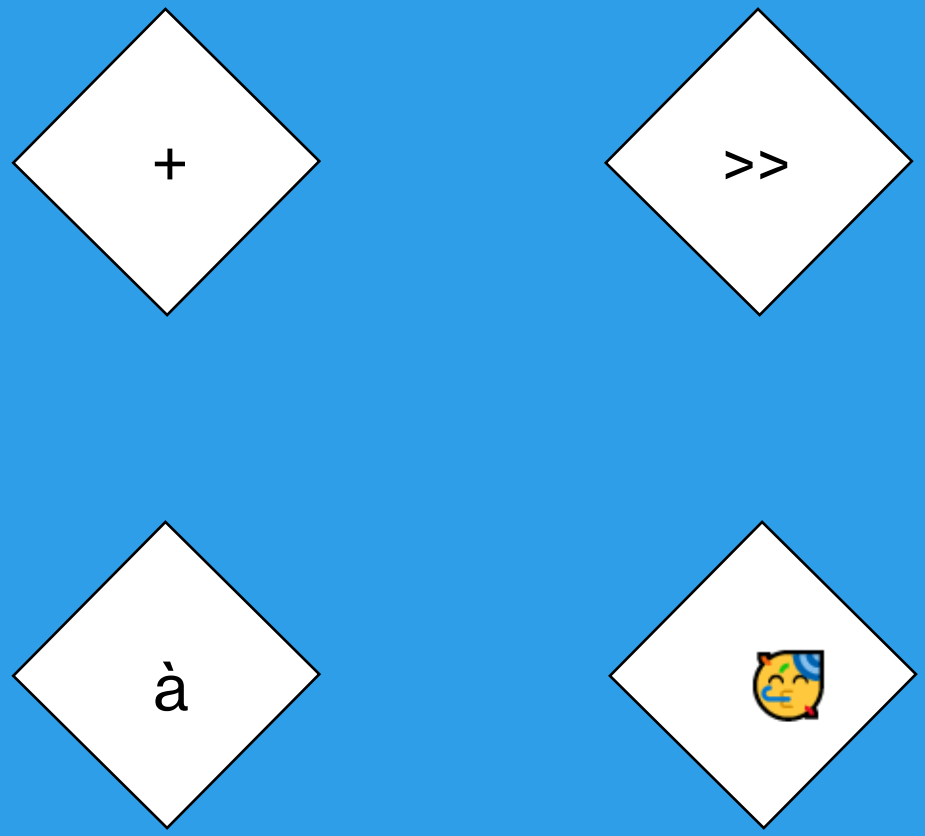


Computer Systems

Binary Arithmetic in Python & Representing Information

CS51 – Fall 2026

Bitwise operations

Bitwise operations

- If we are given two numbers, we can apply the operators NOT, AND, OR, and XOR **bitwise.**
- Treat numbers as two's complement.
- Perform Boolean operation *per bit*.
- Convert result to decimal.

AND (&)

X	Y	X ∧ Y
0	0	0
0	1	0
1	0	0
1	1	1

NOT (~)

X	¬ X
0	1
1	0

OR (|)

X	Y	X ∨ Y
0	0	0
0	1	1
1	0	1
1	1	1

XOR (^)

X	Y	X ⊕ Y
0	0	0
0	1	1
1	0	1
1	1	0

Bitwise operation AND (&)

- 6 & 5 = ?

- Treat numbers as two's complement.
- Perform Boolean operation *per bit.*
- Convert result to decimal.

AND (&)

X	Y	X ∧ Y
0	0	0
0	1	0
1	0	0
1	1	1

Bitwise operation AND (&)

- 6 & 5 = ?

- **Treat numbers as two's complement.**

- Perform Boolean operation *per bit.*

- Convert result to decimal.

$$6_{10} = ?_2$$
$$5_{10} = ?_2$$

AND (&)

X	Y	X ∧ Y
0	0	0
0	1	0
1	0	0
1	1	1

Bitwise operation AND (&)

- 6 & 5 = ?

- **Treat numbers as two's complement.**

- Perform Boolean operation *per bit.*

- Convert result to decimal.

$$6_{10} = 0\,1\,1\,0_2$$

$$5_{10} = 0\,1\,0\,1_2$$

AND (&)

X	Y	X ∧ Y
0	0	0
0	1	0
1	0	0
1	1	1

Bitwise operation AND (&)

- 6 & 5 = ?

- Treat numbers as two's complement.
- **Perform Boolean operation *per bit.***
- Convert result to decimal.

$$6_{10} = 0\,1\,1\,0_2$$
$$\&\quad 5_{10} = 0\,1\,0\,1_2$$

AND (&)

X	Y	X ∧ Y
0	0	0
0	1	0
1	0	0
1	1	1

Bitwise operation AND (&)

- 6 & 5 = ?

