**Pipeline**

Q1 A 5-stage pipelined RISCV datapath is illustrated in Fig. 1. Now we have the time information for each component tabulated in Table 1. (15 points) 

Fig. 1 5-stage pipelined RSICV datapath

Table 1

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Element | Register clk-to-q | Register Setup | MUX | ALU | Mem Read | Mem Write | RegFile Read | RegFile Setup |
| Parameter | tclk-to-q | tsetup | tmux | tALU | tMEMread | tMEMwrite | tRFread | tRFsetup |
| Delay(ps) | 30 | 20 | 25 | 250 | 400 | 500 | 200 | 50 |

1. What is the clock time and frequency of a pipelined CPU (ignore the branch comp and imm.)?

Clock time:

Tclk >= the longest cycle time =

IF: tpc clk-to-q + t\_imem read + t reg setup = 30+400+20=450 ps

Id: tReg clk-to-q + trf read + t reg setup = 30+200 +20 = 250 ps

EX: tReg clk-to-q + tmux +t alu + t reg setup = 30 + 25+250+20 = 325ps

MEM : tReg clk-to-q + tDMEM read + tmux + tReg setup = 30 + 400 + 25 + 20 = 475 ps

WB：　tReg clk-to-q＋ trf setup　＝ 30+20 =50 ps

The frequency of a pipelined CPU :

1\*10^12/475=2105263157.894737 = 2.1GHz

1. What is the purpose of the new registers?

We need to save the result of the last stage so that different instructions in the pipeline would not influence each other. What’s more, we could control each stage by generating appropriate control signals. 我们需要保存上一阶段的结果，这样流水线中的不同指令就不会相互影响。 更重要的是，我们可以通过产生适当的控制信号来控制每个阶段。

1. Why do we add +4 to the PC again in the memory stage?

此处指令的执行Execute阶段，是第三个，而这个第三阶段的Execute和指令的第一个阶段的Fetch取指，相差的值是 3 -1 =2，即两个CPU的Cycle，而每个Cycle都会导致PC=+PC+4，所以，指令到了Execute阶段，才会发现，此时PC已经变成PC=PC+8了。  
The Execute stage of the instruction here is the third of the five-stage pipeline, and the third stage of Execute and the Fetch of the first stage of the instruction are fetched. The difference is 3 -1 = 2, that is, two The cycle of the CPU, and each cycle will cause PC=+PC+4, so the instruction will not find until the Execute stage, at this time the PC has become PC=PC+8. .So we need add +4 to the PC again.

1. Why do we need to save the instruction in a register multiple times?

**Because each pipeline stage needs to receive correct control signals for the current instruction.**

**Hazards**

Data hazards occur due to data dependencies among instructions. Forwarding can solve many data hazards.

Q2. Spot the data dependencies in the code below and figure out how forwarding can resolve

data hazards. (10 points)



指令1和指令2冲突, 因为指令2需要指令1的t0, 指令3和指令1冲突,指令3需要指令1 的 t0.

And s2,t0,a0

Sw s0,100(t0)

Needs the result of addi, so after

ex addi(C2)

we needs to forward data to

ex And s2,t0,a0 #(C3) 转发到指令2的ex也就是C3开始之前

ex Sw s0,100(t0) #(C4) 执行阶段, sw 指令通过t0+offset算出地址, mem阶段, 把s0存入. 因此我们需要把c2的结果转发给sw 的执行阶段c4之前,sw的 reg阶段取过来的就不要了.

Q3. In general, under what conditions will an EX stage need to take in forwarded inputs from

previous instructions? Where should those inputs come from in regards to the current cycle?

Assume you have the signals ALUout(n), rt(n), rs(n), regWrite(n), and regDst(n), where n is 0

for the signal of the current instruction being executed by the EX stage, -1 for the previous, etc.

通常，在什么条件下，EX阶段需要从先前的指令中接收转发的输入？ 就当前周期而言，这些input应从何而来？假设您有信号ALUout（n），rt（n），rs（n），regWrite（n）和regDst（n），其中n为0，表示EX级正在执行的当前指令的信号- 1代表前一个，依此类推。

For example,

Forward ALUout(-1) if (rs2(0) == regDst(-1) || rs1(0) == regDst(-1)) && regWrite(-1)//如果要写回, 那就直接转发

Add s2 s1 s0

Add s3 s2 s5

Please give other conditions. (10 points)

Forward ALUout(-2) if (rs2(0) == regDst(-2) || rs1(0) == regDst(-2)) && regWrite(-2)// 还在dm,没有写回

Forward ALUout(-3) if (rs2(0) == regDst(-3) || rs1(0) == regDst(-3)) && regWrite(-3)//在wb 但是还是来不及.

