Programming for Engineers

Bit Manipulation

UNIVERSITY AT ALBANY
State University of New York

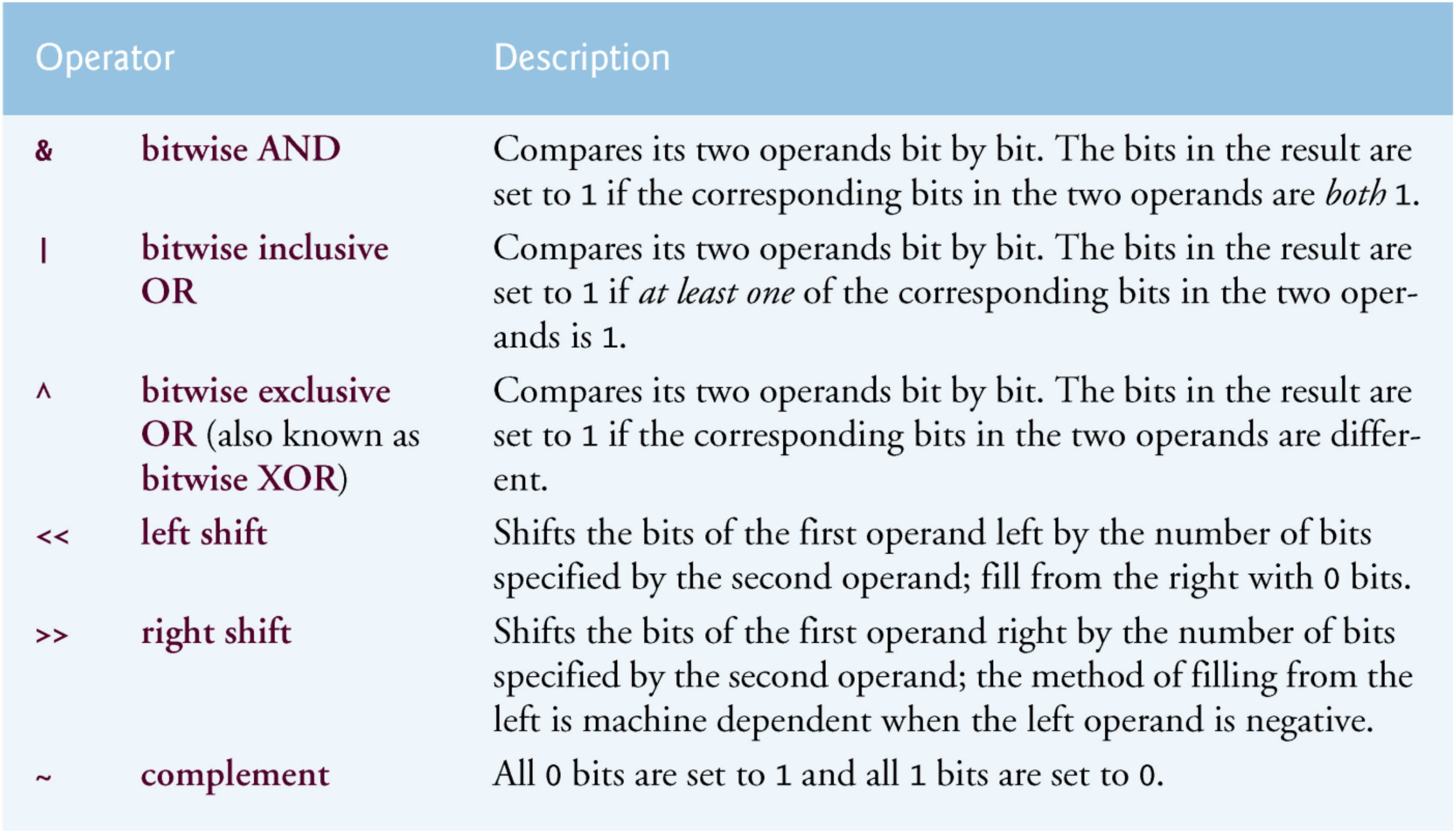
ICEN 200– Spring 2018

Prof. Dola Saha

Bitwise Operation

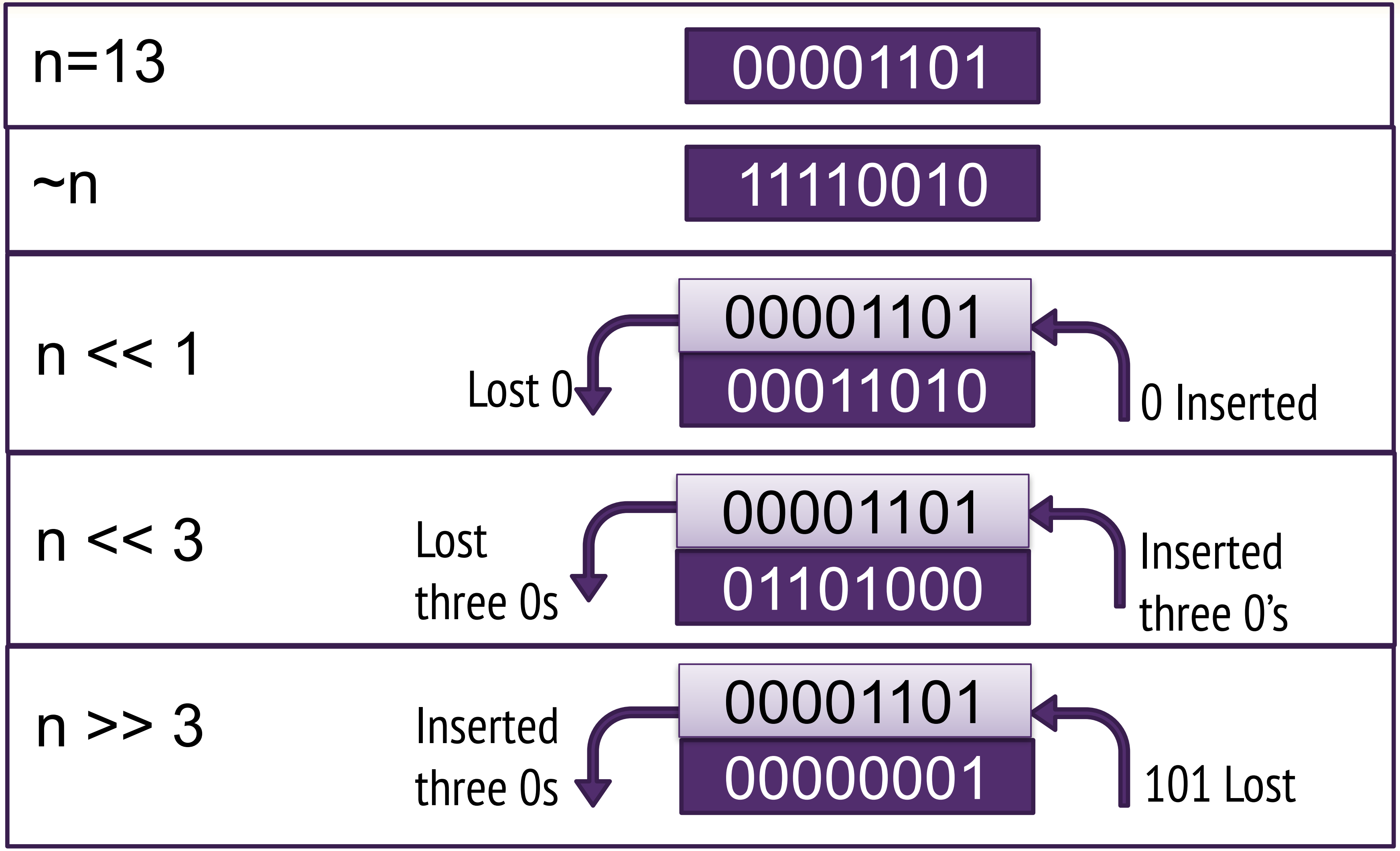
- Computers represent all data internally as sequences of bits.
- Each bit can assume the value 0 or the value 1.
- The bitwise operators are used to manipulate the bits of integral operands both signed and unsigned.
- Unsigned integers are normally used with the bitwise operators.
- Bitwise manipulations are machine dependent.

Bitwise Operator



Operator		Description
&	bitwise AND	Compares its two operands bit by bit. The bits in the result are set to 1 if the corresponding bits in the two operands are *both* 1.
|	bitwise inclusive OR	Compares its two operands bit by bit. The bits in the result are set to 1 if *at least one* of the corresponding bits in the two operands is 1.
^	bitwise exclusive OR (also known as bitwise XOR)	Compares its two operands bit by bit. The bits in the result are set to 1 if the corresponding bits in the two operands are different.
