Homework Assignment 5 Solution

Exercise 4.8

4.8.1 ***(10 points)***

Clock cycle time for pipelined = 350 ps.
Clock cycle time for single-cycle = 1250 ps.

4.8.2 ***(10 points)***

LW latency for pipelined = 1750 ps.
LW latency for single-cycle = 1250 ps.

4.8.3 ***(10 points)***

We should split ID stage because it is the critical path (slowest stage).
Clock cycle time for pipelined = 300 ps.

Exercise 4.9

4.9.1 ***(10 points)***

Instruction sequence	Dependences
I1: OR R1,R2,R3	RAW on R1 from I1 to I2 and I3
I2: OR R2,R1,R4	RAW on R2 from I2 to I3
I3: OR R1,R1,R2	WAR on R2 from I1 to I2
	WAR on R1 from I2 to I3
	WAW on R1 from I1 to I3

4.9.2 ***(5 points)***

In the basic five-stage pipeline WAR and WAW dependences do not cause any hazards. Without forwarding, any RAW dependence between an instruction and the next two instructions (if register read happens in the second half of the clock cycle and the register write happens in the first half) will cause 2 stalls. To eliminates these hazards, we can insert NOP instructions as shown below

Instruction sequence	
OR R1,R2,R3	
NOP	Delay I2 to avoid RAW hazard on R1 from I1
NOP	
OR R2,R1,R4	
NOP	Delay I3 to avoid RAW hazard on R2 from I2
NOP	
OR R1,R1,R2	

4.9.3 ***(5 points)***

With full forwarding, an ALU instruction can forward a value to EX stage of the next instruction without a stall (NOP).

Instruction sequence	
OR R1,R2,R3	
OR R2,R1,R4	No RAW hazard on R1 from I1 (forwarded)
OR R1,R1,R2	No RAW hazard on R2 from I2 (forwarded)

4.9.4 ***(15 points)***

No forwarding: 11 cycles → 11 × 250 = 2750 ps

With forwarding: 7 cycles → 7 × 300 = 2100 ps

Speedup = 2750 / 2100 = 1.31

Exercise 4.13

4.13.1 ***(10 points)***

```
ADD R5,R2,R1
NOP
NOP
LW R3,4(R5)
LW R2,0(R2)
NOP
OR R3,R5,R3
NOP
NOP
SW R3,0(R5)
```

4.13.2 ***(10 points)***

We can usually move up an instruction by swapping its place with another instruction that has no dependences with it, so we can try to fill some NOP slots with such instructions. We can also use R7 to eliminate WAW or WAR dependences so we can have more instructions to move up. Unfortunately in this code, we still need 5 NOPs.

I1: ADD R5,R2,R1 I3: LW R2,0(R2) NOP I2: LW R3,4(R5) NOP NOP I4: OR R3,R5,R3 NOP NOP I5: SW R3,0(R5)	Moved up to fill NOP slot Had to add another NOP here, so there is no performance gain

4.13.3 ***(5 points)***

Code executes correctly (for both loads, there is no RAW dependence between the load and the next instruction).

Exercise 4.15

4.15.1 ***(10 points)***

When the branch outcomes are determined at the EX stage, The branch CPI = 1 if prediction is correct and branch CPI = 3 if the prediction is wrong.

Average branch CPI for always-taken prediction (45% accuracy) = 1×45% + 3×55% = 2.1
The overall CPI would be: 2.1×25% + 1×75% = 1.275
So the extra CPI is 0.275 cycles,

Exercise 4.16

4.16.1 ***(10 points)***

Always Taken	Always not-taken
3/5 = 60%	2/5 = 40%

4.16.2 ***(5 points)***

For the first 4 branches in this pattern, the two-bit predictor gives: (0 is NT, 1 is T)

Outcomes	Predictor value at time of prediction	Correct or Incorrect	Accuracy
T, NT, T, T	0,1,0,1	I,C,I,I	25%

4.16.3 ***(5 points)***

The first few recurrences of this pattern do not have the same accuracy as later ones because the predictor is still warming up. To determine the accuracy in the “steady state”, we must work through the branch predictions until the predictor values start repeating (i.e., until the predictor has the same value at the start of the current and the next recurrence of the pattern, in this case 2,3,2,3,3 or T, T, T, T, T).

Outcomes	Predictor value at time of prediction	Correct or Incorrect (in steady state)	Accuracy in steady state
T, NT, T, T, NT	1st occurrence: 0,1,0,1,2 2nd occurrence: 1,2,1,2,3 3rd occurrence: 2,3,2,3,3 4th occurrence: 2,3,2,3,3	C,I,C,C,I	60%

Extra Credit ***(12 points)***

Technique	Reduces
Dynamic scheduling	**Data hazard stalls**
Branch prediction	**Control stalls**
Multiple Issue	**CPI**
Speculation	**Data and control stalls**
Loop unrolling	**Control hazard stalls**
Compiler pipeline scheduling	**Data hazard stalls**