- Treat numbers as two's complement.
- **Perform Boolean operation *per bit.***
- Convert result to decimal.

$$\begin{aligned} & 6_{10} = 0\,1\,1\,0_2 \\ \& \quad & 5_{10} = 0\,1\,0\,1_2 \\ \hline & 0\,1\,0\,0_2 \end{aligned}$$

AND (&)

X	Y	X ∧ Y
0	0	0
0	1	0
1	0	0
1	1	1

Bitwise operation AND (&)

- 6 & 5 = ?

- Treat numbers as two's complement.
- Perform Boolean operation *per bit.*
- **Convert result to decimal.**

$$
\begin{aligned}
& 6_{10} = 0\,1\,1\,0_2 \\
\& \quad & 5_{10} = 0\,1\,0\,1_2 \\
\hline
& ?_{10} = 0\,1\,0\,0_2
\end{aligned}
$$

AND (&)

X	Y	X ∧ Y
0	0	0
0	1	0
1	0	0
1	1	1

Bitwise operation AND (&)

- 6 & 5 = 4

- Treat numbers as two's complement.
- Perform Boolean operation *per bit.*
- **Convert result to decimal.**

$$\begin{aligned} & 6_{10} = 0\,1\,1\,0_2 \\ \& \quad & 5_{10} = 0\,1\,0\,1_2 \\ \hline & 4_{10} = 0\,1\,0\,0_2 \end{aligned}$$

AND (&)

X	Y	X ∧ Y
0	0	0
0	1	0
1	0	0
1	1	1

Bitwise operation OR (|)

- 11 | -9 = ?

- Treat numbers as two's complement.
- Perform Boolean operation *per bit.*
- Convert result to decimal.

OR (|)

X	Y	X ∧ Y
0	0	0
0	1	1
1	0	1
1	1	1

Bitwise operation OR (|)

- 11 | -9 = ?

- **Treat numbers as two's complement.**

- Perform Boolean operation *per bit.*

- Convert result to decimal.

$$11_{10} = ?_2$$

$$-9_{10} = ?_2$$

OR (|)

X	Y	X ∧ Y
0	0	0
0	1	1
1	0	1
1	1	1

Bitwise operation OR (|)

- 11 | -9 = ?

- **Treat numbers as two's complement.**
- Perform Boolean operation *per bit.*
- Convert result to decimal.

$$11_{10} = 0\ 1\ 0\ 1\ 1_2$$
$$-9_{10} = 1\ 0\ 1\ 1\ 1_2$$

OR (|)

X	Y	X ∧ Y
0	0	0
0	1	1
1	0	1
1	1	1

Bitwise operation OR (|)

- 11 | -9 = ?

- Treat numbers as two's complement.
- **Perform Boolean operation *per bit.***
- Convert result to decimal.

$$11_{10} = 0\ 1\ 0\ 1\ 1_2$$
$$|\ \ -9_{10} = 1\ 0\ 1\ 1\ 1_2$$

OR (|)

X	Y	X ∧ Y
0	0	0
0	1	1
1	0	1
1	1	1

Bitwise operation OR (|)

- 11 | -9 = ?

- Treat numbers as two's complement.
- **Perform Boolean operation *per bit.***
- Convert result to decimal.

$$\begin{array}{r} 11_{10} = 0\ 1\ 0\ 1\ 1_2 \\ |\ \ -9_{10} = 1\ 0\ 1\ 1\ 1_2 \\ \hline 1\ 1\ \ 1\ 1\ 1_2 \end{array}$$

OR (|)

X	Y	X ∧ Y
0	0	0
0	1	1
1	0	1
1	1	1

Bitwise operation OR (|)

- 11 | -9 = ?

- Treat numbers as two's complement.
- Perform Boolean operation *per bit.*
- **Convert result to decimal.**

$$11_{10} = 0\ 1\ 0\ 1\ 1_2$$
$$|\ \ -9_{10} = 1\ 0\ 1\ 1\ 1_2$$
$$\overline{\qquad\qquad\qquad\qquad}$$
$$?_{10} = 1\ 1\ \ 1\ 1\ 1_2$$

OR (|)

X	Y	X ∧ Y
0	0	0
0	1	1
1	0	1
1	1	1

Bitwise operation OR (|)

- 11 | -9 = -1

- Treat numbers as two's complement.
- Perform Boolean operation *per bit.*
- **Convert result to decimal.**

OR (|)

X	Y	X ∧ Y
0	0	0
0	1	1
1	0	1
1	1	1

$$11_{10} = 0\ 1\ 0\ 1\ 1_2$$
$$|\ \ -9_{10} = 1\ 0\ 1\ 1\ 1_2$$
$$-1_{10} = 1\ 1\ \ 1\ 1\ 1_2$$

Bitwise operation NOT (~)

- ~9 = ?

