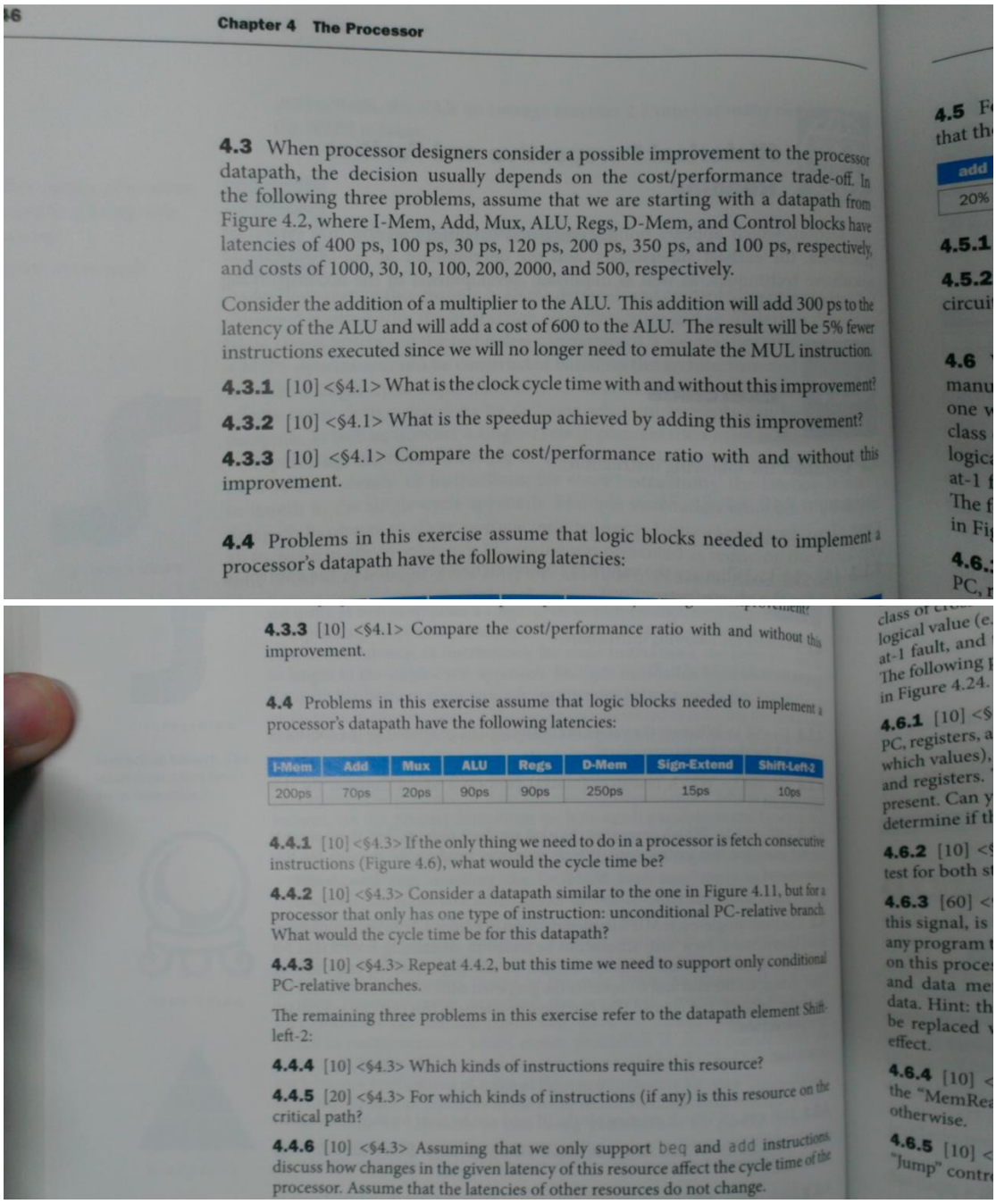
HW4: 4.3, 4.4, 4.7, 4.8, 4.9, 4.10, 4.13, 4.14, 4.15



4.3 When processor designers consider a possible improvement to the processor datapath, the decision usually depends on the cost/performance trade-off. 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 400 ps, 100 ps, 30 ps, 120 ps, 200 ps, 350 ps, and 100 ps, respectively, and costs of 1000, 30, 10, 100, 200, 2000, and 500, respectively.

Consider the addition of a multiplier to the ALU. This addition will add 300 ps to the latency of the ALU and will add a cost of 600 to the ALU. The result will be 5% fewer instructions executed since we will no longer need to emulate the MUL instruction.

4.3.1 [10] <§4.1> What is the clock cycle time with and without this improvement?

