JOINT INSTITUTE
交大密西根学院

Ve370 Introduction to Computer Organization

Homework 4

1. Exercise 4.2.2
2. Exercise 4.2.3
3. Exercise 4.2.4

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.	`SEQ Rd,Rs,Rt`	`Reg[Rd]` = Boolean value (0 or 1) of (`Reg[Rs] == Reg[Rs]`)
b.	`LWI Rt,Rd(Rs)`	`Reg[Rt] = Mem[Reg[Rd]+Reg[Rs]]`

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

4.2.2 [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 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 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.	Add Multiplier to ALU	+300ps for ALU	+600 for ALU	Lets us add MUL instruction. Allows us to execute 5% fewer instructions (MUL no longer emulated).
b.	Simpler Control	+100ps for Control	–400 for Control	Control becomes slower but cheaper logic.

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 speedup achieved by adding this improvement?

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

4. Exercise 4.6.5

The remaining three problems in this exercise refer to the following logic block (resource) in the datapath:

	Resource
a.	Shift-left-2
b.	Registers

4.6.4 [10] <4.3> Which kinds of instructions require this resource?

4.6.5 [20] <4.3> For which kinds of instructions (if any) is this resource on the critical path?

4.6.6 [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.

5. Exercise 4.7.1
6. Exercise 4.7.2
7. Exercise 4.7.3

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.	200ps	70ps	20ps	90ps	90ps	250ps	15ps	10ps
b.	750ps	200ps	50ps	250ps	300ps	500ps	100ps	5ps

4.7.1 [10] <4.3> What is the clock cycle time if the only types 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 have 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?

8. Exercise 4.8.1
9. Exercise 4.8.2

Exercise 4.8

When silicon chips are fabricated, defects in materials (e.g., silicon) and manufacturing errors can result in defective circuits. A very common defect is for one wire to affect the signal in another. This is called a cross-talk fault. A special class of

cross-talk faults is when a signal is connected to a wire that has a constant logical value (e.g., a power supply wire). In this case we have a stuck-at-0 or a stuck-at-1 fault, and the affected signal always has a logical value of 0 or 1, respectively.

The following problems refer to the following signal from Figure 4.24:

	Signal
a.	Registers, input Write Register, bit 0
b.	Add unit in upper right corner, ALU result, bit 0

4.8.1 [10] <4.3, 4.4> Let us assume that processor testing is done by filling the PC, registers, and data and instruction memories with some values (you can choose which values), letting a single instruction execute, then reading the PC, memories, and registers. These values are then examined to determine if a particular fault is present. Can you design a test (values for PC, memories, and registers) that would determine if there is a stuck-at-0 fault on this signal?

4.8.2 [10] <4.3, 4.4> Repeat 4.8.1 for a stuck-at-1 fault. Can you use a single test for both stuck-at-0 and stuck-at-1? If yes, explain how; if no, explain why not.

10. Exercise 4.11.2
11. Exercise 4.11.3

Exercise 4.11

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:

	Instruction word
a.	10101100011000100000000000010100
b.	00000000100000100000100000101010

4.11.1 [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?

4.11.2 [10] <4.4> What are the values of the ALU control unit's inputs for this instruction?

12. Modify the MIPS single cycle implementation based on Figure 4.24 to support the instruction BNE.
13. Modify the MIPS single cycle implementation based on Figure 4.24 to support the instruction JAL.
14. Modify the MIPS single cycle implementation based on Figure 4.24 to support the instruction JR.

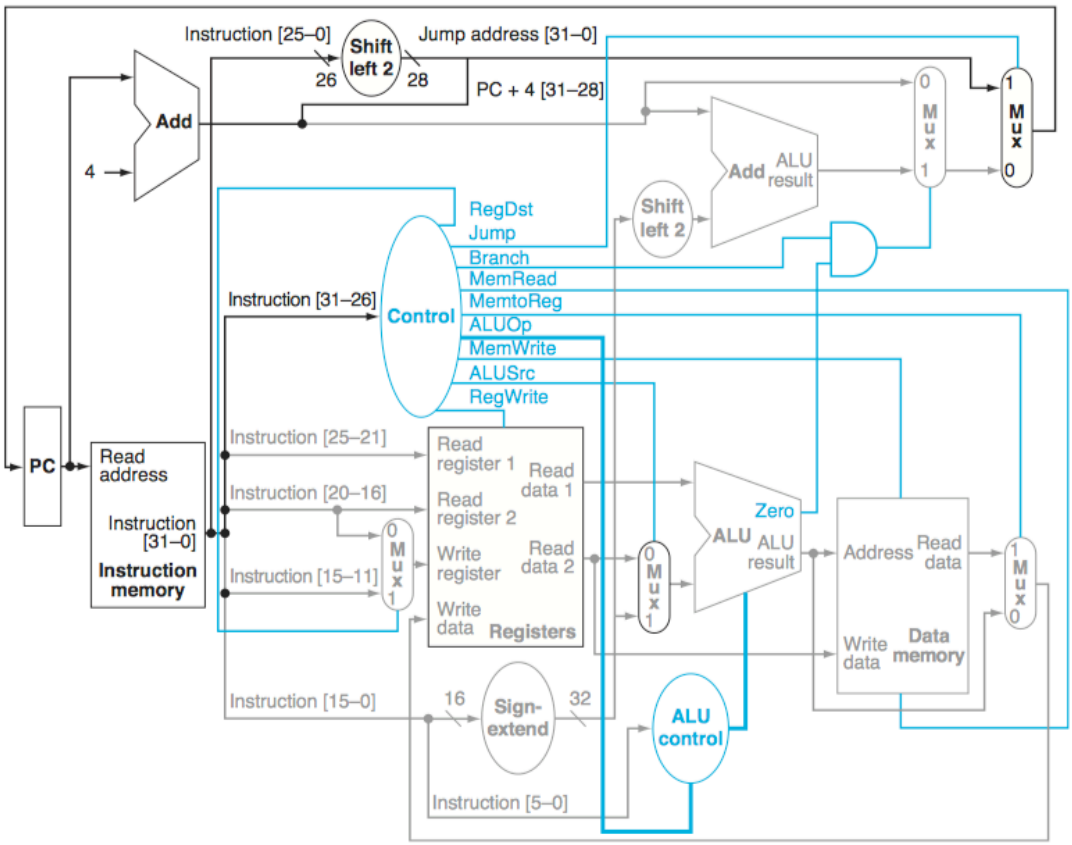

FIGURE 4.24 The simple control and datapath are extended to handle the jump instruction. An additional multiplexor (at the upper right) is used to choose between the jump target and either the branch target or the sequential instruction following this one. This multiplexor is controlled by the jump control signal. The jump target address is obtained by shifting the lower 26 bits of the jump instruction left 2 bits, effectively adding 00 as the low-order bits, and then concatenating the upper 4 bits of PC + 4 as the high-order bits, thus yielding a 32-bit address.