computation is moved to the MEM stage? What is the speedup 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 4.15

In this exercise, we examine how the ISA affects pipeline design. Problems in this exercise refer to the following new instruction:

a.	`ADDM Rd,Rt+Offs(Rs)`	Rd=Rt+Mem[Offs+Rs]
b.	`BEQM Rd,Rt,Offs(Rs)`	if Rt=Mem[Offs+Rs] then PC = Rd

4.15.1 [20] <4.5> What must be changed in the pipelined datapath to add this instruction to the MIPS ISA?

4.15.2 [10] <4.5> Which new control signals must be added to your pipeline from 4.15.1?

4.15.3 [20] <4.5, 4.13> Does support for this instruction introduce any new hazards? Are stalls due to existing hazards made worse?

4.15.4 [10] <4.5, 4.13> Give an example of where this instruction might be useful and a sequence of existing MIPS instructions that are replaced by this instruction.

4.15.5 [10] <4.5, 4.11, 4.13> If this instruction already exists in a legacy ISA, explain how it would be executed in a modern processor like AMD Barcelona.

The last problem in this exercise assumes that each use of the new instruction replaces the given number of original instructions, that the replacement can be made once in the given number of original instructions, and that each time the new instruction is executed the given number of extra stall cycles is added to the program's execution time:

	Replaces	Once in every	Extra Stall Cycles
a.	2	30	2
b.	3	40	1

4.15.6 [10] <4.5> What is the speedup achieved by adding this new instruction? In your calculation, assume that the CPI of the original program (without the new instruction) is 1.

Exercise 4.16

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

	Instruction
a.	SW R16,-100(R6)
b.	OR R2,R1,R0

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 the EX and MEM stages?

The remaining three problems in this exercise refer to the following loop. 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.

	Loop

a.	Loop: ADD R1,R2,R1
	LW R2,0(R1)
	LW R2,16(R2)
	SLT R1,R2,R4
	BEQ R1,R9,Loop

b.	Loop: LW R1,0(R1) AND R1,R1,R2 LW R1,0(R1) LW R1,0(R1) BEQ R1,R0,Loop

4.16.4 [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 of that iteration 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).

4.16.5 [10] <4.6> How often (as a percentage of all cycles) do we have a cycle in which all five pipeline stages are doing useful work?

4.16.6 [10] <4.6> At the start of the cycle in which we fetch the first instruction of the third iteration of this loop, what is stored in the IF/ID register?

Exercise 4.17

Problems in this exercise a
sor are broken down as fol

	ADD	
a.	40%	
b.	60%	

4.17.1 [5] <4.6> Assum
branches are taken, in wha
EX stage generate a value t

4.17.2 [5] <4.6> Assumi
do we actually need to us
same cycle?

4.17.3 [5] <4.6> Assumi
do we use the data memor

Each pipeline stage in F
introduces registers betwe
tional latency. The remai
latencies for logic within
stages:

	IF	ID
a.	200ps	120ps
b.	150ps	200ps

4.17.4 [5] <4.6> Assum
pipelining a single-cycle d

4.17.5 [10] <4.6> We ca
(no offset) and put the m
cycle time if this is done i
that the latency of the new

4.17.6 [10] <4.6> The c
tions to be converted int
of these instructions, wha
5-stage pipeline to the 4-s

Exercise 4.17

Problems in this exercise assume that instructions executed by a pipelined processor are broken down as follows:

	ADD	BEQ	LW	SW
a.	40%	30%	25%	5%
b.	60%	10%	20%	10%

4.17.1 [5] <4.6> Assuming there are no stalls and that 60% of all conditional branches are taken, in what percentage of clock cycles does the branch adder in the EX stage generate a value that is actually used?

4.17.2 [5] <4.6> Assuming there are no stalls, how often (percentage of all cycles) do we actually need to use all three register ports (two reads and a write) in the same cycle?

4.17.3 [5] <4.6> Assuming there are no stalls, how often (percentage of all cycles) do we use the data memory?

Each pipeline stage in Figure 4.33 has some latency. Additionally, pipelining introduces registers between stages (Figure 4.35), and each of these adds an additional latency. The remaining problems in this exercise assume the following latencies for logic within each pipeline stage and for each register between two stages:

	IF	ID	EX	MEM	WB	Pipeline Register
a.	200ps	120ps	150ps	190ps	100ps	15ps
b.	150ps	200ps	200ps	200ps	100ps	15ps

4.17.4 [5] <4.6> Assuming there are no stalls, what is the speedup achieved by pipelining a single-cycle datapath?

4.17.5 [10] <4.6> We can convert all load/store instructions into register-based (no offset) and put the memory access in parallel with the ALU. What is the clock cycle time if this is done in the single-cycle and in the pipelined datapath? Assume that the latency of the new EX/MEM stage is equal to the longer of their latencies.

4.17.6 [10] <4.6> The change in 4.17.5 requires many existing `LW/SW` instructions to be converted into two-instruction sequences. If this is needed for 50% of these instructions, what is the overall speedup achieved by changing from the 5-stage pipeline to the 4-stage pipeline where EX and MEM are done in parallel?

