ALL DATA IS STORED IN BINARY FORM

ALL DATA IS STORED IN BINARY FORM

VARIABLES OF TYPE `char` ARE REPRESENTED BY A SINGLE BYTE, I.E. 8 BITS

VARIABLES OF TYPE `int` ARE USUALLY REPRESENTED BY 4 BYTES, I.E. 32 BITS

`char`, `int`, `long`, `short` ALL CONSIST OF THE STRAIGHT BINARY REPRESENTATION OF DATA

DOUBLES AND FLOATS ARE REPRESENTED BY 8-BYTE AND 4-BYTE DATA ITEMS, BUT THESE HAVE A SPECIAL FORMAT, NOT JUST THE BINARY REPRESENTATION OF A NUMBER

VARIABLES OF TYPE `char` ARE REPRESENTED BY A SINGLE BYTE, I.E. 8 BITS

CHARACTERS ARE REPRESENTED AS NUMBERS USING A CODE CALLED ASCII, WHERE EACH CHARACTER HAS A NUMBER BETWEEN 0 AND 127

TAKE THE CHARACTER 'A', WHICH HAS ASCII VALUE DECIMAL 65

THUS, CHARACTER 'A' CORRESPONDS TO AN 8-BIT BINARY REPRESENTATION OF 01000001

THUS, CHARACTER 'A' CORRESPONDS TO AN 8-BIT BINARY REPRESENTATION OF 01000001

BIT NUMBER

7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	1

BIT VALUE

THERE ARE EASY WAYS TO ACCESS AND MANIPULATE INDIVIDUAL BITS

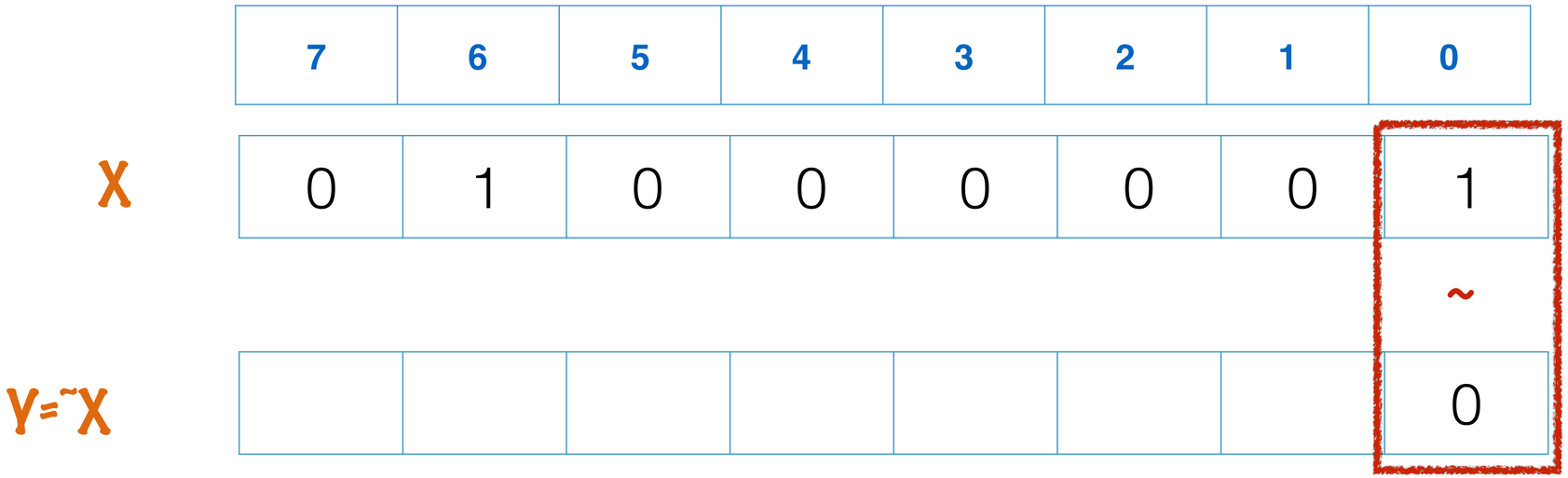
USE THE ~ OPERATOR TO REVERSE EVERY BIT IN A NUMBER

```
char x= 'A';  // 01000001
char y = ~x; // reverse every bit in x
```

	7	6	5	4	3	2	1	0
X	0	1	0	0	0	0	0	1
Y=~X	1	0	1	1	1	1	1	0

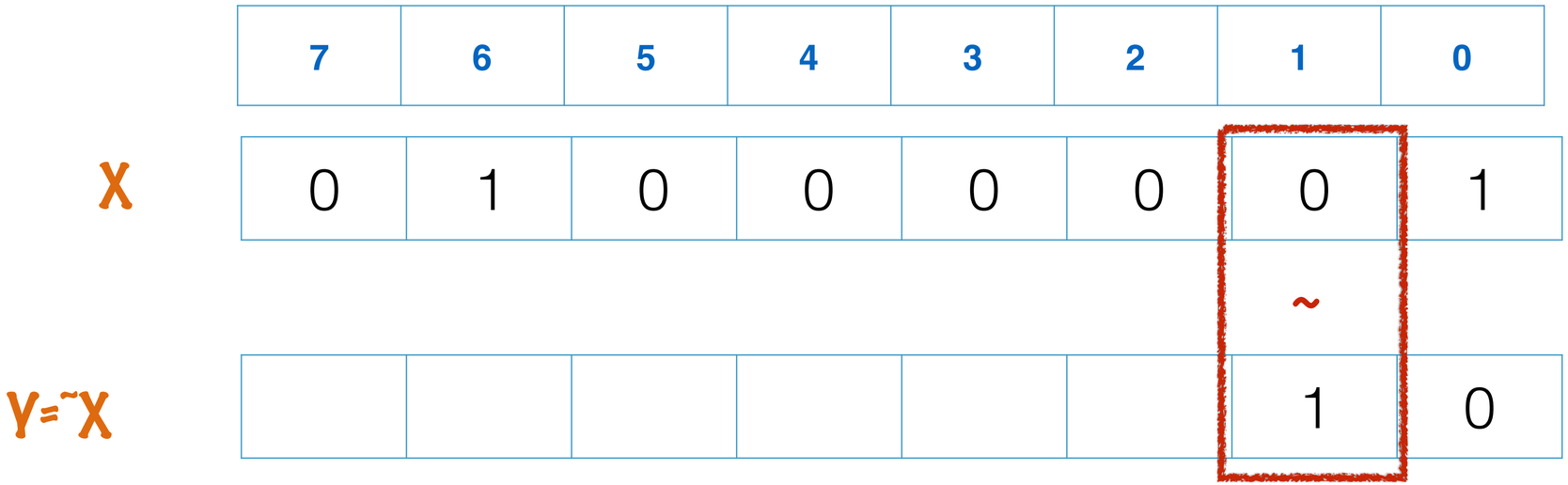
USE THE ~ OPERATOR TO REVERSE EVERY BIT IN A NUMBER

```
char x= 'A';  // 01000001
char y = ~x; // reverse every bit in x
```



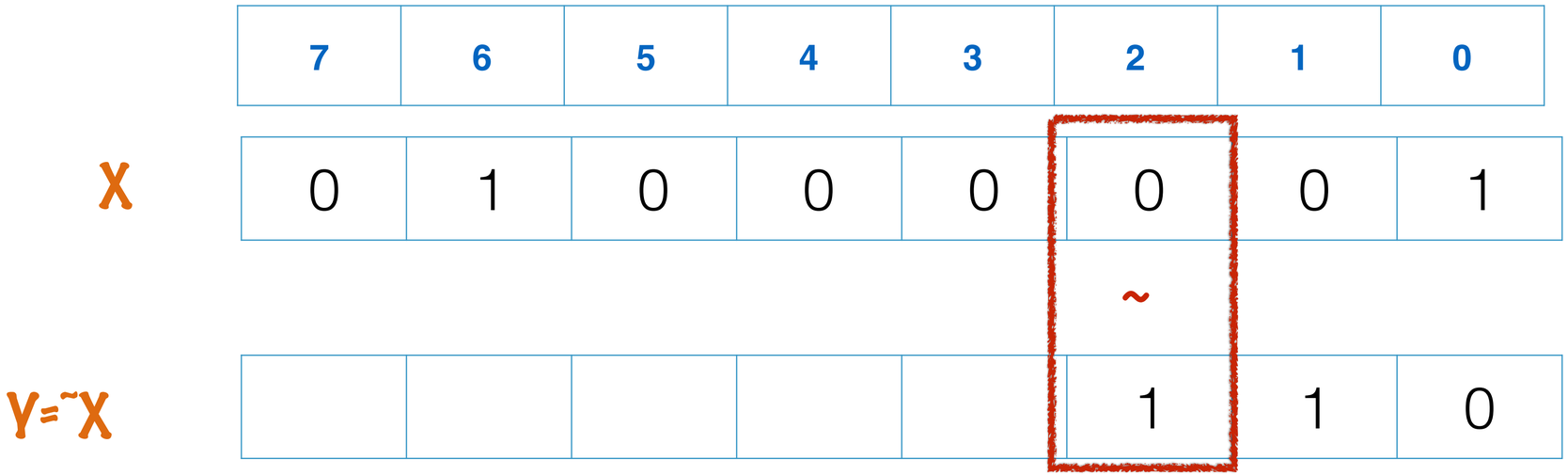
USE THE ~ OPERATOR TO REVERSE EVERY BIT IN A NUMBER

```
char x= 'A';  // 01000001
char y = ~x; // reverse every bit in x
```