- Treat numbers as two's complement.
- Perform Boolean operation *per bit.*
- Convert result to decimal.

NOT (~)

X	¬ X
0	1
1	0

Bitwise operation NOT (~)

- ~9 = ?

NOT (~)

X	¬ X
0	1
1	0

- **Treat numbers as two's complement.**
- Perform Boolean operation *per bit.*
- Convert result to decimal.

$$9_{10} = ?_{2}$$

Bitwise operation NOT (~)

- ~9 = ?

NOT (~)

X	¬ X
0	1
1	0

- **Treat numbers as two's complement.**
- Perform Boolean operation *per bit.*
- Convert result to decimal.

$$9_{10} = 0\,1\,0\,0\,1_2$$

Bitwise operation NOT (~)

NOT (~)

X	¬ X
0	1
1	0

- ~9 = ?

- Treat numbers as two's complement.
- **Perform Boolean operation *per bit.***
- Convert result to decimal.

$$\sim 9_{10} = 0\,1\,0\,0\,1_2$$

Bitwise operation NOT (~)

NOT (~)

X	¬ X
0	1
1	0

- ~9 = ?

- Treat numbers as two's complement.
- **Perform Boolean operation *per bit.***
- Convert result to decimal.

$$\sim 9_{10} = 0\,1\,0\,0\,1_2$$

$$1\,0\,1\,1\,0_2$$

Bitwise operation NOT (~)

NOT (~)

X	¬ X
0	1
1	0

- ~9 = ?

- Treat numbers as two's complement.
- Perform Boolean operation *per bit.*
- **Convert result to decimal.**

$$\sim 9_{10} = 0\,1\,0\,0\,1_2$$

$$?_{10} = 1\,0\,1\,1\,0_2$$

Bitwise operation NOT (~)

NOT (~)

X	¬ X
0	1
1	0

- ~9 = -10

- Treat numbers as two's complement.
- Perform Boolean operation *per bit.*
- **Convert result to decimal.**

$$\sim 9_{10} = 0\ 1\ 0\ 0\ 1_2$$

$$-10_{10} = 1\ 0\ 1\ 1\ 0_2$$

Bitwise operation XOR (^)

- 10^1 = ?

- Treat numbers as two's complement.
- Perform Boolean operation *per bit.*
- Convert result to decimal.

XOR (^)

X	Y	X ⊕ Y
0	0	0
0	1	1
1	0	1
1	1	0

Bitwise operation XOR (^)

- 10^1 = ?

XOR (^)

X	Y	X ⊕ Y
0	0	0
0	1	1
1	0	1
1	1	0

- **Treat numbers as two's complement.**
- Perform Boolean operation *per bit.*
- Convert result to decimal.

$$10_{10} = ?_2$$
$$1_{10} = ?_2$$

Bitwise operation XOR (^)

- 10^1 = ?

XOR (^)

X	Y	X ⊕ Y
0	0	0
0	1	1
1	0	1
1	1	0

- **Treat numbers as two's complement.**
- Perform Boolean operation *per bit.*
- Convert result to decimal.

$$10_{10} = 0\ 1\ 0\ 1\ 0_2$$

$$1_{10} = 0\ 1_2$$

Bitwise operation XOR (^)

- 10^1 = ?

XOR (^)

X	Y	X ⊕ Y
0	0	0
0	1	1
1	0	1
1	1	0

- **Treat numbers as two's complement.**
- Perform Boolean operation *per bit.*
- Convert result to decimal.

$$10_{10} = 0\ 1\ \ 0\ 1\ 0_2$$
$$1_{10} = 0\ 0\ 0\ 0\ 1_2$$

Bitwise operation XOR (^)

- 10^1 = ?

XOR (^)

X	Y	X ⊕ Y
0	0	0
0	1	1
1	0	1
1	1	0

- Treat numbers as two's complement.
- **Perform Boolean operation *per bit.***
- Convert result to decimal.

$$10_{10} = 0\ 1\ 0\ 1\ 0_2$$
$$\hat{}\quad 1_{10} = 0\ 0\ 0\ 0\ 1_2$$

Bitwise operation XOR (^)

- 10^1 = ?

XOR (^)

X	Y	X ⊕ Y
0	0	0
0	1	1
1	0	1
1	1	0

- Treat numbers as two's complement.
- **Perform Boolean operation *per bit.***
- Convert result to decimal.

$$\begin{array}{rr} & 10_{10} = 0\ 1\ \ 0\ 1\ 0_2 \\ \hat{} & \ \ 1_{10} = 0\ 0\ 0\ 0\ 1_2 \\ \hline & 0\ 1\ \ 0\ \ 1\ 1_2 \end{array}$$

Bitwise operation XOR (^)

- 10^1 = ?

