不作答疑，答案供参考

4.16，4.19，4.22，4.26，4.27

4.16

(1)流水线：350；非流水线：1250

(2)流水线：1750；非流水线：1250

(3)将 ID 阶段分割，这将时钟周期时间降低到 300ps。

(4)35%

(5)65%

4.19

x15 = 54（代码将正确运行，因为第一条指令的结果在第五个周期的开始被写回到寄存器文件中，而最后一条指令在本周期后半段读取了更新后的 x1 值。）

4.22

(1)停顿用**标记：

```
sd  x29, 12(x16)  IF ID EX ME WB
ld  x29, 8(x16)      IF ID EX ME WB
sub x17, x15, x14       IF ID EX ME WB
bez x17, label             ** ** IF ID EX ME WB
add x15, x11, x14                 IF ID EX ME WB
sub x15,x30,x14                      IF ID EX ME WB
```

(2)重排代码不会有帮助。每条指令都必须被获取；因此，每次数据访问都会导致一个停顿。重排代码只是会改变发生冲突的指令对。

(3)不能通过插入 NOP 指令来解决这个结构性的冒险，因为即使是 NOP 指令也必须从指令内存中获取。

(4)35%。每次数据访问都会导致一个停顿。

4.26

4.26.1

```
// EX to 1st only:
add x11, x12, x13
add x14, x11, x15
add x5, x6, x7
// MEM to 1st only:
ld x11, 0(x12)
add x15, x11, x13
add x5, x6, x7
// EX to 2nd only:
add x11, x12, x13
add x5, x6, x7
add x14, x11, x12
// MEM to 2nd only:
ld x11, 0(x12)
```

```
add x5, x6, x7
add x14, x11, x13
// EX to 1st and EX to 2nd:
add x11, x12, x13
add x5, x11, x15
add x16, x11, x12
```

4.26.2

```
// EX to 1st only: 2 nops
add x11, x12, x13
nop
nop
add x14, x11, x15
add x5, x6, x7
// MEM to 1st only: 2 stalls
ld x11, 0(x12)
nop
nop
add x15, x11, x13
add x5, x6, x7
// EX to 2nd only: 1 nop
add x11, x12, x13
add x5, x6, x7
nop
add x14, x11, x12
// MEM to 2nd only: 1 nop
ld x11, 0(x12)
add x5, x6, x7
nop
add x14, x11, x13
// EX to 1st and EX to 2nd: 2 nops
add x11, x12, x13
nop
nop
add x5, x11, x15
add x16, x11, x12
```

4.26.3

考虑以下代码：

```
ld x11, 0(x5) #  从内存阶段到第二个阶段---需要一个停顿
add x12, x6, x7 #  从执行阶段到第一个阶段---需要两个停顿
add x13, x11, x12
add x28, x29, x30
```

如果我们分别分析每条指令，我们会得出需要增加 3 个停顿（一个是因为“从内存阶段到第二个阶段”，两个是因为“仅从执行阶段到第一个阶段”）。但是，正如下面所示，我们实际上只需要两个停顿：

ld x11, 0(x5)
add x12, x6, x7
nop # 无操作指令
nop # 无操作指令
add x13, x11, x12
add x28, x29, x30

4.26.4

根据 4.26.2 中的答案进行加权平均，得到 0.05*2+0.2*2+0.05*1+0.1*1+0.1*2= 0.85 每条指令平均 0.85 个停顿（对于 CPI 为 1.85）。这意味着每 1.85 个周期中有 0.85 个停顿，或者说 46%的周期是停顿的。

4.26.5

唯一不能通过转发处理的依赖是从 MEM 阶段到下一条指令的依赖。因此，20%的指令将产生一个停顿，对于 CPI 为 1.2。这意味着在 1.2 个周期中有 0.2 个停顿，或者说 17%的周期是停顿的。

4.26.6

如果我们只从 EX/MEM 寄存器进行数据转发，我们有以下停顿/NOP 指令：
EX to 1st: 0
MEM to 1st: 2
EX to 2nd: 1
MEM to 2nd: 1
EX to 1st and 2nd: 1
这代表了一个平均值为 0.05*0+0.2*2+0.05*1+0.10*1+0.10*1=0.65 每指令 0.65 个停顿。因此，CPI（每指令周期数）为 1.65。

