Selected Exercise for Chapter 4 Part1			
4.1 all	4.2 all	4.3 all	4.5

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
add Rd ,Rs ,Rt	Reg[Rd] = Reg[Rs] + Reg[Rt]

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

4.1.1 Solution: The values of the signals are as follows:

RegWrite	MemRead	ALUSrc	MemWrite	ALUOp	MemToReg	Branch
1	0	0 (Reg)	0	Add	0(ALU)	0

ALUSrc is the control signal that controls the Mux at the ALU input, 0 (Reg) selects the output of the register file and 1 (Imm) selects the immediate from the instruction word as the second input to the ALU. MemToReg is the control signal that controls the Mux at the Data input to the register file, 0 (ALU) selects the output of the ALU and 1 (Mem) selects the output of memory.

A value of X is a “don’t care” (does not matter if signal is 0 or 1)

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

4.1.2 Solution: Resources performing a useful function for this instruction are the following: All except Data Memory and branch Add unit.

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?

4.1.3 Solution: Outputs that are not used : Branch Add. No output: Data Memory

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 Instruction Set Architecture (ISA), but the decision whether or not to do that depends, among other things, on the cost and complexity the proposed addition introduces into the processor datdpath and control. The first three problems in this exercise refer to the new instruction:

Instruction: LWI Rt, Rd (Rs)

Interpretation: Reg [Rt] = Mem [Reg [Rd] + Reg [Rs]]

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

4.2.1 Solution: This instruction uses instruction memory, both register read ports, the ALU to add Rd and Rs together, data memory, and write port in Registers.

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

4.2.2 Solution: None. This instruction can be implemented using existing blocks.

4.2.3 [10] <§4.1 > What new signals do we need (if any) from the control unit to support this instruction?
4.2.3 Solution: None. This instruction can be implemented without adding new control signals. It only requires changes in the Control logic.

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?

I-Mem	Add	Mux	ALU	Reg	D-Mem	Control
400ps	100ps	30ps	120ps	200ps	350ps	100ps
1000	30	10	100	200	2000	500

4.3.1 Solution: Clock cycle time is determined by the critical path, which for the given latencies happens to be to get the data value for the load instruction: I-Mem (read instruction), Regs (takes longer than Control), Mux (select ALU input), ALU, Data Memory, and Mux (select value from memory to be written into Registers). The latency of this path is 400 ps + 200 ps + 30 ps + 120 ps + 350 ps + 30 ps = 1130 ps.
1430 ps (1130 ps + 300 ps, ALU is on the critical path).
(Some answer said additional one register latency should be added (400+200 + 30+120+350+30 +200 = 1330). However, this is incorrect because only one register write occurs in a cycle.

4.3.2 [10] <§4.1 > What is the speedup achieved by adding this improvement?
4.3.2 Solution: The speedup comes from changes in clock cycle time and changes to the number of clock cycles we need for the program: We need 5% fewer cycles for a program, but cycle time is 1430 instead of 1130, so we have a speedup of (1/0.95)*(1130/1430) = 0.83, which means we actually have a slowdown.

4.3.3 [10] <§4.1> Compare the cost/performance ratio with and without this improvement.
4.3.3 Solution: The cost is always the total cost of all components (not just those on the critical path, so the original processor has a cost of I-Mem, Regs, Control, ALU, D-Mem, 2 Add units and 3 Mux units, for a total cost of 1000 + 200 + 500 + 100 + 2000 + 2*30 + 3*10 = 3890.
We will compute cost relative to this baseline. The performance relative to this baseline is the speedup we previously computed, and our cost/ performance relative to the baseline is as follows:
New Cost: 3890 + 600 = 4490
Relative Cost: 4490/3890 = 1.15

Cost/Performance: 1.15/0.83 = 1.39. We are paying significantly more for significantly worse performance; the cost/performance is a lot worse than with the unmodified processor.

4.5 For the 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
20%	20%	0%	25%	25%	10%

4.5.1 [10] < §4.3> In what fraction of all cycles is the data memory used?

4.5.1 Solution: The data memory is used by LW and SW instructions, so the answer is:
25% + 10% = 35%

4.5.2 [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?

4.5.2 The input of the sign-extend circuit is needed for ADDI (to provide the immediate ALU operand), BEQ (to provide the PC-relative off set), and LW and SW (to provide the offset used in addressing memory) so the answer is: 20% + 25% + 25% + 10% = 80%

The sign-extend circuit is actually computing a result in every cycle, but its output is ignored for ADD and NOT instructions.