CS24 Elementary Computer Organization
Chapter 4 Exercises: 4.5, 4.8, 4.11, 4.13

4.5 For the problems in this exercise, assume that there are no pipeline stalls and that the breakdown of executed instructions is as follows:

add	addi	not	beq	lw	sw
20%	20%	0%	25%	25%	10%

4.5.1 [10] <4.3> In what fraction of all cycles is the data memory used?

Of all the instructions, the ones that use data memory for storing and loading words are 'lw' and' sw'. Since lw is 25% of the instructions and sw is 10% of the instructions that means that 35% of the instructions utilize the memory step in their processing. Since all the instructions have the same number of cycles (5 cycles in this case), 35% of those cycles will use the data memory step during the pipeline.

4.8 In this exercise, we examine how pipelining affects the clock cycle time of the processor. Problems in this exercise assume that individual stages of the datapath have the following latencies:

IF	ID	EX	MEM	WB
250ps	230ps	150ps	300ps	200ps

Also, assume that the instructions executed by the processor are broken down as follows:

alu	beq	lw	sw
45%	20%	20%	15%

4.8.1 [5] <4.5> What is the clock cycle time in a pipelined and non-pipelined processor?

The clock cycle time in a non-pipelined processor is defined as the sum of all the latencies for each step of the instruction execution process. The reason one clock cycle is the sum of all the latencies is due to a non-pipelined processor only being capable of executing one instruction at a time, therefore one instruction every clock cycle.

$$250\text{ps} + 350\text{ps} + 150\text{ps} + 300\text{ps} + 200\text{ps}$$
$$=$$
$$\boxed{1250\text{ps}}$$

In a pipelined processor, the clock cycle time is defined as the latency of the step that takes the most time to complete in the instruction execution process. In this case, the largest latency is 350ps. This is due to the fact that we want to be able to execute every step of the instruction execution process in the same amount of time so that we can maintain a constant cycle time as we move throughout the pipeline. Thus, each clock cycle will take $\boxed{350\text{ps}}$. The latency of the entire instruction may be larger than that for the non-pipelined processor, but the through put increases meaning that we have a much higher efficiency in executing instructions as they happen in parallel with each other.

4.8.2 [10] <4.5> What is the latency of an LW instruction in a pipelined and non-pipelined processor?

The total latency of a single instruction in a non-pipelined processor is equal to the sum of each of the latencies of each step of its execution:

$$250\text{ps} + 350\text{ps} + 150\text{ps} + 300\text{ps} + 200\text{ps}$$
$$=$$
$$\boxed{1250\text{ps}}$$

The total latency of a single instruction in a pipelined processor is equal to the sum of each of the latencies of each step of its execution as well:

$$350\text{ps} + 350\text{ps} + 350\text{ps} + 350\text{ps} + 350\text{ps}$$
$$=$$
$$\boxed{1750\text{ps}}$$

Again, the latency for the pipelined processor may be higher than that of the non-pipelined processor, but the through-put is much higher than that of the non-pipelined processor meaning its much more efficient in the amount of instructions it can execute in a given period of time.

4.11 Consider the following loop.
loop:lw r1, 0(r1)
and r1, r1, r2
lw r1, 0(r1)
lw r1, 0(r1)
bew r1, r0, loop vspace0.3cm

Assume that perfect branch prediction is used (no stalls due to control hazards), that there are no delay slots, and that the pipeline has full forwarding support. Also assume that many iterations of this loop are executed before the loop exits.

4.11.1 [10] <4.6> Show a pipeline execution diagram for the third iteration of this loop, from the cycle in which we fetch the first instruction up to (but not including) the cycle in which we can fetch the first instruction of the next iteration. show all instructions that are in the pipeline during these cycles (not just those from the third iteration).

	1	2	3	4	5	6	7	8	9	10	11	12	13
lw r1, 0(r1)	W												
beq r1, r0, loop	D	E	M	W									
lw r1, 0(r1)	F	D	E	M	W								
and r1, r1, r2		()	F	D	E	M	W						
lw r1, 0(r1)			()	F	D	E	M	W					
lw r1, 0(r1)				()	()	F	D	E	M	W			
beq r1, r0, loop					()	()	()	()	F	D	E	M	W
lw r1, 0(r1)						F	D	E	M	W			
and r1, r1, r2							()	F	D	E	M	W	
lw r1, 0(r1)								()	F	D	E	M	W

4.13 This exercise is intended to help you understand the relationship between forwarding, hazard detection, and ISA design. Problems in this exercise refer to the following sequence of instructions, and assume that it is executed on a 5-stage pipelined datapath:

4.13.1 [5] <4.7> If there is no forwarding or hazard detection, insert nops to ensure correct execution.

	1	2	3	4	5	6	7	8	9	10	11	12	13
add r5, r2, r1	F	D	E	M	W								
		nop											
			nop										
lw r3, 4(r5)				F	D	E	M	W					
lw r2, 0(r2)					F	D	E	M	W				
						nop							
or r3, r5, r3							F	D	E	M	W		
								nop					
sw r3, 0(r5)									F	D	E	M	W

The total number of cycles for only being able to use nop's is $\boxed{13 \text{ cycles}}$

Pipeline diagram for the same set of instructions while being able to use forwarding:

	1	2	3	4	5	6	7	8	9
add r5, r2, r1	F	D	E	M	W				
lw r3, 4(r5)		F	D	E	M	W			
lw r2, 0(r2)			F	D	E	M	W		
or r3, r5, r3				F	D	E	M	W	
sw r3, 0(r5)					F	D	E	M	W

The total number of cycles is $\boxed{9 \text{ cycles}}$