<<	left shift	Shifts the bits of the first operand left by the number of bits specified by the second operand; fill from the right with 0 bits.
>>	right shift	Shifts the bits of the first operand right by the number of bits specified by the second operand; the method of filling from the left is machine dependent when the left operand is negative.
~	complement	All 0 bits are set to 1 and all 1 bits are set to 0.

Bitwise Operation Example



n=13		00001101	
~n		11110010	
n << 1	Lost 0	00001101 00011010	0 Inserted
n << 3	Lost three 0s	00001101 01101000	Inserted three 0’s
n >> 3	Inserted three 0s	00001101 00000001	101 Lost

Bitwise Operation Example

Bit 1	Bit 2	Bit 1 & Bit 2
0	0	0
0	1	0
1	0	0
1	1	1

n	00001101
m	01010101
n&m	00000101

Bit 1	Bit 2	Bit 1 | Bit 2
0	0	0
0	1	1
1	0	1
1	1	1

n	00001101
m	01010101
n|m	01011101

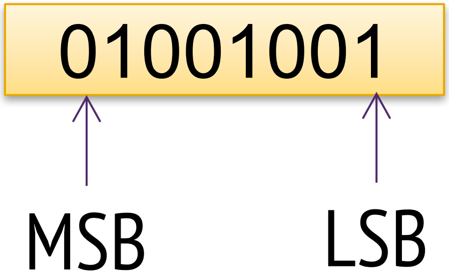
Bit 1	Bit 2	Bit 1 ^ Bit 2
0	0	0
0	1	1
1	0	1
1	1	0

n	00001101
m	01010101
n^m	01011000

Bit Order

- Most Significant Bit (MSB)

- Least Significant Bit (LSB)