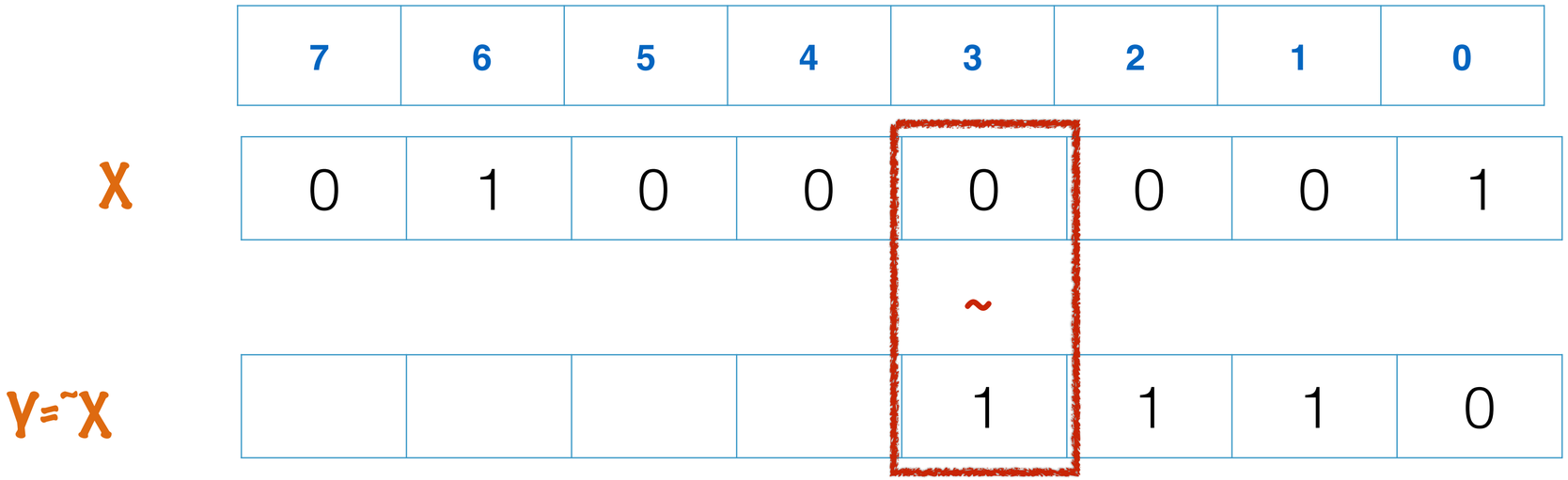
USE THE ~ OPERATOR TO REVERSE EVERY BIT IN A NUMBER

```
char x= 'A';  // 01000001
char y = ~x; // reverse every bit in x
```



USE THE ~ OPERATOR TO REVERSE EVERY BIT IN A NUMBER

```
char x= 'A';  // 01000001
char y = ~x; // reverse every bit in x
```



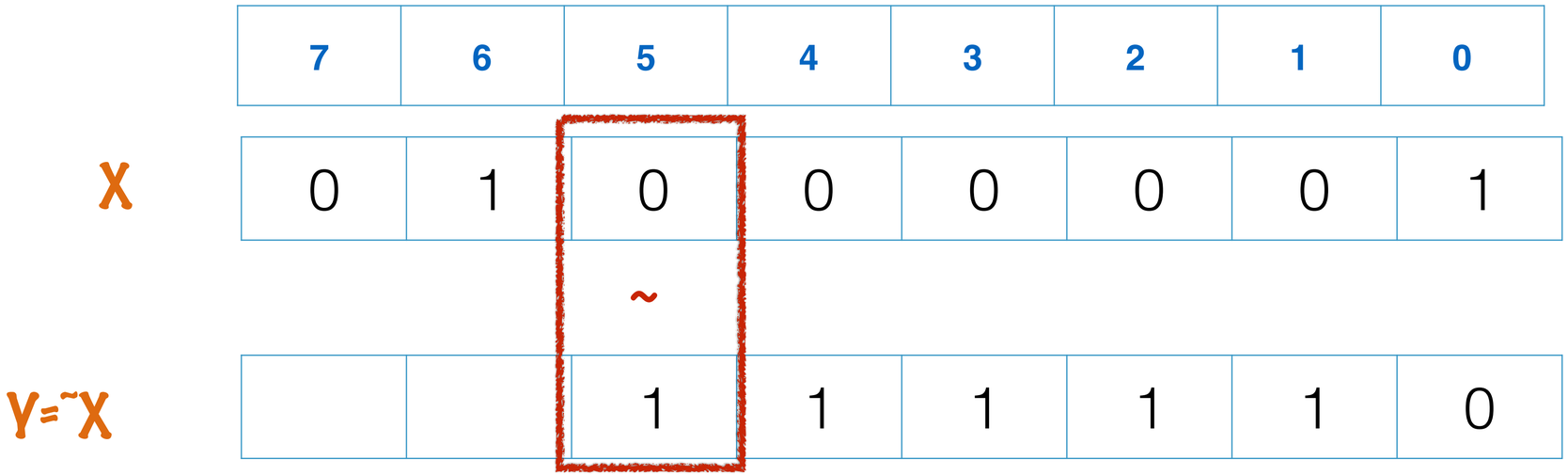
USE THE ~ OPERATOR TO REVERSE EVERY BIT IN A NUMBER

```
char x= 'A';  // 01000001
char y = ~x; // reverse every bit in x
```

	7	6	5	4	3	2	1	0
X	0	1	0	0	0	0	0	1
				~				
Y=~X				1	1	1	1	0

USE THE ~ OPERATOR TO REVERSE EVERY BIT IN A NUMBER

```
char x= 'A';  // 01000001
char y = ~x; // reverse every bit in x
```



USE THE ~ OPERATOR TO REVERSE EVERY BIT IN A NUMBER

```
char x= 'A';  // 01000001
char y = ~x; // reverse every bit in x
```

	7	6	5	4	3	2	1	0
X	0	1	0	0	0	0	0	1
		~						
Y=~X		0	1	1	1	1	1	0

USE THE ~ OPERATOR TO REVERSE EVERY BIT IN A NUMBER

```
char x= 'A';  // 01000001
char y = ~x; // reverse every bit in x
```

	7	6	5	4	3	2	1	0
X	0	1	0	0	0	0	0	1
	~							
Y=~X	1	0	1	1	1	1	1	0

USE THE & OPERATOR TO 'AND' THE BITS OF TWO NUMBERS

```
char x= 'A';    // 01000001
char y= 'L';    // 01001100
char z= x & y; // bitwise and each operator
```

	7	6	5	4	3	2	1	0
X	0	1	0	0	0	0	0	1
Y	0	1	0	0	1	1	0	0
Z = X & Y	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**

USE THE & OPERATOR TO 'AND' THE BITS OF TWO NUMBERS

```
char x= 'A';    // 01000001
char y= 'L';    // 01001100
char z= x & y; // bitwise and each operator
```

	7	6	5	4	3	2	1	0
X	0	1	0	0	0	0	0	1
Y	0	1	0	0	1	1	0	0
Z = X & Y								0

&

USE THE & OPERATOR TO 'AND' THE BITS OF TWO NUMBERS

```
char x= 'A';    // 01000001
char y= 'L';    // 01001100
char z= x & y; // bitwise and each operator
```

	7	6	5	4	3	2	1	0
X	0	1	0	0	0	0	0	1
Y	0	1	0	0	1	1	0	0
Z = X & Y							0	0

&

USE THE & OPERATOR TO 'AND' THE BITS OF TWO NUMBERS

```
char x= 'A';    // 01000001
char y= 'L';    // 01001100
char z= x & y; // bitwise and each operator
```

	7	6	5	4	3	2	1	0
X	0	1	0	0	0	0	0	1
Y	0	1	0	0	1	1	0	0
Z = X & Y						0	0	0

&

USE THE & OPERATOR TO 'AND' THE BITS OF TWO NUMBERS

```
char x= 'A';    // 01000001
char y= 'L';    // 01001100
char z= x & y; // bitwise and each operator
```

	7	6	5	4	3	2	1	0
X	0	1	0	0	0	0	0	1
					&			
Y	0	1	0	0	1	1	0	0
Z = X & Y					0	0	0	0

USE THE & OPERATOR TO 'AND' THE BITS OF TWO NUMBERS

```
char x= 'A';    // 01000001
char y= 'L';    // 01001100
char z= x & y; // bitwise and each operator
```

	7	6	5	4	3	2	1	0
X	0	1	0	0	0	0	0	1
				&				
Y	0	1	0	0	1	1	0	0
Z = X & Y				0	0	0	0	0

USE THE & OPERATOR TO 'AND' THE BITS OF TWO NUMBERS

```
char x= 'A';    // 01000001
char y= 'L';    // 01001100
char z= x & y; // bitwise and each operator
```

	7	6	5	4	3	2	1	0
X	0	1	0	0	0	0	0	1
Y	0	1	0	0	1	1	0	0
Z = X & Y			0	0	0	0	0	0

&

USE THE & OPERATOR TO 'AND' THE BITS OF TWO NUMBERS

```
char x= 'A';    // 01000001
char y= 'L';    // 01001100
char z= x & y; // bitwise and each operator
```

	7	6	5	4	3	2	1	0
X	0	1	0	0	0	0	0	1
		&						
Y	0	1	0	0	1	1	0	0
Z = X & Y		1	0	0	0	0	0	0

USE THE & OPERATOR TO 'AND' THE BITS OF TWO NUMBERS

```
char x= 'A';    // 01000001
char y= 'L';    // 01001100
char z= x & y; // bitwise and each operator
```

	7	6	5	4	3	2	1	0
X	0	1	0	0	0	0	0	1
&								
Y	0	1	0	0	1	1	0	0
Z = X & Y	0	1	0	0	0	0	0	0

USE THE | OPERATOR TO 'OR' THE BITS OF TWO NUMBERS

```
char x= 'A';    // 01000001
char y= 'L';    // 01001100
char z= x | y; // bitwise and each operator
```