Exercise 4.18

The first three problems in this exercise refer to the execution of the following instruction in the pipelined datapath from Figure 4.51, and assume the following clock cycle time, ALU latency, and Mux latency:

	Instruction	Clock Cycle Time	ALU Latency	Mux Latency
a.	LW R1,32(R2)	50ps	30ps	15ps
b.	OR R1,R5,R6	200ps	170ps	25ps

4.18.1 [10] <4.6> For each stage of the pipeline, what are the values of the control signals asserted by this instruction in that pipeline stage?

4.18.2 [10] <4.6, 4.7> How much time does the control unit have to generate the ALUSrc control signal? Compare this to a single-cycle organization.

4.18.3 What is the value of the PCSrc signal for this instruction? This signal is generated early in the MEM stage (only a single AND gate). What would be a reason in favor of doing this in the EX stage? What is the reason against doing it in the EX stage?

The remaining problems in this exercise refer to the following signals from Figure 4.48:

	Signal 1	Signal 2
a.	ALUSrc	PCSrc
b.	Branch	RegWrite

4.18.4 [5] <4.6> For each of these signals, identify the pipeline stage in which it is generated and the stage in which it is used.

4.18.5 [5] <4.6> For which MIPS instruction(s) are both of these signals set to 1?

4.18.6 [10] <4.6> One of these signals goes back through the pipeline. Which signal is it? Is this a time-travel paradox? Explain.

Exercise 4.19

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

have a particular type of RAW
is identified by the stage that p
that consumes the result (1st
result, 2nd instruction that fo
done in the first half of the clo
half of the cycle, so "EX to 3
because they cannot result in
sor is 1 if there are no data ha

	EX to 1st Only	MEM to 1st Only
a.	5%	20%
b.	20%	10%

4.19.1 [10] <4.7> If we use
due to data hazards?

4.19.2 [5] <4.7> If we use
warded), what fraction of cyc

4.19.3 [10] <4.7> Let us ass
that are needed for full forw
only from the EX/MEM pip
the MEM/WB pipeline regist
results in fewer data stall cycl

The remaining three problem
individual pipeline stages. Fo
cessor without forwarding ar

	IF	ID	EX (no FW)	EX
a.	150ps	100ps	120ps	
b.	300ps	200ps	300ps	

4.19.4 [10] <4.7> For the g
what is the speedup achieve
forwarding?

4.19.5 [10] <4.7> What w
sor with forwarding) if we a

have 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 3rd" dependences are not counted because they cannot result in data hazards. Also, assume that the CPI of the processor is 1 if there are no data hazards.

	EX to 1st Only	MEM to 1st Only	EX to 2nd only	MEM to 2nd Only	EX to 1st and MEM to 2nd	Other RAW Dependences
a.	5%	20%	5%	10%	10%	10%
b.	20%	10%	15%	10%	5%	0%

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 staling due to data hazards?

4.19.3 [10] <4.7> Let us assume that we cannot 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.	150ps	100ps	120ps	150ps	140ps	130ps	120ps	100ps
b.	300ps	200ps	300ps	350ps	330ps	320ps	290ps	100ps

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

4.19.5 [10] <4.7> What would be the additional speedup (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 4.19.3 but this time determine which of the two options results in shorter time per instruction.

Exercise 4.20

Problems in this exercise refer to the following instruction sequences:

	Instruction Sequence

a.	ADD R1,R2,R1
	LW R2,0(R1)
	LW R1,4(R1)
	OR R3,R1,R2

b.	LW R1,0(R1)
	AND R1,R1,R2
	LW R2,0(R1)
	LW R1,0(R3)

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 5-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 4.20.2 for this 6-stage pipeline.

The remaining three problems in this exercise assume that, before any of the above is executed, all values in data memory are zeroes and that registers R0 through R3 have the following initial values:

	R0	R1	R2	R3
a.	0	–1	31	1500
b.	0	4	63	3000

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> F
sequence to ensure co

Exercise 4.21

This exercise is intend
ing, hazard detection,
ing sequences of instr
datapath:

a.	ADD R5,R2,R1
	LW R3,4(R5)
	LW R2,0(R2)
	OR R3,R5,R3
	SW R3,0(R5)

b.	LW R2,0(R1)
	AND R1,R2,R1
	LW R3,0(R2)
	LW R1,0(R1)
	SW R1,0(R2)

4.21.1 [5] <4.7> If
ensure correct execut

4.21.2 [10] <4.7>
avoided by changing
can be used to hold t

4.21.3 [10] <4.7>
the hazard detection

4.21.4 [20] <4.7>
tion of this code, spe
tion and forwarding

4.21.5 [10] <4.7>
do we need for the
sequence as an exam

4.21.6 [20] <4.7>
output signals it asse
code.

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

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 5-stage pipelined datapath:

	Instruction sequence

a.	ADD R5,R2,R1
	LW R3,4(R5)
	LW R2,0(R2)
	OR R3,R5,R3
	SW R3,0(R5)

b.	LW R2,0(R1)
	AND R1,R2,R1
	LW R3,0(R2)
	LW R1,0(R1)
	SW R1,0(R2)

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 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 4.21.5, specify which output signals it asserts in each of the first five cycles during the execution of this code.

