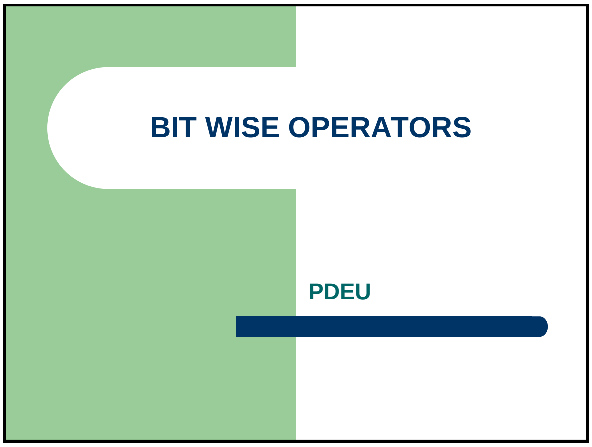
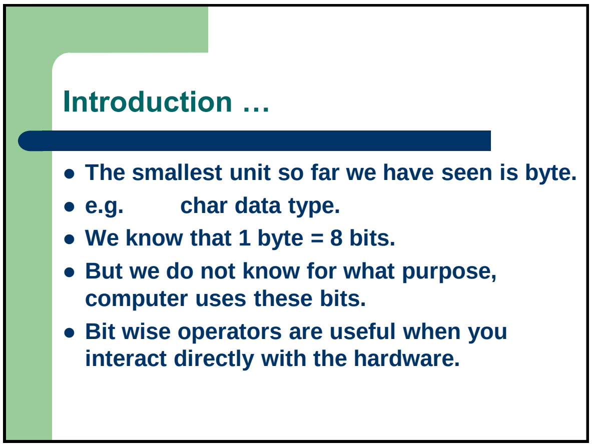


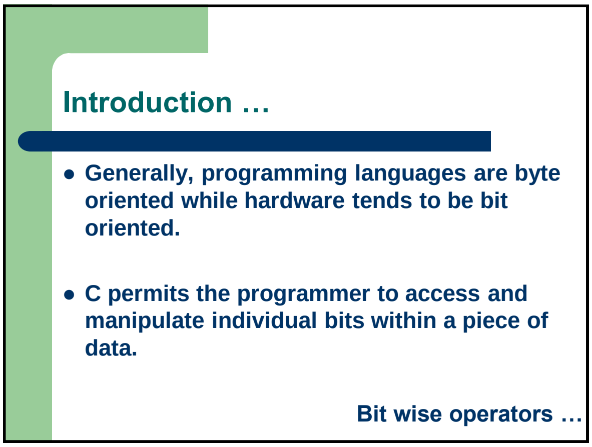
BIT WISE OPERATORS

PDEU



Introduction …

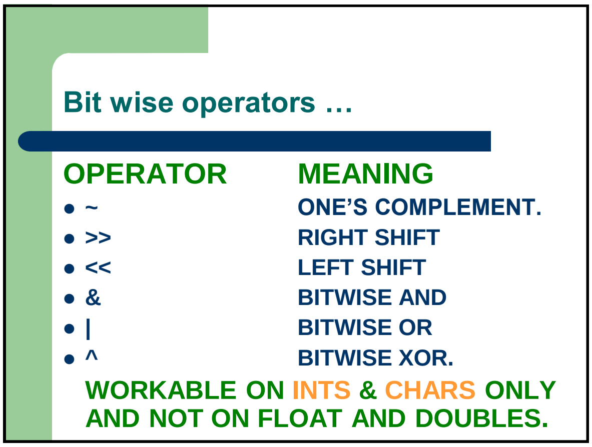
- The smallest unit so far we have seen is byte.
- e.g.   char data type.
- We know that 1 byte = 8 bits.
- But we do not know for what purpose, computer uses these bits.
- Bit wise operators are useful when you interact directly with the hardware.



Introduction …

- Generally, programming languages are byte oriented while hardware tends to be bit oriented.
- C permits the programmer to access and manipulate individual bits within a piece of data.

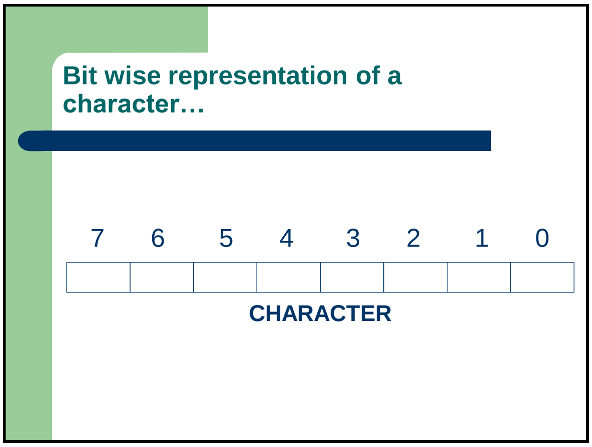
Bit wise operators …



Bit wise operators …

OPERATOR	MEANING
~	ONE’S COMPLEMENT.
>>	RIGHT SHIFT
<<	LEFT SHIFT
&	BITWISE AND
|	BITWISE OR
^	BITWISE XOR.

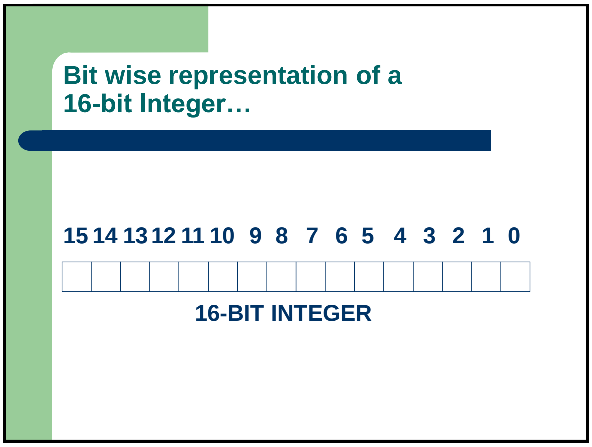
WORKABLE ON INTS & CHARS ONLY AND NOT ON FLOAT AND DOUBLES.