	7	6	5	4	3	2	1	0
X	0	1	0	0	0	0	0	1
Y	0	1	0	0	1	1	0	0
Z = X | Y	**0**	**1**	**0**	**0**	**1**	**1**	**0**	**1**

USE THE | OPERATOR TO 'OR' THE BITS OF TWO NUMBERS

```
char x= 'A';    // 01000001
char y= 'L';    // 01001100
char z= x | y; // bitwise and each operator
```

	7	6	5	4	3	2	1	0
X	0	1	0	0	0	0	0	1
Y	0	1	0	0	1	1	0	0
Z = X | Y								1

&

USE THE | OPERATOR TO 'OR' THE BITS OF TWO NUMBERS

```
char x= 'A';    // 01000001
char y= 'L';    // 01001100
char z= x | y; // bitwise and each operator
```

	7	6	5	4	3	2	1	0
X	0	1	0	0	0	0	0	1
Y	0	1	0	0	1	1	0	0
Z = X | Y							0	1

&

USE THE | OPERATOR TO 'OR' THE BITS OF TWO NUMBERS

```
char x= 'A';    // 01000001
char y= 'L';    // 01001100
char z= x | y; // bitwise and each operator
```

	7	6	5	4	3	2	1	0
X	0	1	0	0	0	0	0	1
Y	0	1	0	0	1	1	0	0
Z = X | Y						1	0	1

&

USE THE | OPERATOR TO 'OR' THE BITS OF TWO NUMBERS

```
char x= 'A';    // 01000001
char y= 'L';    // 01001100
char z= x | y; // bitwise and each operator
```

	7	6	5	4	3	2	1	0
X	0	1	0	0	0	0	0	1
Y	0	1	0	0	1	1	0	0
Z = X | Y					1	1	0	1

&

USE THE | OPERATOR TO 'OR' THE BITS OF TWO NUMBERS

```
char x= 'A';    // 01000001
char y= 'L';    // 01001100
char z= x | y; // bitwise and each operator
```

	7	6	5	4	3	2	1	0
X	0	1	0	0	0	0	0	1
Y	0	1	0	0	1	1	0	0
Z = X | Y				0	1	1	0	1

&

USE THE | OPERATOR TO 'OR' THE BITS OF TWO NUMBERS

```
char x= 'A';    // 01000001
char y= 'L';    // 01001100
char z= x | y; // bitwise and each operator
```

	7	6	5	4	3	2	1	0
X	0	1	0	0	0	0	0	1
			&					
Y	0	1	0	0	1	1	0	0
Z = X | Y			0	0	1	1	0	1

USE THE | OPERATOR TO 'OR' THE BITS OF TWO NUMBERS

```
char x= 'A';    // 01000001
char y= 'L';    // 01001100
char z= x | y; // bitwise and each operator
```

	7	6	5	4	3	2	1	0
X	0	1	0	0	0	0	0	1
		&						
Y	0	1	0	0	1	1	0	0
Z = X | Y		1	0	0	1	1	0	1

USE THE | OPERATOR TO 'OR' THE BITS OF TWO NUMBERS

```
char x= 'A';    // 01000001
char y= 'L';    // 01001100
char z= x | y; // bitwise and each operator
```

	7	6	5	4	3	2	1	0
X	0	1	0	0	0	0	0	1
&								
Y	0	1	0	0	1	1	0	0
Z = X | Y	0	1	0	0	1	1	0	1

USE THE ^ OPERATOR TO 'XOR' THE BITS OF TWO NUMBERS

```
char x= 'A';    // 01000001
char y= 'L';    // 01001100
char z= x ^ y; // bitwise and each operator
```

	7	6	5	4	3	2	1	0
X	0	1	0	0	0	0	0	1
Y	0	1	0	0	1	1	0	0
Z = X ^ Y	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**1**

USE THE ^ OPERATOR TO 'XOR' THE BITS OF TWO NUMBERS

```
char x= 'A';    // 01000001
char y= 'L';    // 01001100
char z= x ^ y; // bitwise and each operator
```

	7	6	5	4	3	2	1	0
X	0	1	0	0	0	0	0	1
								^
Y	0	1	0	0	1	1	0	0
Z = X ^ Y	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**1**

USE THE ^ OPERATOR TO 'XOR' THE BITS OF TWO NUMBERS

```
char x= 'A';    // 01000001
char y= 'L';    // 01001100
char z= x ^ y; // bitwise and each operator
```

	7	6	5	4	3	2	1	0
X	0	1	0	0	0	0	0	1
							^	
Y	0	1	0	0	1	1	0	0
Z = X ^ Y	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**1**

USE THE ^ OPERATOR TO 'XOR' THE BITS OF TWO NUMBERS

```
char x= 'A';    // 01000001
char y= 'L';    // 01001100
char z= x ^ y; // bitwise and each operator
```

	7	6	5	4	3	2	1	0
X	0	1	0	0	0	0	0	1
						^		
Y	0	1	0	0	1	1	0	0
Z = X ^ Y	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**1**

USE THE ^ OPERATOR TO 'XOR' THE BITS OF TWO NUMBERS

```
char x= 'A';    // 01000001
char y= 'L';    // 01001100
char z= x ^ y; // bitwise and each operator
```

	7	6	5	4	3	2	1	0
X	0	1	0	0	0	0	0	1
					^			
Y	0	1	0	0	1	1	0	0
Z = X ^ Y	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**1**

USE THE ^ OPERATOR TO 'XOR' THE BITS OF TWO NUMBERS

```
char x= 'A';    // 01000001
char y= 'L';    // 01001100
char z= x ^ y; // bitwise and each operator
```

	7	6	5	4	3	2	1	0
X	0	1	0	0	0	0	0	1
				^				
Y	0	1	0	0	1	1	0	0
Z = X ^ Y	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**1**

USE THE ^ OPERATOR TO 'XOR' THE BITS OF TWO NUMBERS

```
char x= 'A';    // 01000001
char y= 'L';    // 01001100
char z= x ^ y; // bitwise and each operator
```

	7	6	5	4	3	2	1	0
X	0	1	0	0	0	0	0	1
			^					
Y	0	1	0	0	1	1	0	0
Z = X ^ Y	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**1**

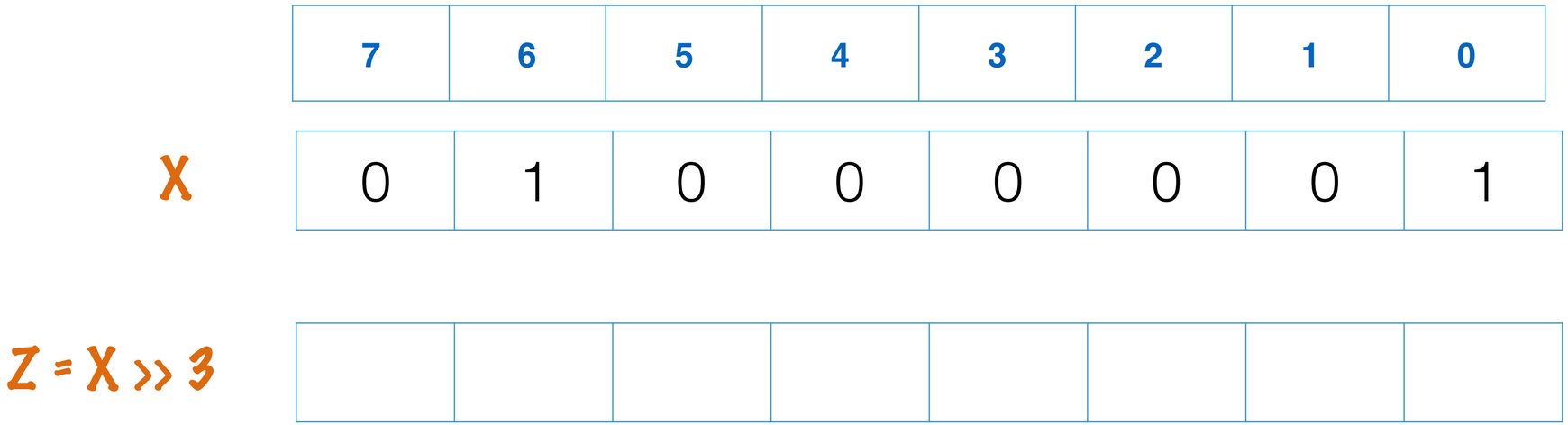
USE THE ^ OPERATOR TO 'XOR' THE BITS OF TWO NUMBERS

```
char x= 'A';    // 01000001
char y= 'L';    // 01001100
char z= x ^ y; // bitwise and each operator
```

	7	6	5	4	3	2	1	0
X	0	1	0	0	0	0	0	1
		^						
Y	0	1	0	0	1	1	0	0
Z = X ^ Y	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**1**

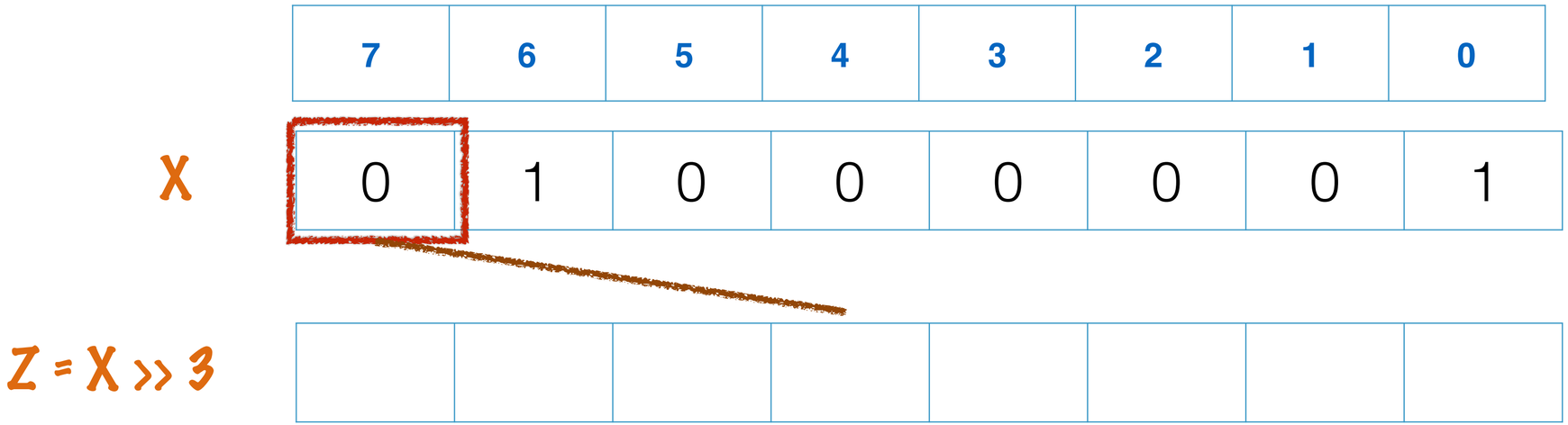
USE THE >> OPERATOR TO 'SHIFT RIGHT' THE BITS OF A NUMBER

```
char x= 'A';    // 01000001
char z= x >> 3; // shift each bit 3 places to the right
```