XOR (^)

X	Y	X ⊕ Y
0	0	0
0	1	1
1	0	1
1	1	0

- Treat numbers as two's complement.
- Perform Boolean operation *per bit.*
- **Convert result to decimal.**

$$\begin{array}{r} 10_{10} = 0\ 1\ \ 0\ 1\ 0_2 \\ \hat{}\quad 1_{10} = 0\ 0\ 0\ 0\ 1_2 \\ \hline ?_{10} = 0\ 1\ \ 0\ \ 1\ 1_2 \end{array}$$

Bitwise operation XOR (^)

- 10^1 = 11

XOR (^)

X	Y	X ⊕ Y
0	0	0
0	1	1
1	0	1
1	1	0

- Treat numbers as two's complement.
- Perform Boolean operation *per bit.*
- **Convert result to decimal.**

$$\begin{array}{r} 10_{10} = 0\ 1\ \ 0\ 1\ 0_2 \\ \hat{}\quad 1_{10} = 0\ 0\ 0\ 0\ 1_2 \\ \hline 11_{10} = 0\ 1\ \ 0\ \ 1\ 1_2 \end{array}$$

Bitwise operation XOR (^)

- 10^-1 = ?

- Treat numbers as two's complement.
- Perform Boolean operation *per bit.*
- Convert result to decimal.

XOR (^)

X	Y	X ⊕ Y
0	0	0
0	1	1
1	0	1
1	1	0

Bitwise operation XOR (^)

- 10^-1 = ?

- **Treat numbers as two's complement.**
- Perform Boolean operation *per bit.*
- Convert result to decimal.

$$10_{10} = ?_2$$

$$-1_{10} = ?_2$$

XOR (^)

X	Y	X ⊕ Y
0	0	0
0	1	1
1	0	1
1	1	0

Bitwise operation XOR (^)

- 10^-1 = ?

XOR (^)

X	Y	X ⊕ Y
0	0	0
0	1	1
1	0	1
1	1	0

- **Treat numbers as two's complement.**
- Perform Boolean operation *per bit.*
- Convert result to decimal.

$$10_{10} = 0\ 1\ 0\ 1\ 0_2$$

$$-1_{10} = 1_2$$

Bitwise operation XOR (^)

- 10^-1 = ?

XOR (^)

X	Y	X ⊕ Y
0	0	0
0	1	1
1	0	1
1	1	0

- **Treat numbers as two's complement.**
- Perform Boolean operation *per bit.*
- Convert result to decimal.

$$10_{10} = 0\ 1\ 0\ 1\ 0_2$$
$$-1_{10} = 1\ 1\ 1\ 1\ 1_2$$

Bitwise operation XOR (^)

- 10^-1 = ?

XOR (^)

X	Y	X ⊕ Y
0	0	0
0	1	1
1	0	1
1	1	0

- Treat numbers as two's complement.
- **Perform Boolean operation *per bit.***
- Convert result to decimal.

$$10_{10} = 0\ 1\ 0\ 1\ 0_2$$
$$\hat{}\ \ -1_{10} = 1\ 1\ 1\ 1\ 1_2$$

Bitwise operation XOR (^)

- 10^-1 = ?

XOR (^)

X	Y	X ⊕ Y
0	0	0
0	1	1
1	0	1
1	1	0

- Treat numbers as two's complement.
- **Perform Boolean operation *per bit.***
- Convert result to decimal.

$$\begin{array}{rr} & 10_{10} = 0\ 1\ \ 0\ 1\ 0_2 \\ \hat{}\ & -1_{10} = 1\ \ 1\ \ 1\ \ 1\ 1_2 \\ \hline & 1\ \ 0\ \ 1\ 0\ 1_2 \end{array}$$

Bitwise operation XOR (^)

- 10^-1 = ?

XOR (^)

X	Y	X ⊕ Y
0	0	0
0	1	1
1	0	1
1	1	0

- Treat numbers as two's complement.
- Perform Boolean operation *per bit.*
- **Convert result to decimal.**

$$10_{10} = 0\ 1\ 0\ 1\ 0_2$$
$$\hat{}\ \ -1_{10} = 1\ 1\ 1\ 1\ 1_2$$
$$?_{10} = 1\ 0\ 1\ 0\ 1_2$$

Bitwise operation XOR (^)

- 10^-1 = -11

XOR (^)

X	Y	X ⊕ Y
0	0	0
0	1	1
1	0	1
1	1	0

- Treat numbers as two's complement.
- Perform Boolean operation *per bit.*
- **Convert result to decimal.**

$$\begin{array}{r} 10_{10} = 0\ 1\ \ 0\ 1\ 0_2 \\ \hat{}\quad 1_{10} = 1\ \ 1\ \ 1\ \ 1\ 1_2 \\ \hline -11_{10} = 1\ \ 0\ \ 1\ 0\ 1_2 \end{array}$$

