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?

Solution 4.13

4.13.1

	Instruction sequence	Dependences

a.	`I1: lw $1,40($6)`
	`I2: add $6,$2,$2`
	`I3: sw $6,50($1)`
		RAW on \$1 from I1 to I3
		RAW on \$6 from I2 to I3
		WAR on \$6 from I1 to I2 and I3

b.	`I1: lw $5,-16($5)`
	`I2: sw $5,-16($5)`
	`I3: add $5,$5,$5`
		RAW on \$5 from I1 to I2 and I3
		WAR on \$5 from I1 and I2 to I3
		WAW on \$5 from I1 to I3

4.13.2 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). The code that eliminates these hazards by inserting `nop` instructions is:

	Instruction sequence	

a.	`lw $1,40($6)`
	`add $6,$2,$2`
	`nop`
	`sw $6,50($1)`
		Delay I3 to avoid RAW hazard on \$1 from I1

b.	`lw $5,-16($5)`
	`nop`
	`nop`
	`sw $5,-16($5)`
	`add $5,$5,$5`
		Delay I2 to avoid RAW hazard on \$5 from I1
		
		Note: no RAW hazard from on \$5 from I1 now

4.13.3 With full forwarding, an ALU instruction can forward a value to EX stage of the next instruction without a hazard. However, a load cannot forward to the EX stage of the next instruction (by can to the instruction after that). The code that eliminates these hazards by inserting `nop` instructions is:

	Instruction sequence	

a.	`lw $1,40($6)`
	`add $6,$2,$2`
	`sw $6,50($1)`
		No RAW hazard on \$1 from I1 (forwarded)

b.	`lw $5,-16($5)`
	`nop`
	`sw $5,-16($5)`
	`add $5,$5,$5`
		Delay I2 to avoid RAW hazard on \$5 from I1
		Value for \$5 is forwarded from I2 now
		Note: no RAW hazard from on \$5 from I1 now

4.13.4 The total execution time is the clock cycle time times the number of cycles. Without any stalls, a three-instruction sequence executes in 7 cycles (5 to complete the first instruction, then one per instruction). The execution without forwarding must add a stall for every `nop` we had in 4.13.2, and execution forwarding must add a stall cycle for every `nop` we had in 4.13.3. Overall, we get:

	No forwarding	With forwarding	Speed-up due to forwarding
a.	(7 + 1) × 300ps = 2400ps	7 × 400ps = 2800ps	0.86 (This is really a slowdown)
b.	(7 + 2) × 200ps = 1800ps	(7 + 1) × 250ps = 2000ps	0.90 (This is really a slowdown)

4.13.5 With ALU-ALU-only forwarding, an ALU instruction can forward to the next instruction, but not to the second-next instruction (because that would be forwarding from MEM to EX). A load cannot forward at all, because it determines the data value in MEM stage, when it is too late for ALU-ALU forwarding. We have:

	Instruction sequence	

a.	`lw $1,40($6)`
	`add $6,$2,$2`
	`nop`
	`sw $6,50($1)`
		Can't use ALU-ALU forwarding, ($1 loaded in MEM)

b.	`lw $5,-16($5)`
	`nop`
	`nop`
	`sw $5,-16($5)`
	`add $5,$5,$5`
		Can't use ALU-ALU forwarding ($5 loaded in MEM)

4.13.6

	No forwarding	With ALU-ALU forwarding only	Speed-up with ALU-ALU forwarding
a.	(7 + 1) × 300ps = 2400ps	(7 + 1) × 360ps = 2880ps	0.83 (This is really a slowdown)
b.	(7 + 2) × 200ps = 1800ps	(7 + 2) × 220ps = 1980ps	0.91 (This is really a slowdown)

Exercise 4.16

The first three problems in this exercise refer to the following MIPS instruction:

	Instruction
a.	`lw  $1,40($6)`
b.	`add $5,$5,$5`

4.16.1 [5] <4.6> As this instruction executes, what is kept in each register located between two pipeline stages?

4.16.2 [5] <4.6> Which registers need to be read, and which registers are actually read?

4.16.3 [5] <4.6> What does this instruction do in EX and MEM stages?

Solution 4.16

4.16.1 For every instruction, the IF/ID register keeps the PC + 4 and the instruction word itself. The ID/EX register keeps all control signals for the EX, MEM, and WB stages, PC + 4, the two values read from Registers, the sign-extended lowermost 16 bits of the instruction word, and Rd and Rt fields of the instruction word (even for instructions whose format does not use these fields). The EX/MEM register keeps control signals for MEM and WB stages, the PC + 4 + Offset (where Offset is the sign-extended lowermost 16 bits of the instructions, even for instructions that have no offset field), the ALU result and the value of its Zero output, the value that was read from the second register in the ID stage (even for instructions that never need this value), and the number of the destination register (even for instructions that need no register writes; for these instructions the number of the destination register is simply a "random" choice between Rd or Rt). The MEM/WB register keeps the WB control signals, the value read from memory (or a "random" value if there was no memory read), the ALU result, and the number of the destination register.

4.16.2

	Need to be read	Actually read
a.	$6	$6, $1
b.	$5	$5 (twice)

4.16.3

	EX	MEM
a.	40 + $6	Load value from memory
b.	$5 + $5	Nothing

Exercise 4.20

Problems in this exercise refer to the following instruction sequences:

	Instruction sequence

a.	`lw $1,40($2)`
	`add $2,$3,$3`
	`add $1,$1,$2`
	`sw $1,20($2)`

b.	`add $1,$2,$3`
	`sw $2,0($1)`
	`lw $1,4($2)`
	`add $2,$2,$1`

4.20.1 [5] <4.7> Find all data dependences in this instruction sequence.