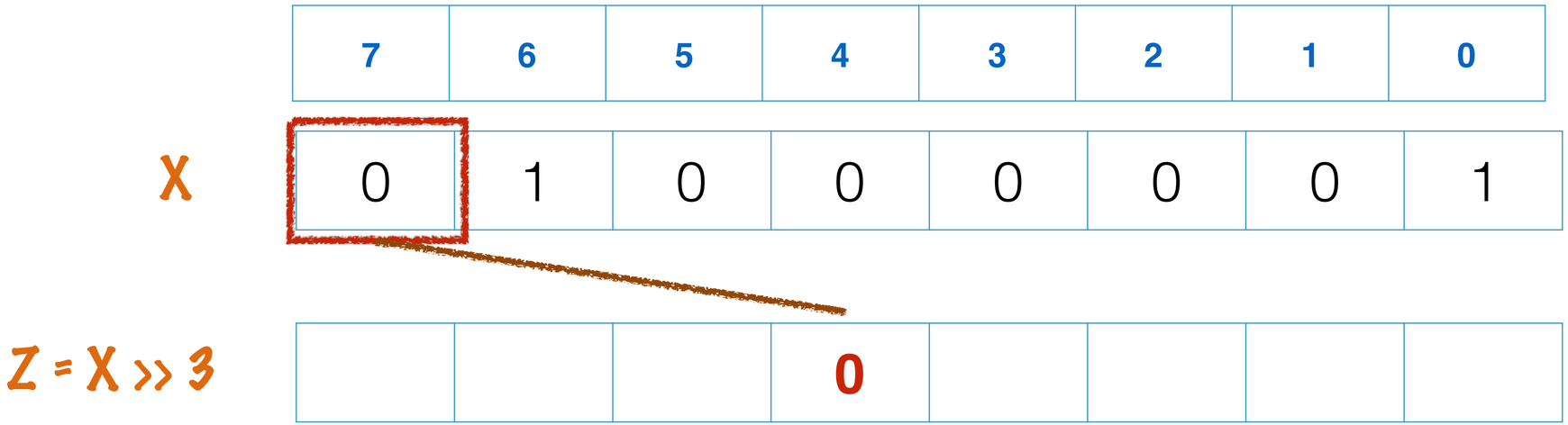
USE THE >> OPERATOR TO 'SHIFT RIGHT' THE BITS OF A NUMBER

```
char x= 'A';    // 01000001
char z= x >> 3; // shift each bit 3 places to the right
```



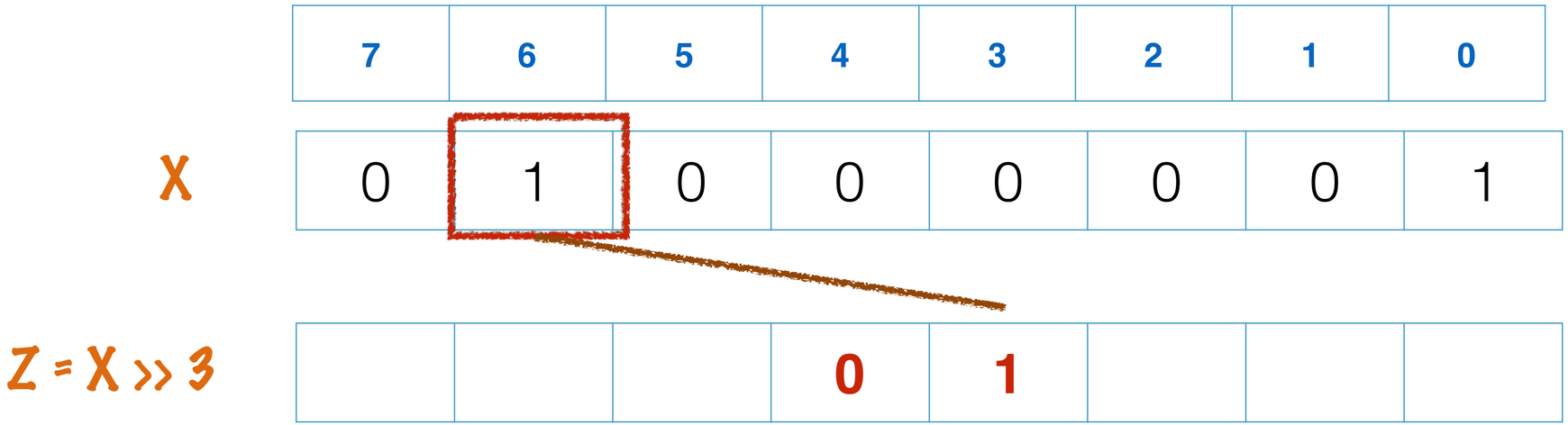
USE THE >> OPERATOR TO 'SHIFT RIGHT' THE BITS OF A NUMBER

```
char x= 'A';    // 01000001
char z= x >> 3; // shift each bit 3 places to the right
```



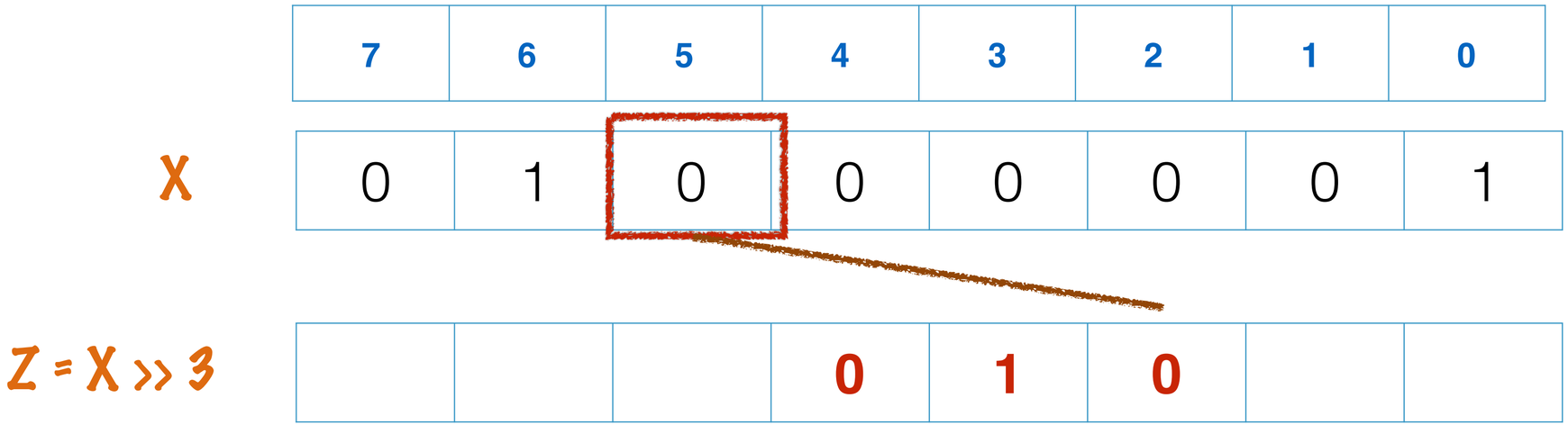
USE THE >> OPERATOR TO 'SHIFT RIGHT' THE BITS OF A NUMBER

```
char x= 'A';    // 01000001
char z= x >> 3; // shift each bit 3 places to the right
```



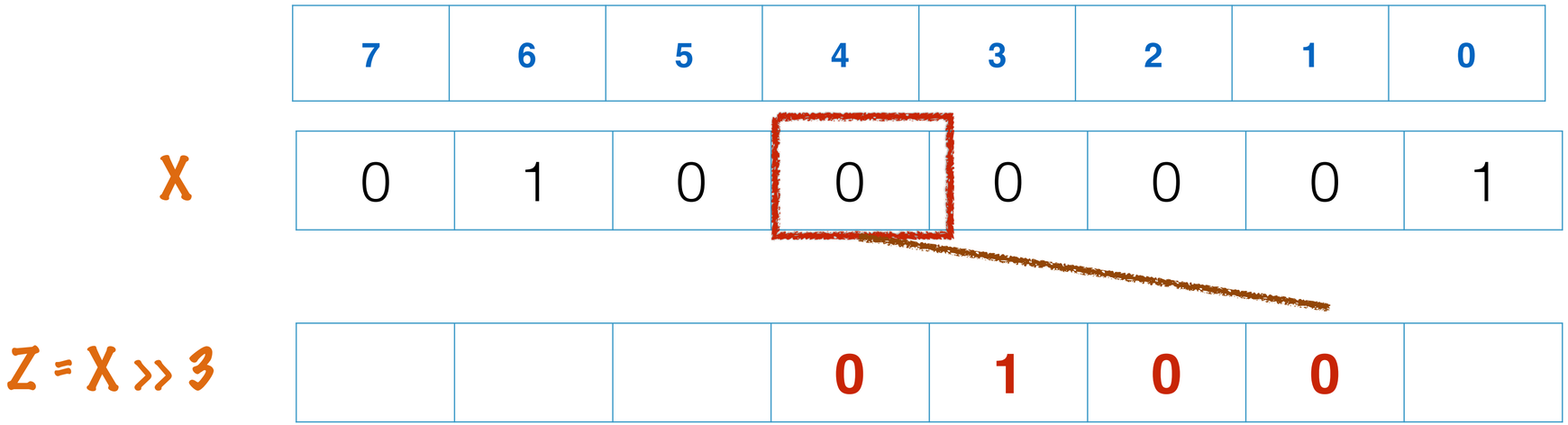
USE THE >> OPERATOR TO 'SHIFT RIGHT' THE BITS OF A NUMBER

```
char x= 'A';    // 01000001
char z= x >> 3; // shift each bit 3 places to the right
```