Bit wise representation of a character…

7	6	5	4	3	2	1	0

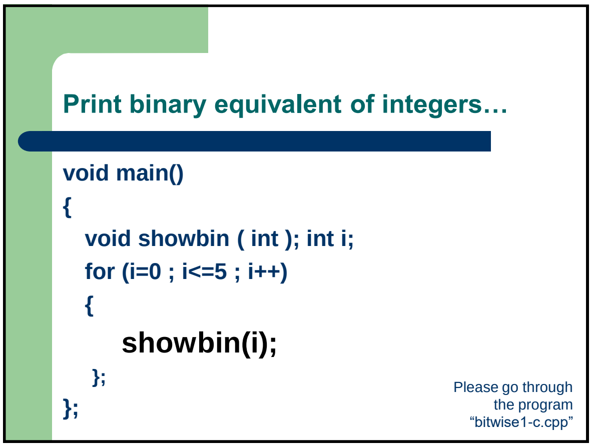
CHARACTER



Bit wise representation of a 16-bit Integer…

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

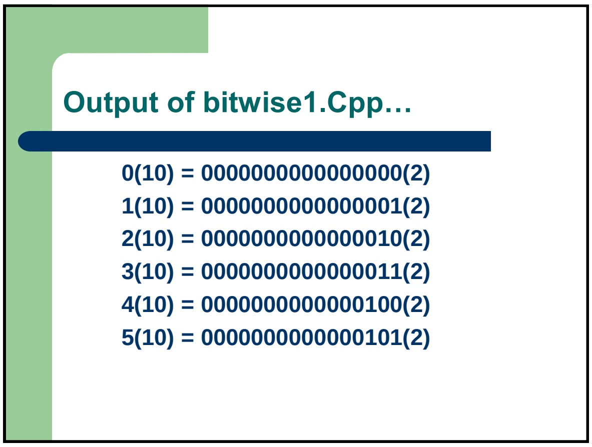
16-BIT INTEGER



Print binary equivalent of integers…

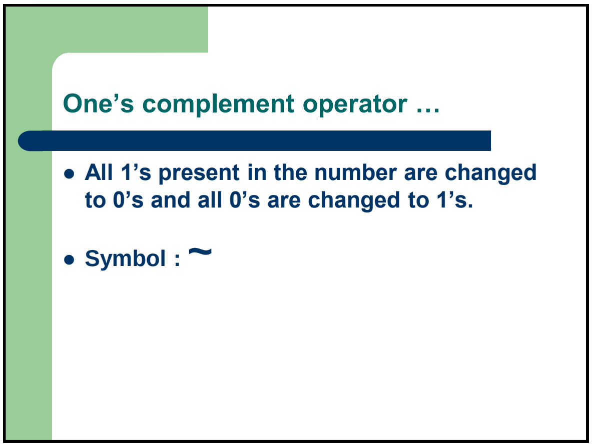
```
void main()
{
   void showbin ( int ); int i;
   for (i=0 ; i<=5 ; i++)
   {
        showbin(i);
    };
};
```

Please go through
the program
“bitwise1-c.cpp”



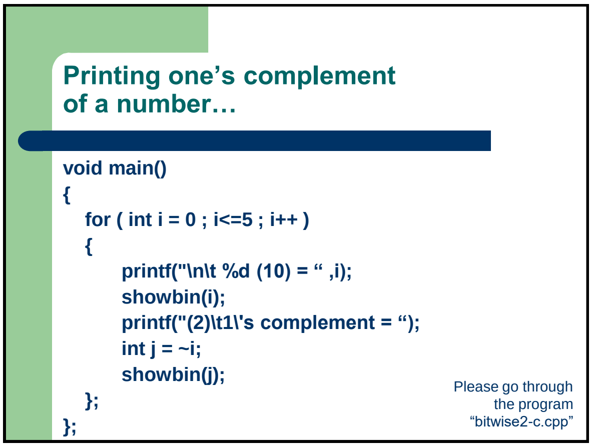
Output of bitwise1.Cpp…

```
0(10) = 0000000000000000(2)
1(10) = 0000000000000001(2)
2(10) = 0000000000000010(2)
3(10) = 0000000000000011(2)
4(10) = 0000000000000100(2)
5(10) = 0000000000000101(2)
```



One’s complement operator …

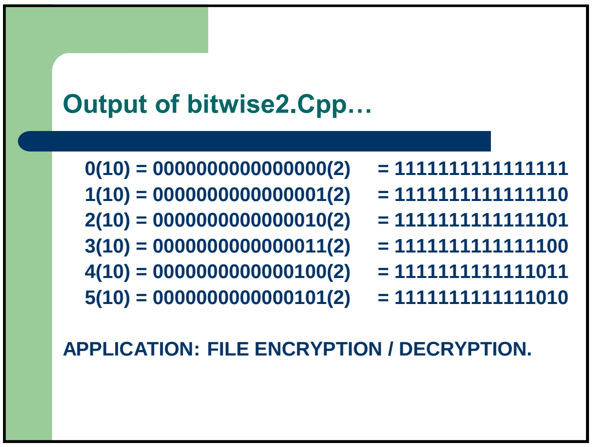
- All 1’s present in the number are changed to 0’s and all 0’s are changed to 1’s.
- Symbol : ~



Printing one’s complement of a number…

```
void main()
{
  for ( int i = 0 ; i<=5 ; i++ )
  {
       printf("\n\t %d (10) = “ ,i);
       showbin(i);
       printf("(2)\t1\'s complement = “);
       int j = ~i;
       showbin(j);
  };
};
```

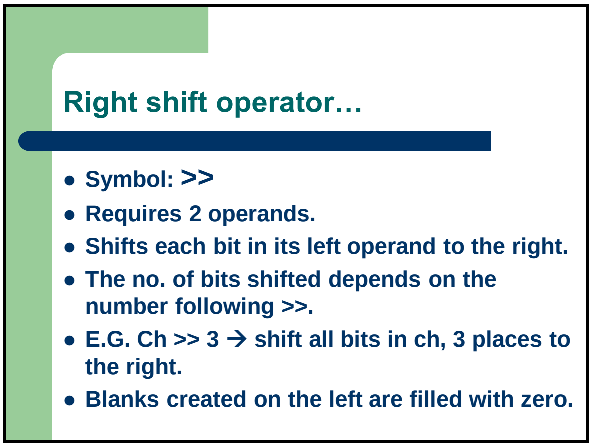
Please go through the program “bitwise2-c.cpp”



Output of bitwise2.Cpp…

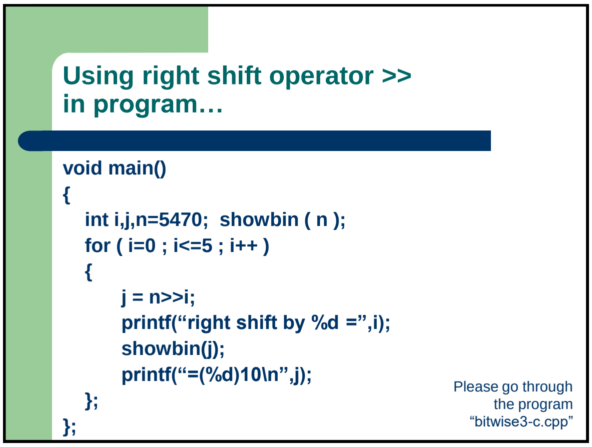
```
0(10) = 0000000000000000(2)    = 1111111111111111
1(10) = 0000000000000001(2)    = 1111111111111110
2(10) = 0000000000000010(2)    = 1111111111111101
3(10) = 0000000000000011(2)    = 1111111111111100
4(10) = 0000000000000100(2)    = 1111111111111011
5(10) = 0000000000000101(2)    = 1111111111111010
```

APPLICATION: FILE ENCRYPTION / DECRYPTION.