4.20.2 [10] <4.7> Find all hazards in this instruction sequence for a five-stage pipeline with and then without forwarding.

4.20.3 [10] <4.7> To reduce clock cycle time, we are considering a split of the MEM stage into two stages. Repeat Exercise 4.20.2 for this six-stage pipeline.

The remaining three problems in this exercise assume that, before any of the above is executed, all values in data memory are 0s and that registers $0 through $3 have the following initial values:

	$0	$1	$2	$3
a.	0	1	31	1000
b.	0	–2	63	2500

4.20.4 [5] <4.7> Which value is the first one to be forwarded and what is the value it overrides?

4.20.5 [10] <4.7> If we assume forwarding will be implemented when we design the hazard detection unit, but then we forget to actually implement forwarding, what are the final register values after this instruction sequence?

4.20.6 [10] <4.7> For the design described in Exercise 4.20.5, add `nops` to this instruction sequence to ensure correct execution in spite of missing support for forwarding.

Solution 4.20

4.20.1

	Instruction sequence	RAW	WAR	WAW

a.	`I1: lw $1,40($2)`
	`I2: add $2,$3,$3`
	`I3: add $1,$1,$2`
	`I4: sw $1,20($2)`
		(\$1) I1 to I3
		(\$2) I2 to I3, I4
		(\$1) I3 to I4
			(\$2) I1 to I2	(\$1) I1 to I3

b.	`I1: add $1,$2,$3`
	`I2: sw $2,0($1)`
	`I3: lw $1,4($2)`
	`I4: add $2,$2,$1`
		(\$1) I1 to I2
		(\$1) I3 to I4
			(\$2) I1, I2, I3 to I4
			(\$1) I1, I2 to I3
				(\$1) I1 to I3

4.20.2 Only RAW dependences can become data hazards. With forwarding, only RAW dependences from a load to the very next instruction become hazards.

Without forwarding, any RAW dependence from an instruction to one of the following three instructions becomes a hazard:

	Instruction sequence	With forwarding	Without forwarding

a.	`I1: lw $1,40($2)`
	`I2: add $2,$3,$3`
	`I3: add $1,$1,$2`
	`I4: sw $1,20($2)`
			(\$1) I1 to I3
			(\$2) I2 to I3, I4
			(\$1) I3 to I4

b.	`I1: add $1,$2,$3`
	`I2: sw $2,0($1)`
	`I3: lw $1,4($2)`
	`I4: add $2,$2,$1`
		(\$1) I3 to I4	(\$1) I1 to I2
			(\$1) I3 to I4

4.20.3 With forwarding, only RAW dependences from a load to the next two instructions become hazards because the load produces its data at the end of the second MEM stage. Without forwarding, any RAW dependence from an instruction to one of the following 4 instructions becomes a hazard:

	Instruction sequence	With forwarding	RAW

a.	`I1: lw $1,40($2)`
	`I2: add $2,$3,$3`
	`I3: add $1,$1,$2`
	`I4: sw $1,20($2)`
		(\$1) I1 to I3	(\$1) I1 to I3
			(\$2) I2 to I3, I4
			(\$1) I3 to I4

b.	`I1: add $1,$2,$3`
	`I2: sw $2,0($1)`
	`I3: lw $1,4($2)`
	`I4: add $2,$2,$1`
		(\$1) I3 to I4	(\$1) I1 to I2
			(\$1) I3 to I4

4.20.4

	Instruction sequence	RAW

a.	`I1: lw $1,40($2)`
	`I2: add $2,$3,$3`
	`I3: add $1,$1,$2`
	`I4: sw $1,20($2)`
		(\$1) I1 to I3 (0 overrides 1)
		(\$2) I2 to I3 (2000 overrides 31)

b.	`I1: add $1,$2,$3`
	`I2: sw $2,0($1)`
	`I3: lw $1,4($2)`
	`I4: add $2,$2,$1`
		(\$1) I1 to I2 (2563 overrides 63)

4.20.5 A register modification becomes "visible" to the EX stage of the following instructions only two cycles after the instruction that produces the register value leaves the EX stage. Our forwarding-assuming hazard detection unit only adds a one-cycle stall if the instruction that immediately follows a load is dependent on the load. We have:

	Instruction sequence with forwarding stalls	Execution without forwarding	Values after execution
a.	I1: lw $1,40($2) I2: add $2,$3,$3 I3: add $1,$1,$2 I4: sw $1,20($2)	$1 = 0 (I4 and after) $2 = 2000 (after I4) $1 = 32 (after I4)	$0 = 0 $1 = 32 $2 = 2000 $3 = 1000
b.	I1: add $1,$2,$3 I2: sw $2,0($1) I3: lw $1,4($2) Stall I4: add $2,$2,$1	$1 = 2563 (Stall and after) $1 = 0 (after I4) $2 = 2626 (after I4)	$0 = 0 $1 = 0 $2 = 2626 $3 = 2500

4.20.6

	Instruction sequence with forwarding stalls	Correct execution	Sequence with NOPs
a.	I1: lw $1,40($2) I2: add $2,$3,$3 I3: add $1,$1,$2 I4: sw $1,20($2)	I1: lw $1,40($2) I2: add $2,$3,$3 Stall Stall I3: add $1,$1,$2 Stall Stall I4: sw $1,20($2)	lw $1,40($2) add $2,$3,$3 nop nop add $1,$1,$2 nop nop sw $1,20($2)
b.	I1: add $1,$2,$3 I2: sw $2,0($1) I3: lw $1,4($2) Stall I4: add $2,$2,$1	I1: add $1,$2,$3 Stall Stall I2: sw $2,0($1) I3: lw $1,4($2) Stall Stall I4: add $2,$2,$1	add $1,$2,$3 nop nop sw $2,0($1) lw $1,4($2) nop nop add $2,$2,$1