1. Processor Implementation Details (30 points)

P&H(4.2) <§4.1>. 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 datapath 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]]

a. Which existing blocks (if any) can be used for this instruction?

The register file for getting rs and rd values
The ALU to add rs and rd to form the memory address
Mux to prevent the immediate being selected
Data Memory to get the added memory address to read from
As well, all PC and I-Mem operations would still occur and need those parts

b. Which new functional blocks (if any) do we need for this instruction?

We should need two new multiplexers before the read 2 register # and write register # for the register file. This is because rt is normally sent to read 2 and rd is write. So we 2 multiplexers to switch the rt and rd locations. The one in front of read 2 should have rt as 0 and rd as 1. The in front of write should be rd as 0 and rt as 1.

c. What new signals do we need (if any) from the control unit to support this instruction?

We need only one new signal to control both multiplexers as they should both either swap the instructions or not at same time. As of right now, we only need them to be the same.

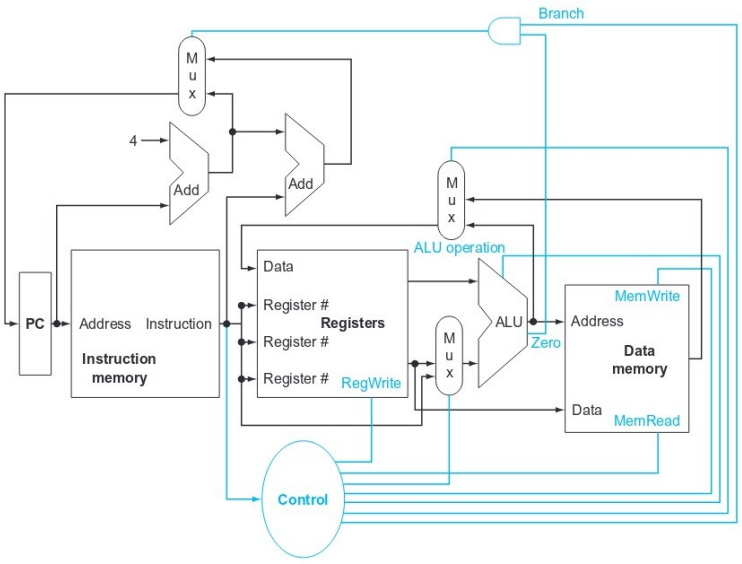

FIGURE 4.2 The basic implementation of the MIPS subset, including the necessary multiplexors and control lines.

2. Processor Cycle Time Determination (40 points)
Assume the following latencies for the logic blocks in Figure 4.17 from the textbook.

I-Mem	Adder	MUX	ALU	Reg Read	D-Mem	Sign-Extend	Shift-Left-2	Control	ALU Control	AND Gate
200ps	70ps	20ps	90ps	90ps	250ps	10ps	5ps	40ps	20ps	10ps

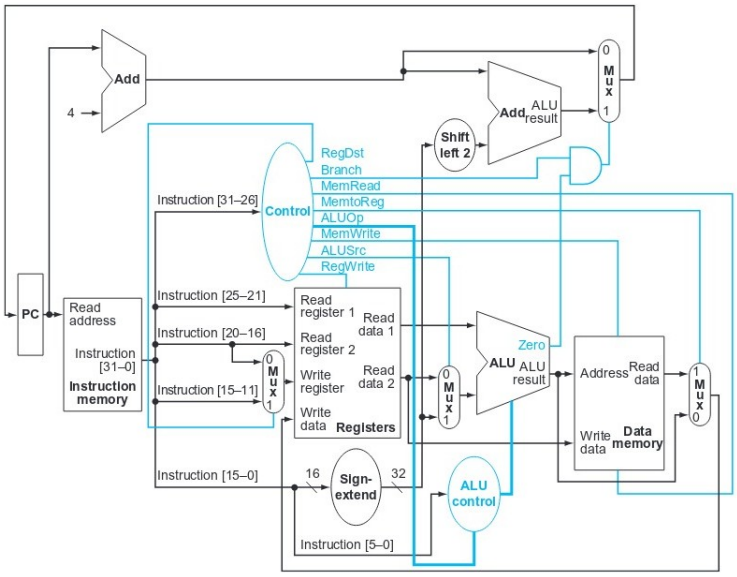

FIGURE 4.17 The simple datapath with the control unit. The input to the control unit is the 6-bit opcode field from the instruction.

a. Identify and quantify (i.e., give the path through the blocks and the time for that path) the worst-case path for each of the following: an arithmetic R-format instruction, a lw instruction, and a conditional branch instruction. [Note that in your project you will actually synthesize your processor and be asked to identify its critical path—you'll find out the specific delays for the components you've designed when they are mapped to an FPGA. This problem should help you develop an intuitive sense of how to reason about critical paths.]