Right shift operator…

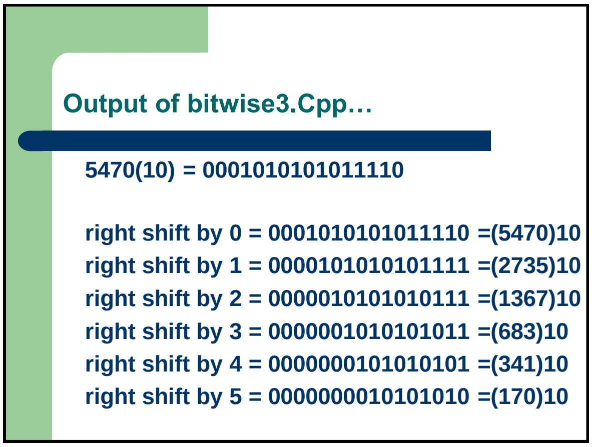
- Symbol: >>
- Requires 2 operands.
- Shifts each bit in its left operand to the right.
- The no. of bits shifted depends on the number following >>.
- E.G. Ch >> 3 → shift all bits in ch, 3 places to the right.
- Blanks created on the left are filled with zero.



Using right shift operator >> in program…

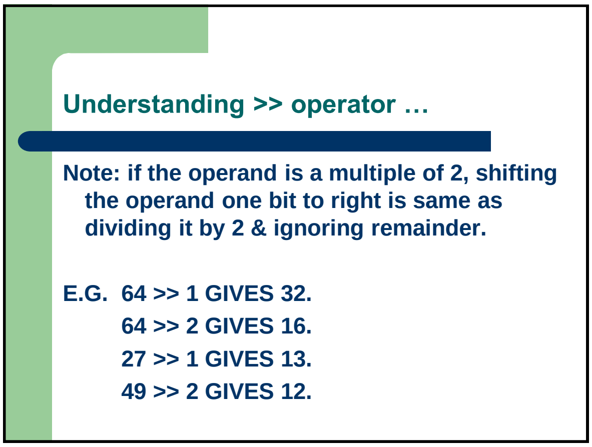
```
void main()
{
   int i,j,n=5470;  showbin ( n );
   for ( i=0 ; i<=5 ; i++ )
   {
        j = n>>i;
        printf(“right shift by %d =”,i);
        showbin(j);
        printf(“=(%d)10\n”,j);
   };
};
```

Please go through the program “bitwise3-c.cpp”



Output of bitwise3.Cpp…

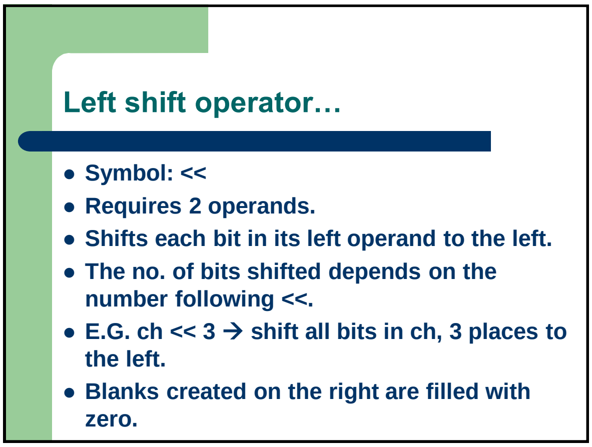
```
5470(10) = 0001010101011110

right shift by 0 = 0001010101011110 =(5470)10
right shift by 1 = 0000101010101111 =(2735)10
right shift by 2 = 0000010101010111 =(1367)10
right shift by 3 = 0000001010101011 =(683)10
right shift by 4 = 0000000101010101 =(341)10
right shift by 5 = 0000000010101010 =(170)10
```



Understanding >> operator …

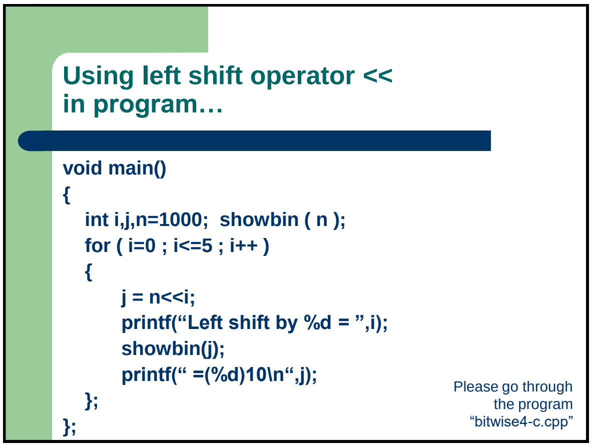
Note: if the operand is a multiple of 2, shifting the operand one bit to right is same as dividing it by 2 & ignoring remainder.

E.G. 64 >> 1 GIVES 32.
64 >> 2 GIVES 16.
27 >> 1 GIVES 13.
49 >> 2 GIVES 12.



Left shift operator…

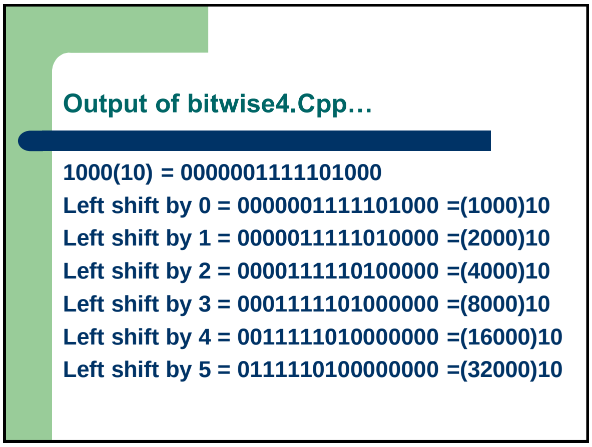
- Symbol: <<
- Requires 2 operands.
- Shifts each bit in its left operand to the left.
- The no. of bits shifted depends on the number following <<.
- E.G. ch << 3 → shift all bits in ch, 3 places to the left.
- Blanks created on the right are filled with zero.



