A. x & 0xFF

B. x^~0xFF

C. x | 0xFF

这些表达式是在执行低级位运算中经常发现的典型类型。表达式~0xFF创建一个掩码，该掩码8个最低位等于0，而其余的位为1。可以观察到，这些掩码的产生和字长无关。而相比之下，表达式0xFFFFFF00只能工作在32位的机器上。

2.13　这个问题帮助你思考布尔运算和程序员应用掩码运算的典型方式之间的关系。代码如下：

```
/* Declarations of functions implementing operations bis and bic */
int bis(int x, int m);
int bic(int x, int m);

/* Compute x|y using only calls to functions bis and bic */
int bool_or(int x, int y) {
  int result = bis(x,y);
  return result;
}

/* Compute x^y using only calls to functions bis and bic */
int bool_xor(int x, int y) {
  int result = bis(bic(x,y), bic(y,x));
  return result;
}
```

bis运算等价于布尔OR——如果x中或者m中的这一位置位了，那么z中的这一位就置位。另一方面，bic(x, m)等价于x&~m；我们想实现只有当x对应的位为1且m对应的位为0时，该位等于1。

由此，可以通过对bis的一次调用来实现|。为了实现^，我们利用以下属性

$$x \hat{} y = (x \& \sim y) \mid (\sim x \& y)$$

2.14　这个问题突出了位级布尔运算和C语言中的逻辑运算之间的关系。常见的编程错误是在想用逻辑运算的时候用了位级运算，或者反过来。

表达式	值	表达式	值
x & y	0x20	x && y	0x01
x | y	0x7F	x || y	0x01
~x | ~y	0xDF	!x || !y	0x00
x & !y	0x00	x && ~y	0x01

2.15　这个表达式是!(x ^ y)。

也就是，当且仅当x的每一位和y相应的每一位匹配时，x ^ y等于零。然后，我们利用!来判定一个字是否包含任何非零位。

没有任何实际的理由要去使用这个表达式，因为可以简单地写成x==y，但是它说明了位级运算和逻辑运算之间的一些细微差别。

2.16　这个练习可以帮助你理解各种移位运算。

x		x<<3		（逻辑）x>>2		（算术）x>>2	
十六进制	二进制	二进制	十六进制	二进制	十六进制	二进制	十六进制
0xC3	[11000011]	[00011000]	0x18	[00110000]	0x30	[11110000]	0xF0
0x75	[01110101]	[10101000]	0xA8	[00011101]	0x1D	[00011101]	0x1D
0x87	[10000111]	[00111000]	0x38	[00100001]	0x21	[11100001]	0xE1
0x66	[01100110]	[00110000]	0x30	[00011001]	0x19	[00011001]	0x19