

숫자의 표현과 2진수 비트 연산

00101011 00101011 00101011 00101011
01101010 01101010 01101010 01101010
101110101 101110101 101110101 101110101
11011000 11011000 11011000 11011000
10100110 10100110 10100110 10100110

00101011 00101011 00101011 00101011
01101010 01101010 01101010 01101010
101110101 101110101 101110101 101110101

숫자의 표현과
2진수 비트 연산

00101011
01101010
101110101
11011000
10100110

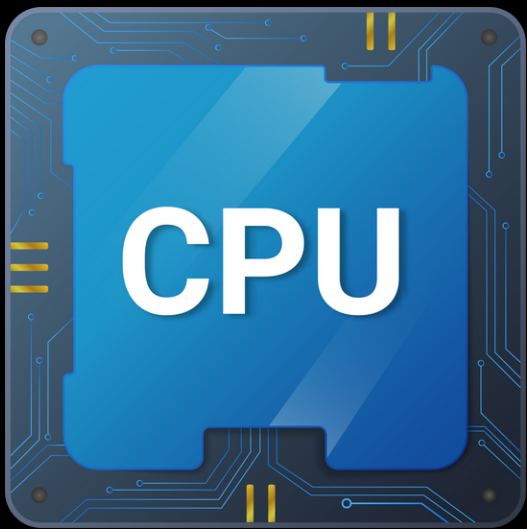
JS 진짜!
자바스크립트



10진수



5 + 6



2진수



ADD 5 6

1100 00000101 00000110

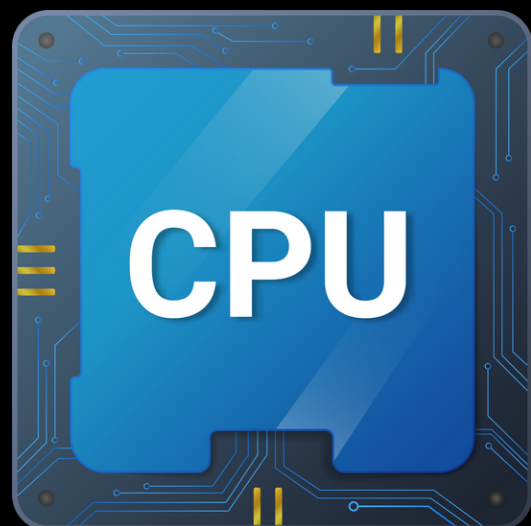
ADD 명령어가 1100 이고
8비트 컴퓨터라면



숫자의 표현과 2진수 비트 연산

00101011
01101010
101110101
11011000
10100110

JS 진짜!
자바스크립트



000

0000

2진수

8진수

16진수

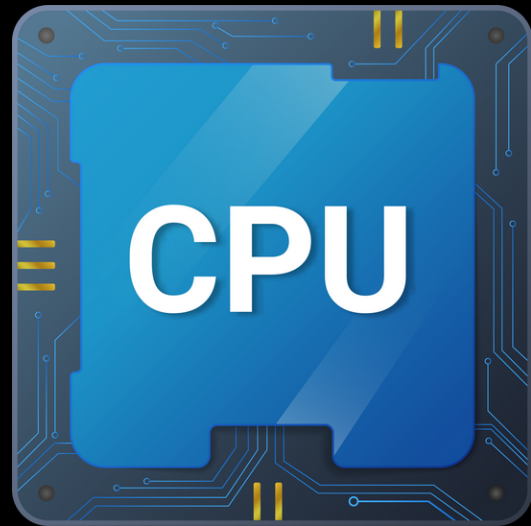


A B C D E F

숫자의 표현과 2진수 비트 연산

00101011
01101010
101110101
11011000
10100110

JS 진짜!
자바스크립트



000

0000

2진수

8진수

16진수



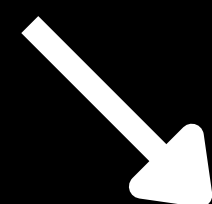
A B C D E F

10진수의 한자리 수의 범위와 올림수

$$9 = 9 \rightarrow 9 * 10^0$$

$$9 + 1 = 10 \rightarrow 1 * 10^1 + 0 * 10^0$$

$$99 + 1 = 100$$

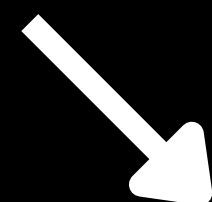

$$1 * 10^2 + 0 * 10^1 + 0 * 10^0$$

2진수의 한자리 수의 범위와 올림수

$$1 = 1 \rightarrow 1 * 2^0$$

$$1 + 1 = 10 \rightarrow 1 * 2^1 + 0 * 2^0$$

$$11 + 1 = 100$$


$$1 * 2^2 + 0 * 2^1 + 0 * 2^0$$

8진수 한자리 수의 범위는 왜 0~7까지일까?