Exercise 4.22

This exercise is intended to help you understand the relationship between delay slots, control hazards, and branch execution in a pipelined processor. In this exercise, we assume that the following MIPS code is executed on a pipelined processor with a 5-stage pipeline, full forwarding, and a predict-taken branch predictor:

a.
```
Label1: LW   R2,0(R2)
        BEQ  R2,R0,Label  ; Taken once, then not taken
        OR   R2,R2,R3
        SW   R2,0(R5)
```

b.
```
        LW   R2,0(R1)
Label1: BEQ  R2,R0,Label2 ; Not taken once, then taken
        LW   R3,0(R2)
        BEQ  R3,R0,Label1 ; Taken
        ADD  R1,R3,R1
Label2: SW   R1,0(R2)
```

4.22.1 [10] <4.8> Draw the pipeline execution diagram for this code, assuming there are no delay slots and that branches execute in the EX stage.

4.22.2 [10] <4.8> Repeat 4.22.1, but assume that delay slots are used. In the given code, the instruction that follows the branch is now the delay slot instruction for that branch.

4.22.3 [20] <4.8> One way to move the branch resolution one stage earlier is to not need an ALU operation in conditional branches. The branch instructions would be "`BEZ Rd,Label`" and "`BNEZ Rd,Label`", and it would branch if the register has and does not have a zero value, respectively. Change this code to use these branch instructions instead of `BEQ`. You can assume that register `R8` is available for you to use as a temporary register, and that an `SEQ` (set if equal) R-type instruction can be used.

Section 4.8 describes how the severity of control hazards can be reduced by moving branch execution into the ID stage. This approach involves a dedicated comparator in the ID stage, as shown in Figure 4.62. However, this approach potentially adds to the latency of the ID stage, and requires additional forwarding logic and hazard detection.

4.22.4 [10] <4.8> Using the first branch instruction in the given code as an example, describe the hazard detection logic needed to support branch execution in the ID stage as in Figure 4.62. Which type of hazard is this new logic supposed to detect?

4.22.5 [10] <4.8> Fo
branch execution into t
tion, assume that the a
cycle time.

4.22.6 [10] <4.8> Us
example, describe the f
execution in the ID stag
the complexity of the e

Exercise 4.23

The importance of hav
tional branches are exe
determine how much t
exercise, assume that th
tion categories is as foll

	R-Type	
a.	40%	
b.	60%	

Also, assume the follow

	Always-Taken
a.	45%
b.	65%

4.23.1 [10] <4.8> St
What is the extra CPI d
tor? Assume that branc
no data hazards, and th

4.23.2 [10] <4.8> Re

4.23.3 [10] <4.8> Re

4.23.4 [10] <4.8> W
we could convert half
instruction with an AL
dicted instructions hav

4.22.5 [10] <4.8> For the given code, what is the speedup achieved by moving branch execution into the ID stage? Explain your answer. In your speedup calculation, assume that the additional comparison in the ID stage does not affect clock cycle time.

4.22.6 [10] <4.8> Using the first branch instruction in the given code as an example, describe the forwarding support that must be added to support branch execution in the ID stage. Compare the complexity of this new forwarding unit to the complexity of the existing forwarding unit in Figure 4.62.

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.	40%	25%	5%	25%	5%
b.	60%	8%	2%	20%	10%

Also, assume the following branch predictor accuracies:

	Always-Taken	Always-Not-Taken	2-Bit
a.	45%	55%	85%
b.	65%	35%	98%

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 4.23.1 for the "always-not-taken" predictor.

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

4.23.4 [10] <4.8> With the 2-bit predictor, what speedup 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 speedup 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 4.24

This exercise examines the accuracy of various branch predictors for the following repeating pattern (e.g., in a loop) of branch outcomes:

	Branch Outcomes
a.	T, T, NT, NT
b.	T, NT, T, T, NT

4.24.1 [5] <4.8> What is the accuracy of always-taken and always-not-taken predictors for this sequence of branch outcomes?

4.24.2 [5] <4.8> What is the accuracy of the two-bit predictor for the first 4 branches in this pattern, assuming that the predictor starts off in the bottom left state from Figure 4.63 (predict not taken)?

4.24.3 [10] <4.8> What is the accuracy of the two-bit predictor if this pattern is repeated forever?

4.24.4 [30] <4.8> Design a predictor that would achieve a perfect accuracy if this pattern is repeated forever. You predictor should be a sequential circuit with one output that provides a prediction (1 for taken, 0 for not taken) and no inputs other than the clock and the control signal that indicates that the instruction is a conditional branch.

4.24.5 [10] <4.8> What is the accuracy of your predictor from 4.24.4 if it is given a repeating pattern that is the exact opposite of this one?

4.24.6 [20] <4.8> Repeat 4.24.4, but now your predictor should be able to eventually (after a warm-up period during which it can make wrong predictions) start perfectly predicting both this pattern and its opposite. Your predictor should have an input that tells it what the real outcome was. Hint: this input lets your predictor determine which of the two repeating patterns it is given.

Exercise 4.25

This exercise explores how exce
three problems in this exercise r

	Instruction 1
a.	BNE R1,R2,Label
b.	JUMP Label

4.25.1 [5] <4.9> Which excep
each of these exceptions, specify

4.25.2 [10] <4.9> If there is a
how the pipeline organization m
You can assume that the address
is designed.