Using left shift operator << in program…

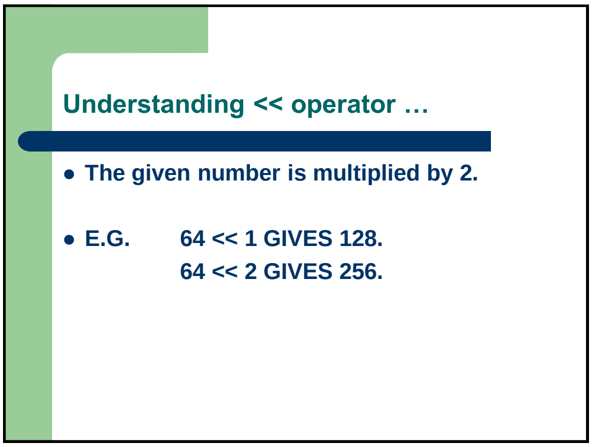
```
void main()
{
   int i,j,n=1000;  showbin ( n );
   for ( i=0 ; i<=5 ; i++ )
   {
        j = n<<i;
        printf(“Left shift by %d = ”,i);
        showbin(j);
        printf(“ =(%d)10\n“,j);
   };
};
```

Please go through the program “bitwise4-c.cpp”



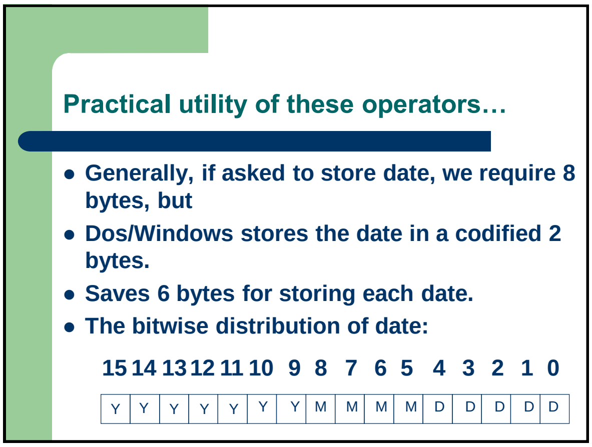
Output of bitwise4.Cpp…

```
1000(10) = 0000001111101000
Left shift by 0 = 0000001111101000 =(1000)10
Left shift by 1 = 0000011111010000 =(2000)10
Left shift by 2 = 0000111110100000 =(4000)10
Left shift by 3 = 0001111101000000 =(8000)10
Left shift by 4 = 0011111010000000 =(16000)10
Left shift by 5 = 0111110100000000 =(32000)10
```



Understanding << operator …

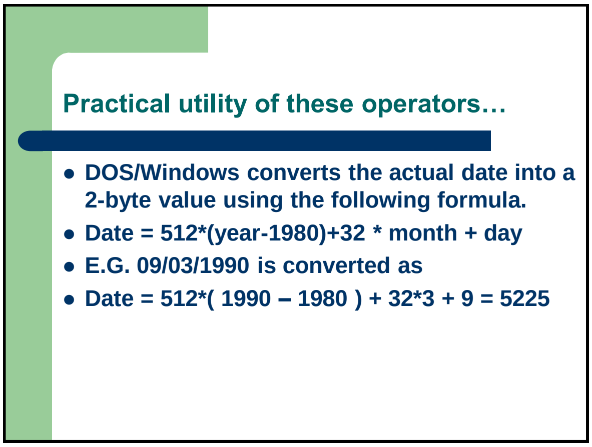
- The given number is multiplied by 2.
- E.G. 64 << 1 GIVES 128.
  64 << 2 GIVES 256.



Practical utility of these operators…

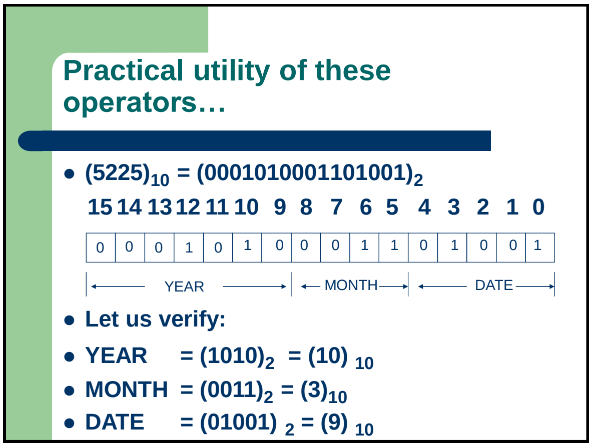
- Generally, if asked to store date, we require 8 bytes, but
- Dos/Windows stores the date in a codified 2 bytes.
- Saves 6 bytes for storing each date.
- The bitwise distribution of date:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Y	Y	Y	Y	Y	Y	Y	M	M	M	M	D	D	D	D	D



Practical utility of these operators…

- DOS/Windows converts the actual date into a 2-byte value using the following formula.
- Date = 512*(year-1980)+32 * month + day
- E.G. 09/03/1990 is converted as
- Date = 512*(1990 – 1980) + 32*3 + 9 = 5225



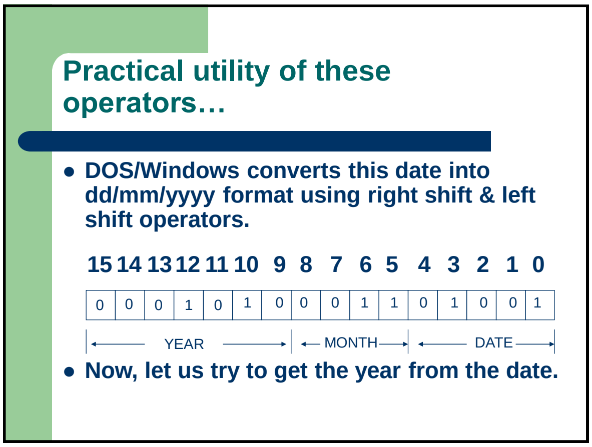
Practical utility of these operators…

- $(5225)_{10}$ = $(0001010001101001)_2$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	1	0	0	0	1	1	0	1	0	0	1

YEAR
MONTH
DATE

- Let us verify:
- YEAR = $(1010)_2$ = $(10)_{10}$
- MONTH = $(0011)_2$ = $(3)_{10}$
- DATE = $(01001)_2$ = $(9)_{10}$



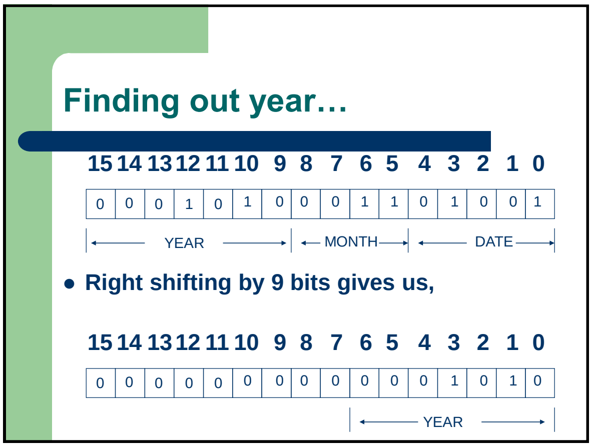
Practical utility of these operators…

- DOS/Windows converts this date into dd/mm/yyyy format using right shift & left shift operators.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	1	0	0	0	1	1	0	1	0	0	1

YEAR
MONTH
DATE

- Now, let us try to get the year from the date.