16진수 한자리 수의 범위는 왜 0~15(0~9, A,B,C,D,E, F)까지 일까?

$$000 = 0$$

$$0 * 2^2 + 0 * 2^1 + 0 * 2^0$$

$$0000 = 0$$

$$0 * 2^3 + 0 * 2^2 + 0 * 2^1 + 0 * 2^0$$

$$111 = 7$$

$$1 * 2^2 + 1 * 2^1 + 1 * 2^0$$

$$1111 = 15 = F$$

$$1 * 2^3 + 1 * 2^2 + 1 * 2^1 + 1 * 2^0$$

8진수 한자리 수의 범위는 왜 0~7까지일까?

16진수 한자리 수의 범위는 왜 0~15(0~9, A,B,C,D,E, F)까지 일까?

000 = 0

001 = 1

010 = 2

011 = 3

100 = 4

101 = 5

110 = 6

111 = 7

0000 = 0

0001 = 1

0010 = 2

0011 = 3

0100 = 4

0101 = 5

0110 = 6

0111 = 7

1000 = 8

1001 = 9

1010 = 10 = A

1011 = 11 = B

1100 = 12 = C

1101 = 13 = D

1110 = 14 = E

1111 = 15 = F

0xFF 16진수 두 숫자로 1바이트 표현이 가능하기 때문에
대부분 16진수로 컴퓨터 수를 표현한다.

숫자의 표현과
2진수 비트 연산

00101011
01101010
101110101
11011000
10100110




자바스크립트 코드 실습

비트 연산자


& | ^ ~ << >> >>>

AND




INPUT		OUTPUT
A	B	
0	0	0
1	0	0
0	1	0
1	1	1

OR



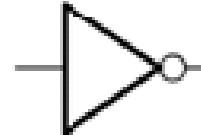
INPUT		OUTPUT
A	B	
0	0	0
1	0	1
0	1	1
1	1	1

XOR



INPUT		OUTPUT
A	B	
0	0	0
1	0	1
0	1	1
1	1	0

NOT



INPUT	OUTPUT
A	
0	1
1	0

AND(&)

OR(|)

XOR(^)

NOT(~)

비트 이동 연산자

LEFT SHIFT: $a \ll b$

- 값 a 의 비트를 왼쪽으로 b 만큼 이동한다.
- 왼쪽으로 이동할 때 맨 오른쪽에 추가되는 비트는 항상 0이다.
- $00101 \ll 2 \Rightarrow 10100 \Rightarrow 00101$ 에 2^2 을 곱한 것과 같다. ($a \ll b === a * 2^b$)

RIGHT SHIFT: $a \gg b$

- 값 a 의 비트를 오른쪽으로 b 만큼 이동한다.
- 오른쪽으로 이동할 때 왼쪽에 추가되는 비트는 MSB 이다.(양수면 0, 음수면 1)
- $00101 \gg 2 \Rightarrow 00001 \Rightarrow 00101$ 에 2^2 으로 나눈 것과 같다. (양수일 때, $a \gg b === a / 2^b$)
- 5비트 타입에서 MSB가 부호 비트라면, $10101 \gg 2 \Rightarrow 11101$

UNSIGNED RIGHT SHIFT: $a \ggg b$

- 값 a 의 비트를 오른쪽으로 b 만큼 이동한다.
- 오른쪽으로 이동할 때 왼쪽에 추가되는 비트가 항상 0이다. 그래서 부호 비트가 제거된다.
- $00101 \ggg 2 \Rightarrow 00001 \Rightarrow 00101$ 에 2^2 으로 나눈 것과 같다.

숫자의 표현과
2진수 비트 연산

00101011
01101010
101110101
11011000
10100110



자바스크립트 코드 실습

보수

Complement = 보완하다. 보충하다. 덧붙이다.

