Problems

Dr Bhanu

Decimal to Hex Conversion

Divide by 16	Quotient	Remainder	Hex Value
1228 ÷ 16	76	12	C
76 ÷ 16	4	12	C
4 ÷ 16	0	4	4

Therefore, $1228_{10} = 4CC_{16}$

Decimal to Hex conversion

- First, divide the decimal number by 16, considering the number as an integer.
- Keep aside the remainder left.
- Again divide the quotient by 16 and repeat till you get the quotient value equal to zero.
- Now take the values of the remainder’s left, in the reverse order, to get the hexadecimal numbers.

$$23457_{10} = ???_{16}$$

$$23457_{10} = 5BA1_{16}$$

Hex to Binary conversion

- Write down the hex number and represent each hex digit by its binary equivalent number.
- Use 4 bits and add insignificant leading zeros if the binary number has less than 4 bits. E.g. Write 10_2 (2 decimal) as 0010_2.
- Then concatenate or string all the digits together.
- Discard any leading zeros at the left of the binary number.

$$23457_{10} = 5BA1_{16}$$

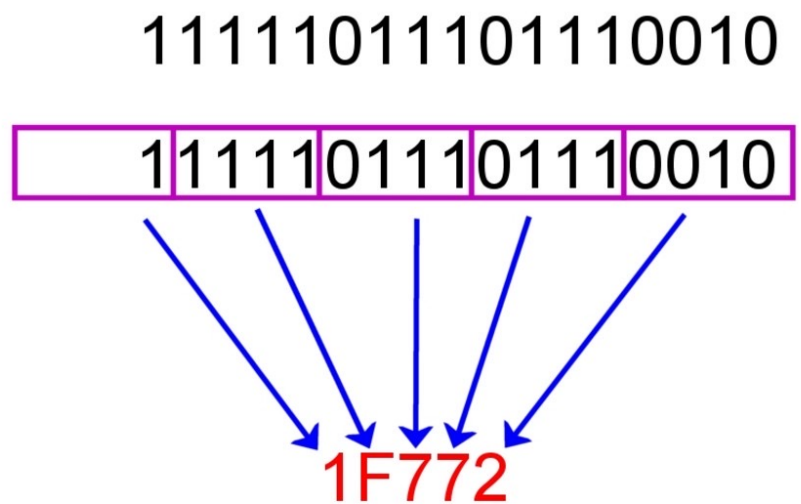
5	B	A	1
0101	1011	1010	0001

$$5BA1_{16} = 0101101110100001_{2}$$

Different forms of representation

- Int a = 23457;
- Int a = 0x5BA1;
- Int a = 055641
- Int a = 0b**0101101110100001;**

Converting Binary to Hex



Starting from the right, divide up into blocks of 4 digits

Write down the hex value of each block

$$1111101111011110010_2 = 1F772_{16}$$

New Assignment Operators

<<=	Left shift AND assignment operator	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator	C &= 2 is same as C = C & 2
^=	bitwise exclusive OR and assignment operator	C ^= 2 is same as C = C ^ 2
|=	bitwise inclusive OR and assignment operator	C |= 2 is same as C = C | 2

Find the answer

```
int a = 10;
int b = 45;
int c = a & b;
printf("a is : %d\n", a);
printf("c is : %d\n", c);
```

Find the answer

```
int a = 10;
int b = 450000;
int c = a & b;
printf("a is : %d\n", a);
printf("c is : %d\n", c);
```

Find the answer

```
int c = 011100;
c <<= 2;
printf("c << 2 is : %d", c);
```

Find the answer

```
int c = 0x11100;
c <<= 2;
printf("c << 2 is : %d", c);
```

Find the answer

```
int c = 0b11100;
c <<= 2;
printf("c << 2 is : %d", c);
```

Problem 1

Consider int val=0xCAFE; Write expressions using bitwise operators that perform the following:

(a) test if at least three of last four bits (LSB) are on

(b) reverse the byte order (i.e., produce val=0xFECA)

(c) rotate four bits (i.e., produce val=0xECAF)

Back to Problem 1

int val=0xCAFE;

(a) test if at least three of last four bits (LSB) are on

- **0xCAFE = 1100 1010 1111 1110**

Masking

In a given binary number, find if the 4^{th} bit is a 1

	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Value	128	64	32	16	8	4	2	1
A	1	0	0	0	0	1	1	1
Mask	0	0	0	1	0	0	0	0
&	0	0	0	0	0	0	0	0

Masks

												P	Q	R	S	
1	**1**	**0**	**0**	**1**	**0**	**1**	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	←. Given number
0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	←. M1 checks for PQRS
0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	←. M2 checks for QRS
0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	←. M3 checks for PQR
0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	←. M4 checks for PQS
0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	←. M5 checks for PRS

Masks

- M1 = 000F Mask all bits except PQRS
- M2 = 0007 Mask all bits except QRS
- M3 = 000E Mask all bits except PQR
- M4 = 000D Mask all bits except PQS
- M5 = 000B Mask all bits except PRS

Solve

(a) test if at least three of last four bits (LSB) are on

- **0xCAFE = 1100 1010 1111 1110**

```
a = 0xCAFE;
b = a & 0xF;
if ((b == 0xF) || (b == 0x7) || (b == 0xD) || (b == 0xB || (b == 0xE))
```

at least three of last four bits (LSB) are 1.