Finding out year…

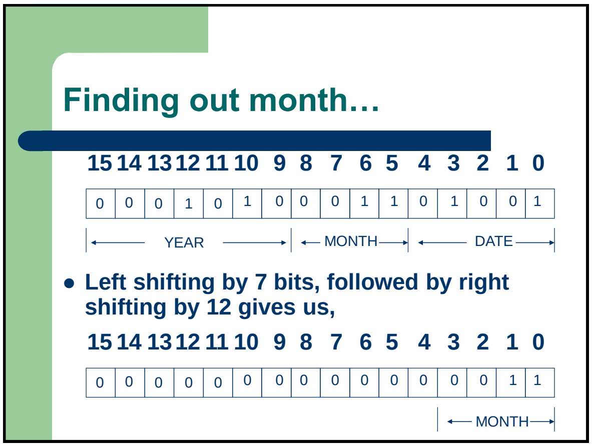
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	1	0	0	0	1	1	0	1	0	0	1

YEAR
MONTH
DATE

- Right shifting by 9 bits gives us,

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0

YEAR



Finding out month…

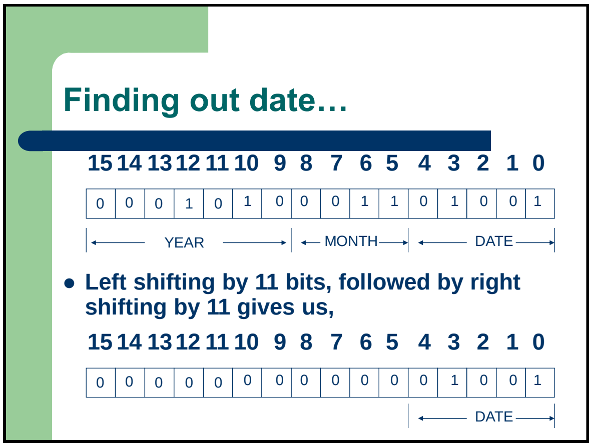
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	1	0	0	0	1	1	0	1	0	0	1

YEAR MONTH DATE

- Left shifting by 7 bits, followed by right shifting by 12 gives us,

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

MONTH



Finding out date…

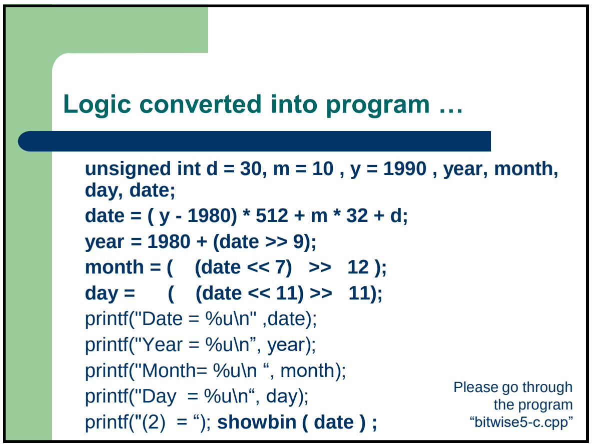
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	1	0	0	0	1	1	0	1	0	0	1

YEAR MONTH DATE

- Left shifting by 11 bits, followed by right shifting by 11 gives us,

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1

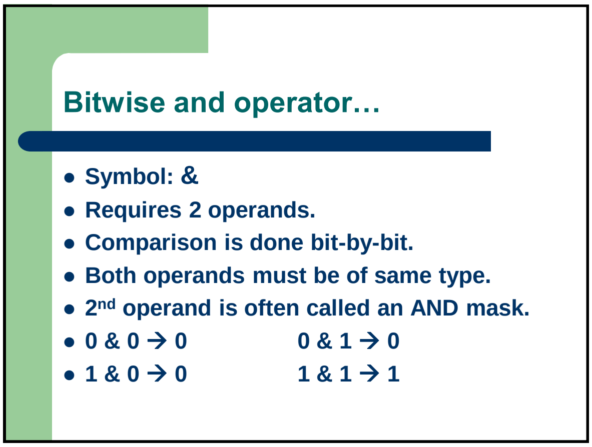
DATE



Logic converted into program …

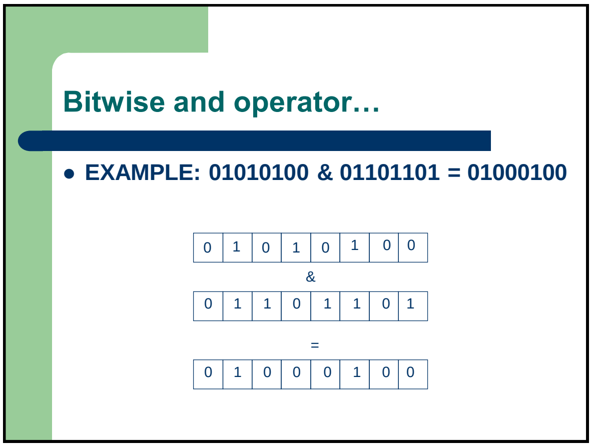
```
unsigned int d = 30, m = 10 , y = 1990 , year, month,
day, date;
date = ( y - 1980) * 512 + m * 32 + d;
year = 1980 + (date >> 9);
month = (    (date << 7)   >>   12 );
day =      (    (date << 11) >>   11);
printf("Date = %u\n" ,date);
printf("Year = %u\n”, year);
printf("Month= %u\n “, month);
printf("Day  = %u\n“, day);
printf("(2)  = “); showbin ( date ) ;
```

Please go through the program “bitwise5-c.cpp”



Bitwise and operator…

- Symbol: &
- Requires 2 operands.
- Comparison is done bit-by-bit.
- Both operands must be of same type.
- 2nd operand is often called an AND mask.
- 0 & 0 → 0 0 & 1 → 0
- 1 & 0 → 0 1 & 1 → 1



