Bài Tập Chương 4 - Pipeline

Exercise 4.12

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
a.	300ps	400ps	350ps	500ps	100ps
b.	200ps	150ps	120ps	190ps	140ps

4.12.1 [5] <4.5> What is the clock cycle time in a pipelined and nonpipelined processor?

4.12.2 [10] <4.5> What is the total latency of a `lw` instruction in a pipelined and nonpipelined processor?

4.12.3 [10] <4.5> If we can split one stage of the pipelined datapath into two new stages, each with half the latency of the original stage, which stage would you split and what is the new clock cycle time of the processor?

The remaining problems in this exercise assume that instructions executed by the processor are broken down as follows:

	ALU	beq	lw	sw
a.	50%	25%	15%	10%
b.	30%	25%	30%	15%

4.12.4 [10] <4.5> Assuming there are no stalls or hazards, what is the utilization (% of cycles used) of the data memory?

4.12.5 [10] <4.5> Assuming there are no stalls or hazards, what is the utilization of the write-register port of the "Registers" unit?

4.12.6 [30] <4.5> Instead of a single-cycle organization, we can use a multi-cycle organization where each instruction takes multiple cycles but one instruction finishes before another is fetched. In this organization, an instruction only goes through stages it actually needs (e.g., ST only takes four cycles because it does not need the WB stage). Compare clock cycle times and execution times with single-cycle, multi-cycle, and pipelined organization.

Exercise 4.13

In this exercise, we examine how data dependences affect execution in the basic five-stage pipeline described in Section 4.5. Problems in this exercise refer to the following sequence of instructions:

	Instruction sequence

a.	`lw $1,40($6)`
	`add $6,$2,$2`
	`sw $6,50($1)`

b.	`lw $5,-16($5)`
	`sw $5,-16($5)`
	`add $5,$5,$5`

4.13.1 [10] <4.5> Indicate dependences and their type.

4.13.2 [10] <4.5> Assume there is no forwarding in this pipelined processor. Indicate hazards and add `nop` instructions to eliminate them.

4.13.3 [10] <4.5> Assume there is full forwarding. Indicate hazards and add `nop` instructions to eliminate them. The remaining problems in this exercise assume the following clock cycle times:

	Without forwarding	With full forwarding	With ALU-ALU forwarding only
a.	300ps	400ps	360ps
b.	200ps	250ps	220ps

4.13.4 [10] <4.5> What is the total execution time of this instruction sequence without forwarding and with full forwarding? What is the speed-up achieved by adding full forwarding to a pipeline that had no forwarding?

4.13.5 [10] <4.5> Add `nop` instructions to this code to eliminate hazards if there is ALU-ALU forwarding only (no forwarding from the MEM to the EX stage)?

4.13.6 [10] <4.5> What is the total execution time of this instruction sequence with only ALU-ALU forwarding? What is the speed-up over a no-forwarding pipeline?

Exercise 4.14

In this exercise, we examine how resource hazards, control hazards, and ISA design can affect pipelined execution. Problems in this exercise refer to the following fragment of MIPS code:

	Instruction sequence

a.	`lw $1,40($6)`
	`beq $2,$0,Label ; Assume $2 == $0`
	`sw $6,50($2)`
	`Label: add $2,$3,$4`
	`sw $3,50($4)`

b.	`lw $5,-16($5)`
	`sw $4,-16($4)`
	`lw $3,-20($4)`
	`beq $2,$0,Label ; Assume $2 != $0`
	`add $5,$1,$4`

4.14.1 [10] <4.5> For this problem, assume that all branches are perfectly predicted (this eliminates all control hazards) and that no delay slots are used. If

we only have one memory (for both instructions and data), there is a structural hazard every time we need to fetch an instruction in the same cycle in which another instruction accesses data. To guarantee forward progress, this hazard must always be resolved in favor of the instruction that accesses data. What is the total execution time of this instruction sequence in the five-stage pipeline that only has one memory? We have seen that data hazards can be eliminated by adding `nops` to the code. Can you do the same with this structural hazard? Why?

4.14.2 [20] <4.5> For this problem, assume that all branches are perfectly predicted (this eliminates all control hazards) and that no delay slots are used. If we change load/store instructions to use a register (without an offset) as the address, these instructions no longer need to use the ALU. As a result, MEM and EX stages can be overlapped and the pipeline has only four stages. Change this code to accommodate this changed ISA. Assuming this change does not affect clock cycle time, what speed-up is achieved this instruction sequence?

4.14.3 [10] <4.5> Assuming stall-on-branch and no delay slots, what speed-up is achieved on this code if branch outcomes are determined in the ID stage, relative to the execution where branch outcomes are determined in the EX stage?

The remaining problems in this exercise assume that individual pipeline stages have the following latencies:

	IF	ID	EX	MEM	WB
a.	100ps	120ps	90ps	130ps	60ps
b.	180ps	100ps	170ps	220ps	60ps

4.14.4 [10] <4.5> Given these pipeline stage latencies, repeat the speed-up calculation from 4.14.2, but take into account the (possible) change in clock cycle time. When EX and MEM are done in a single stage, most of their work can be done in parallel. As a result, the resulting EX/MEM stage has a latency that is the larger of the original two, plus 20ps needed for the work that could not be done in parallel.

4.14.5 [10] <4.5> Given these pipeline stage latencies, repeat the speed-up calculation from Exercise 4.14.3, taking into account the (possible) change in clock cycle time. Assume that the latency ID stage increases by 50% and the latency of the EX stage decreases by 10ps when branch outcome resolution is moved from EX to ID.

4.14.6 [10] <4.5> Assuming stall-on-branch and no delay slots, what is the new clock cycle time and execution time of this instruction sequence if `beq` address

computation is moved to the MEM stage? What is the speed-up from this change? Assume that the latency of the EX stage is reduced by 20ps and the latency of the MEM stage is unchanged when branch outcome resolution is moved from EX to MEM.