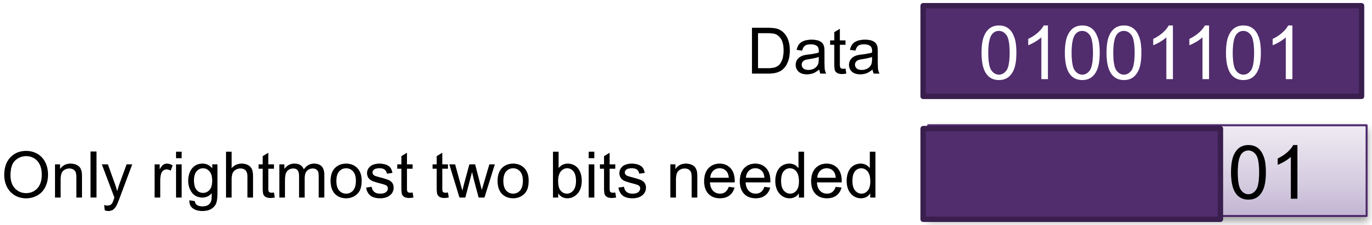
01001001

MSB

LSB

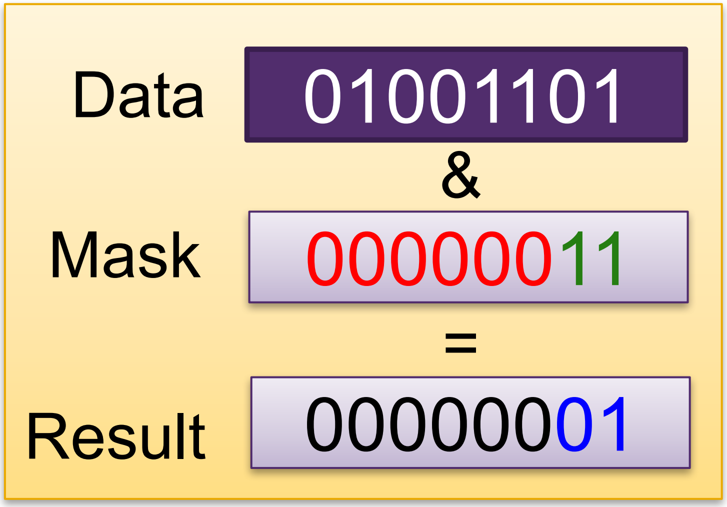
Field Extraction: Mask

- ANDing a bit with 0 produces 0.
- ANDing a bit with 1 produces the original bit.