PRACTICE TIME – Bitwise operations

- Let's assume we have $x = 5_{10}$ and $y = -9_{10}$
  - $\sim x$
  - $\sim y$
  - $x \& y$
  - $x | y$
  - $x \wedge y$

AND (&)

X	Y	X ∧ Y
0	0	0
0	1	0
1	0	0
1	1	1

NOT (~)

X	¬ X
0	1
1	0

OR (|)

X	Y	X ∨ Y
0	0	0
0	1	1
1	0	1
1	1	1

XOR (^)

X	Y	X ⊕ Y
0	0	0
0	1	1
1	0	1
1	1	0

ANSWER – Bitwise operations

- Let's assume we have $x = 5_{10}$ and $y = -9_{10}$
  - $\sim x = 11010_2 = -6_{10}$
  - $\sim y = 01000_2 = 8_{10}$
  - $x \& y = 00101_2 = 5_{10}$
  - $x | y = 10111_2 = -9_{10}$
  - $x \wedge y = 10010_2 = -14_{10}$

AND (&)

X	Y	X ∧ Y
0	0	0
0	1	0
1	0	0
1	1	1

NOT (~)

X	¬ X
0	1
1	0

OR (|)

X	Y	X ∨ Y
0	0	0
0	1	1
1	0	1
1	1	1

XOR (^)

X	Y	X ⊕ Y
0	0	0
0	1	1
1	0	1
1	1	0

Masking

- We can use bitwise operations to **mask** a certain number of bits in a number.
- By masking, we can reset, set, or invert certain bits in a number.
- Bitwise AND will reset a subset of the bits to 0
- Bitwise OR will set a subset of the bits to 1
- Bitwise XOR will invert a subset of the bits

Bitwise AND – Resetting bits to 0

We want to reset the three least significant bits to 0

&

Bit	6^{th}	5^{th}	4^{th}	3^{rd}	2^{nd}	1^{st}	0^{th}
Data	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Mask							
Result							

What number should we bitwise-AND (&) to do this?

Bitwise AND – Resetting bits to 0

We want to reset the three least significant bits to 0

&

Bit	6^{th}	5^{th}	4^{th}	3^{rd}	2^{nd}	1^{st}	0^{th}
Data	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Mask	1	1	1	1	0	0	0
Result							

What number should we bitwise-AND (&) to do this?

Bitwise AND – Resetting bits to 0

We want to reset the three least significant bits to 0

&

Bit	6^{th}	5^{th}	4^{th}	3^{rd}	2^{nd}	1^{st}	0^{th}
Data	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Mask	1	1	1	1	0	0	0
Result							

What number is this in decimal?

Bitwise AND – Resetting bits to 0

We want to reset the three least significant bits to 0

& -8_{10}

Bit	6^{th}	5^{th}	4^{th}	3^{rd}	2^{nd}	1^{st}	0^{th}
Data	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Mask	1	1	1	1	0	0	0
Result							

What number is this in decimal?

Bitwise AND – Resetting bits to 0

Say we have 47 & -8. *What's the result?*

47_{10}
& -8_{10}

Bit	6^{th}	5^{th}	4^{th}	3^{rd}	2^{nd}	1^{st}	0^{th}
Data	0	1	0	1	1	1	1
Mask	1	1	1	1	0	0	0
Result	0	1	0	1	**0**	**0**	**0**

Bitwise AND – Resetting bits to 0

Say we have 47 & -8. *What's the result?*

47_{10}
& -8_{10}
40_{10}

Bit	6^{th}	5^{th}	4^{th}	3^{rd}	2^{nd}	1^{st}	0^{th}
Data	0	1	0	1	1	1	1
Mask	1	1	1	1	0	0	0
Result	0	1	0	1	**0**	**0**	**0**

PRACTICE TIME - Bitwise AND

- What would the result be of applying the mask 42_{10} on the input 85_{10} using bitwise AND?

ANSWER - Bitwise AND

- What would the result be of applying the mask 42_{10} on the input 85_{10} using bitwise AND? First, we convert from decimal to binary, working with 8 bits for both numbers:
  - $42_{10} = 00101010_2$
  - $85_{10} = 01010101_2$
- We then apply the bitwise AND operation which will reset the the 7th (no real effect), 6th, 4th, 2nd, and 0th bit of the input, resulting to 0_{10}:

85_{10}
$\& \ 42_{10}$
0_{10}

Bit	7th	6th	5th	4th	3rd	2nd	1st	0th
Data	0	1	0	1	0	1	0	1
Mask	0	0	1	0	1	0	1	0
Result	**0**	**0**	0	**0**	0	**0**	0	**0**

Bitwise OR – Setting bits to 1

We want to set the two most significant bits to 1

Bit	6^{th}	5^{th}	4^{th}	3^{rd}	2^{nd}	1^{st}	0^{th}
Data	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Mask							
Result							

What number should we bitwise-OR (|) to do this?