Lw的转发不同,是在mem 后才有结果

Forward memDst(-1) if(rs2(0) == regDst(-1) || rs1(0) == regDst(-1))&& regWrite(-1)

Q4. Spot the data dependencies in the code below and figure out why forwarding cannot

resolve this hazard. What can we do to solve this data hazard? (10 points)



即使使用了旁路转发机制, 仍然会产生一次阻塞, 存储器访问的输出到执行级的输入之间的路径在时间上是倒着的,第二条指令需要在C3之前获得第一条指令mem的结果, 而第一条指令C3之后才完成mem.

我们可以插入气泡来解决这个数据冒险,或者重新排列指令.  
Even if the bypass mechanism is used, a blockage will still occur, and the path from the output of the memory access to the input of the execution stage is reversed in time, which is obviously impossible.

We can insert bubbles to solve this data adventure, or rearrange the instructions,

Q5. Given the RISC-V code below and a pipelined CPU with no forwarding, how many hazards

would there be? What types are each hazard? Consider all possible hazards from all pairs of

instructions. How many stalls would there need to be in order to fix the data hazard(s)? What about the

control hazard(s)? (10 points)



And s2,t0,a0 有数据冒险, 依赖前面的数据t0. 要停三个时钟周期, 写回后才能取reg.

Sw s2,100(t0) 有数据冒险, 依赖前面的数据t0(停到C4之后再reg)和s2(停到C3之后再执行),停两个周期.

Beq s0 ,s3,label 控制冒险, 因为可能跳转, 但是后面的读入进流水线了.要停三个周期, 直到mem级才知道是否执行分支, 我们可以分支预测来消除.

Addi t2,x0,x0

Q6. Besides stalling, what can we do to resolve control hazards? (5 points)

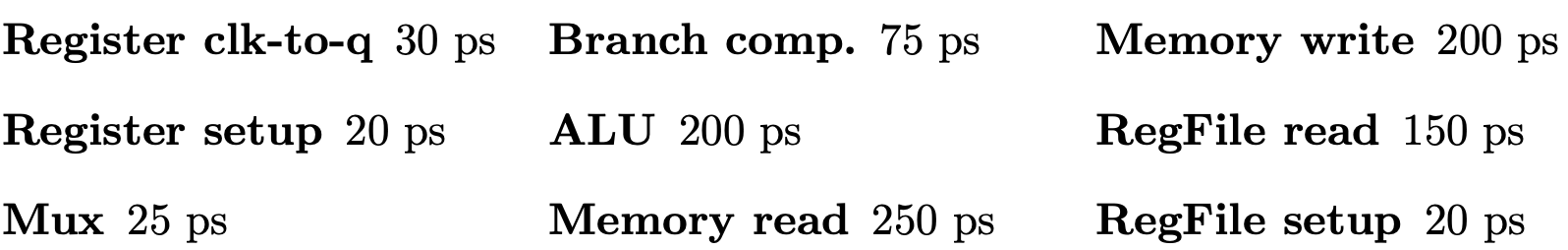
1.分支预测,这样流水线可以全速执行, 甚至可以动态预测, 根据本次预测的结果来调整下一次的预测行为. 历史记录足够多时, 正确率可以达到90%, 预测错误时, 让后面的指令执行不生效, flush 流水线, 在正确的分支地址处重新开始启动流水线. 一种方法是预测分支总是不发生.

2.延迟分支, 比如用一些不影响分支的指令来隐藏分支延迟, 但是只有分支延迟较短的时候延迟分支才有效, 对更长的延迟分支, 一般都使用硬件分支预测器.  
Branch prediction, one method is to predict that the branch will not occur, so that the pipeline can be executed at full speed, and can even be dynamically predicted, according to the results of this prediction to adjust the next prediction behavior. When there are enough historical records, the accuracy rate can reach 90%, and the prediction When an error occurs, the execution of the following instructions will not take effect, and the pipeline will be restarted at the correct branch address.

There is also a delayed branch.For example, some instructions that do not affect the branch are used to hide the branch delay, but the delayed branch is only effective when the branch delay is short.For longer delayed branches, hardware branch predictors are generally used.

**Performance Analysis**

Q7. (10 points)



1. With the delays provided above for each of the datapath components, what would be the fastest possible clock time for a single cycle datapath?