Data 01001101

Only rightmost two bits needed 01



Data 01001101

&

Mask 00000011

=

Result 00000001

Field Extraction: Mask and Shift

Data 01001001

2^{nd} & 3^{rd} bits needed 11

Data 01001001

&

Mask 00001100

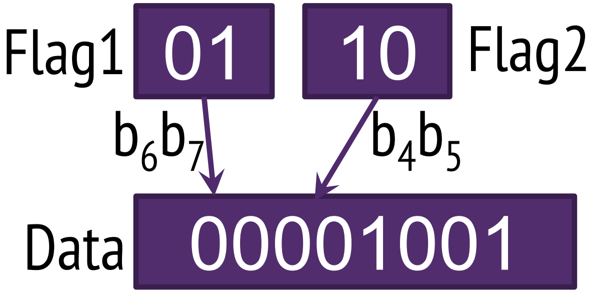
=

00001000

>> 2

Result 00000010

Field Insertion



Flag1 00000001

&

Mask 00000011

=

00000001

<< 6

Shifted 01000000

|

Data 00001001

=

01001001

Make sure that you have only two bits in Flag1

Flip bits

Data 01001001

Flip 2nd & 3rd bits 01000101

Data 01001001

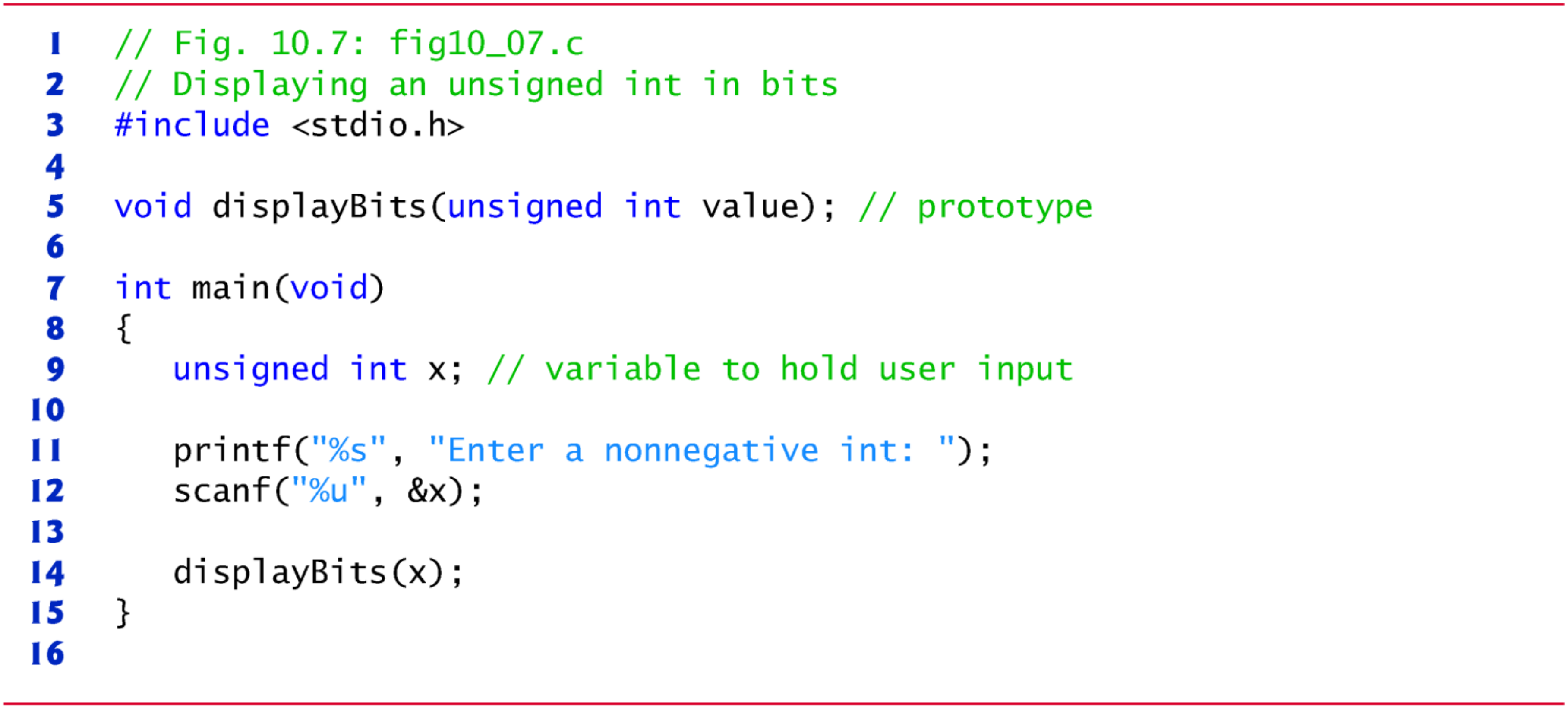
^

Mask 00001100

=

01000101

Display Bits Example (1)



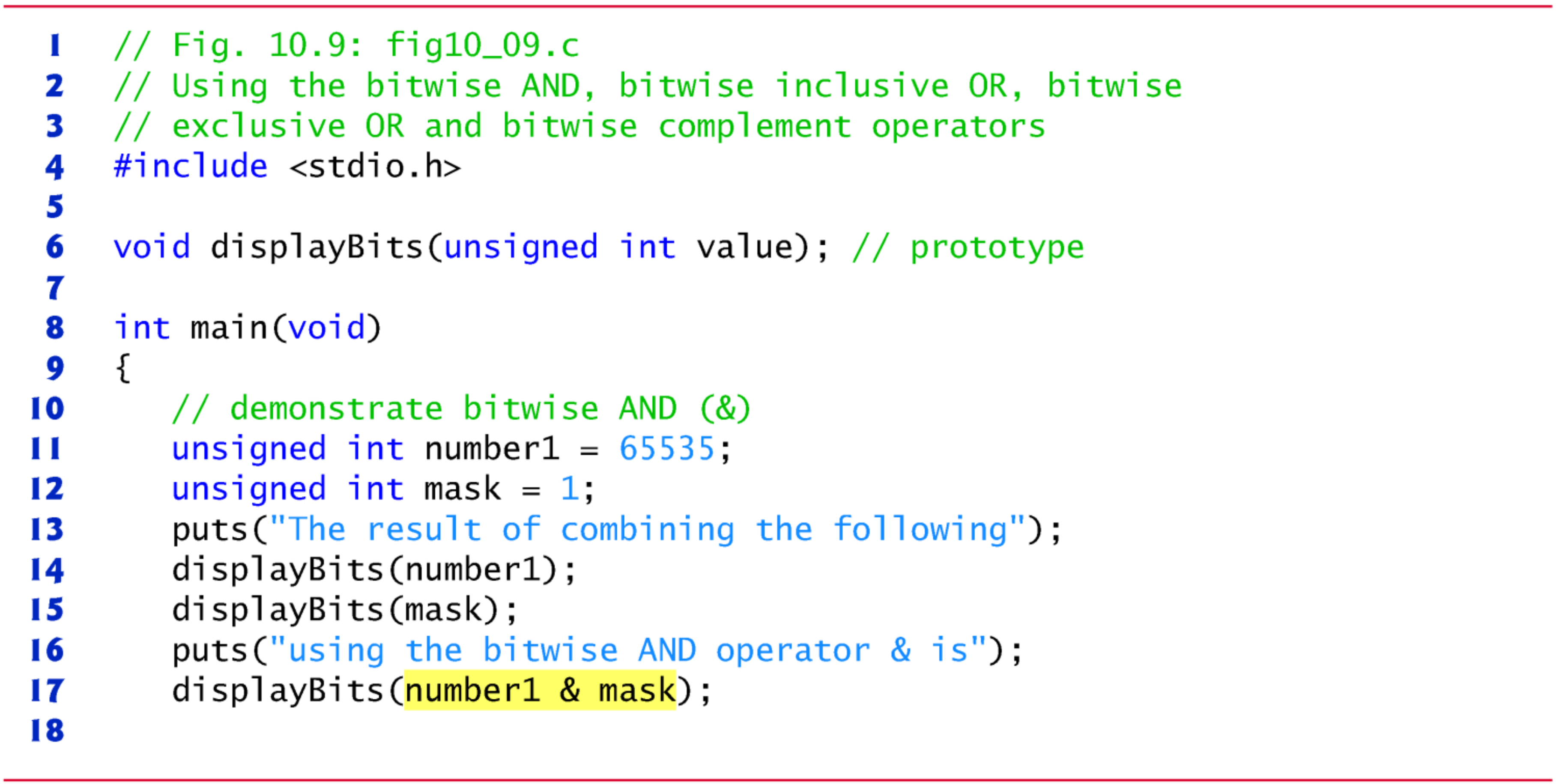
```c
// Fig. 10.7: fig10_07.c
// Displaying an unsigned int in bits
#include <stdio.h>

void displayBits(unsigned int value); // prototype

int main(void)
{
   unsigned int x; // variable to hold user input

   printf("%s", "Enter a nonnegative int: ");
   scanf("%u", &x);

   displayBits(x);
}