Bitwise OR – Setting bits to 1

We want to set the two most significant bits to 1

Bit	6^{th}	5^{th}	4^{th}	3^{rd}	2^{nd}	1^{st}	0^{th}
Data	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Mask	1	1	0	0	0	0	0
Result							

What number should we bitwise-OR (|) to do this?

Bitwise OR – Setting bits to 1

We want to set the two most significant bits to 1

Bit	6^{th}	5^{th}	4^{th}	3^{rd}	2^{nd}	1^{st}	0^{th}
Data	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Mask	1	1	0	0	0	0	0
Result							

What number is this in decimal?

Bitwise OR – Setting bits to 1

We want to set the two most significant bits to 1

	Bit	6^{th}	5^{th}	4^{th}	3^{rd}	2^{nd}	1^{st}	0^{th}
	Data	0/1	0/1	0/1	0/1	0/1	0/1	0/1
$\mid -32_{10}$	**Mask**	1	1	0	0	0	0	0
	Result							

What number is this in decimal?

Bitwise OR – Setting bits to 1

Say we have 47 | - 32. *What's the result?*

47_{10}
$| \ -32_{10}$

Bit	6^{th}	5^{th}	4^{th}	3^{rd}	2^{nd}	1^{st}	0^{th}
Data	0	1	0	1	1	1	1
Mask	1	1	0	0	0	0	0
Result	**1**	**1**	0	1	1	1	1

Bitwise OR – Setting bits to 1

Say we have 47 | -32. *What's the result?*

47_{10}
$| \ -32_{10}$
-17_{10}

Bit	6^{th}	5^{th}	4^{th}	3^{rd}	2^{nd}	1^{st}	0^{th}
Data	0	1	0	1	1	1	1
Mask	1	1	0	0	0	0	0
Result	**1**	**1**	0	1	1	1	1

PRACTICE TIME - Bitwise OR

- What would the result be of applying the mask 17_{10} on the input -27_{10} using bitwise OR?

ANSWER - Bitwise OR

- What would the result be of applying the mask 17_{10} on the input -27_{10} using bitwise OR? You can assume you have 8 bits.
- First, we convert from decimal to binary:
  - $17_{10} = 010001_2$
  - $-27_{10} = 100101_2$
- We then apply the bitwise OR operation which will set the the 4th and 0th bit of the input to 1, resulting to -11_{10}:

-27_{10}
$| \ 17_{10}$
-11_{10}

Bit	5th	4th	3rd	2nd	1st	0th
Data	1	0	0	1	0	1
Mask	0	1	0	0	0	1
Result	1	**1**	0	1	0	**1**

Bitwise XOR – Inverting bits

We want to invert the two most and two least significant bits.

^

Bit	7^{th}	6^{th}	5^{th}	4^{th}	3^{rd}	2^{nd}	1^{st}	0^{th}
Data	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Mask								
Result								

What number should we bitwise-XOR (^) to do this?

Bitwise XOR – Inverting bits

We want to invert the two most and two least significant bits.

^

Bit	7^{th}	6^{th}	5^{th}	4^{th}	3^{rd}	2^{nd}	1^{st}	0^{th}
Data	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Mask	1	1	0	0	0	0	1	1
Result								

What number should we bitwise-XOR (^) to do this?

Bitwise XOR – Inverting bits

We want to invert the two most and two least significant bits.

^

Bit	7^{th}	6^{th}	5^{th}	4^{th}	3^{rd}	2^{nd}	1^{st}	0^{th}
Data	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Mask	1	1	0	0	0	0	1	1
Result								

What number is this in decimal?

Bitwise XOR – Inverting bits

We want to invert the two most and two least significant bits.

^ -61_{10}

Bit	7th	6th	5th	4th	3rd	2nd	1st	0th
Data	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Mask	1	1	0	0	0	0	1	1
Result								

What number is this in decimal?

Bitwise XOR – Inverting bits

Say we have 117 ^ -61. *What's the result?*

117_{10}
$\hat{}\ -61_{10}$

Bit	7th	6th	5th	4th	3rd	2nd	1st	0th
Data	0	1	1	1	0	1	0	1
Mask	1	1	0	0	0	0	1	1
Result								

Bitwise XOR – Inverting

Say we have 117 ^ -61. *What's the result?*

117_{10}
$\hat{}\ -61_{10}$
-74_{10}

Bit	7th	6th	5th	4th	3rd	2nd	1st	0th
Data	0	1	1	1	0	1	0	1
Mask	1	1	0	0	0	0	1	1
Result	**1**	**0**	1	1	0	1	**1**	**0**

Digital logic mysteries

- Bitwise operations can be quite useful in digital logic and can make code both more concise and faster.
  - We can represent many Boolean values with a single integer (e.g., in memory constrained environments)
  - Are fast, so can speed up computation (if you can pose your problem as a bitwise operation)
  - Interacting with low-level hardware
- Next assignment will give you more practice with bitwise operations.
- In the meantime, let's work on some mystery functions.