4.25.3 [10] <4.9> If the secon
the instruction from the first tab
first instruction causes the first
execution diagram from the tim
first instruction of the exceptio

The remaining three problems i
located at the following addresse

	Overflow	Invalid Data Address
a.	0x1000CB05	0x1000D230
b.	0x450064E8	0xC8203E20

4.25.4 [5] <4.9> What is th
What happens if there is an i
memory?

4.25.5 [20] <4.9> In vectored
addresses is in data memory at
implement this exception handl
Pipeline and vectored exception

4.25.6 [15] <4.9> We want to
4.25.5) on a machine that has o
should be at that fixed address.
the right address from the exce
handler.

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.	BNE R1,R2,Label	LW R1,0(R1)
b.	JUMP Label	SW R5,0(R1)

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 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.	0x1000CB05	0x1000D230	0x1000D780	0x1000E230	00x1000F254
b.	0x450064E8	0xC8203E20	0xC8203E20	0x678A0000	0x00000010

4.25.4 [5] <4.9> What is the address of the exception handler in 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 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 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 4.26

This exercise explores how exception handling affects control unit design and processor clock cycle time. The first three problems in this exercise refer to the following MIPS instruction that triggers an exception:

	Instruction	Exception
a.	BNE R1,R2,Label	Invalid target address
b.	SUB R2,R4,R5	Arithmetic overflow

4.26.1 [10] <4.9> For each stage of the pipeline, determine the values of exception-related control signals from Figure 4.66 as this instruction passes through that pipeline stage.

4.26.2 [5] <4.9> Some of the control signals generated in the ID stage are stored into the ID/EX pipeline register, and some go directly into the EX stage. Explain why, using this instruction as an example.

4.26.3 [10] <4.9> We can make the EX stage faster if we check for exceptions in the stage after the one in which the exceptional condition occurs. Using this instruction as an example, describe the main disadvantage of this approach.

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

	IF	ID	EX	MEM	WB
a.	220ps	150ps	250ps	200ps	200ps
b.	175ps	150ps	200ps	175ps	140ps

4.26.4 [10] <4.9> If an overflow exception occurs once for every 100,000 instructions executed, what is the overall speedup if we move overflow checking into the MEM stage? Assume that this change reduces EX latency by 30ns and that the IPC achieved by the pipelined processor is 1 when there are no exceptions.

4.26.5 [20] <4.9> Can we generate exception control signals in EX instead of in ID? Explain how this will work or why it will not work, using the "BNE R4,R5,Label" instruction and these pipeline stage latencies as an example.

4.26.6 [10] <4.9> Assuming that each Mux has a latency of 40ps, determine how much time does the control unit have to generate the flush signals? Which signal is the most critical?

Exercise 4.27

This exercise examines h
store instructions. Proble
tion and the correspondi

a.	BEQ R5,R4,Label SLT R5,R15,R4
b.	BEQ R1,R0,Label LW R1,0(R1)

4.27.1 [20] <4.9> Ass
then the instruction at "
in each pipeline stage fo
decoded up to the cycle
fetched.

4.27.2 [10] <4.9> Rep
delay slot also causes a h

4.27.3 [10] <4.9> Wh
delay slot causes an exce
handler is completed?

The remaining three pr
instruction:

a.	SW R5,-40(R15)
b.	SW R1,0(R1)

4.27.4 [10] <4.9> Wh
is an invalid instructior
instruction given above

4.27.5 [10] <4.9> If
overflow exception dete
to explain what happen

4.27.6 [10] <4.9> Fo
ticular value is written
registers, WADDR and

Exercise 4.27

This exercise examines how exception handling interacts with branch and load/store instructions. Problems in this exercise refer to the following branch instruction and the corresponding delay slot instruction:

	Branch and Delay Slot
a.	`BEQ R5,R4,Label` `SLT R5,R15,R4`
b.	`BEQ R1,R0,Label` `LW R1,0(R1)`

4.27.1 [20] <4.9> Assume that this branch is correctly predicted as taken, but then the instruction at "Label" is an undefined instruction. Describe what is done in each pipeline stage for each cycle, starting with the cycle in which the branch is decoded up to the cycle in which the first instruction of the exception handler is fetched.

4.27.2 [10] <4.9> Repeat 4.27.1, but this time assume that the instruction in the delay slot also causes a hardware error exception when it is in MEM stage.

4.27.3 [10] <4.9> What is the value in the EPC if the branch is taken but the delay slot causes an exception? What happens after the execution of the exception handler is completed?

The remaining three problems in this exercise also refer to the following store instruction:

	Store Instruction
a.	`SW R5,-40(R15)`
b.	`SW R1,0(R1)`

4.27.4 [10] <4.9> What happens if the branch is taken, the instruction at "Label" is an invalid instruction, the first instruction of the exception handler is the SW instruction given above, and this store accesses an invalid data address?

4.27.5 [10] <4.9> If LD/ST address computation can overflow, can we delay overflow exception detection into the MEM stage? Use the given store instruction to explain what happens.

4.27.6 [10] <4.9> For debugging, it is useful to be able to detect when a particular value is written to a particular memory address. We want to add two new registers, WADDR and WVAL. The processor should trigger an exception when the

value equal to WVAL is about to be written to address WADDR. How would you change the pipeline to implement this? How would this SW instruction be handled by your modified datapath?