4.3.2 [10] <§4.1> What is the speedup achieved by adding this improvement?

4.3.3 [10] <§4.1> Compare the cost/performance ratio with and without this improvement.

4.4 Problems in this exercise assume that logic blocks needed to implement a processor's datapath have the following latencies:

I-Mem	Add	Mux	ALU	Regs	D-Mem	Sign-Extend	Shift-Left-2
200ps	70ps	20ps	90ps	90ps	250ps	15ps	10ps

4.4.1 [10] <§4.3> If the only thing we need to do in a processor is fetch consecutive instructions (Figure 4.6), what would the cycle time be?

4.4.2 [10] <§4.3> Consider a datapath similar to the one in Figure 4.11, but for a processor that only has one type of instruction: unconditional PC-relative branch. What would the cycle time be for this datapath?

4.4.3 [10] <§4.3> Repeat 4.4.2, but this time we need to support only conditional PC-relative branches.

The remaining three problems in this exercise refer to the datapath element Shift-left-2:

4.4.4 [10] <§4.3> Which kinds of instructions require this resource?

4.4.5 [20] <§4.3> For which kinds of instructions (if any) is this resource on the critical path?

4.4.6 [10] <§4.3> Assuming that we only support `beq` and `add` instructions, discuss how changes in the given latency of this resource affect the cycle time of the processor. Assume that the latencies of other resources do not change.

4.7 In this exercise we examine in detail how an instruction is executed in a single-cycle datapath. Problems in this exercise refer to a clock cycle in which the processor fetches the following instruction word:

`10101100011000100000000000010100`.

Assume that data memory is all zeros and that the processor's registers have the following values at the beginning of the cycle in which the above instruction word is fetched:

r0	r1	r2	r3	r4	r5	r6	r8	r12	r31
0	-1	2	-3	-4	10	6	8	2	-16

4.7.1 [5] <§4.4> What are the outputs of the sign-extend and the jump "Shift left 2" unit (near the top of Figure 4.24) for this instruction word?

4.7.2 [10] <§4.4> What are the values of the ALU control unit's inputs for this instruction?

4.7.3 [10] <§4.4> What is the new PC address after this instruction is executed? Highlight the path through which this value is determined.

4.7.4 [10] <§4.4> For each Mux, show the values of its data output during the execution of this instruction and these register values.

4.7.5 [10] <§4.4> For the ALU and the two add units, what are their data input values?

4.7.6 [10] <§4.4> What are the values of all inputs for the "Registers" unit?

4.8 In this exercise, we examine how pipelining affects the clock cycle time of the

4.8.3 [10] <§4.5
stages, each with
and what is the n

4.8.4 [10] <§4.5
of the data memo

4.8.5 [10] <§4.5
of the write-regis

4.8.6 [30] <§4.5
organization wh
finishes before a
through stages I
need the WB st
cycle, multi-cycl

4.9 In this exe
basic 5-stage pip
following seque

```
or r1,r2,r3
or r2,r1,r4
or r1,r1,r2
```

Also, assume th

Without For
250ps

4.9.1 [10] <§

s instruction is executed?
d.

s data output during the

what are their data input

he "Registers" unit?

he clock cycle time of the
ual stages of the datapath

WB
200ps

sor are broken down as

lined and non-pipelined

in a pipelined

4.9 In this exercise, we examine how data dependences affect execution in the basic 5-stage pipeline described in Section 4.5. Problems in this exercise refer to the following sequence of instructions:

```
or r1,r2,r3
or r2,r1,r4
or r1,r1,r2
```

Also, assume the following cycle times for each of the options related to forwarding:

Without Forwarding	With Full Forwarding	With ALU-ALU Forwarding Only
250ps	300ps	290ps

4.9.1 [10] <§4.5> Indicate dependences and their type.

4.9.2 [10] <§4.5> Assume there is no forwarding in this pipelined processor. Indicate hazards and add `nop` instructions to eliminate them.

4.9.3 [10] <§4.5> Assume there is full forwarding. Indicate hazards and add `NOP` instructions to eliminate them.

4.9.4 [10] <§4.5> What is the total execution time of this instruction sequence without forwarding and with full forwarding? What is the speedup achieved by adding full forwarding to a pipeline that had no forwarding?

4.9.5 [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).

4.9.6 [10] <§4.5> What is the total execution time of this instruction sequence with only ALU-ALU forwarding? What is the speedup over a no-forwarding pipeline?