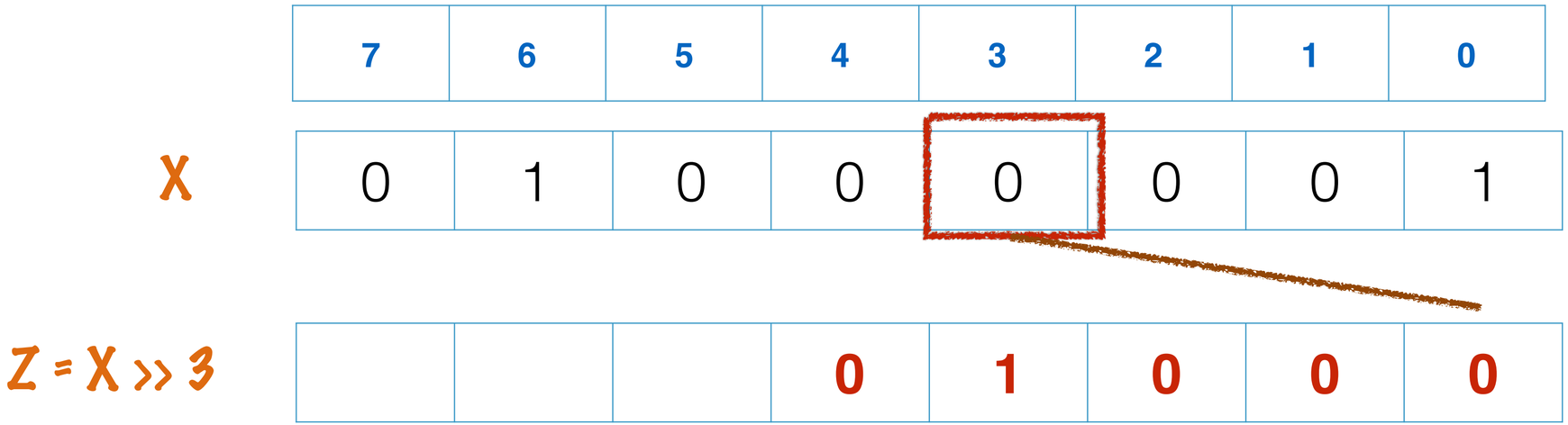
USE THE >> OPERATOR TO 'SHIFT RIGHT' THE BITS OF A NUMBER

```
char x= 'A';    // 01000001
char z= x >> 3; // shift each bit 3 places to the right
```



USE THE >> OPERATOR TO 'SHIFT RIGHT' THE BITS OF A NUMBER

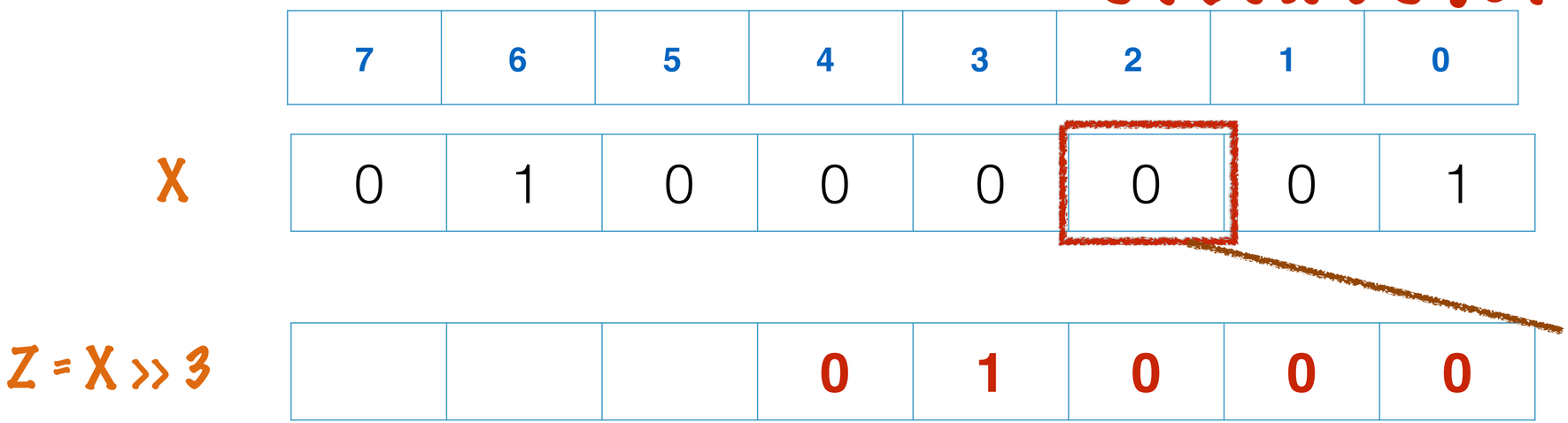
```
char x= 'A';    // 01000001
char z= x >> 3; // shift each bit 3 places to the right
```



USE THE >> OPERATOR TO 'SHIFT RIGHT' THE BITS OF A NUMBER

```
char x= 'A';    // 01000001
char z= x >> 3; // shift each bit 3 places to the right
```

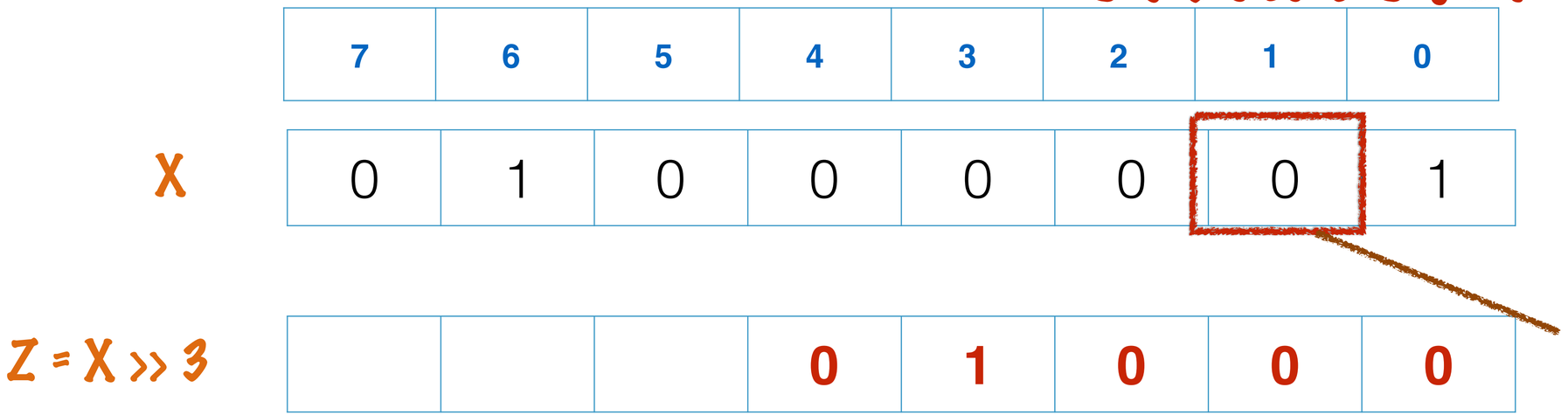
OVERFLOW!



USE THE >> OPERATOR TO 'SHIFT RIGHT' THE BITS OF A NUMBER

```
char x= 'A';    // 01000001
char z= x >> 3; // shift each bit 3 places to the right
```

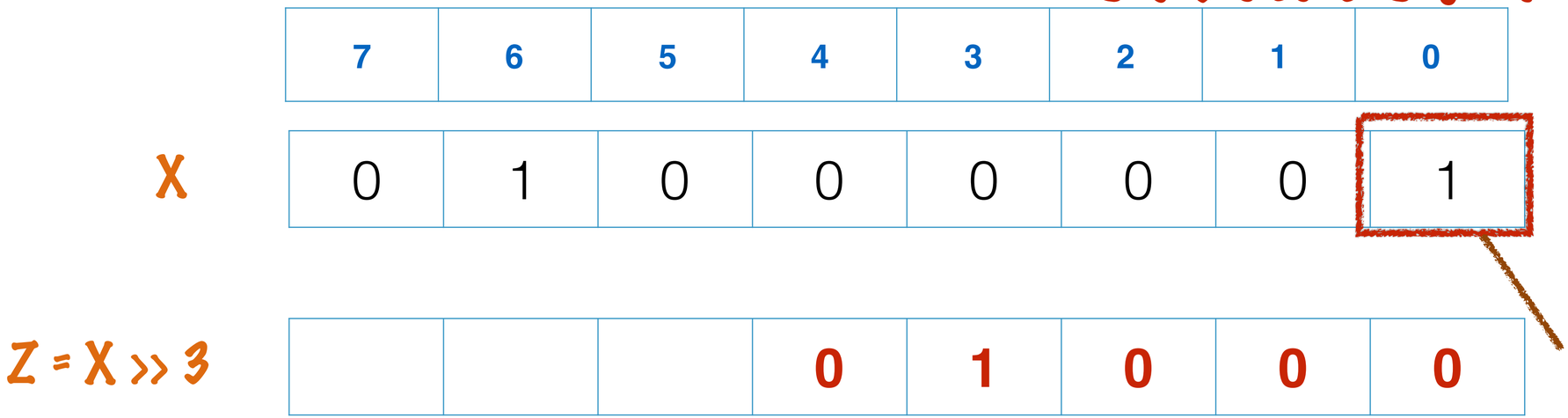
OVERFLOW!



