4.9

I1: OR R1,R2,R3

I2: OR R2,R1,R4

I3: OR R1,R1,R2

1) RAW on R1：在I1指令中需要写入R1，I2， I3都需要读入R1

RAW on R2：在I2指令中需要写入R2，I3需要读入R2

WAR on R2 ： 在I1指令中需要读入R2，I3需要写入R2

WAR on R1： 在I2指令中需要读入R1，I3需要写入R1

WAW on R1：在I1指令中需要写入R1，I3也需要写入R1

2)若是没有旁路，在上述讨论中WAW, WAR不会造成冒险，RAW会，由于R1, R2在I1, I2之后都被写入，每次写入作用域为3条指令，所以I1， I2指令后都需要加入两条NOP：

```
I1: OR R1,R2,R3
NOP
NOP
I2: OR R2,R1,R4
NOP
NOP
I3: OR R1,R1,R2
```

3)当有充分的旁路时，上述R类型指令间的数据冒险不会发生(ALU阶段计算的结果会被转发到EX阶段，解决RAW)，不需要插入NOP：

```
I1: OR R1,R2,R3

I2: OR R2,R1,R4

I3: OR R1,R1,R2
```

4)若是无旁路时，总共有7条指令，共需要11个周期，时间为： $11 \times 250 = 2750ps$

若是有旁路时，总共有3条指令，共需要7个周期，时间为： $7 \times 300 = 2100ps$

加速比为 $2750/2100 = 1.31$

5)若是没有MEM到EX的旁路，那么次相邻指令之间是不能有转发的，I1指令对于R1的改变就传不到I3, 这样的数据改变没有转发的话，只能进行阻塞，但是如果只在相邻指令之间加入一条NOP, 原本相邻的转发就会变成次相邻的转发，还是会产生冒险。

所以若是没有MEM到EX的转发，只能在相邻指令间加入两条NOP，其效果和无旁路一样：

```
I1: OR R1,R2,R3
NOP
NOP
I2: OR R2,R1,R4
NOP
NOP
I3: OR R1,R1,R2
```

6)由上所述，周期数没有减少，但是周期长度增加，总执行时间为 $11 \times 290 = 3190ps$

加速比为$2750/3190 = 0.86$

4.13

1)

```
I1: ADD R5,R2,R1                    //R5改变，作用域为3条
NOP
NOP
I2: LW R3,4(R5)                          //R3改变，作用域为3条
I3: LW R2,0(R2)
NOP
I4: OR R3,R5,R3                          //R3改变，作用域为3条
NOP
NOP
I5: SW R3,0(R5)
```

2)按照题意来移动指令顺序来减少冒险，由于每个寄存器都有用处，R7用不上：

```
I1: ADD R5,R2,R1
I3: LW R2,0(R2)                                  //I3指令用不到R5，因此提前
NOP                                             //R5改变作用域为3，必须插入1条NOP
I2: LW R3,4(R5)
NOP                                             //R3改变作用域为3，必须插入2条NOP
NOP
I4: OR R3,R5,R3                                  //R3改变作用域为3，必须插入2条NOP
NOP
NOP
I5: SW R3,0(R5)
```

3)实现了旁路，但是没有冒险检测单元，还是有问题的。比如在数据转发时需要判断当前需要改变值的寄存器是不是下一条指令要用的，这需要冒险检测单元的判断，如果是的，则需要转发，不是则无需转发。旁路需要冒险检测单元的决策。

4.14

4)第一条分支指令是beq类型，可能有三种需要检测：

①上一条是R指令，当beq进入ID阶段时，R指令还在ALU计算可能要用到的寄存器的值，需要阻塞一个周期

②上一条是lw指令，当beq进入ID阶段时，lw指令还在MEM阶段，可能要用的寄存器还未写回，需要阻塞两个周期

③上上条是lw指令，当beq进入ID阶段时，可能要用的寄存器还未写回，需要阻塞一个周期

由于上述冒险都是在数据没有准备好前使用，所以是数据冒险。

5)当分支在EX阶段执行，且分支总预测发生时，总共需要14个周期，当分支在ID阶段执行，且分支总预测发生时，总共需要15个周期(原本9个周期，两个lw指令阻塞4周期，两次跳转预测错误跳转2周期)

加速比为 $14/15 = 0.93$

6）当把执行分支移到ID阶段，需要比较两个寄存器的值，这值可能被上一条指令的ALU刚刚计算更新过，或者次上条还在MEM阶段，但是都还未写回，所以这时候需要往上两个阶段的旁路，所以在比较器的前面需要两个数据选择器，这与ALU的旁路单元是类似的，所以复杂度还是一样。

4.16

1) 分支总发生： $3/5 = 60\%$　　分支总不发生： $2/5 = 40\%$

2)

分支模式	**T**	**NT**	**T**	**T**
模式变化	0 → 1	1 → 0	0 → 1	1 → 2
判断情况	×	√	×	×

正确率为25%

3)

分支模式	**T**	**NT**	**T**	**T**	**NT**	**准确率**
1	0 → 1(×)	1 → 0(√)	0 → 1(×)	1 → 2(×)	2 → 1(×)	20%
2	1 → 2(×)	2 → 1(×)	1 → 2(×)	2 → 3(√)	3 → 2(×)	20%
3	2 → 3(√)	3 → 2(×)	2 → 3(√)	3 → 3(√)	3 → 2(×)	60%
4	2 → 3(√)	3 → 2(×)	2 → 3(√)	3 → 3(√)	3 → 2(×)	60%
…	2 → 3(√)	3 → 2(×)	2 → 3(√)	3 → 3(√)	3 → 2(×)	60%

发现前两个循环状态还不够稳定，准确率只有20%，第三个循环开始处于稳定状态，准确率维持在60%