4.10 In this exercise, we examine how resource hazards, control haza
Instruction Set Architecture (ISA) design can affect pipelined execution. F
in this exercise refer to the following fragment of MIPS code:

```
sw  r16,12(r6)
lw  r16,8(r6)
beq r5,r4,Label # Assume r5!=r4
add r5,r1,r4
slt r5,r15,r4
```

Assume that individual pipeline stages have the following latencies:

IF	ID	EX	MEM	WB
200ps	120ps	150ps	190ps	100ps

4.10.1 [10] <§4.5> For this problem, assume that all branches are pe
predicted (this eliminates all control hazards) and that no delay slots are usec
only have one memory (for both instructions and data), there is a structural h
every time we need to fetch an instruction in the same cycle in which an
instruction accesses data. To guarantee forward progress, this hazard must a
be resolved in favor of the instruction that accesses data. What is the total exec
time of this instruction sequence in the 5-stage pipeline that only has one mer
We have seen that data hazards can be eliminated by adding `nop`s to the code
you do the same with this structural hazard? Why?

4.10.2 [20] <§4.5> For this problem, assume that all branches are per
predicted (this eliminates all control hazards) and that no delay slots are
If we change load/store instructions to use a register (without an offset) a
address, these instructions no longer need to use the ALU. As a result, MEM
EX stages can be overlapped and the pipeline has only 4 stages. Change this co
accommodate this changed ISA. Assuming this change does not affect clock c
time, what speedup is achieved in this instruction sequence?

4.10.3 [10] <§4.5> Assuming stall-on-branch and no delay slots, what speed
achieved on this code if branch outcomes are determined in the ID stage, relati
the execution where branch outcomes are determined in the EX stage?

4.10.4 [10] <§4.5> Given these pipeline stage latencies, repeat the spee
calculation from 4.10.2, but take into account the (possible) change in clock c
time. When EX and MEM are done in a single stage, most of their work can
done in parallel. As a result, the resulting EX/MEM stage has a latency that is
larger of the original two, plus 20 ps needed for the work that could not be d
in parallel.

4.10.5 [10] <§4.5> Given these pipeline stage latencies, repeat the speed
calculation from 4.10.3, taking into account the (possible) change in clock cy
time. Assume that the latency ID stage increases by 50% and the latency of the
stage decreases by 10ps when branch outcome resolution is moved from EX to I

hazards, and
on. Problems

4.10.6 [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 speedup from this change? Assume that the latency of the EX stage is reduced by 20 ps and the latency of the MEM stage is unchanged when branch outcome resolution is moved from EX to MEM.

4.11 Consider the following loop.

```
loop:lw  r1,0(r1)
     and r1,r1,r2
     lw  r1,0(r1)
     lw  r1,0(r1)
     beq r1,r0,loop
```

re perfectly
used. If we

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

executed before the loop exits.

4.12.6 [20] <§4.7> Repeat 4.12.3 but this time determine which of the two options results in shorter time per instruction.

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:

```
add r5,r2,r1
lw  r3,4(r5)
lw  r2,0(r2)
or  r3,r5,r3
sw  r3,0(r5)
```

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

label2: s

4.14.1 [10]
there are no

4.14.2 [10]
given code, t
for that bran

4.14.3 [20]
not need an
be "bez rd,
and does not
instructions
to use as a te
be used.

ges. For the EX stage,
ing and for a processor

W EM/ ly)	MEM	WB
ps	120 ps	100 ps

f cycles are we stalling

ll results that can be
a hazards?

ave three-input Muxes
it is better to forward
arding) or only from
ich of the two options

pipeline stage latencies,
that had no

Which of the two options

d pipeline stage latencies,
to a pipeline that had no

up (relative to a processor
that eliminates all data
l circuitry adds 100 ps to

rmine which of the two

the relationship between
s in this exercise refer to
it is executed on a 5-stage

l detection, insert nops to

4.13.2 [10] <§4.7> Repeat 4.13.1 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.

4.13.3 [10] <§4.7> If the processor has forwarding, but we forgot to implement the hazard detection unit, what happens when this code executes?

4.13.4 [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.

4.13.5 [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.

4.13.6 [20] <§4.7> For the new hazard detection unit from 4.13.5, specify which output signals it asserts in each of the first five cycles during the execution of this code.

4.14 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:

```
        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)
```

4.14.1 [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.

4.14.2 [10] <§4.8> Repeat 4.14.1, 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.

4.14.3 [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.

4.14.4 [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?

4.14.5 [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.

4.14.6 [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.

4.15 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
40%	25%	5%	25%	5%

Also, assume the following branch predictor accuracies:

Always-Taken	Always-Not-Taken	2-Bit
45%	55%	85%

4.15.1 [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.

4.15.2 [10] <§4.8> Repeat 4.15.1 for the "always-not-taken" predictor.

4.15.3 [10] <§4.8> Repeat 4.15.1 for for the 2-bit predictor.

4.15.4 [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.