USE THE >> OPERATOR TO 'SHIFT RIGHT' THE BITS OF A NUMBER

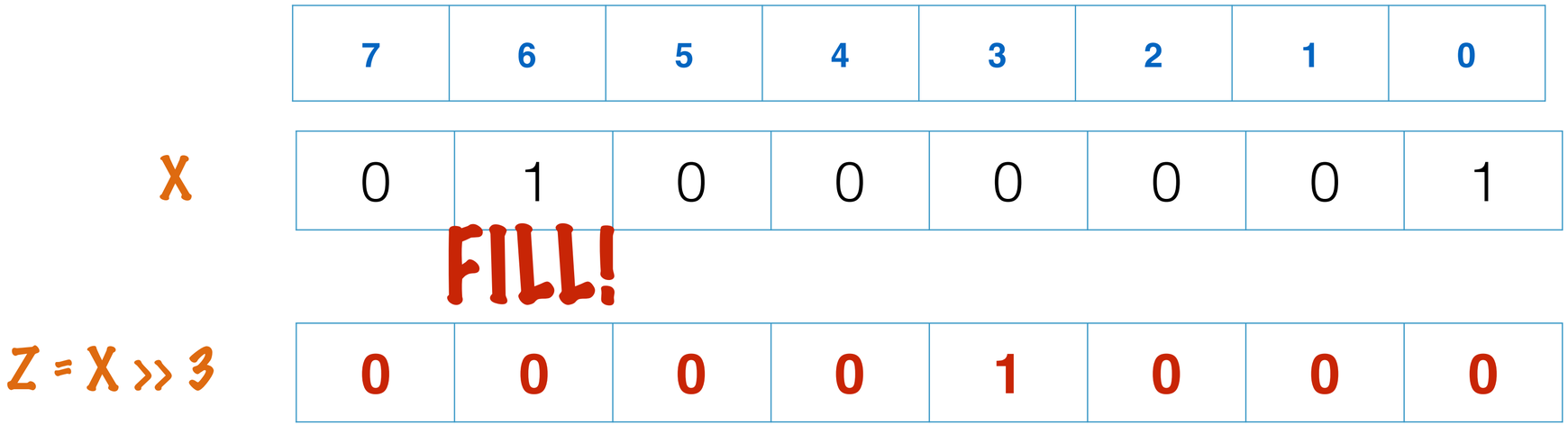
```
char x= 'A';    // 01000001
char z= x >> 3; // shift each bit 3 places to the right
```

OVERFLOW!



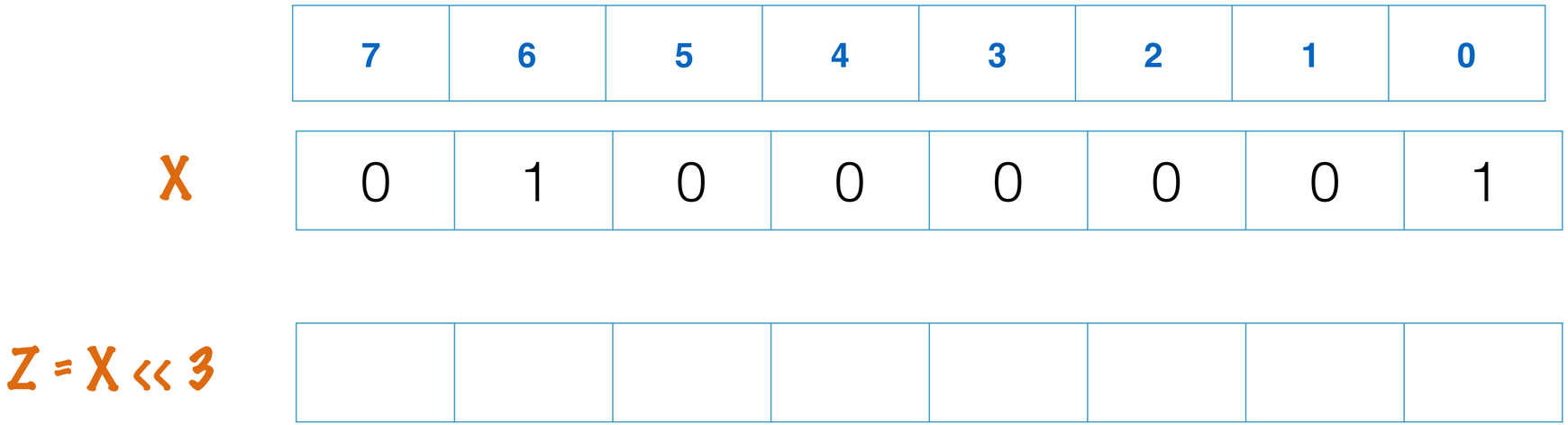
USE THE >> OPERATOR TO 'SHIFT RIGHT' THE BITS OF A NUMBER

```
char x= 'A';    // 01000001
char z= x >> 3; // shift each bit 3 places to the right
```



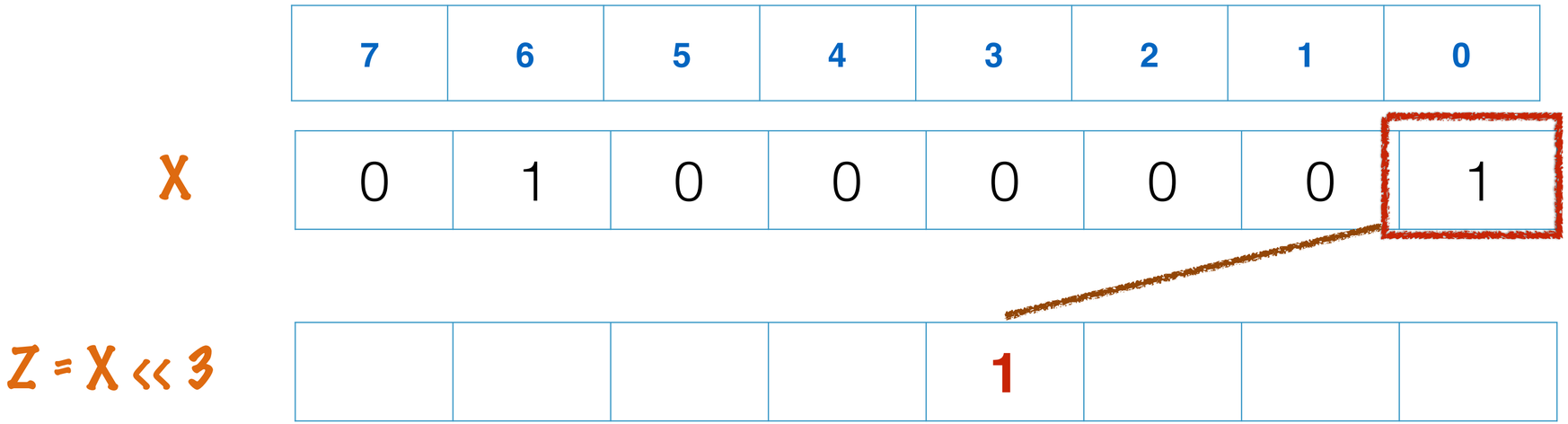
USE THE << OPERATOR TO 'SHIFT LEFT' THE BITS OF A NUMBER

```
char x= 'A';    // 01000001
char z= x << 3; // shift each bit 3 places to the right
```



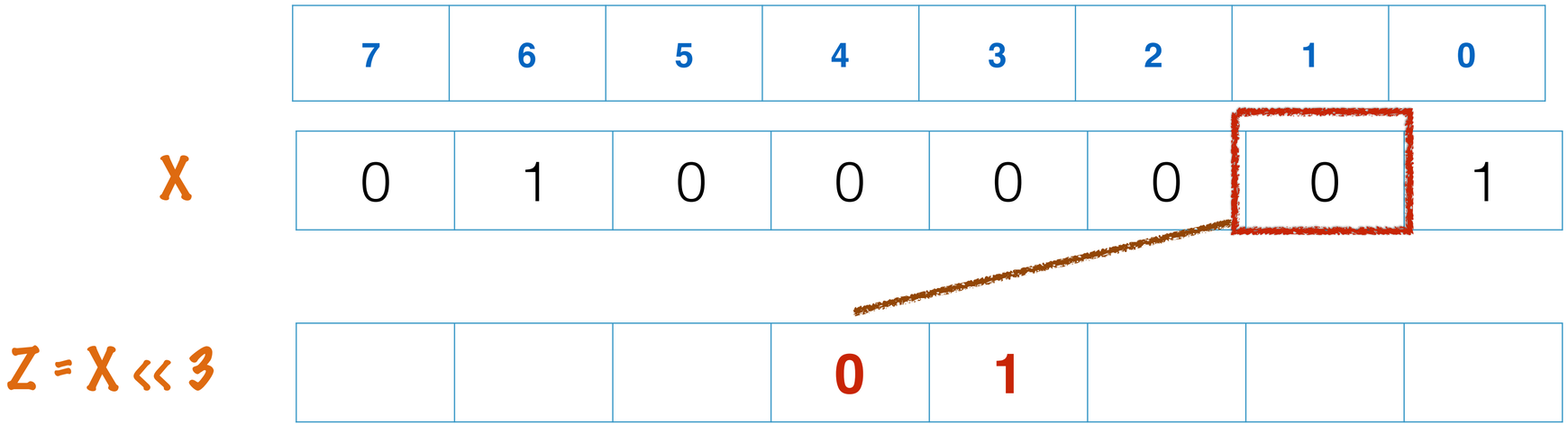
USE THE << OPERATOR TO 'SHIFT LEFT' THE BITS OF A NUMBER

```
char x= 'A';    // 01000001
char z= x << 3; // shift each bit 3 places to the right
```