아래 10진수 연산을 뺄셈이 아닌 덧셈으로 구현하시오

$$9 - 6 = 3$$

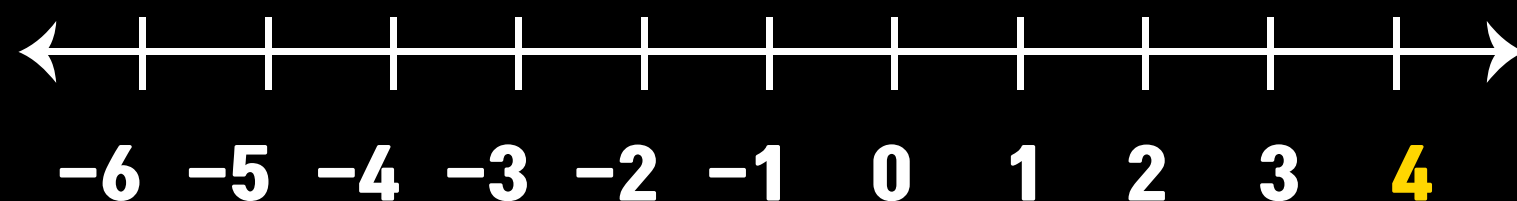
$$9 + ? = 3$$

보수 Complement = 보완하다. 보충하다. 덧붙이다.

10진수에서 아래 연산을 뺄셈이 아닌 덧셈으로 구현하시오

$$9 - 6 = 3$$

$$9 + ? = 3 \quad \text{"-6에 대해 10의 보수를 구하시오" 와 같은 질문}$$



10의 보수는 -6에서 10칸 양수쪽으로 이동한 것

$$9 + 4 = 13 = 3$$

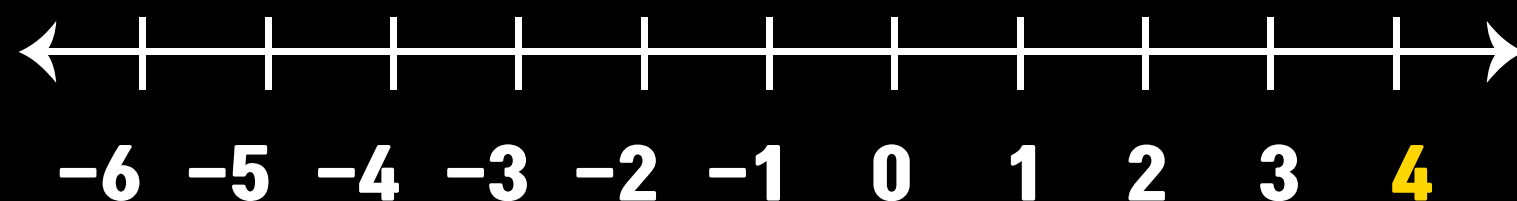
보수는 올림수를 버린다.

보수 Complement = 보완하다. 보충하다. 덧붙이다.

10진수에서 아래 연산을 뺄셈이 아닌 덧셈으로 구현하시오

$$9 - 6 = 3$$

$$9 + ? = 3 \quad \text{"-6에 대해 10의 보수를 구하시오" 와 같은 질문}$$



10의 보수는 -6에서 10칸 양수쪽으로 이동한 것

$$9 + 4 = 13 = 3$$

보수는 올림수를 버린다.

하지만
이 방법은 10진수에
만 적용할 수 있다.
좀 더 보편적인 보수
구하기를 알아 보자.

n의 보수

- n의 보수는 $n-1$ 보수에 1을 더한 값.
- $n-1$ 의 보수는 각 자릿수를 $n-1$ 에서 뺀 값.

9 - 6 = 3 에서 -6에 대한 10의 보수는
9의 보수 = $9 - 6 = 3$ 구한 후 1을 더해서
10의 보수 = $3 + 1 = 4$ 를 하면 구해진다.

2진수를 바탕으로 하는 2의 보수에도
그대로 적용된다.

1의 보수

- 1에 대한 1의 보수는
 - 0의 보수를 먼저 구한다.
 - $0 - 1 = -1$
 - 1의 보수는 위 0의 보수에 1을 더한다
 - $-1 + 1 = 0$
 - 이 값은 1비트머신에서
 - ~ 1 과 동일하다.
- 0에 대한 1의 보수는
 - 0의 보수를 먼저 구한다.
 - $0 - 0 = 0$
 - 1의 보수는 위 0의 보수에 1을 더한다
 - $0 + 1 = 1$
 - 이 값은 1비트머신에서
 - ~ 0 과 동일하다.

