4.28　这个题目非常有趣，它试图在一组划分中找到优化平衡。它提供了大量的机会来计算许多流水线的吞吐量和延迟。

A. 对一个两阶段流水线来说，最好的划分是块 A、B 和 C 在第一阶段，块 D、E 和 F 在第二阶段。第一阶段的延迟为 170ps，所以整个周期的时长为 170＋20＝190ps。因此吞吐量为 5.26 GIPS，而延迟为 380ps。

B. 对一个三阶段流水线来说，应该使块 A 和 B 在第一阶段，块 C 和 D 在第二阶段，而块 E 和 F 在第三阶段。前两个阶段的延迟均为 110ps，所以整个周期时长为 130ps，而吞吐量为 7.69 GIPS。延迟为 390ps。

C. 对一个四阶段流水线来说，块 A 为第一阶段，块 B 和 C 在第二阶段，块 D 是第三阶段，而块 E 和 F 在第四阶段。第二阶段需要 90ps，所以整个周期时长为 110ps，而吞吐量为 9.09 GIPS。延迟为 440ps。

D. 最优的设计应该是五阶段流水线，除了 E 和 F 处于第五阶段以外，其他每个块是一个阶段。周期时长为 80＋20＝100ps，吞吐量为大约 10.00 GIPS，而延迟为 500ps。变成更多的阶段也不会有帮助了，因为不可能使流水线运行得比以 100ps 为一周期还要快了。

4.29　每个阶段的组合逻辑都需要 300/$k$ps，而流水线寄存器需要 20ps。

A. 整个的延迟应该是 300＋20$k$ps，而吞吐量(以 GIPS 为单位)应该是

$$\frac{1\,000}{\frac{300}{k}+20}=\frac{1\,000k}{300+20k}$$

B. 当 k 趋近于无穷大，吞吐量变为 1 000/20＝50 GIPS。当然，这也使得延迟为无穷大。

这个练习题量化了很深的流水线引起的收益下降。当我们试图将逻辑分割为很多阶段时，流水线寄存器的延迟成为了一个制约因素。

4.30　这段代码非常类似于 SEQ 中相应的代码，除了我们还不能确定数据内存是否会为这条指令产生一个错误信号。

```
# Determine status code for fetched instruction
word f_stat = [
        imem_error: SADR;
        !instr_valid : SINS;
        f_icode == IHALT : SHLT;
        1 : SAOK;
];
```

4.31　这段代码只是简单地给 SEQ 代码中的信号名前加上前缀“d_”和“D_”。

```
word d_dstE = [
        D_icode in { IRRMOVQ, IIRMOVQ, IOPQ} : D_rB;
        D_icode in { IPUSHQ, IPOPQ, ICALL, IRET } : RRSP;
        1 : RNONE;  # Don't write any register
];
```

4.32　由于 popq 指令(第 4 行)造成的加载/使用冒险，rrmovq 指令(第 5 行)会暂停一个周期。当它进入译码阶段，popq 指令处于访存阶段，使 M_dstE 和 M_dstM 都等于%rsp。如果两种情况反过来，那么来自 M_valE 的写回优先级较高，导致增加了的栈指针被传送到 rrmovq 指令作为参数。这与练习题 4.8 中确定的处理 popq %rsp 的惯例不一致。

4.33　这个问题让你体验一下处理器设计中一个很重要的任务——为一个新处理器设计测试程序。通常，我们的测试程序应该能测试所有的冒险可能性，而且一旦有相关不能被正确处理，就会产生错误的结果。

对于此例，我们可以使用对练习题 4.32 中所示的程序稍微修改的版本：