```

Display Bits Example (2)

```c
// display bits of an unsigned int value
void displayBits(unsigned int value)
{
   // define displayMask and left shift 31 bits
   unsigned int displayMask = 1 << 31;

   printf("%10u = ", value);

   // loop through bits
   for (unsigned int c = 1; c <= 32; ++c) {
      putchar(value & displayMask ? '1' : '0');
      value <<= 1; // shift value left by 1

      if (c % 8 == 0) { // output space after 8 bits
         putchar(' ');
      }
   }

   putchar('\n');
}
```

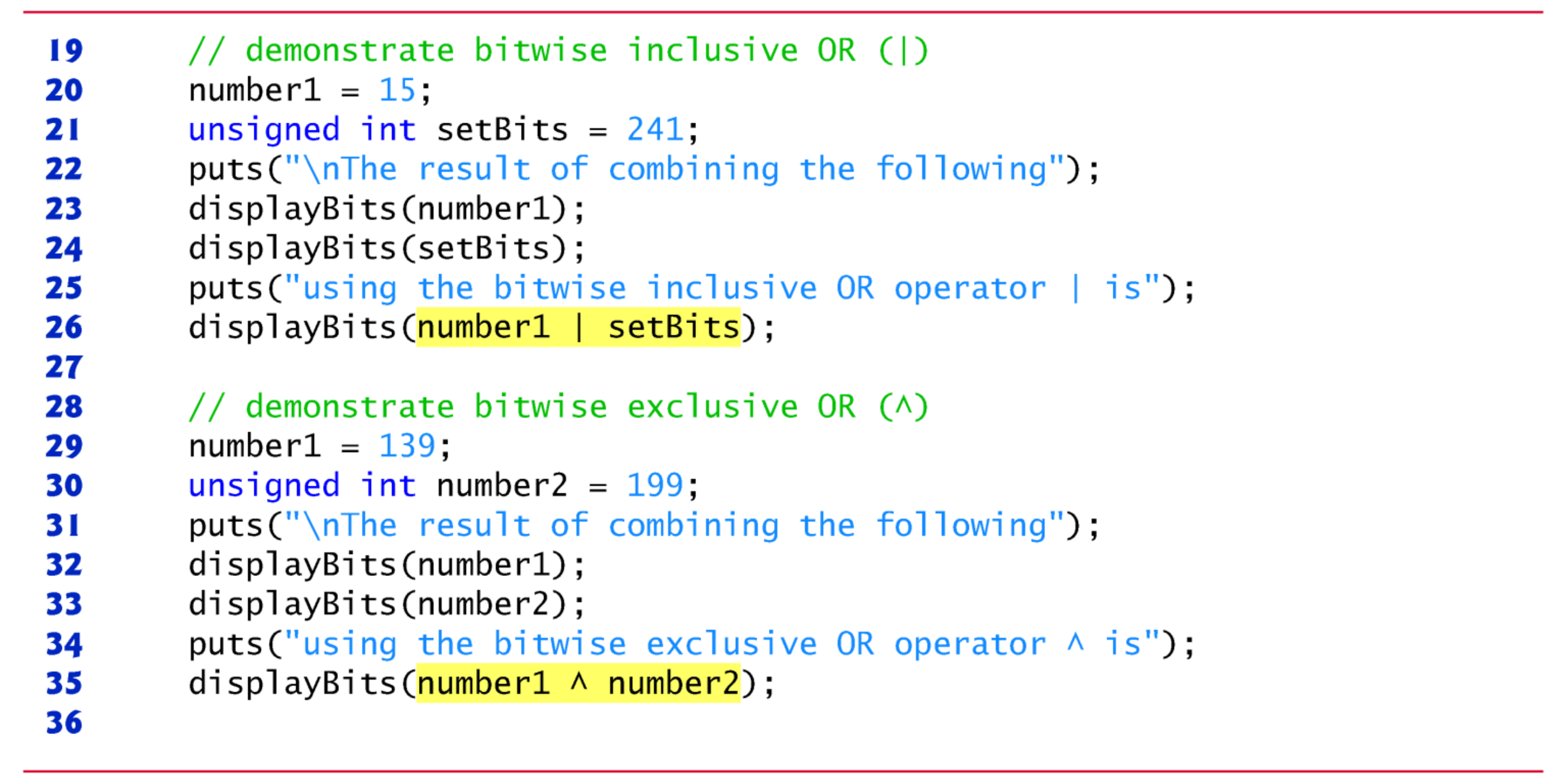
```
Enter a nonnegative int: 65000
     65000 = 00000000 00000000 11111101 11101000
```

Bitwise Operation Example Code (1)