PRACTICE TIME - Mystery 1

- What does the following Python function do?

```
def mystery1(a, b):
        return (a ^ b) == 0
```

ANSWER - Mystery 1

- What does the following Python function do?

```
def mystery1(a, b):
        return (a ^ b) == 0
```

- The function takes two numbers (a,b), applies the bitwise XOR (a^b), turns it into a decimal and compares the result to 0. If it is equal to 0, it returns True, otherwise it returns False.
- If both numbers have n bits, we have $a_{n-1}a_{n-2} \dots a_0$
- Tests if a and b are equal.
- $5_{10} \wedge 5_{10}$ is $0101_2 \wedge 0101_2 = 0000_2 = 0_{10}$
- $5_{10} \wedge 6_{10}$ is $0101_2 \wedge 0110_2 = 0100_2 = 4_{10}$

PRACTICE TIME - Mystery 2

- What does the following Python function do? You can assume we are working with 4 bits

```
def mystery2(a):
        return (a & (1<<3)) != 0
```

ANSWER - Mystery 2

- What does the following Python function do? You can assume we are working with 4 bits

```
def mystery2(a):
        return (a & (1<<3)) != 0
```

- Checks whether a is negative by isolating the MSB.

PRACTICE TIME - Mystery 3

- What does the following Python function do?

```
def mystery3(a,b):
        return (a & (1<<b)) != 0
```

ANSWER - Mystery 3

- What does the following Python function do?

```
def mystery3(a,b):
        return (a & (1<<b)) != 0
```

- Tests if the b-th bit in a is 1.

PRACTICE TIME - Mystery 4

- What does the following Python function do?

```
def mystery4(a):
        return a | 1
```

ANSWER - Mystery 4

- What does the following Python function do?

```
def mystery4(a):
        return a | 1
```

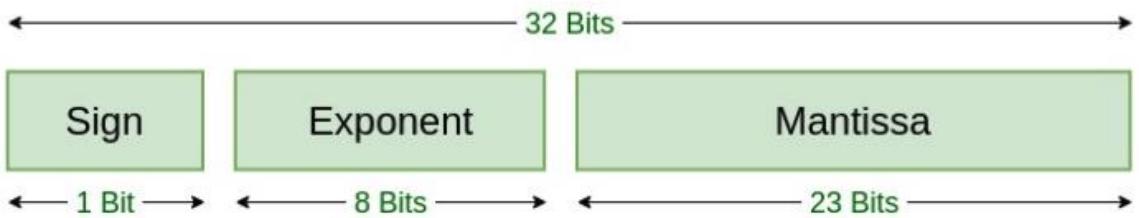
- If a is even, it returns a+1, if a is odd, it leaves it unchanged.

Beyond numbers

Representing information

- So far, we have seen that computers store information in bits and we have interpreted these bits as integers.
- But what about non-integer numbers, such as floating-point numbers, or any other type of information, such as letters, emojis, colors, sound etc?
- There are different international standards that determine how binary is **encoded** into other representations.

Floating point numbers



Single Precision
IEEE 754 Floating-Point Standard

Double Precision
IEEE 754 Floating-Point Standard

Encoding text - ASCII

- **ASCII** (American Standard Code for Information Interchange) which was established in the ‘60s and used 7 bits to represent 128 characters: These included the capital and lowercase letters of the English alphabet, digits 0-9, punctuation and special symbols like @.
  - For example, 01000011 01010011 00110101 00110001 00100001 represents the text “CS51!”
- ASCII was adopted by different computer manufacturers ensuring **interoperability**.
- Soon an 8th bit was used to expand the available set to 255 characters.
- In the US, this was used for accented characters and mathematical symbols and many other languages used these extra characters to encode their own alphabets.
- But that meant that text written in different languages encoded to gibberish when read in a different context.

Encoding text - Unicode

- In 1990s, **Unicode** expanded encoding to thousands of characters to account for all different schemes individual languages used and even includes emojis!
- The most common implementations of Unicode are **UTF-8** , which was designed for backward compatibility with ASCII, and **UTF-16**.