그래서 n의 1의 보수를 $\sim n$ 으로 구하는 것이다.

2의 보수

- 1의 보수에 + 1을 한다.
- 따라서 n 에 대한 2의 보수는
- $\sim n + 1$ 이다.

2의 보수를 사용하는 이유

		절대값	1의보수 $\sim n$	2의보수 $\sim n + 1$
MSB가 부호 비트인 3비트 number 타입 0 이면 양수 1 이면 음수	+0	000	000	000
	+1	001	001	001
	+2	010	010	010
	+3	011	011	011
	-0	100	111	1000
보수는 음수만 적용 적용하는 것이다.	-1	101	110	111
	-2	110	101	110
	-3	111	100	101

2의 보수를 사용하는 이유

ALU를 구현할 때, 덧셈으로 뺄셈을 구현할 수 있기 때문이다.
만약 부호포함 4비트 숫자라면

$$7 - 3 = 4$$

$$0111 + ? = 0100$$

-3의 2의 보수 \Rightarrow 3의 1의 보수 + 1

1의 보수: $\sim(0011) \Rightarrow 1100$

\Rightarrow 양수3에 1의 보수를 하면 부호비트가 생기므로
절대값인 3의 1의 보수를 구한다.

2의 보수: $1100 + 1 = 1101$