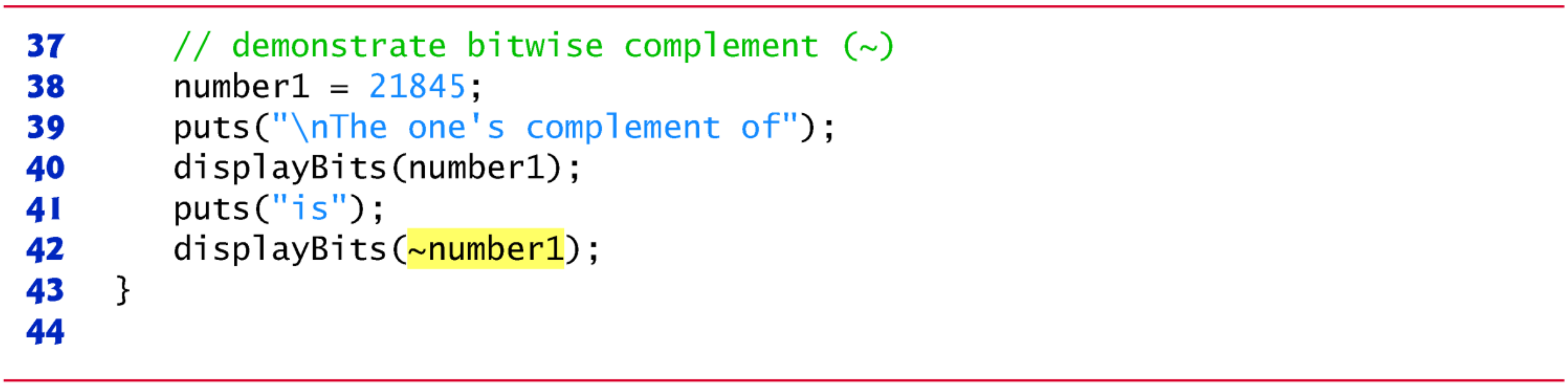
```c
// Fig. 10.9: fig10_09.c
// Using the bitwise AND, bitwise inclusive OR, bitwise
// exclusive OR and bitwise complement operators
#include <stdio.h>

void displayBits(unsigned int value); // prototype

int main(void)
{
   // demonstrate bitwise AND (&)
   unsigned int number1 = 65535;
   unsigned int mask = 1;
   puts("The result of combining the following");
   displayBits(number1);
   displayBits(mask);
   puts("using the bitwise AND operator & is");
   displayBits(number1 & mask);

```

Bitwise Operation Example Code (2)



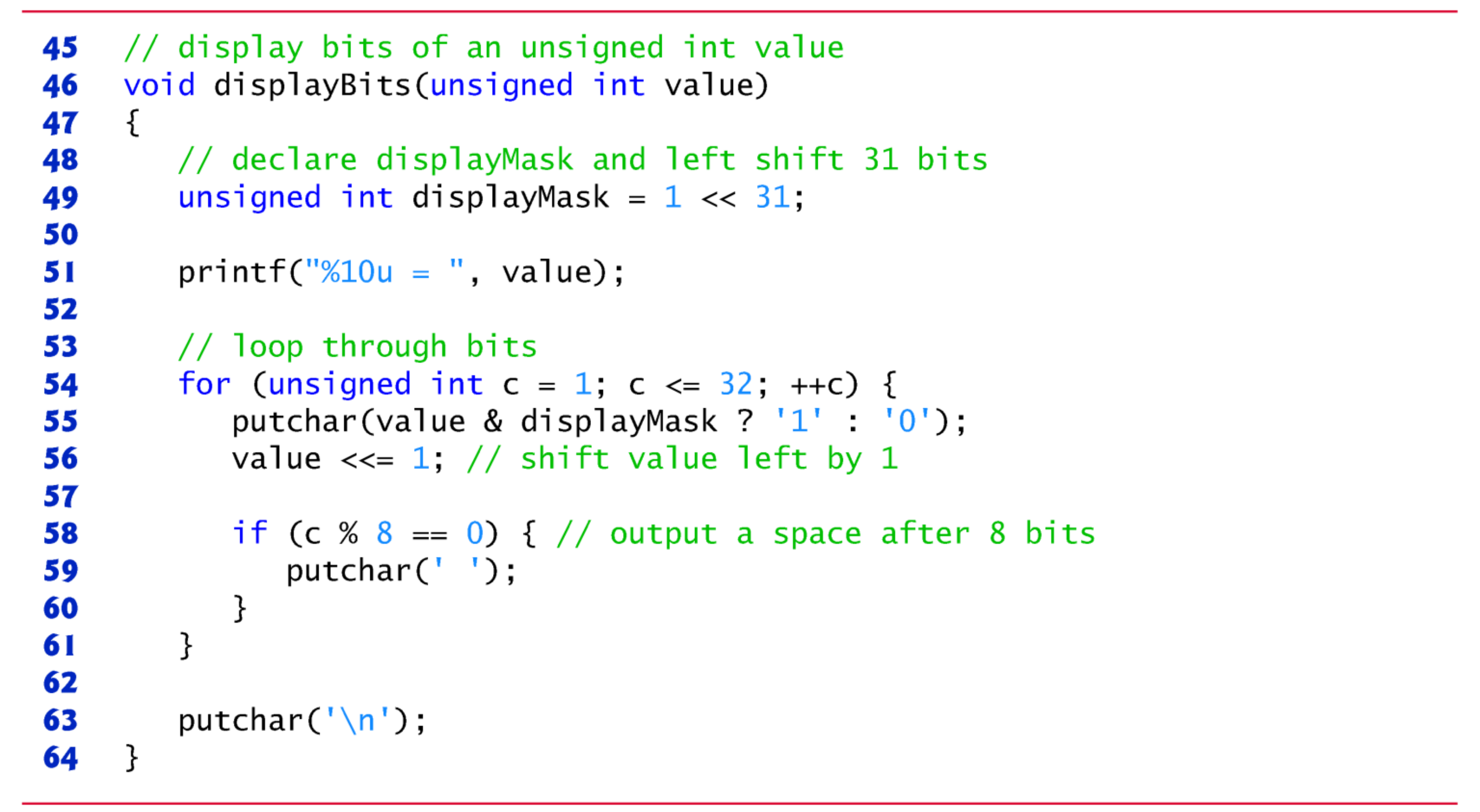
```
   // demonstrate bitwise inclusive OR (|)
   number1 = 15;
   unsigned int setBits = 241;
   puts("\nThe result of combining the following");
   displayBits(number1);
   displayBits(setBits);
   puts("using the bitwise inclusive OR operator | is");
   displayBits(number1 | setBits);

   // demonstrate bitwise exclusive OR (^)
   number1 = 139;
   unsigned int number2 = 199;
   puts("\nThe result of combining the following");
   displayBits(number1);
   displayBits(number2);
   puts("using the bitwise exclusive OR operator ^ is");
   displayBits(number1 ^ number2);