Bitwise and operator…

- EXAMPLE: 01010100 & 01101101 = 01000100

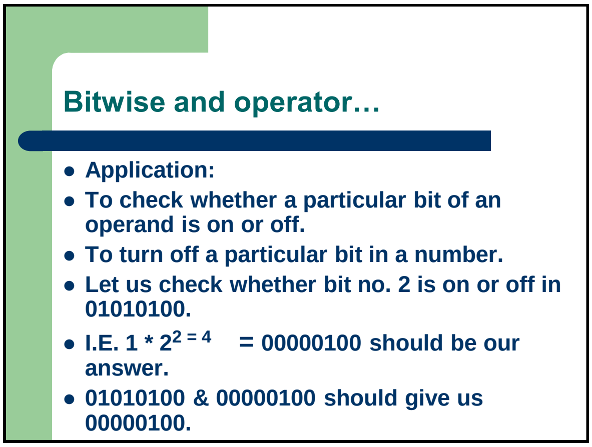
0	1	0	1	0	1	0	0

&

0	1	1	0	1	1	0	1

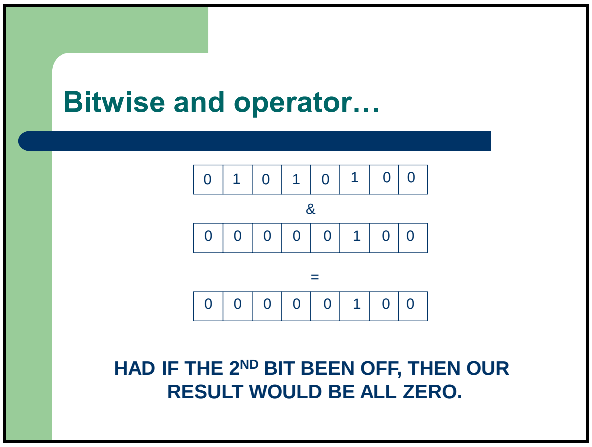
=

0	1	0	0	0	1	0	0



Bitwise and operator…

- Application:
- To check whether a particular bit of an operand is on or off.
- To turn off a particular bit in a number.
- Let us check whether bit no. 2 is on or off in 01010100.
- I.E. 1 * $2^{2 = 4}$ = 00000100 should be our answer.
- 01010100 & 00000100 should give us 00000100.



Bitwise and operator…

0	1	0	1	0	1	0	0

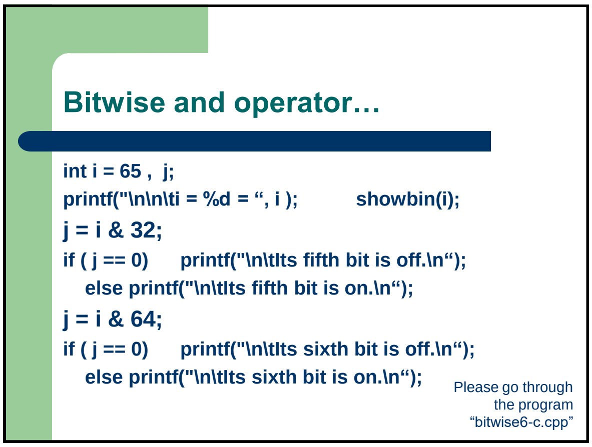
&

0	0	0	0	0	1	0	0

=

0	0	0	0	0	1	0	0

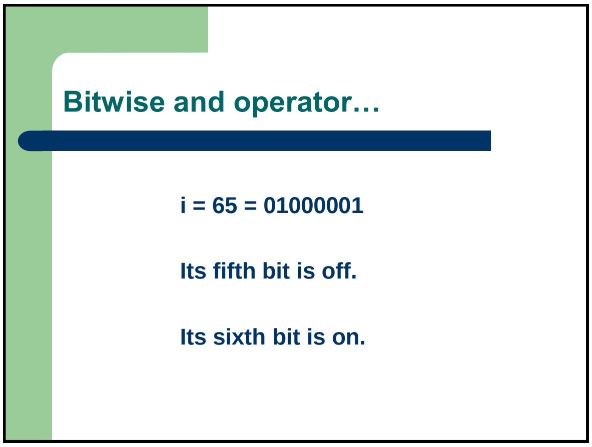
HAD IF THE 2^{ND} BIT BEEN OFF, THEN OUR RESULT WOULD BE ALL ZERO.



Bitwise and operator…

```
int i = 65 ,  j;
printf("\n\n\ti = %d = “, i );          showbin(i);
j = i & 32;
if ( j == 0)      printf("\n\tIts fifth bit is off.\n“);
   else printf("\n\tIts fifth bit is on.\n“);
j = i & 64;
if ( j == 0)      printf("\n\tIts sixth bit is off.\n“);
   else printf("\n\tIts sixth bit is on.\n“);
```

Please go through the program “bitwise6-c.cpp”

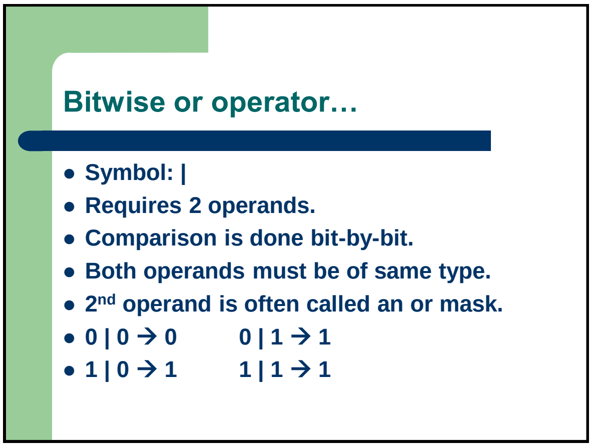


Bitwise and operator…

i = 65 = 01000001

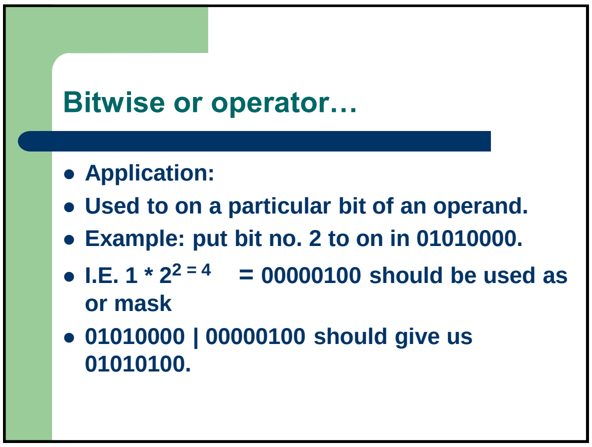
Its fifth bit is off.

