4.7　虽然难以想象这条特殊的指令有什么实际的用处，但是在设计一个系统时，在描述中避免任何歧义是很重要的。我们想要为这条指令的行为确定一个合理的规则，并且保证每个实现都遵循这个规则。

在这个测试中，subq指令将%rsp的起始值与压入栈中的值进行了比较。这个减法的结果为0，表明压入的是%rsp的旧值。

4.8　更难以想象为什么会有人想要把值弹出到栈指针。我们还是应该确定一个规则，并且坚持它。这段代码序列将0xabcd压入栈中，弹出到%rsp，然后返回弹出的值。由于结果等于0xabcd，我们可以推断出popq %rsp将栈指针设置为从内存中读出来的那个值。因此，它等价于指令movq (%rsp), %rsp。

4.9　EXCLUSIVE-OR函数要求两个位有相反的值：

```
bool xor = (!a && b) || (a && !b);
```

通常，信号eq和xor是互补的。也就是，一个等于1，另一个就等于0。

4.10　EXCLUSIVE-OR电路的输出是位相等值的补。根据德摩根定律(网络旁注DATA:BOOL)，我们能用OR和NOT实现AND，得到如图4-71所示的电路：

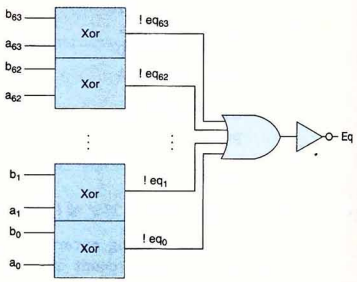

图4-71　练习题4.10的答案

4.11　我们可以看到情况表达式的第二部分可以写为

```
B <=  C          : B;
```

由于第一行将检测出A为最小元素的情况，因此第二行就只需要确定B还是C是最小元素。

4.12　这个设计只是对从三个输入中找出最小值的简单改变。

```
word Med3 = [
        A <= B && B <= C : B;
        C <= B && B <= A : B;
        B <= A && A <= C : A;
        C <= A && A <= B : A;
        1                : C;
];
```

4.13　这些练习使各个阶段的计算更加具体。从目标代码中我们可以看到，指令位于地址0x016。它由10个字节组成，前两个字节为0x30和0xf4。后八个字节是0x0000000000000080(十进制128)按字节反过来的形式。

阶段	通用	具体
	irmovq V, rB	irmovq $128, %rsp
取指	icode:ifun ← M_1[PC] rA:rB ← M_1[PC+1] valC ← M_8[PC+2] valP ← PC+10	icode:ifun ← M_1[0x016]=3:0 rA:rB ← M_1[0x017]=f:4 valC ← M_8[0x018]=128 valP ← 0x016+10=0x020
译码		
执行	valE ← 0+valC	valE ← 0+128=128
访问		
写回	R[rB]← valE	R[%rsp]← valE=128
更新PC	PC ← valP	PC ← valP=0x020

这个指令将寄存器%rsp设为128，并将PC加10。

4.14　我们可以看到指令位于地址0x02c，由两个字节组成，值分别为0xb0和0x00f。pushq指令(第6行)将寄存器%rsp设为了120，并且将9存放在了这个内存位置。