Chapter 4 Exercises

2013-10-28/690 Points

Pls. mail your homework (converted into PDF File) to coa.2012.assignment@gmail.com.

File name: ID_YourName_HW_Sequence.pdf, ex: **5123581321_Obama_HW_01.pdf** （You could save the doc/docx as PDF in word 2007 or later, using ‘Save as…’）

Exercise 4.1

Exercise 4.1

Different instructions utilize different hardware blocks in the basic single-cycle implementation. The next three problems in this exercise refer to the following instruction:

	Instruction	Interpretation
a.	add Rd,Rs,Rt	Reg[Rd]=Reg[Rs]+Reg[Rt]
b.	lw Rt,Offs(Rs)	Reg[Rt]=Mem[Reg[Rs]+Offs]

4.1.1 [5] <4.1> What are the values of control signals generated by the control in Figure 4.2 for this instruction?

4.1.2 [5] <4.1> Which resources (blocks) perform a useful function for this instruction?

4.1.3 [10] <4.1> Which resources (blocks) produce outputs, but their outputs are not used for this instruction? Which resources produce no outputs for this instruction?

Different execution units and blocks of digital logic have different latencies (time needed to do their work). In Figure 4.2 there are seven kinds of major blocks. Latencies of blocks along the critical (longest-latency) path for an instruction determine the minimum latency of that instruction. For the remaining three problems in this exercise, assume the following resource latencies:

	I-Mem	Add	Mux	ALU	Regs	D-Mem	Control
a.	400ps	100ps	30ps	120ps	200ps	350ps	100ps
b.	500ps	150ps	100ps	180ps	220ps	1000ps	65ps

4.1.4 [5] <4.1> What is the critical path for a MIPS AND instruction?

4.1.5 [5] <4.1> What is the critical path for a MIPS load (LD) instruction?

4.1.6 [10] <4.1> What is the critical path for a MIPS BEQ instruction?

Exercise 4.2

The basic single-cycle MIPS implementation in Figure 4.2 can only implement some instructions. New instructions can be added to an existing ISA, but the decision whether or not to do that depends, among other things, on the cost and complexity such an addition introduces into the processor datapath and control. The first three problems in this exercise refer to this new instruction:

	Instruction	Interpretation
a.	`add3 Rd,Rs,Rt,Rx`	`Reg[Rd]=Reg[Rs]+Reg[Rt]+Reg[Rx]`
b.	`sll Rt,Rd,Shift`	`Reg[Rd]= Reg[Rt] << Shift` (shift left by Shift bits)

4.2.1 [10] <4.1> Which existing blocks (if any) can be used for this instruction?

4.2.2 [10] <4.1> Which new functional blocks (if any) do we need for this instruction?

4.2.3 [10] <4.1> What new signals do we need (if any) from the control unit to support this instruction?

When processor designers consider a possible improvement to the processor datapath, the decision usually depends on the cost/performance tradeoff. In the following three problems, assume that we are starting with a datapath from Figure 4.2, where I-Mem, Add, Mux, ALU, Regs, D-Mem, and Control blocks have latencies of 400ps, 100ps, 30ps, 120ps, 200ps, 350ps, and 100ps, respectively, and costs of 1000, 30, 10, 100, 200, 2000, and 500, respectively. The remaining three problems in this exercise refer to the following processor improvement:

	Improvement	Latency	Cost	Benefit
a.	Faster Add	–20ps for Add units	+20 per Add unit	Replaces existing Add units with faster ones.
b.	Larger Registers	+100ps for Regs	+200 for Regs	Fewer loads and stores needed to save and restore register values. This results in 5% fewer instructions.

4.2.4 [10] <4.1> What is the clock cycle time with and without this improvement?

4.2.5 [10] <4.1> What is the speed-up achieved by adding this improvement?

4.2.6 [10] <4.1> Compare the cost/performance ratio with and without this improvement.

Exercise 4.7

In this exercise we examine how latencies of individual components of the datapath affect the clock cycle time of the entire datapath, and how these components are utilized by instructions. For problems in this exercise, assume the following latencies for logic blocks in the datapath:

	I-Mem	Add	Mux	ALU	Regs	D-Mem	Sign-extend	Shift-left-2
a.	400ps	100ps	30ps	120ps	200ps	350ps	20ps	0ps
b.	500ps	150ps	100ps	180ps	220ps	1000ps	90ps	20ps

4.7.1 [10] <4.3> What is the clock cycle time if the only type of instructions we need to support are ALU instructions (`add`, `and`, etc.)?

4.7.2 [10] <4.3> What is the clock cycle time if we only had to support `lw` instructions?

4.7.3 [20] <4.3> What is the clock cycle time if we must support `add`, `beq`, `lw`, and `sw` instructions?

For the remaining 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
a.	30%	15%	5%	20%	20%	10%
b.	25%	5%	5%	15%	35%	15%

4.7.4 [10] <4.3> In what fraction of all cycles is the data memory used?

4.7.5 [10] <4.3> In what fraction of all cycles is the input of the sign-extend circuit needed? What is this circuit doing in cycles in which its input is not needed?