$$\begin{array}{r} 0111 \\ + 1101 \\ \hline = 10100 = 4 \end{array}$$

숫자의 표현과
2진수 비트 연산

00101011
01101010
101110101
11011000
10100110



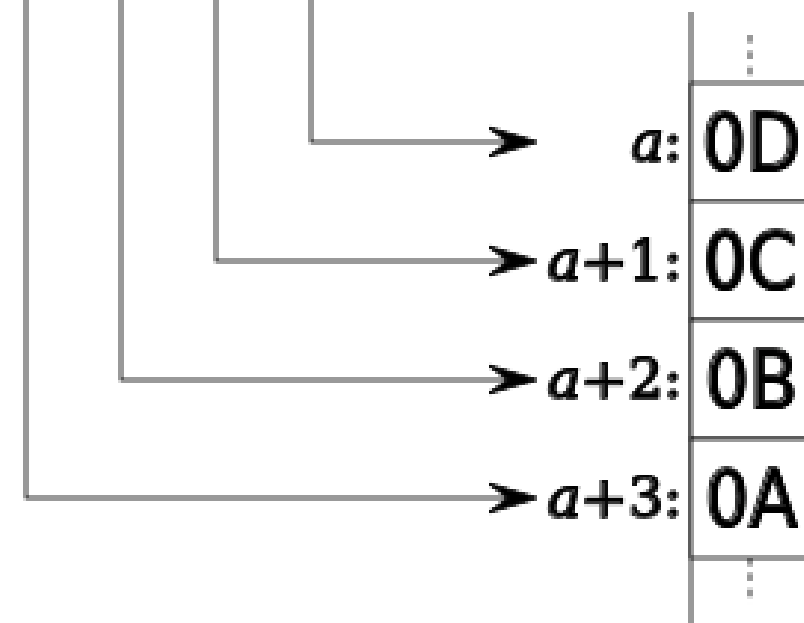
자바스크립트 코드 실습

Endian

one 32-bit integer

0A0B0C0D

arranged as
four bytes in
memory



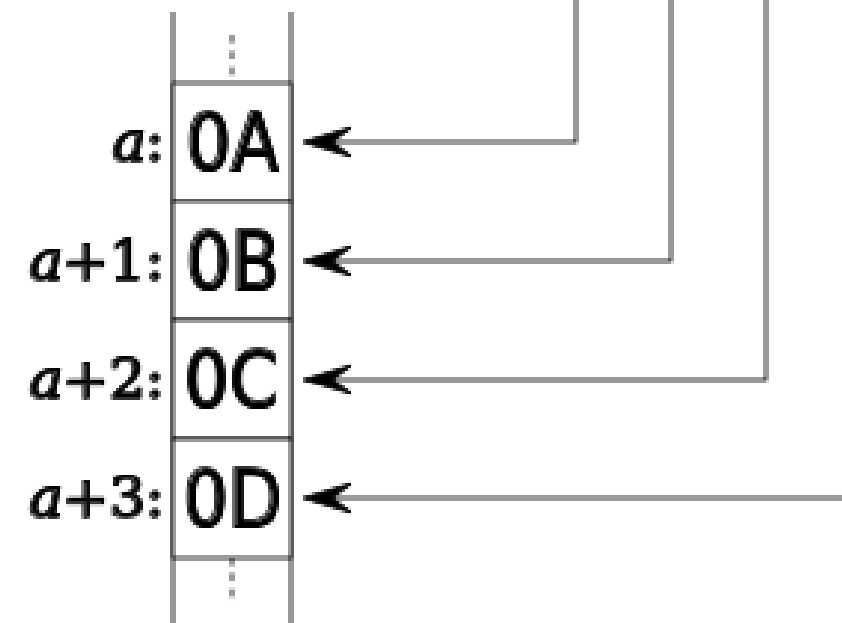
Little-endian

낮은 메모리 주소에
낮은 위치의 바이트 값을 쓴다.

arranged as
four bytes in
memory

one 32-bit integer

0A0B0C0D



Big-endian

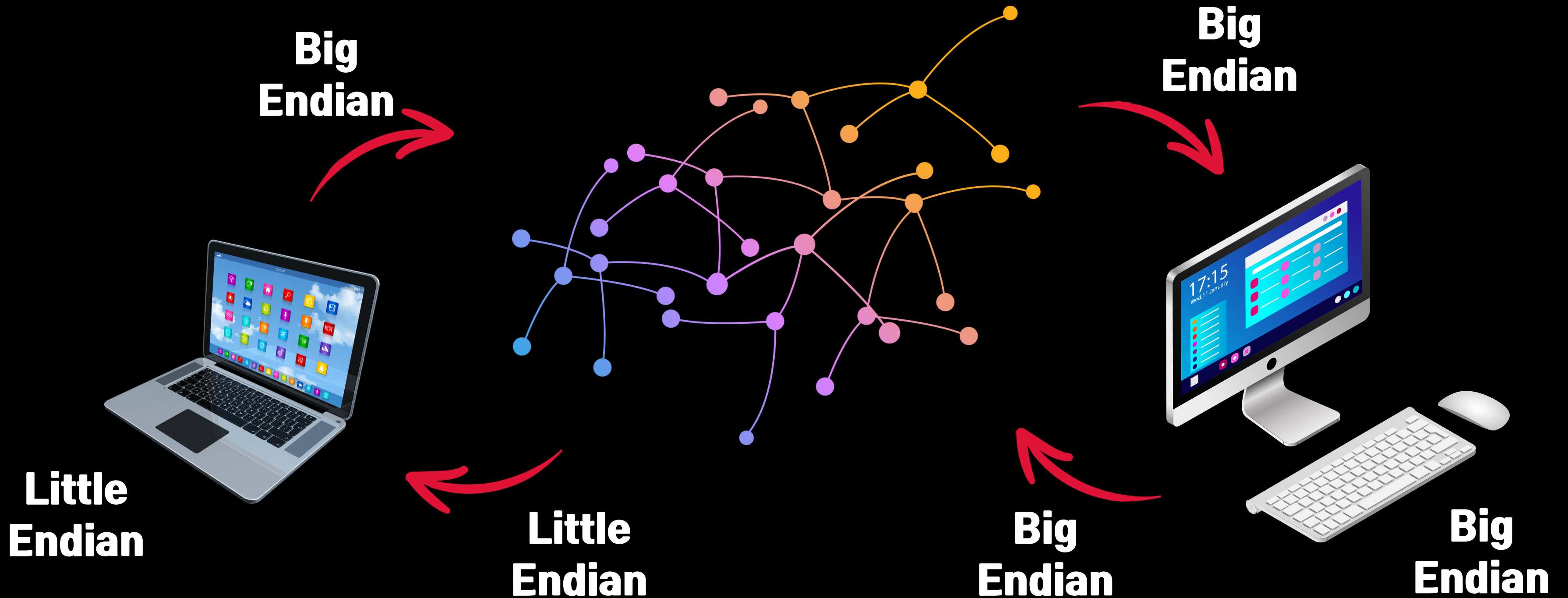
낮은 메모리 주소에
높은 위치의 바이트 값을 쓴다.

숫자의 표현과
2진수 비트 연산

00101011
01101010
101110101
11011000
10100110

JS 진짜!
자바스크립트

Endian



숫자의 표현과
2진수 비트 연산

00101011
01101010
101110101
11011000
10100110



자바스크립트 코드 실습