Solve

(b) reverse the byte order (i.e., produce val=0xFECA)

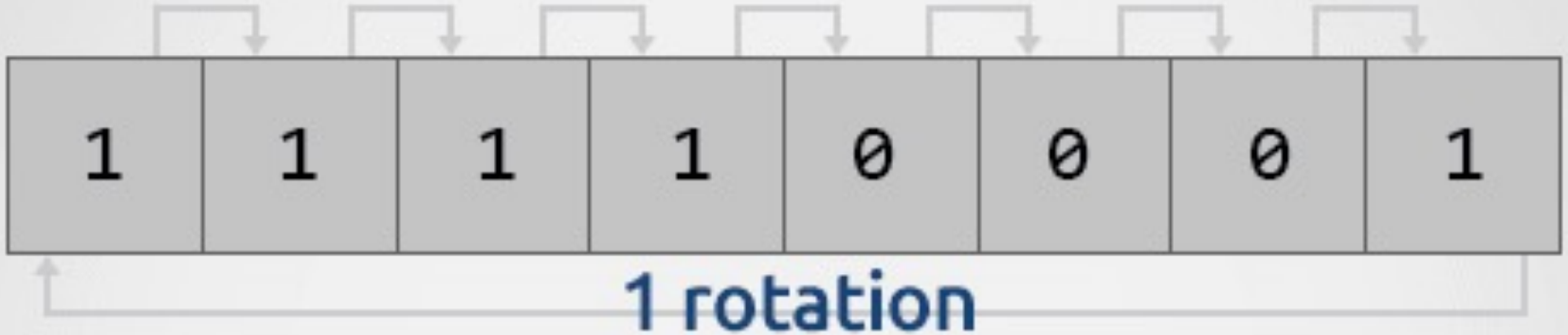
```
int c = 0xcafe;
int d, e, f;
d = (c >> 8);
printf("d is : %x\n", d);
e =  (c & 0xff);
e = e << 8;
printf("e is : %x\n", e);
f  = d | e;
printf("f is : %x\n", f);
```

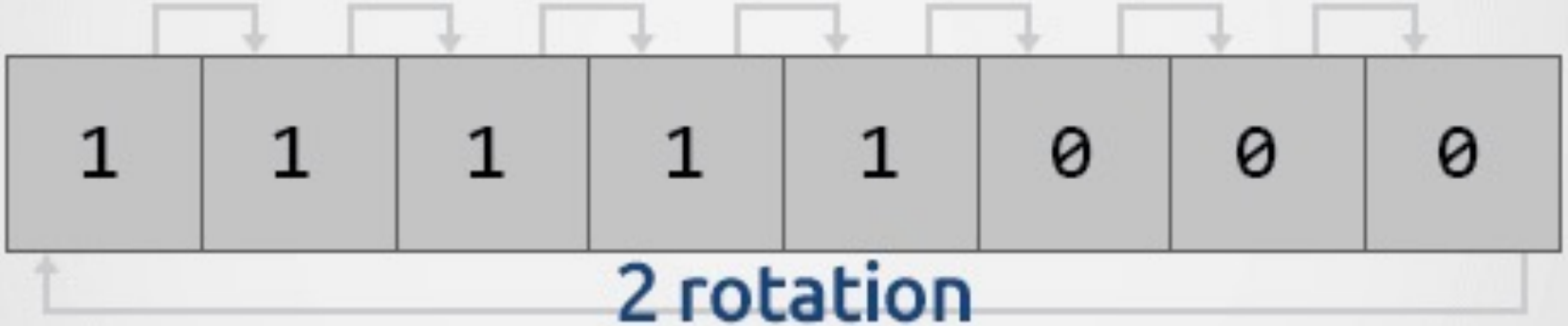
Solve

(c) rotate four bits (i.e., produce val=0xECAF)

- Bit Rotation: A rotation (or circular shift) is an operation similar to shift except that the bits that fall off at one end are put back to the other end.
- In left rotation, the bits that fall off at left end are put back at right end.
- In right rotation, the bits that fall off at right end are put back at left end.

Rotate -15 (11110001) to 2 times right





Final bits

Solve

(c) rotate four bits (i.e., produce val=0xECAF)

```
int c = 0xcafe;
int d, e, f;
d = (c >> 4);
printf("d is : %x\n", d);
e =  (c & 0xf);
 e = e << 12;
 printf("e is : %x\n", e);
 f  = d | e;
 printf("f is : %x\n", f);
```