I think I would need a reg write, pc read, and pc write speeds.
Yellow is the worst case path though somethings may be affected by other paths.

R-Format	I-Mem / 0 ps	Reg / 200ps	ALUSrc Mux / 290ps	ALU / 310ps	MemtoReg Mux / 400ps	Rdst Mux / 420ps	Reg / 440ps
		Ctrl / 200ps	ALU Ctrl / 240ps				
	Adder / 0ps				And / 400ps	Branch Mux / 410ps	PC / 430ps

Lw	I-Mem / 0 ps	Reg / 200ps	ALUSrc Mux / 290ps	ALU / 310ps	DMem/ 400ps	MemtoReg Mux / 650ps	Rdst Mux / 670ps	Reg / 690ps
		Ctrl / 200ps	ALU Ctrl / 240ps					
	Adder / 00ps				And / 400ps	Branch Mux / 410ps	PC / 430ps	

Branch	I-Mem / 0 ps	Reg / 200ps	ALUSrc Mux / 290ps	ALU / 310ps	And / 400ps	Branch Mux / 410ps	PC / 430ps
		Ctrl / 200ps	ALU Ctrl / 240ps				
	Adder / 0ps	Extend / 200ps	Shift L2 / 210ps	Adder / 215ps			

b. Rank the following design approaches in terms of which improve the cycle time the most. You must justify your ranking.

Changing the ISA memory addressing mode to directly use the base register's value (i.e., no offset addition)

This would be the best to improve cycle time as our load instruction is our bottleneck. This would remove the alu related step in a load, which I think would save about 110 ps (ALUSrc and ALU). Cutting it down from 690 to 580ps.

Developing an "extra fast adder" that reduces the Adder and ALU latencies

This would be next best as we have 70 ps to save with the adder and assuming this is the bottleneck of the ALU we could cut it down until a different internal module is slower. This likely doesn't actually save much.

Designing a lower-latency control unit

Making this fast does nothing since in our design the control and register file operate at the same time. The control is already fast so the ALU/ALUSrc Mux would still be waiting on the register file to read values.

3. Amdahl's Law (30 points) Solve problem 1.14 from the textbook.
NOTE: 106 means 10^6(1 million)in the book.

1.14 Assume a program requires the execution of 50 × 10^6 FP instructions, 110 × 10^6 INT instructions, 80 × 10^6 L/S instructions, and 16 × 10^6 branch instructions. The CPI for each type of instruction is 1, 1, 4, and 2, respectively.
Assume that the processor has a 2 GHz clock rate.

Total Cycles = (50 + 110 + 320 + 32) = 512 * 10^6
F_{FP} = 50 / 512 = 25 / 256
F_{INT} = 110 / 512 = 55 / 256
$F_{L/S}$ = 320 / 512 = 5 / 8
F_{Branch} = 32 / 512 = 1 / 16

1.14.1 [10] <§1.10> By how much must we improve the CPI of FP instructions if we want the program to run two times faster?

SU = 2 = 1/((1-25/256) + (25/256)/a)
a → infinity = 2 = 1/(231/256)
= 2 != 1.108225

Not possible to double speed from just FP instructions

1.14.2 [10] <§1.10> By how much must we improve the CPI of L/S instructions if we want the program to run two times faster?

SU = 2 = 1/((1-5/8) + (5/8)/a)
½ = 3/8 + (5/8) / a
1/8 = (5/8) / a
a = 5

Improve speed of L/S instructions by 5 times

1.14.3 [5] <§1.10> By how much is the execution time of the program improved if the CPI of INT and FP instructions is reduced by 40% and the CPI of L/S and Branch is reduced by 30?

512 * 10^6 Cycles / 2 GHz = 512 000 000 / 2000 000 000
= 512 / 2000 = .256 seconds

New Cycles = (.6*50 + .6*110 + .7*320 + .7*32) = (30 + 60+ 224 + 22.4) = 314 + 23 = 337 * 10^6

337 * 10^6 Cycles / 2 GHz = 337 000 000 / 2000 000 000
= 337 / 2000 = .1685 seconds

.256 - .1685 = .0875 seconds improved