```

Bitwise Operation Example Code (3)



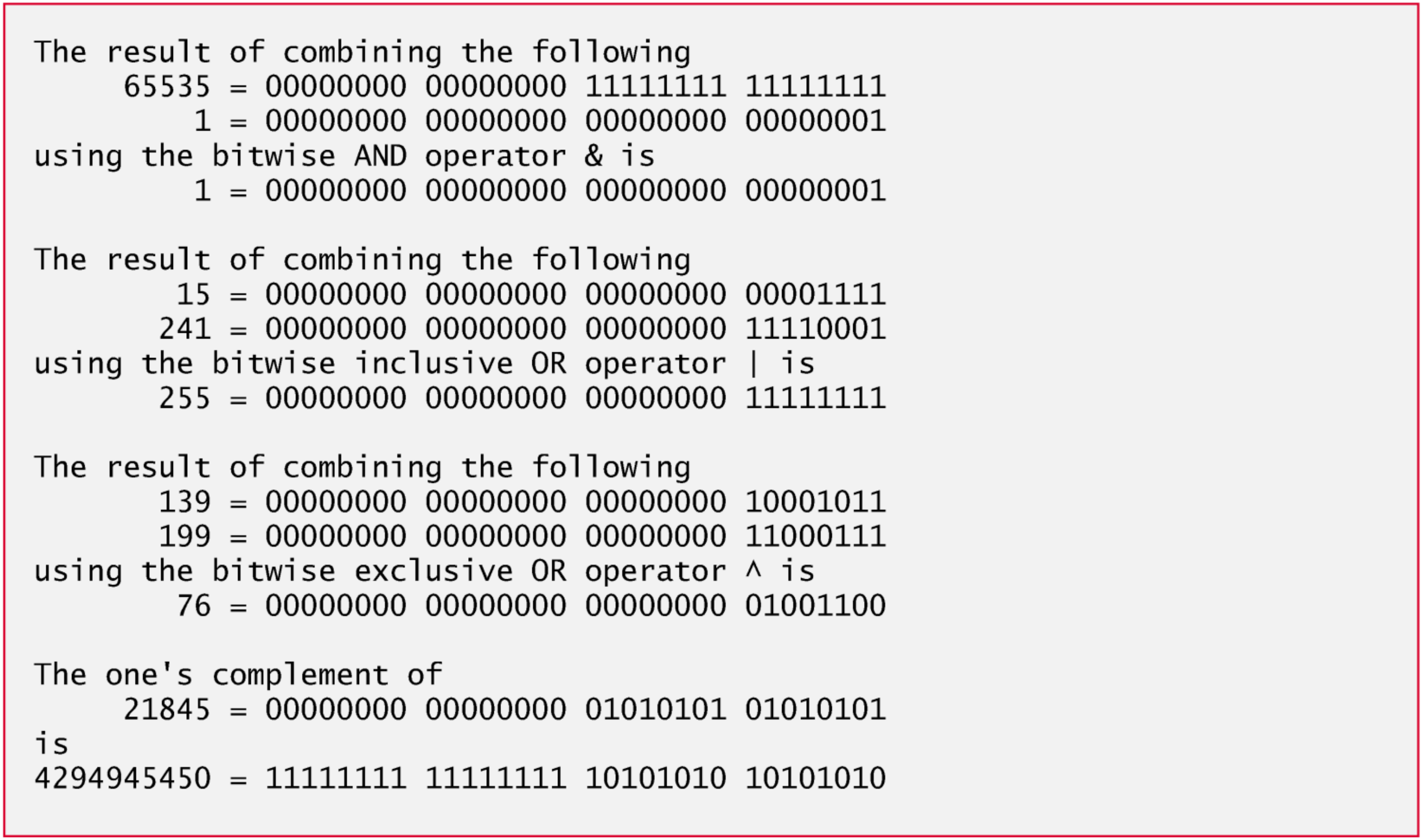
```c
        // demonstrate bitwise complement (~)
        number1 = 21845;
        puts("\nThe one's complement of");
        displayBits(number1);
        puts("is");
        displayBits(~number1);
    }

```

Bitwise Operation Example Code (4)



```c
// display bits of an unsigned int value
void displayBits(unsigned int value)
{
   // declare displayMask and left shift 31 bits
   unsigned int displayMask = 1 << 31;

   printf("%10u = ", value);

   // loop through bits
   for (unsigned int c = 1; c <= 32; ++c) {
      putchar(value & displayMask ? '1' : '0');
      value <<= 1; // shift value left by 1

      if (c % 8 == 0) { // output a space after 8 bits
         putchar(' ');
      }
   }

   putchar('\n');
}
```