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;i!=j;i+=2)` `  a[i+1]=a[i];`
b.	`for(i=0;i!=j;i+=2)` `  b[i]=a[i]-a[i+1];`

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.	R2	R8	R9	R10	R11	R3,R4,R5
b.	R5	R6	R1	R2	R3	R10,R11,R12

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 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 two instructions (not just consecutive instructions) in the same cycle.

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

4.28.5 [10] <4.10> What is the speedup of going from a 1-issue processor to a 2-issue processor from Figure 4.69? Use your code from 4.28.1 for both 1-issue and 2-issue, and assume that 1,000,000 iterations of the loop are executed. As in

4.28.2, assume that the pr
processor can fetch any t

4.28.6 [10] <4.10> Rep
cessor one of the instruct
other must be a non-mer

Exercise 4.29

In this exercise, we consi
scalar processor. To simp
tion types can execute i
instructions can be 3 ALU
combination of these in
straint, but data and cont
in this exercise refer to th

a.	`Loop: ADDI R1,R1`
	`LW R2,0(R1`
	`LW R3,16(R`
	`ADD R2,R2,`
	`ADD R2,R2,`
	`BEQ R2,zer`

b.	`Loop: LW R1,0(R`
	`AND R1,R1,`
	`LW R2,0(R`
	`BEQ R1,zer`

4.29.1 [10] <4.10> If r
determine the fraction o
scalar processor.

4.29.2 [10] <4.10> If
cuted, determine the fr
static superscalar proce
from 4.29.1.

4.29.3 [10] <4.10> If r
determine the fraction o
in a 3-issue static superso

4.29.4 [20] <4.10> U
superscalar processor. A

4.28.2, assume that the processor has perfect branch predictions, and that a 2-issue processor can fetch any two instructions in the same cycle.

4.28.6 [10] <4.10> Repeat 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.

Exercise 4.29

In this exercise, we consider the execution of a loop in a statically scheduled superscalar processor. To simplify the exercise, assume that any combination of instruction types can execute in the same cycle, e.g., in a 3-issue superscalar, the three instructions can be 3 ALU operations, 3 branches, 3 load/store instructions, or any combination of these instructions. Note that this only removes a resource constraint, but data and control dependences must still be handled correctly. Problems in this exercise refer to the following loop:

	Loop

a.	`Loop: ADDI R1,R1,4`
	`LW R2,0(R1)`
	`LW R3,16(R1)`
	`ADD R2,R2,R1`
	`ADD R2,R2,R3`
	`BEQ R2,zero,Loop`

b.	`Loop: LW R1,0(R1)`
	`AND R1,R1,R2`
	`LW R2,0(R2)`
	`BEQ R1,zero,Loop`

4.29.1 [10] <4.10> If many (e.g., 1,000,000) iterations of this loop are executed, determine the fraction of all register reads that are useful in a 2-issue static superscalar processor.

4.29.2 [10] <4.10> If many (e.g., 1,000,000) iterations of this loop are executed, determine the fraction of all register reads that are useful in a 3-issue static superscalar processor. Compare this to your result for a 2-issue processor from 4.29.1.

4.29.3 [10] <4.10> If many (e.g., 1,000,000) iterations of this loop are executed, determine the fraction of cycles in which two or three register write ports are used in a 3-issue static superscalar processor.

4.29.4 [20] <4.10> Unroll this loop once and schedule it for a 2-issue static superscalar processor. Assume that the loop always executes an even number of

iterations. You can use registers R10 through R20 when changing the code to eliminate dependences.

4.29.5 [20] <4.10> What is the speedup of using your code from 4.29.4 instead of the original code with a 2-issue static superscalar processor? Assume that the loop has many (e.g., 1,000,000) iterations.

4.29.6 [10] <4.10> What is the speedup of using your code from 4.29.4 instead of the original code with a pipelined (1-issue) processor? Assume that the loop has many (e.g., 1,000,000) iterations.

Exercise 4.30

In this exercise, we make several assumptions. First, we assume that an N-issue superscalar processor can execute any N instructions in the same cycle, regardless of their types. Second, we assume that every instruction is independently chosen, without regard for the instruction that precedes or follows it. Third, we assume that there are no stalls due to data dependences, that no delay slots are used, and that branches execute in the EX stage of the pipeline. Finally, we assume that instructions executed in the program are distributed as follows:

	ALU	Correctly Predicted BEQ	Incorrectly Predicted BEQ	LW	SW
a.	40%	20%	5%	25%	10%
b.	45%	4%	1%	30%	20%

4.30.1 [5] <4.10> What is the CPI achieved by a 2-issue static superscalar processor on this program?

4.30.2 [10] <4.10> In a 2-issue static superscalar whose predictor can only handle one branch per cycle, what speedup is achieved by adding the ability to predict two branches per cycle? Assume a stall-on-branch policy for branches that the predictor cannot handle.

4.30.3 [10] <4.10> In a 2-issue static superscalar processor that only has one register write port, what speedup is achieved by adding a second register write port?

4.30.4 [5] <4.10> For a 2-issue static superscalar processor with a classic 5-stage pipeline, what speedup is achieved by making the branch prediction perfect?

4.30.5 [10] <4.10> Repeat 4.30.4, but for a 4-issue processor. What conclusion can you draw about the importance of good branch prediction when the issue width of the processor is increased?

