《计算机组成原理》第六次作业

信息安全 胡博浩 2212998

4.16

1)

流水线处理器时钟周期：350ps

非流水线处理器的时钟周期：250+350+150+300+200=1250ps

2)

流水线：350×5=1750ps

非流水线：250+350+150+300+200=1250ps

3)

选择拆分ID字段，拆分后

IF	ID1	ID2	EX	MEM	WB
250ps	175ps	175ps	150ps	300ps	200ps

处理器的时钟周期是300ps

4)

利用率是(20+15)/100=35%

5)

利用率是(45+20)/100=65%

4.19

最终结果为54

代码将正确运行，因为第一条指令的结果在第5个周期开始时被写回寄存器文件，而最后一条指令在该周期的后半部分读取x1的更新值。

4.22

1)

```
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)

重新排序代码是没有用的。每条指令都必须执行。

因此，每次数据访问都会导致停机，重新排序代码，只会改变有冲突的指令对。

3)

是的。我们使用nop无法解决这种结构上的危险，因为即使是nop也必须从指令存储器中获取。

4)

会产生35%的阻塞，每一次数据的访问都会导致失速。

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
```

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
```

3)

使用下面的代码:

```
ld  x11, 0(x5)
add x12, x6, x7
add x13, x11, x12
add x28, x29, x30
```

如果我们单独分析每条指令，我们将计算出我们需要增加3个stall(一个用于“MEM到第2位”，两个用于“EX到第1位”)。然而，如下图所示，我们只需要两个stall:

```
ld x11, 0(x5)
add  x12, x6, x7
nop
nop
add  x13, x11, x12
add  x28, x29, x30
```

4)

对4.26.2的答案进行加权平均，得到CPI为1.85时，每条指令(平均)的stall为0.05×2+0.2×2+0.05×1+0.1×1+0.1×2=0.85。这意味着0.85/1.85个周期，结果是46%，所以是失速。

5)

唯一不能通过转发处理的依赖关系是从MEM阶段到下一条指令。因此，当CPI为1.2时，20%的指令将产生一个失速。这意味着1.2个周期中有0.2个，即17%是停滞。

6)

如果我们只从EX/MEM登记册转发，我们有以下stall/nop:

- EX to 1st: 0
- MEM to 1st: 2
- EX to 2nd: 1
- MEM to 2nd: 1
- EX to 1st and 2nd: 1

这表示平均为0.05×0 + 0.2×2 + 0.05×1 + 0.10×1 + 0.10×1 = 0.65 个stall/指令。因此，

CPI为1.65。

如果我们只从MEM/WB转发，我们有以下stalls/nops:

- EX to 1st: 1
- MEM to 1st: 1
- EX to 2nd: 0
- MEM to 2nd: 0
- EX to 1st and 2nd: 1

这表示平均为 `0.05*1 + 0.2*1 + 0.1*1 = 0.35` 个stalls/指令。

因此，CPI为1.35。

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

8)

完全转发时CPI为1.2

“穿越”转发时CPI为1.0时钟周期

完全转发时CPI为130时钟周期

“穿越”转发时CPI为230时钟周期

Speedup=(1.2*130)/(1*230)=0.68

这意味着“穿越”转发实际上使CPU变慢了

9)

“EX/MEM”转发时，“EX to 1st”不产生stall，而“EX to 1st and EX to 2nd”产生一个stall。但是，“MEM to 1st"和“MEM to 1 and MEM to 2nd”将始终生成相同数量的stall。(无论转发的类型如何，所有“MEM to 1st”

依赖都会导致停机。这导致第二个指令的ID阶段与基本指令的WB阶段重叠，在这种情况下不需要转发。）

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)
```

2)

做不到减少nop指令的数量

3)

代码正确执行。只有当加载后的指令使用加载的结果时，我们才需要危险检测来插入一个失速，但是这种情况不会发生。

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总是1,ID/EX之前的mux总是设置为传递控制值。

(1) ForwardA = X;

(2) ForwardA = X;

(3) ForwardA = 0;ForwardB = 0(无转发)

(4) ForwardA = 2;ForwardB = 0(基寄存器取自上一条指令的结果)

(5)ForwardB = 1(基寄存器取自前两条指令的结果)

(6)ForwardA = 0;ForwardB= 2 (rs1 = x15从寄存器;rs2 = x13，取自第1个ld - 2条指令前的结果)

(7)ForwardA = 0;ForwardB = 2(从寄存器文件中获取的基本寄存器，要写入的数据取自先前的指令)

5)

危害检测单元还需要从MEM/WB寄存器中输出的rd值。如果当前处于ID阶段的指令依赖于由EX中的指令或MEM阶段中的指令产生(或从该指令转发)的值，则需要停止该指令。所以我们需要检查这两条指令的目的寄存器。Hazard单元已经有来自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止管道。

EX/MEM中的rd值用于检测add和下一条指令之间的数据危险，MEM/WB中的rd值用于检测第一条 old 指令和or指令之间的数据冒险。

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