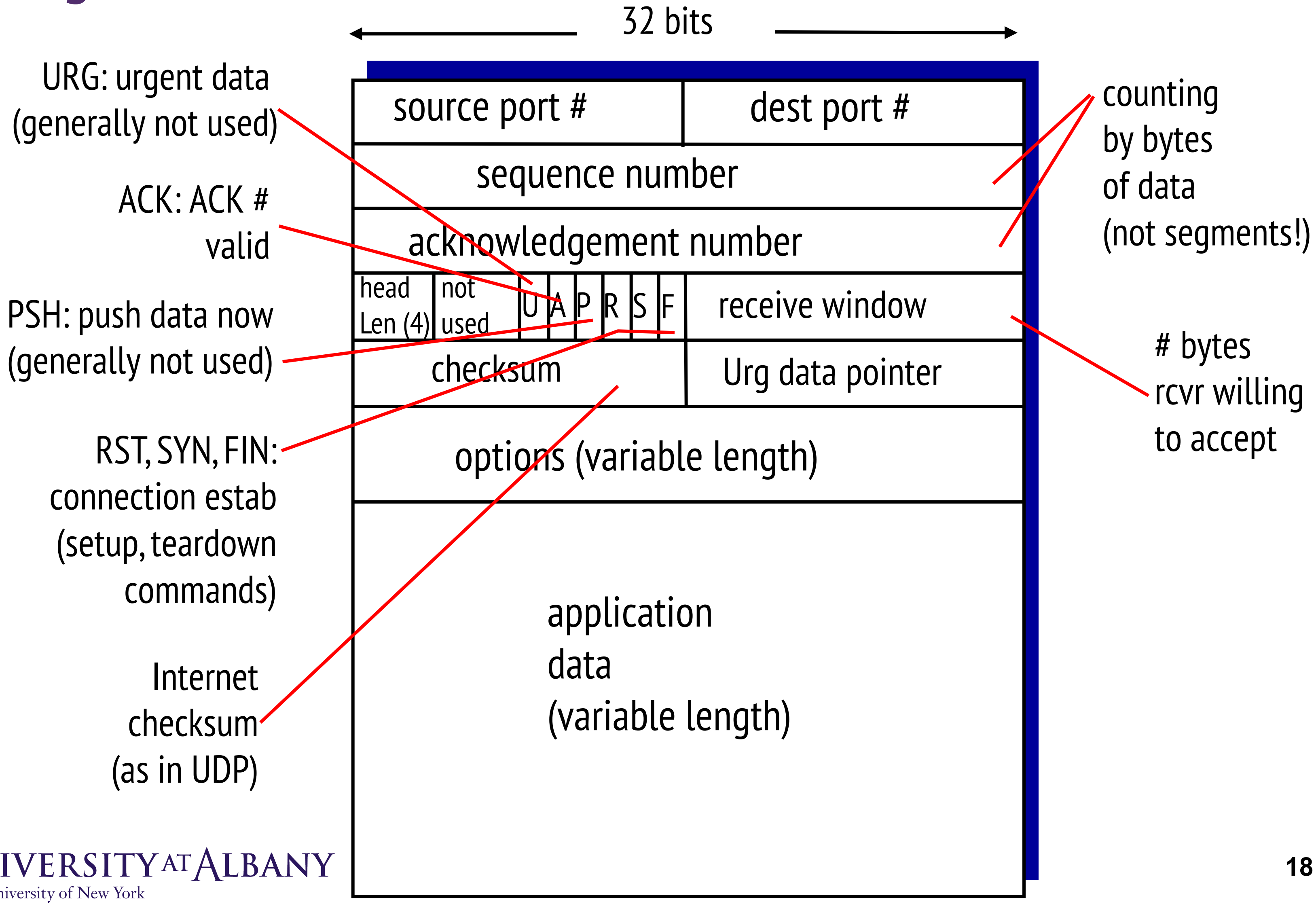
Bitwise Operation Example Code Output



```
The result of combining the following
     65535 = 00000000 00000000 11111111 11111111
         1 = 00000000 00000000 00000000 00000001
using the bitwise AND operator & is
         1 = 00000000 00000000 00000000 00000001

The result of combining the following
        15 = 00000000 00000000 00000000 00001111
       241 = 00000000 00000000 00000000 11110001
using the bitwise inclusive OR operator | is
       255 = 00000000 00000000 00000000 11111111

The result of combining the following
       139 = 00000000 00000000 00000000 10001011
       199 = 00000000 00000000 00000000 11000111
using the bitwise exclusive OR operator ^ is
        76 = 00000000 00000000 00000000 01001100

The one's complement of
     21845 = 00000000 00000000 01010101 01010101
is
4294945450 = 11111111 11111111 10101010 10101010
```

Bitwise Operation Application:

TCP segment structure


32 bits
source port #
dest port #
sequence number
acknowledgement number
head Len (4)
not used
U
A
P
R
S
F
receive window
checksum
Urg data pointer
options (variable length)
application data (variable length)
URG: urgent data (generally not used)
ACK: ACK # valid
PSH: push data now (generally not used)
RST, SYN, FIN: connection estab (setup, teardown commands)
Internet checksum (as in UDP)
counting by bytes of data (not segments!)
# bytes rcvr willing to accept

Multiply and Divide by Bitwise Operation

- Left Shift
  - Multiply

positional powers of 2:	2^4	2^3	2^2	2^1	2^0
decimal positional value:	**16**	**8**	**4**	**2**	**1**
binary number:	0	0	1	1	1

$4 + 2 + 1 = 7_{10}$

Left shift:	0	1	1	1	0

$8 + 4 + 2 = 14_{10}$

- Right Shift (without rotate)
  - Divide

Multiplication using shift

- x * 10
  - $x * 10 = x * (8 + 2) = (x * 8) + (x * 2) = (x * 2^3) + (x * 2^1) = (x << 3) + (x << 1)$

- x * 20
  - $x * 20 = x * (16 + 4) = (x * 16) + (x * 4) = (x * 2^4) + (x * 2^2) = (x << 4) + (x << 2)$

- x * 15
  - $x * 15 = x * (16 - 1) = (x * 16) - x = (x * 2^4) - x = (x << 4) - x$