如果我们只从 MEM/WB 进行数据转发，我们有以下停顿/NOP 指令：
EX to 1st: 1
MEM to 1st: 1
EX to 2nd: 0
MEM to 2nd: 0
EX to 1st and 2nd: 1
这代表了一个平均值为 0.05*1+0.2*1+0.1*1=0.35 每指令 0.35 个停顿。因此，CPI 为 1.35。

4.26.7

	No forwarding	EX/MEM	MEM/WB	Full Forwarding
CPI	1.85	1.65	1.35	1.2
Period	120	120	1.20	130
Time	222n	198n	162n	156n
Speedup	–	1.12	1.37	1.42

4.26.8

完全转发时的 CPI 为 1.2，“time travel”转发时的 CPI 为 1.0；完全转发时的时钟周期为 130，“time travel”转发时的时钟周期为 230。则加速比为(1.2*130) / (1*230) = 0.68（这意味着“time travel”转发实际上降低了 CPU 的速度。）

4.26.9

在考虑 4.26.6 中的“EX/MEM”转发时，“EX 到第一个”不会产生停顿，但是“EX 同时到第一和第二个”会产生一个停顿。然而，“MEM 到第一个”和“MEM 同时到第一和第二个”将始终产生相同数量的停顿。（所有“MEM 到第一个”的依赖都会导致停顿，无论转发类型如何。这个停顿导致第二条指令的 ID 阶段与基准指令的 WB 阶段重叠，在这种情况下不需要转发。）

4.27

4.27.1

```
add x15, x12, x11
nop
nop
ld x13, 4(x15)
ld x12, 0(x2)
nop
or x13, x15, x13
nop
nop
sd x13, 0(x15)
```

4.27.2

不可能减少 NOP 指令的数量。

4.27.3

这段代码执行是正确的。我们只需要冒险检测来在紧随加载指令之后的指令使用加载结果时插入一个停顿。但在这个案例中并没有发生这种情况。

4.27.4

Cycle	1	2	3	4	5	6	7	8	
add	IF	ID	EX	ME	WB				
ld		IF	ID	EX	ME	WB			
ld			IF	ID	EX	ME	WB		
or				IF	ID	EX	ME	WB	
sd					IF	ID	EX	ME	WB

因为在这段代码中没有停顿，PCWrite 和 IF/IDWrite 始终为 1，而 ID/EX 之前的 mux 始终设置为传递控制值。

(1) ForwardA = X; ForwardB = X （执行阶段还没有指令）
(2) ForwardA = X; ForwardB = X （执行阶段还没有指令）
(3) ForwardA = 0; ForwardB = 0 （不进行转发；值从寄存器中获取）
(4) ForwardA = 2; ForwardB = 0 （基础寄存器从之前指令的结果中获取）
(5) ForwardA = 1; ForwardB = 1 （基础寄存器从两个指令之前的结果中获取）
(6) ForwardA = 0; ForwardB = 2 （rs1 = x15 从寄存器中获取；rs2 = x13 从第一条 ld 指令的结果中获取—两个指令之前）
(7) ForwardA = 0; ForwardB = 2 （基础寄存器从寄存器文件中获取。要写入的数据从之前的指令中获取）

4.27.5

冒险检测单元还需要从 MEM/WB 寄存器中输出的 rd 值。当前处于 ID 阶段的指令如果依赖于由执行阶段（EX）的指令或内存阶段（MEM）的指令产生的值（或从这些指令转发过来的值），则需要被停顿。因此，我们需要检查这两个指令的目标寄存器。冒险单元已经有了来自 EX/MEM 寄存器的 rd 值作为输入，所以我们只需要添加来自 MEM/WB 寄存器的值。

不需要额外的输出。我们可以使用我们已经拥有的三个输出信号来停顿流水线。

来自 EX/MEM 的 rd 值需要用来检测加法指令和紧随其后的 ld 指令之间的数据冒险。来自 MEM/WB 的 rd 值需要用来检测第一条 ld 指令和 or 指令之间的数据冒险。

4.27.6

Cycle	1	2	3	4	5	6
add	IF	ID	EX	ME	WB	
ld		IF	ID	–	–	EX
ld			IF	–	–	ID

(1) PCWrite = 1; IF/IDWrite = 1; control mux = 0

(2) PCWrite = 1; IF/IDWrite = 1; control mux = 0

(3) PCWrite = 1; IF/IDWrite = 1; control mux = 0

(4) PCWrite = 0; IF/IDWrite = 0; control mux = 1

(5) PCWrite = 0; IF/IDWrite = 0; control mux = 1