4.8

4.8.1) In a non-pipelined processor it executes 1 instruction per cycle so it’s as long as it takes to get through all the stages

Cycle time = 250ps + 350ps + 150ps +300ps +200ps = 1250ps

In a pipelined processor it still executes, as long as no stalls are needed, 1 instruction per cycle except the cycles are only as long as the longest stage, which in this case is the ID stage, therefore:

Cycle time = 350ps

4.8.2) The amount of time it takes to complete a single instruction is not reduced by pipelining so the amount of time it takes to complete a lw instruction in a pipelined processor is the same amount of time in a non-pipelined processor so they both take 1250ps to complete a lw instruction

4.8.3) I would split the ID stage into two different stages so the new clock cycle time would become 250 rather than 350. To see the improvement let’s take a set of 5 instructions as an example. In the original pipelined design with a clock cycle time of 350ps it would take:

(# of stages \* ct) +((#instructions -1) \*ct) = 5 \* 350 + 4 \*350 = 3150ps

In the new pipeline design it would take:

6 \* 250 + 4 \* 250 = 2500ps.

4.9

4.9.1) There is a RAW dependency on r1 between instruction 1 and 2. There is a WAR on r2 between instructions 2 and 1. There’s a RAW dependency on r1 between instructions 1 and 3. There’s a WAR on r1 between instruction 3 and 2. There’s a RAW dependency on r2 between instructions 2 and 3. There’s a WAW dependency on r1 between instruction 1 and 3.

4.9.2) or r1,r2,r3 RAW

Noop

Noop

or r2,r1,r4 RAW

Noop

Noop

or r1,r1,r2

These 2 are the only hazards that require no-ops to be added. The other dependencies were

noted above in 4.9.1.

4.9.3)

or r1,r2,r3 RAW

no op

or r2,r1,r4 RAW

no op

or r1,r1,r2

4.9.4)

Without forwarding: 11 cycles \* 250ps =2750ps

With forwarding: 9 cycles = 2700ps

The speedup is about 1.02.

4.13

4.13.1

add r5,r2,r1 RAW

no op

no op

lw r3,4(r5) WAW on the destination of instruction 4 and RAW on the target

lw r2,0(r2) WAR

or r3,r5,r3 WAW on the destination of instruction 5 and WAR on the destination

sw r3,0(r5)

4.13.2

Add r5, r2,r1

Lw r2,0(r2)

Or r3,r5,r3

Lw r3,4(r5)

Sw r3, 0(r5)

4.13.3

All the instructions would run fine up till lw r3,4(r5) which would need a stall in between it and Add r5, r2,r1 but in the end it wouldn’t matter too much as the value of r3 is determined by the sw statement at the end.

4.15

4.15.1

In order to determine the extra CPI due to mispredicted branches we will need to use the frequency of the branch instruction, the new CPI because of the misprediction and the prediction inaccuracy for the prediction method, since the stall will only occur when it makes an incorrect prediction.

Beq occurrence =25%

Prediction method inaccuracy = 1 - .45 = .55

CPI with stall = 2

Extra CPI = .25 \* .55 \* 2 = .275

New cpi = 1+.275 = 1.275

4.16

4.16.1

The accuracy of always taken in this scenario is 3/5 or 60% and the accuracy for always not taken is 2/5 or 40%.

4.16.2

For the first 4 branches it will only have a 25% accuracy since it will start off by predicting not taken which it is taken so it moves to the second predict not taken. In this branch it will make the correct decision which means it will predict not taken again for the next branch which is incorrect. Since there is only one incorrect decision in a row currently it will make the same prediction as before which was not taken which is again incorrect. Therefore it was only correct for one of the four branches.

4.16.3

If this cycle continued infinitely it would eventually work out to an accuracy of 60%. The first two cycles of the branch sequence T NT T T NT would only be right once per cycle but after that it is right for 3/5 of the branches and since the cycle is repeated forever it will approximately be right 60% of the time.