4.30.6 <4.10> Repeat 4.30.5, but now assume that the 4-issue processor has 50 pipeline stages. Assume that each of the original 5 stages is broken into 10 new stages, and that branches are executed in the first of ten new EX stages. What

conclusion can you dra
the pipeline depth of th

Exercise 4.31

Problems in this exercise
as an MIPS translation of
tions before it exits. Whe
determine what the perfo
the last few iterations of
perfect branch predictio
about are resource hazar
problem have two operar
indicates both the first so
second source data value
example, "sub (edx),eax"
that value from register e

	x86 Inst		
a.	Label:	mov	-4(
		mov	-4(
		add	(ed
		mov	edx
		mov	-4(
		cmp	0,
		jne	Lab
b.	Label:	add	4,
		mov	(ed
		add	4(e
		add	8(e
		mov	eax
		test	edx
		jl	Lab

4.31.1 [20] <4.11> W
is executed on a 1-issue

4.31.2 [20] <4.11> W
is executed on a 1-issu
The stages of the pipeli
are similar to those in
the memory location t

conclusion can you draw about the importance of good branch prediction when the pipeline depth of the processor is increased?

Exercise 4.31

Problems in this exercise refer to the following loop, which is given as x86 code and also as an MIPS translation of that code. You can assume that this loop executes many iterations before it exits. When determining performance, this means that you only need to determine what the performance would be in the "steady state," not for the first few and the last few iterations of the loop. Also, you can assume full forwarding support and perfect branch prediction without delay slots, so the only hazards you have to worry about are resource hazards and data hazards. Note that most x86 instructions in this problem have two operands each. The last (usually second) operand of the instruction indicates both the first source data value and the destination. If the operation needs a second source data value, it is indicated by the other operand of the instruction. For example, "sub (edx),eax" reads the memory location pointed by register edx, subtracts that value from register eax, and puts the result back in register eax.

	x86 Instructions	MIPS-like Translation

a.	`Label: mov -4(esp), eax`
	`mov -4(esp), edx`
	`add (edi,eax,4),edx`
	`mov edx, -4(esp)`
	`mov -4(esp),eax`
	`cmp 0, (edi,eax,4)`
	`jne Label`
		`Label: lw r2,-4(sp)`
		`lw r3,-4(sp)`
		`sll r2,r2,2`
		`add r2,r2,r4`
		`lw r2,0(r2)`
		`add r3,r3,r2`
		`sw r3,-4(sp)`
		`lw r2,-4(sp)`
		`sll r2,r2,2`
		`add r2,r2,r4`
		`lw r2,0(r2)`
		`bne r2,zero,Label`

b.	`Label: add 4, edx`
	`mov (edx), eax`
	`add 4(edx), eax`
	`add 8(edx), eax`
	`mov eax, -4(edx)`
	`test edx, edx`
	`jl Label`
		`Label: addi r4,r4,4`
		`lw r3,0(r4)`
		`lw r2,4(r4)`
		`add r2,r2,r3`
		`lw r3,8(r4)`
		`add r2,r2,r3`
		`sw r2,-4(r4)`
		`slt r1,r4,zero`
		`bne r1,zero,Label`

4.31.1 [20] <4.11> What CPI would be achieved if the MIPS version of this loop is executed on a 1-issue processor with static scheduling and a 5-stage pipeline?

4.31.2 [20] <4.11> What CPI would be achieved if the X86 version of this loop is executed on a 1-issue processor with static scheduling and a 7-stage pipeline? The stages of the pipeline are IF, ID, ARD, MRD, EXE, and WB. Stages IF and ID are similar to those in the 5-stage MIPS pipeline. ARD computes the address of the memory location to be read, MRD performs the memory read, EXE executes

the operation, and WB writes the result to register or memory. The data memory has a read port (for instructions in the MRD stage) and a separate write port (for instructions in the WB stage).

4.31.3 [20] <4.11> What CPI would be achieved if the X86 version of this loop is executed on a processor that internally translates these instructions into MIPS-like micro-operations, then executes these micro-operations on a 1-issue 5-stage pipeline with static scheduling. Note that the instruction count used in CPI computation for this processor is the X86 instruction count.

4.31.4 [20] <4.11> What CPI would be achieved if the MIPS version of this loop is executed on a 1-issue processor with dynamic scheduling? Assume that our processor is not doing register renaming, so you can only reorder instructions that have no data dependences.

4.31.5 [30] <4.10, 4.11> Assuming that there are many free registers available, rename the MIPS version of this loop to eliminate as many data dependences as possible between instructions in the same iteration of the loop. Now repeat 4.31.4, using your new renamed code.

4.31.6 [20] <4.10, 4.11> Repeat 4.31.4, but this time assume that the processor assigns a new name to the result of each instruction as that instruction is decoded, and then renames registers used by subsequent instructions to use correct register values.

Exercise 4.32

Problems in this exercise assume that branches represent the following fraction of all executed instructions, and the following branch predictor accuracy. Assume that the processor is never stalled by data and resource dependences, i.e., the processor always fetches and executes the maximum number of instructions per cycle if there are no control hazards. For control dependences, the processor uses branch prediction and continues fetching from the predicted path. If the branch has been mispredicted, when the branch outcome is resolved the instructions fetched after the mispredicted branch are discarded, and in the next cycle the processor starts fetching from the correct path.