30+75+200+0(setup ignored)+200+150+25+250+0(setup ignored)=930ps

IF: tpc clk-to-q + t\_imem read + t reg setup = 30+250+20=300 ps

Id: tReg clk-to-q + trf read + t reg setup = 30+150+20 = 200 ps

EX: tReg clk-to-q + tmux +t alu + t reg setup = 30 + 25+200+20 = 275ps

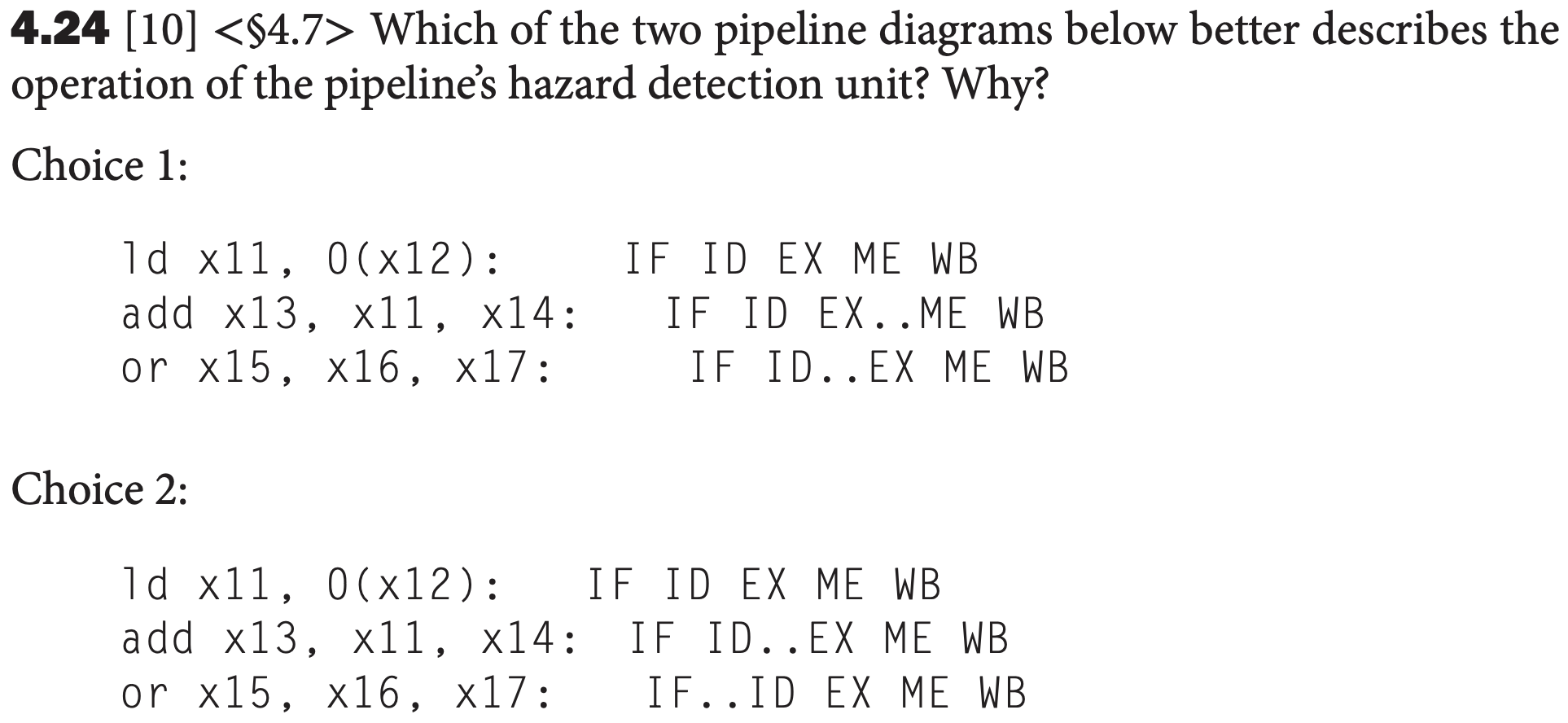
MEM : tReg clk-to-q + tDMEM read + tmux + tReg setup = 30+250+25+20=325 ps