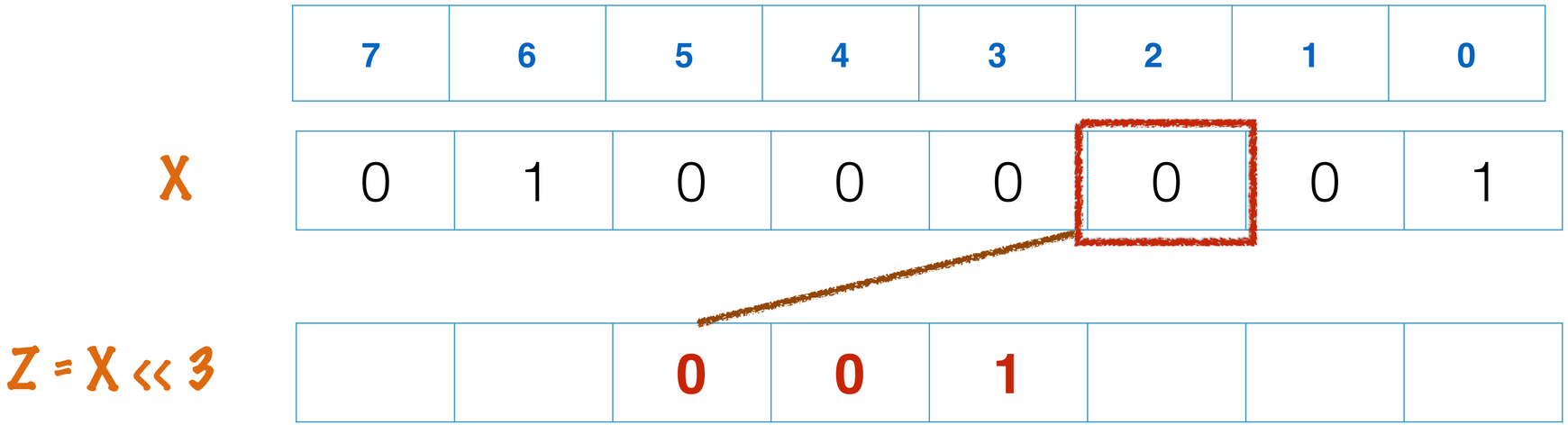
USE THE << OPERATOR TO 'SHIFT LEFT' THE BITS OF A NUMBER

```
char x= 'A';    // 01000001
char z= x << 3; // shift each bit 3 places to the right
```



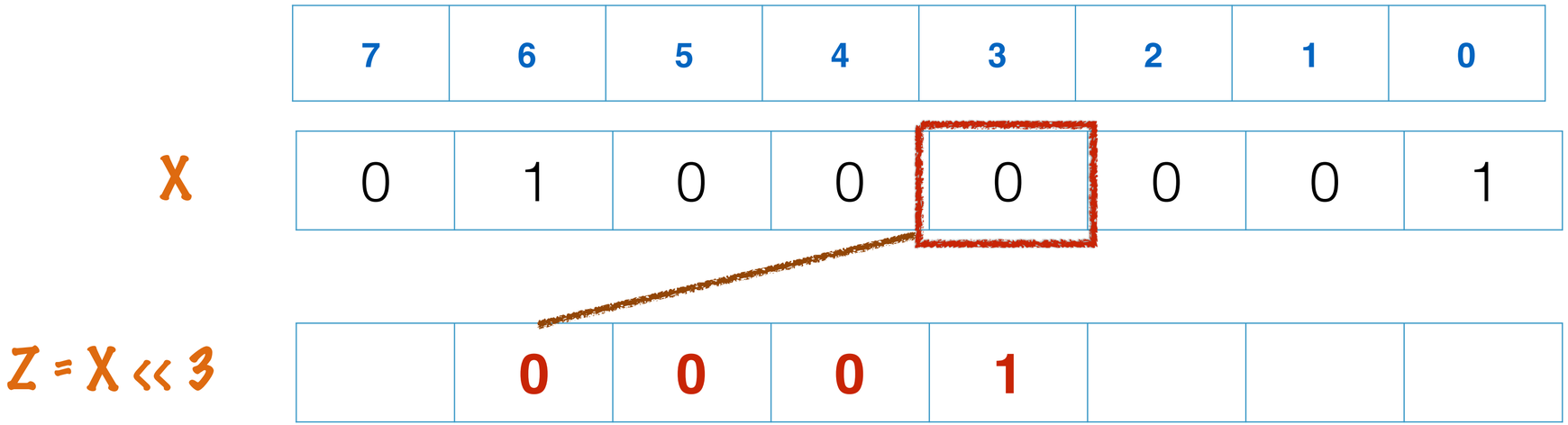
USE THE << OPERATOR TO 'SHIFT LEFT' THE BITS OF A NUMBER

```
char x= 'A';    // 01000001
char z= x << 3; // shift each bit 3 places to the right
```



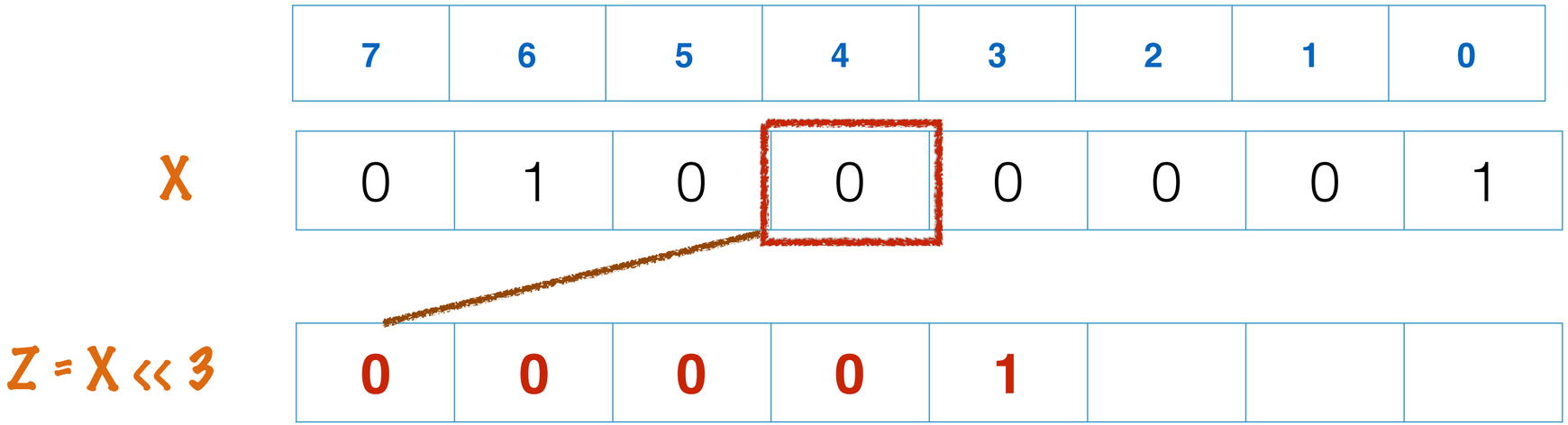
USE THE << OPERATOR TO 'SHIFT LEFT' THE BITS OF A NUMBER

```
char x= 'A';    // 01000001
char z= x << 3; // shift each bit 3 places to the right
```



USE THE << OPERATOR TO 'SHIFT LEFT' THE BITS OF A NUMBER

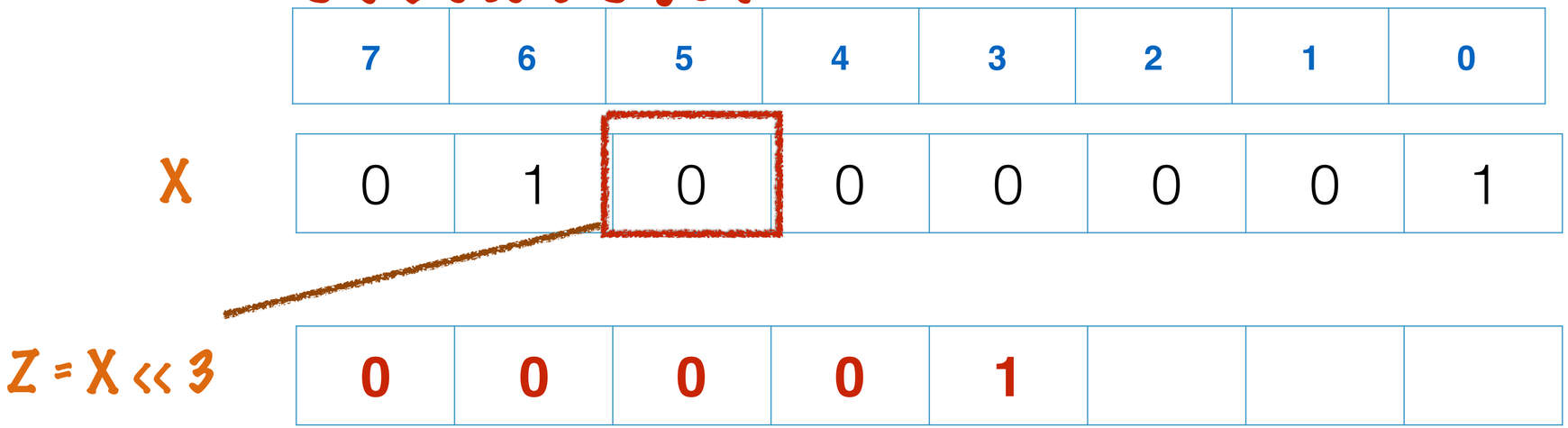
```
char x= 'A';    // 01000001
char z= x << 3; // shift each bit 3 places to the right
```