	Branches as a % of All Executed Instructions	Branch Prediction Accuracy
a.	25	95%
b.	25	99%

4.32.1 [5] <4.11> How many instructions are expected to be executed between the time one branch misprediction is detected and the time the next branch misprediction is detected?

The remaining problem
that the branch outcom
from stage 1):

	Pipeline
a.	15
b.	30

4.32.2 [5] <4.11> In
many branch instructio
not yet committed) at a

4.32.3 [5] <4.11> Ho
each branch mispredicti

4.32.4 [10] <4.11> W
4-issue to 8-issue? Assu
the number of instructi
branch resolution stage,

4.32.5 [10] <4.11> W
a 4-issue processor?

4.32.6 [10] <4.11> W
an 8-issue processor? Di
4.32.5.

Exercise 4.33

This exercise explores h
lined multiple-issue pr
the following number o

	Pipelin
a.	1
b.	3

4.33.1 [10] <4.11> H
avoid any resource haza

4.33.2 [10] <4.11> I
dences, what is the exp
with the classical 5-sta
proportion to the numb

The remaining problems in this exercise assume the following pipeline depth and that the branch outcome is determined in the following pipeline stage (counting from stage 1):

	Pipeline Depth	Branch Outcome Known in Stage
a.	15	12
b.	30	20

4.32.2 [5] <4.11> In a 4-issue processor with these pipeline parameters, how many branch instructions can be expected to be "in progress" (already fetched but not yet committed) at any given time?

4.32.3 [5] <4.11> How many instructions are fetched from the wrong path for each branch misprediction in a 4-issue processor?

4.32.4 [10] <4.11> What is the speedup achieved by changing the processor from 4-issue to 8-issue? Assume that the 8-issue and the 4-issue processor differ only in the number of instructions per cycle, and are otherwise identical (pipeline depth, branch resolution stage, etc.).

4.32.5 [10] <4.11> What is the speedup of executing branches 1 stage earlier in a 4-issue processor?

4.32.6 [10] <4.11> What is the speedup of executing branches 1 stage earlier in an 8-issue processor? Discuss the difference between this result and the result from 4.32.5.

Exercise 4.33

This exercise explores how branch prediction affects performance of a deeply pipelined multiple-issue processor. Problems in this exercise refer to a processor with the following number of pipeline stages and instructions issued per cycle:

	Pipeline Depth	Issue Width
a.	15	2
b.	30	8

4.33.1 [10] <4.11> How many register read ports should the processor have to avoid any resource hazards due to register reads?

4.33.2 [10] <4.11> If there are no branch mispredictions and no data dependences, what is the expected performance improvement over a 1-issue processor with the classical 5-stage pipeline? Assume that the clock cycle time decreases in proportion to the number of pipeline stages.

4.33.3 [10] <4.11> Repeat 4.33.2, but this time every executed instruction has a RAW data dependence to the instruction that executes right after it. You can assume that no stall cycles are needed, i.e., forwarding allows consecutive instructions to execute in back-to-back cycles.

For the remaining three problems in this exercise, unless the problem specifies otherwise, assume the following statistics about what percentage of instructions are branches, predictor accuracy, and performance loss due to branch mispredictions:

	Branches as a Fraction of All Executed Instructions	Branches Execute in Stage	Predictor Accuracy	Performance Loss
a.	10%	9	96%	5%
b.	10%	5	98%	1%

4.33.4 [10] <4.11> If we have the given fraction of branch instructions and branch prediction accuracy, what percentage of all cycles are entirely spent fetching wrong-path instructions? Ignore the performance loss number.

4.33.5 [20] <4.11> If we want to limit stalls due to mispredicted branches to no more than the given percentage of the ideal (no stalls) execution time, what should be our branch prediction accuracy? Ignore the given predictor accuracy number.

4.33.6 [10] <4.11> What should the branch prediction accuracy be if we are willing to have a speedup of 0.5 (one half) relative to the same processor with an ideal branch predictor?

Exercise 4.34

This exercise is designed to help you understand the discussion of the "Pipelining is easy" fallacy from Section 4.13. The first four problems in this exercise refer to the following MIPS instruction:

	Instruction	Interpretation
a.	`AND Rd,Rs,Rt`	`Reg[Rd]=Reg[Rs] AND Reg[Rt]`
b.	`SW Rt,Offs(Rs)`	`Mem[Reg[Rs]+Offs] = Reg[Rt]`

4.34.1 [10] <4.13> Describe a pipelined datapath needed to support only this instruction. Your datapath should be designed with the assumption that the only instructions that will ever be executed are instances of this instruction.

4.34.2 [10] <4.13> Describe the requirements of forwarding and hazard detection units for your datapath from 4.34.1.

4.34.3 [10] <4.13> W
exceptions in your dat
exception should be tri
of instruction.

The remaining two pro

	Instructio
a.	`ADD Rd,Rs,Rt`
b.	`ADDI Rt,Rs,Imm`

4.34.4 [10] <4.13> D
also support this instru
support instances of th

4.34.5 [10] <4.13> R

4.34.6 [10] <4.13> R

Exercise 4.35

This exercise is intend
ISA design and pipelin
tiple-issue pipelined
instructions issued pe
branch predictor accu

	Pipeline Depth	Issue Width
a.	15	2
b.	25	4

4.35.1 [5] <4.8, 4.13
slots. How many dela
control hazards in this

4.35.2 [10] <4.8, 4.
four branch delay slo
there are no data dep
can be filled with use
instructions. To make
predicted branch inst
i.e., no instructions t
from the wrong path.