4.7.6 [10] <4.3> If we can improve the latency of one of the given datapath components by 10%, which component should it be? What is the speed-up from this improvement?

Exercise 4.12

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

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.

Exercise 4.19

This exercise is intended to help you understand the cost/complexity/performance tradeoffs of forwarding in a pipelined processor. Problems in this exercise refer to pipelined datapaths from Figure 4.45. These problems assume that, of all instructions executed in a processor, the following fraction of these instructions

has a particular type of RAW data dependence. The type of RAW data dependence is identified by the stage that produces the result (EX or MEM) and the instruction that consumes the result (1st instruction that follows the one that produces the result, 2nd instruction that follows, or both). We assume that the register write is done in the first half of the clock cycle and that register reads are done in the second half of the cycle, so "EX to 3rd" and "MEM to 2nd" dependences are not counted because they can not result in data hazards. Also, assume that the CPI of the processor is 1 if there are no data hazards.

	EX to 1st only	EX to 1st and 2nd	EX to 2nd only	MEM to 1st
a.	10%	10%	5%	25%
b.	15%	5%	10%	20%

4.19.1 [10] <4.7> If we use no forwarding, what fraction of cycles are we stalling due to data hazards?

4.19.2 [5] <4.7> If we use full forwarding (forward all results that can be forwarded), what fraction of cycles are we stalling due to data hazards?

4.19.3 [10] <4.7> Let us assume that we can not afford to have three-input Muxes that are needed for full forwarding. We have to decide if it is better to forward only from the EX/MEM pipeline register (next-cycle forwarding) or only from the MEM/WB pipeline register (two-cycle forwarding). Which of the two options results in fewer data stall cycles?

The remaining three problems in this exercise refer to the following latencies for individual pipeline stages. For the EX stage, latencies are given separately for a processor without forwarding and for a processor with different kinds of forwarding.

	IF	ID	EX (no FW)	EX (full FW)	EX (FW from EX/MEM only)	EX (FW from MEM/WB only)	MEM	WB
a.	100ps	50ps	75ps	110ps	100ps	100ps	100ps	60ps
b.	250ps	300ps	200ps	350ps	320ps	310ps	300ps	200ps

4.19.4 [10] <4.7> For the given hazard probabilities and pipeline stage latencies, what is the speed-up achieved by adding full forwarding to a pipeline that had no forwarding?

4.19.5 [10] <4.7> What would be the additional speed-up (relative to a processor with forwarding) if we added time-travel forwarding that eliminates all data hazards? Assume that the yet-to-be-invented time-travel circuitry adds 100ps to the latency of the full-forwarding EX stage.

4.19.6 [20] <4.7> Repeat Exercise 4.19.3 but this time determine which of the two options results in shorter time per instruction.

Exercise 4.21

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 sequences of instructions, and assume that it is executed on a five-stage pipelined datapath:

	Instruction sequence

a.	`lw  $1,40($6)`
	`add $2,$3,$1`
	`add $1,$6,$4`
	`sw  $2,20($4)`
	`and $1,$1,$4`

b.	`add $1,$5,$3`
	`sw  $1,0($2)`
	`lw  $1,4($2)`
	`add $5,$5,$1`
	`sw  $1,0($2)`

4.21.1 [5] <4.7> If there is no forwarding or hazard detection, insert `nops` to ensure correct execution.

4.21.2 [10] <4.7> Repeat Exercise 4.21.1 but now use `nops` only when a hazard cannot be avoided by changing or rearranging these instructions. You can assume register R7 can be used to hold temporary values in your modified code.

4.21.3 [10] <4.7> If the processor has forwarding, but we forgot to implement the hazard detection unit, what happens when this code executes?

4.21.4 [20] <4.7> If there is forwarding, for the first five cycles during the execution of this code, specify which signals are asserted in each cycle by hazard detection and forwarding units in Figure 4.60.

4.21.5 [10] <4.7> If there is no forwarding, what new inputs and output signals do we need for the hazard detection unit in Figure 4.60? Using this instruction sequence as an example, explain why each signal is needed.

4.21.6 [20] <4.7> For the new hazard detection unit from Exercise 4.21.5, specify which output signals it asserts in each of the first five cycles during the execution of this code.

Exercise 4.23

Exercise 4.23

The importance of having a good branch predictor depends on how often conditional branches are executed. Together with branch predictor accuracy, this will determine how much time is spent stalling due to mispredicted branches. In this exercise, assume that the breakdown of dynamic instructions into various instruction categories is as follows:

	R-Type	beq	jmp	lw	sw
a.	50%	15%	10%	15%	10%
b.	30%	10%	5%	35%	20%

Also, assume the following branch predictor accuracies:

	Always-taken	Always not-taken	2-bit
a.	40%	60%	80%
b.	60%	40%	95%

4.23.1 [10] <4.8> Stall cycles due to mispredicted branches increase the CPI. What is the extra CPI due to mispredicted branches with the always-taken predictor? Assume that branch outcomes are determined in the EX stage, that there are no data hazards, and that no delay slots are used.