Its sixth bit is on.



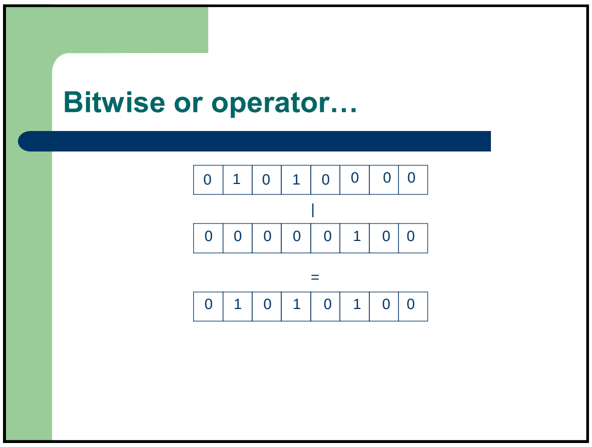
Bitwise or operator…

- Symbol: |
- Requires 2 operands.
- Comparison is done bit-by-bit.
- Both operands must be of same type.
- 2nd operand is often called an or mask.
- 0 | 0 → 0 0 | 1 → 1
- 1 | 0 → 1 1 | 1 → 1



Bitwise or operator…

- Application:
- Used to on a particular bit of an operand.
- Example: put bit no. 2 to on in 01010000.
- I.E. $1 * 2^{2 = 4}$ = 00000100 should be used as or mask
- 01010000 | 00000100 should give us 01010100.



Bitwise or operator…

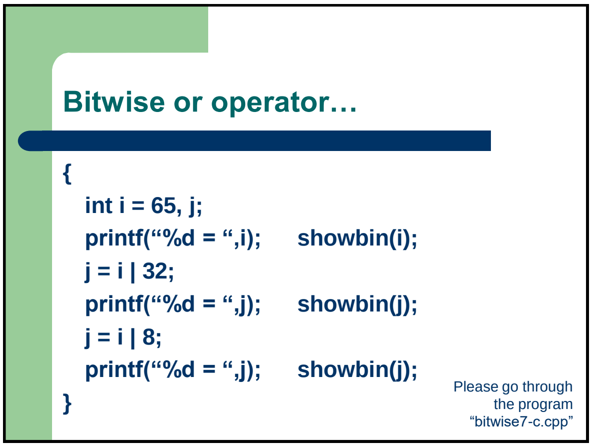
0 1 0 1 0 0 0 0

|

0 0 0 0 0 1 0 0

=

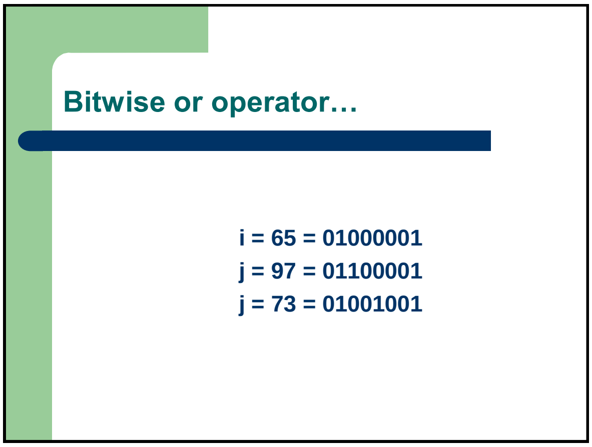
0 1 0 1 0 1 0 0



Bitwise or operator…

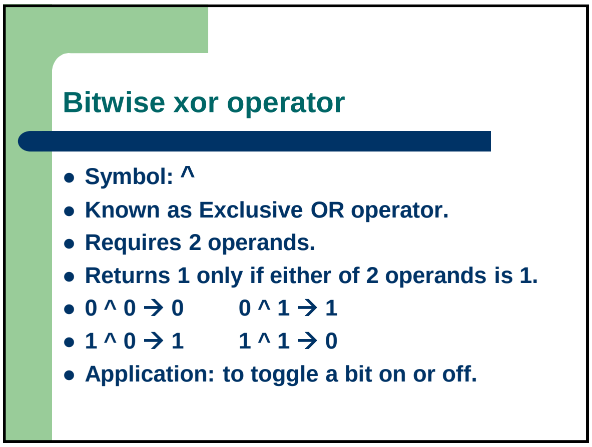
```
{
   int i = 65, j;
   printf(“%d = “,i);     showbin(i);
   j = i | 32;
   printf(“%d = “,j);     showbin(j);
   j = i | 8;
   printf(“%d = “,j);     showbin(j);
}
```

Please go through
the program
“bitwise7-c.cpp”



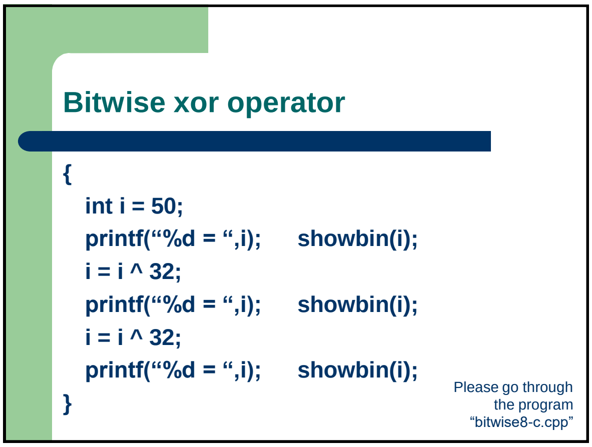
Bitwise or operator…

```
i = 65 = 01000001
j = 97 = 01100001
j = 73 = 01001001
```



Bitwise xor operator

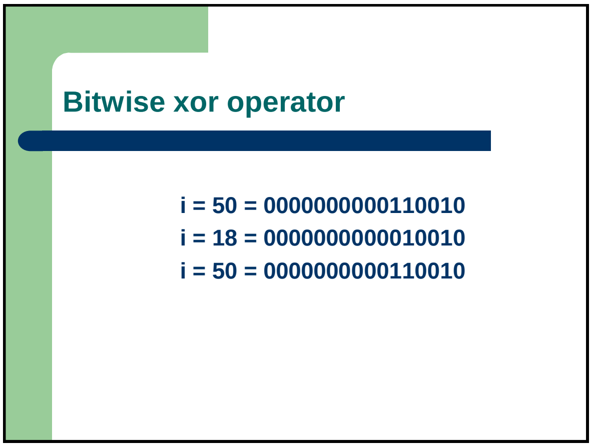
- Symbol: ^
- Known as Exclusive OR operator.
- Requires 2 operands.
- Returns 1 only if either of 2 operands is 1.
- 0 ^ 0 → 0   0 ^ 1 → 1
- 1 ^ 0 → 1   1 ^ 1 → 0
- Application: to toggle a bit on or off.