4.34.3 [10] <4.13> What needs to be done to support undefined instruction exceptions in your datapath from 4.34.1? Note that the undefined instruction exception should be triggered whenever the processor encounters any other kind of instruction.

The remaining two problems in this exercise also refer to this MIPS instruction:

	Instruction	Interpretation
a.	`ADD Rd,Rs,Rt`	`Reg[Rd]=Reg[Rs]+Reg[Rt]`
b.	`ADDI Rt,Rs,Imm`	`Reg[Rt]=Reg[Rs]+Imm`

4.34.4 [10] <4.13> Describe how to extend your datapath from 4.34.1 so it can also support this instruction. Your extended datapath should be designed to only support instances of these two instructions.

4.34.5 [10] <4.13> Repeat 4.34.2 for your extended datapath from 4.34.4.

4.34.6 [10] <4.13> Repeat 4.34.3 for your extended datapath from 4.34.4.

Exercise 4.35

This exercise is intended to help you better understand the relationship between ISA design and pipelining. Problems in this exercise assume that we have a multiple-issue pipelined processor with the following number of pipeline stages, instructions issued per cycle, stage in which branch outcomes are resolved, and branch predictor accuracy:

	Pipeline Depth	Issue Width	Branches Execute in Stage	Branch Predictor Accuracy	Branches as a % of Instructions
a.	15	2	10	90%	25%
b.	25	4	15	96%	15%

4.35.1 [5] <4.8, 4.13> Control hazards can be eliminated by adding branch delay slots. How many delay slots must follow each branch if we want to eliminate all control hazards in this processor?

4.35.2 [10] <4.8, 4.13> What is the speedup that would be achieved by using four branch delay slots to reduce control hazards in this processor? Assume that there are no data dependences between instructions and that all four delay slots can be filled with useful instructions without increasing the number of executed instructions. To make your computations easier, you can also assume that the mispredicted branch instruction is always the last instruction to be fetched in a cycle, i.e., no instructions that are in the same pipeline stage as the branch are fetched from the wrong path.

4.35.3 [10] <4.8, 4.13> Repeat 4.35.2, but now assume that 10% of executed branches have all four delay slots filled with useful instruction, 20% have only three useful instructions in delay slots (the fourth delay slot is an NOP), 30% have only two useful instructions in delay slots, and 40% have no useful instructions in their delay slots.

The remaining four problems in this exercise refer to the following C loop:

a.
```
for(i=0;i!=j;i++){
  c+=a[i];
}
```

b.
```
for(i=0;i!=j;i+=2){
  c+=a[i]-a[i+1];
}
```

4.35.4 [10] <4.8, 4.13> Translate this C loop into MIPS instructions, assuming that our ISA requires one delay slot for every branch. Try to fill delay slots with non-NOP instructions when possible. You can assume that variables `a`, `b`, `c`, `i`, and `j` are kept in registers `r1`, `r2`, `r3`, `r4`, and `r5`.

4.35.5 [10] <4.7, 4.13> Repeat 4.35.4 for a processor that has two delay slots for every branch.

4.35.6 [10] <4.10, 4.13> How many iterations of your loop from 4.35.4 can be "in flight" within this processor's pipeline? We say that an iteration is "in flight" when at least one of its instructions has been fetched and has not yet been committed.

Exercise 4.36

This exercise is intended to help you better understand the last pitfall from Section 4.13—failure to consider pipelining in instruction set design. The first four problems in this exercise refer to the following new MIPS instruction:

	Instruction	Interpretation
a.	`SWINC Rt,Offset(Rs)`	`Mem[Reg[Rs]+Offset]=Reg[Rt]` `Reg[Rs]=Reg[Rs]+4`
b.	`SWI Rt,Rd(Rs)`	`Mem[Reg[Rd]+Reg[Rs]]= Reg[Rt]`

4.36.1 [10] <4.11, 4.13> Translate this instruction into MIPS micro-operations.

4.36.2 [10] <4.11, 4.13> How would you change the 5-stage MIPS pipeline to add support for micro-op translation needed to support this new instruction?

4.36.3 [20] <4.13> If
the changes to the pipeli
needed to directly (with

4.36.4 [10] <4.13> H
you think that we would

The remaining two pr
instruction to the ISA. I
lems assume the followi
tion is completed in th
completing):

	ADD	BEQ
a.	25%	20%
b.	25%	10%

4.36.5 [10] <4.13> G
sor with direct support
replacing this instructio
Assume that the ADDM i
cal 5-stage pipeline with

4.36.6 [10] <4.13> Re
adding a pipeline stage.
and, as a result, half of th
elimination applies only
added by the ADDM tran

Exercise 4.37

This exercise explores s
cycle time and utilizatio
exercise refer to the follo
that the processor does

a.
```
        SW R16,-1
        LW R16,8(
        BEQ R5,R4
        ADD R5,R1
        SLT R5,R1
```

b.
```
        OR R1,R2,
        SW R1,0(R
        BEQ R1,R0
        OR R2,R1,
Label:  ADD R1,R1
```