WB：　tReg clk-to-q＋ trf setup　＝ 30+20 =50 ps

1. What is the fastest possible clock time for a pipelined datapath?

the fastest possible clock time = The longest stage time = MEM =325ps

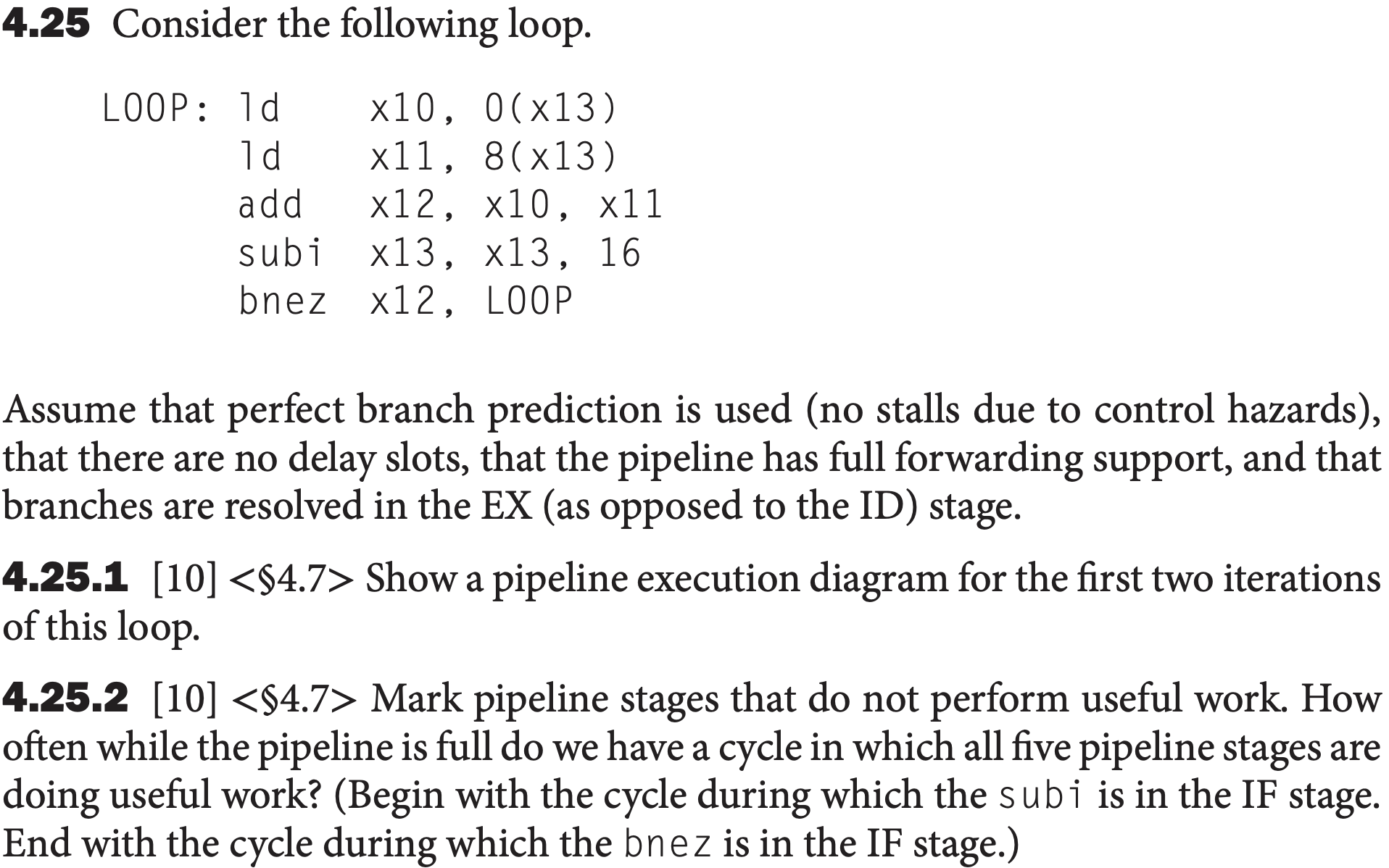
Q8. (10 points)



在add 执行前检测, 因为不一定读入的x11更新了.

在or 译码前检测. 检测装载指令, 如果id 指令被阻塞, if也必须阻塞, 否则已经取到的指令就会丢失.

Q9. (20 points)



As opposed to the ID stage.

Assume that perfect branch prediction is used(no stalls due to control hazards)是说没有control hazards造成的停, 而不是说没有stall.

4.25.1

Show a pipeline execution diagream for the first two iterations of this loop.

如下图所示



显示此循环的前两次迭代的流水线执行图

4.25.2

How often就是说占总的百分比多少.

8和9是five pipeline stages are doing useful work , 5到12总共8个,因此 占25%