USE THE << OPERATOR TO 'SHIFT LEFT' THE BITS OF A NUMBER

```
char x= 'A';    // 01000001
char z= x << 3; // shift each bit 3 places to the right
```

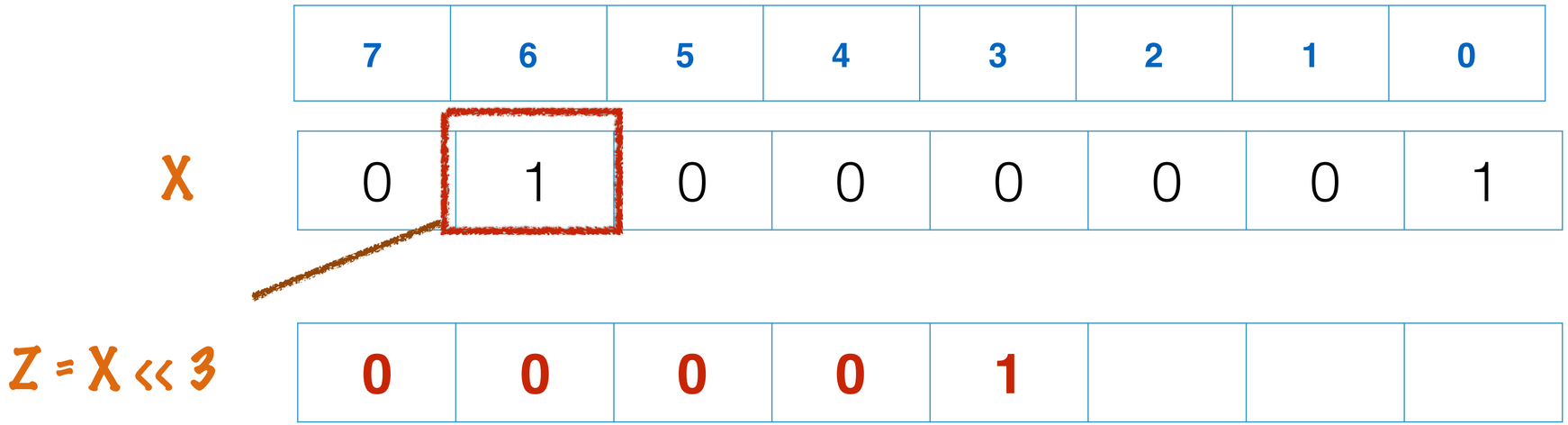
OVERFLOW!



USE THE << OPERATOR TO 'SHIFT LEFT' THE BITS OF A NUMBER

```
char x= 'A';    // 01000001
char z= x << 3; // shift each bit 3 places to the right
```

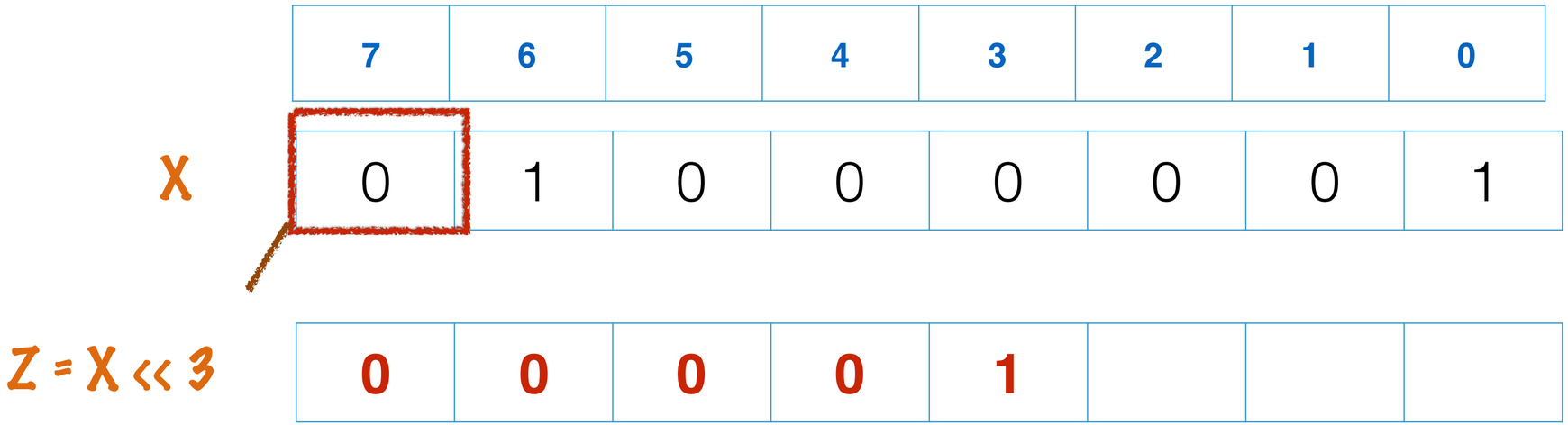
OVERFLOW!



USE THE << OPERATOR TO 'SHIFT LEFT' THE BITS OF A NUMBER

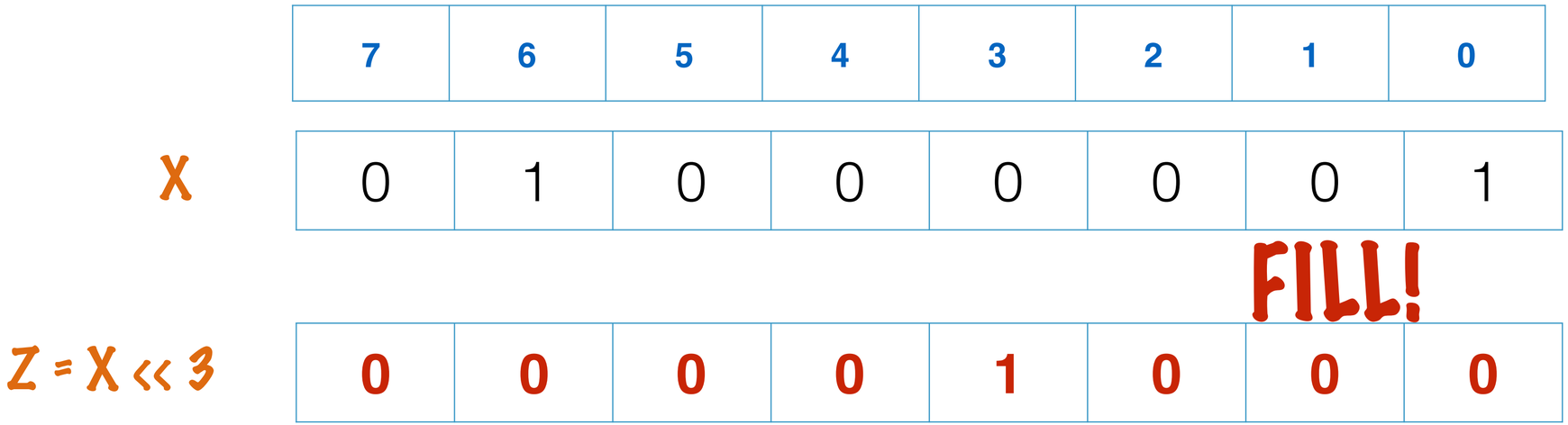
```
char x= 'A';    // 01000001
char z= x << 3; // shift each bit 3 places to the right
```

OVERFLOW!



USE THE << OPERATOR TO 'SHIFT LEFT' THE BITS OF A NUMBER

```
char x= 'A';    // 01000001
char z= x << 3; // shift each bit 3 places to the right
```



THE LEFT AND RIGHT SHIFT OPERATORS ARE VERY DIFFICULT TO USE CORRECTLY

THE OVERFLOW ISSUE WE JUST SAW IS ONE REASON

THE FILL IS ALSO SYSTEM-DEPENDENT, ESPECIALLY FOR SIGNED NUMBERS

BIT MANIPULATION IS
INCREDIBLY POWERFUL

BIT MANIPULATION IS INCREDIBLY POWERFUL

```
// find if the n-th bit of a byte is 1
int n = 3;
int value = 2345;
int checkBit = 1 << n;
bool isOne = (value & checkBit) == checkBit;
```

BIT MANIPULATION IS INCREDIBLY POWERFUL

```
// find if the n-th bit of a byte is 0
int n = 3;
int value = 2345;
int checkBit = 1 << n;
bool isZero = !((value & checkBit) == checkBit);
```

// count the number of 1s in a number

```c
#include<stdio.h>

/* Function to get no of set bits in binary
   representation of passed binary no. */
int countSetBits(int n)
{
    unsigned int count = 0;
    while (n)
    {
      n &= (n-1) ;
      count++;
    }
    return count;
}

/* Program to test function countSetBits */
int main()
{
    int i = 9;
    printf("%d", countSetBits(i));
    getchar();
    return 0;
}
```

BIT MANIPULATION IS INCREDIBLY POWERFUL

KEY INSIGHT: SUBTRACTING 1 FROM A NUMBER TOGGLES ALL BITS FROM THE RIGHT UNTIL THE FIRST 1

//reverse the bits of a number

```
unsigned int reverseBits(unsigned int num)
{
    unsigned int count = sizeof(num) * 8 - 1;
    unsigned int reverse_num = num;

    num >>= 1;
    while(num)
    {
       reverse_num <<= 1;
       reverse_num |= num & 1;
       num >>= 1;
       count--;
    }
    reverse_num <<= count;
    return reverse_num;
}

int main()
{
    unsigned int x = 1;
    printf("%u", reverseBits(x));
    getchar();
}
```

BIT MANIPULATION IS INCREDIBLY POWERFUL