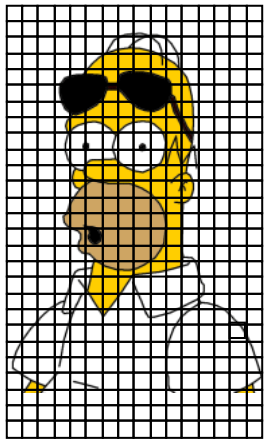
Smileys & Emotion

face-smiling

№	Code	Browser	Sample	GMail	SB	DCM	KDDI	CLDR Short Name
1	U+1F600				—	—	—	grinning face
2	U+1F603							grinning face with big eyes
3	U+1F604					—	—	grinning face with smiling eyes
4	U+1F601							beaming face with smiling eyes
5	U+1F606				—		—	grinning squinting face
6	U+1F605				—		—	grinning face with sweat
7	U+1F923			—	—	—	—	rolling on the floor laughing
8	U+1F602					—		face with tears of joy
9	U+1F642				—	—	—	slightly smiling face
10	U+1F643			—	—	—	—	upside-down face

How is an image represented?

How is an image represented?



- images are made up of pixels
- for a color image, each pixel corresponds to an RGB value (i.e. three numbers)

Encoding images

- Each pixel stores a **color** using three **color channels**: Red, Green, Blue (RGB).
- Each color channel can be encoded using binary numbers
  - Typical: 8 bits per channel, that is 24 bits per pixel. With 8 bits, we represent 0,..., 255
  - For example, 00000000 represents no red and 11111111 represents full red. Same for green, blue.
- Putting it together, we have triples of numbers to represent pixels. E.g.,
  - Pure red = $11111111\ 00000000\ 00000000 \rightarrow$ (255,0,0) – only red
  - Pure white = $11111111\ 11111111\ 11111111 \rightarrow$ (255,255,255) – all three colors
  - Pure black = $00000000\ 00000000\ 00000000 \rightarrow$ (0, 0, 0) – absence of all three colors
- An entire image would be represented as a matrix of pixels (width × height), each pixel's RGB values encoded in binary

PRACTICE TIME - Encoding images

- If we have 8 bits for each of the three color channels, how many colors can we support in a 24-bit RGB system?

ANSWER - Encoding images

- If we have 8 bits for each of the three color channels, how many colors can we support in a 24-bit RGB system?
- Each channel supports $2^8 = 256$ possible values.
- Thus, the total number of supported colors is $256 \times 256 \times 256 = 16{,}777{,}216$
- That means that a 24-bit RGB system, can represent over 16 million colors.

RGBA format

- Sometimes, the RGB format is supplemented with one more channel called the **alpha channel**.
- The alpha channel indicates how opaque a pixel is.
- By convention RGBA colors are stored in hex. For example, for alpha, 00 would be fully transparent and FF fully opaque. For colors, 00 would be lack of color and FF would be pure color.
- For example, the RGBA color #FF00FF80 is a semi-transparent purple:
  - FF_{16} stands for a full red in the red channel
  - 00_{16} stands for no green in the green channel
  - FF_{16} stands for a full blue in the blue channel
  - and $80_{16} = 128_{10}$ represents 50% opacity.

Practice Problems

Practice Problems – Problem 1

- Convert the following decimal numbers to their *unsigned* binary representation and add them:
  - $45_{10} + 27_{10}$
  - How many bits do you need to not have overflow?
- Assume 6-bit signed numbers in two's complement representation. Add them and state whether overflow occurs:
  - $101110_2 + 010101_2$
  - Convert these three numbers to decimal to double check your work.
- Add the hex numbers 0xA7+0x5C
  - Convert these three numbers to decimal to double check your work.

Practice Problems – Problem 2

- What will the results of the following operations be in binary and decimal assuming 8 bits?

- $-28_{10} \gg 2$

Practice Problems – Problem 3

- Compute the following expressions in Python:
- $101101012_2 \& 11110000_2$
- $01011011_2 | 00101101_2$
- $11001010_2 \wedge 10101100_2$

Practice Problems – Answer 1

- Convert the following decimal numbers to their *unsigned* binary representation and add them:
  - $45_{10} + 27_{10} = 101101_2 + 011011_2 = 1001000$
  - How many bits do you need to not have overflow? 7
- Assume 6-bit signed numbers in two's complement representation. Add them and state whether overflow occurs:
  - $101110_2 + 010101_2 = 000011_2$. No overflow as we drop the left most bit (1).
  - Convert these three numbers to decimal to double check your work. -18+21=-3
- Add the hex numbers $A7_{16} + 5C_{16}$
  - 103_{16}
  - Convert these three numbers to decimal to double check your work. $167_{10} + 92_{10} = 259_{10}$

Practice Problems – Answer 2

- What will the results of the following operations be in binary and decimal assuming 8 bits?

- $-28_{10} \gg 2 = 11100100_2 \gg 2 = 11111001_2 = -7_{10}$

Practice Problems – Answer 3

- Compute the following expressions in Python:
- $101101012_2 \& 11110000_2 = 10110000_2$
- $01011011_2 | 00101101_2 = 01111111_2$
- $11001010_2 \wedge 10101100_2 = 01100110_2$