4.23.2 [10] <4.8> Repeat Exercise 4.23.1 for the "always not-taken" predictor.

4.23.3 [10] <4.8> Repeat Exercise 4.23.1 for the 2-bit predictor.

4.23.4 [10] <4.8> With the 2-bit predictor, what speed-up would be achieved if we could convert half of the branch instructions in a way that replaces a branch instruction with an ALU instruction? Assume that correctly and incorrectly predicted instructions have the same chance of being replaced.

4.23.5 [10] <4.8> With the 2-bit predictor, what speed-up would be achieved if we could convert half of the branch instructions in a way that replaced each branch instruction with two ALU instructions? Assume that correctly and incorrectly predicted instructions have the same chance of being replaced.

4.23.6 [10] <4.8> Some branch instructions are much more predictable than others. If we know that 80% of all executed branch instructions are easy-to-predict loop-back branches that are always predicted correctly, what is the accuracy of the 2-bit predictor on the remaining 20% of the branch instructions?

Exercise 4.25

This exercise explores how exception handling affects pipeline design. The first three problems in this exercise refer to the following two instructions:

	Instruction 1	Instruction 2
a.	`add $0,$1,$2`	`bne $1,$2,Label`
b.	`lw $2,40($3)`	`nand $1,$2,$3`

4.25.1 [5] <4.9> Which exceptions can each of these instructions trigger? For each of these exceptions, specify the pipeline stage in which it is detected.

4.25.2 [10] <4.9> If there is a separate handler address for each exception, show how the pipeline organization must be changed to be able to handle this exception. You can assume that the addresses of these handlers are known when the processor is designed.

4.25.3 [10] <4.9> If the second instruction from this table is fetched right after the instruction from the first table, describe what happens in the pipeline when the first instruction causes the first exception you listed in Exercise 4.25.1.

Show the pipeline execution diagram from the time the first instruction is fetched until the time the first instruction of the exception handler is completed.

The remaining three problems in this exercise assume that exception handlers are located at the following addresses:

	Overflow	Invalid data address	Undefined instruction	Invalid instruction address	Hardware malfunction
a.	0xFFFFF000	0xFFFFF100	0xFFFFF200	0xFFFFF300	0xFFFFF400
b.	0x00000008	0x00000010	0x00000018	0x00000020	0x00000028

4.25.4 [5] <4.9> What is the address of the exception handler in Exercise 4.25.3? What happens if there is an invalid instruction at that address in instruction memory?

4.25.5 [20] <4.9> In vectored exception handling, the table of exception handler addresses is in data memory at a known (fixed) address. Change the pipeline to implement this exception handling mechanism. Repeat Exercise 4.25.3 using this modified pipeline and vectored exception handling.

4.25.6 [15] <4.9> We want to emulate vectored exception handling (described in Exercise 4.25.5) on a machine that has only one fixed handler address. Write the code that should be at that fixed address. Hint: this code should identify the exception, get the right address from the exception vector table, and transfer execution to that handler.

Exercise 4.28

In this exercise we compare the performance of 1-issue and 2-issue processors, taking into account program transformations that can be made to optimize for 2-issue execution. Problems in this exercise refer to the following loop (written in C):

	C code
a.	`for(i=0;!=j;i++)` `  b[i]=a[i];`
b.	`for(i=0;a[i]!=a[i+1];i++)` `  a[i]=0;`

When writing MIPS code, assume that variables are kept in registers as follows, and that all registers except those indicated as Free are used to keep various variables, so they cannot be used for anything else.

	i	j	a	b	c	Free
a.	$1	$2	$3	$4	$5	$6,$7,$8
b.	$4	$5	$6	$7	$8	$1,$2,$3

4.28.1 [10] <4.10> Translate this C code into MIPS instructions. Your translation should be direct, without rearranging instructions to achieve better performance.

4.28.2 [10] <4.10> If the loop exits after executing only two iterations, draw a pipeline diagram for your MIPS code from Exercise 4.28.1 executed on a 2-issue processor shown in Figure 4.69. Assume the processor has perfect branch prediction and can fetch any 2 instructions (not just consecutive instructions) in the same cycle.

4.28.3 [10] <4.10> Rearrange your code from Exercise 4.28.1 to achieve better performance on a 2-issue statically scheduled processor from Figure 4.69.

4.28.4 [10] <4.10> Repeat Exercise 4.28.2, but this time use your MIPS code from Exercise 4.28.3.

4.28.5 [10] <4.10> What is the speed-up of going from a 1-issue processor to a 2-issue processor from Figure 4.69. Use your code from Exercise 4.28.1 for both

1-issue and 2-issue, and assume that 1,000,000 iterations of the loop are executed. As in Exercise 4.28.2, assume that the processor has perfect branch predictions, and that a 2-issue processor can fetch any 2 instructions in the same cycle.

4.28.6 [10] <4.10> Repeat Exercise 4.28.5, but this time assume that in the 2-issue processor one of the instructions to be executed in a cycle can be of any kind, and the other must be a non-memory instruction.