Bitwise xor operator

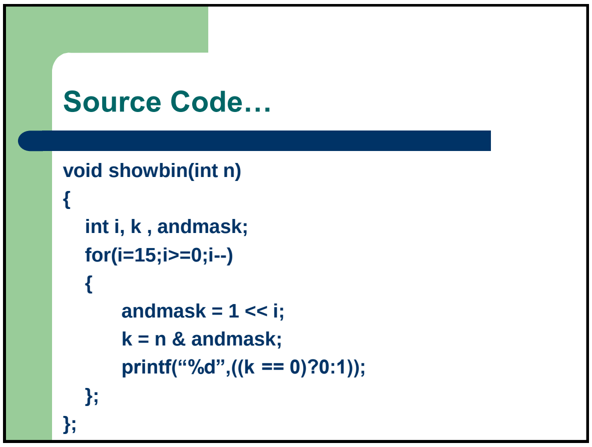
```
{
   int i = 50;
   printf(“%d = “,i);     showbin(i);
   i = i ^ 32;
   printf(“%d = “,i);     showbin(i);
   i = i ^ 32;
   printf(“%d = “,i);     showbin(i);
}
```

Please go through
the program
“bitwise8-c.cpp”



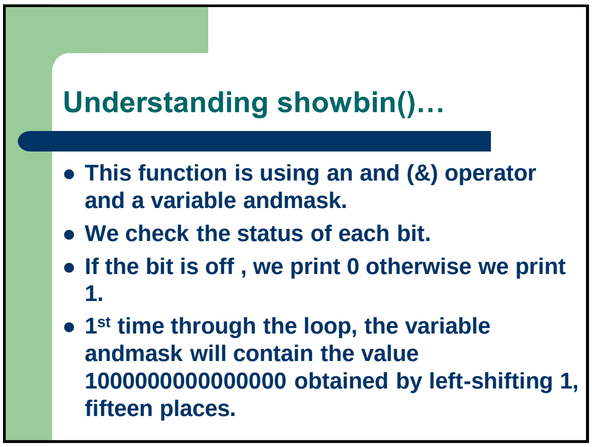
Bitwise xor operator

```
i = 50 = 0000000000110010
i = 18 = 0000000000010010
i = 50 = 0000000000110010
```



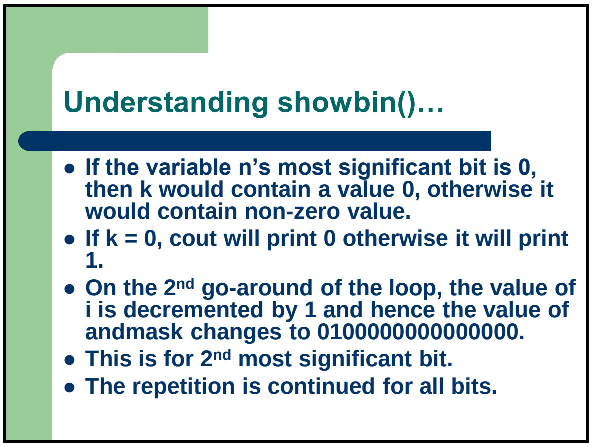
Source Code…

```
void showbin(int n)
{
   int i, k , andmask;
   for(i=15;i>=0;i--)
   {
        andmask = 1 << i;
        k = n & andmask;
        printf("%d",((k == 0)?0:1));
   };
};
```



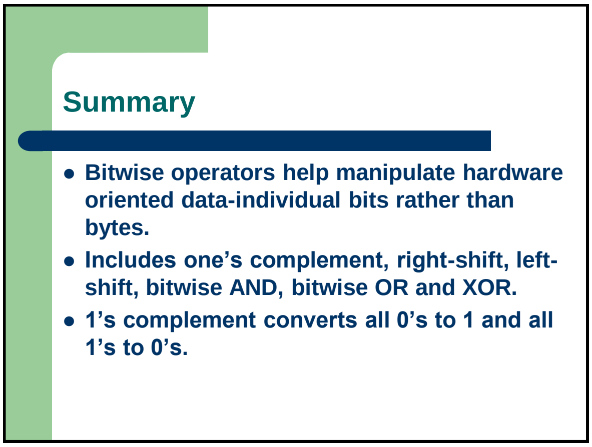
Understanding showbin()…

- This function is using an and (&) operator and a variable andmask.
- We check the status of each bit.
- If the bit is off , we print 0 otherwise we print 1.
- 1st time through the loop, the variable andmask will contain the value 1000000000000000 obtained by left-shifting 1, fifteen places.



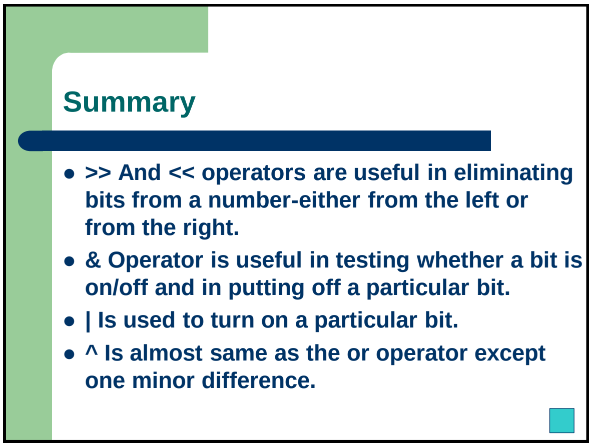
Understanding showbin()…

- If the variable n's most significant bit is 0, then k would contain a value 0, otherwise it would contain non-zero value.
- If k = 0, cout will print 0 otherwise it will print 1.
- On the 2nd go-around of the loop, the value of i is decremented by 1 and hence the value of andmask changes to 0100000000000000.
- This is for 2nd most significant bit.
- The repetition is continued for all bits.



Summary

- Bitwise operators help manipulate hardware oriented data-individual bits rather than bytes.
- Includes one’s complement, right-shift, left-shift, bitwise AND, bitwise OR and XOR.
- 1’s complement converts all 0’s to 1 and all 1’s to 0’s.



Summary

- >> And << operators are useful in eliminating bits from a number-either from the left or from the right.
- & Operator is useful in testing whether a bit is on/off and in putting off a particular bit.
- | Is used to turn on a particular bit.
- ^ Is almost same as the or operator except one minor difference.