VE370 RC4

Lecture Slides

Exercise 4.2.2

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]]`

4.2.2 [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?

(a). A new mux is required before Write Data of Registers to selected between Zero output from ALU, data read from Data Memory, and Result from ALU.

(b). No new functional block.

Exercise 4.2.3

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]]`

4.2.3 [10] <4.1> What new signals do we need (if any) from the control unit to support this instruction?

(a). **New signal:** ZerotoReg to control the mux.

When it is 1, select Zero to Write Data of Registers

(b). **New signal:** No

RegDst=1 ALUOp=0010 MemtoReg=1 RegWrite=1 ALUSrc=0 MemRead=1 MemWrite=0 Branch=0

By changing the original signals, we can support this instruction.

Exercise 4.2.4

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?

Critical path (lw): I-Mem(read instruction) + Regs (takes longer than Control) + Mux (select ALU input) + ALU + Data Memory + Mux (select value from memory to be written into Registers) + Regs = 1330ps

(a). ALU+300ps ➔ total+300ps=1630ps

(b). No change. (Control not in the critical path)

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

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

(a). Jump instructions

(b). All instructions except jump

Exercise 4.7.1

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.)?

Critical path:I-Mem + Regs + Mux + ALU + Mux + Regs

(a).

200ps+90ps+20ps+90ps+20ps+90ps=510ps

(b).

750ps+300ps+50ps+250ps+50ps+300ps=1700ps

Exercise 4.7.2-3

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

4.7.2 [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?

4.7.2

Critical path I-Mem+ Regs + Mux + ALU + Data Memory + Mux + Regs

(a).
200ps+90ps+20ps+90ps+250ps+20ps+90ps=760ps

(b).

750ps+300ps+50ps+250ps+500ps+50ps+300ps=2200ps

4.7.3

As same as 4.7.2, since the critical path is lw instruction

Exercise 4.8.1(a)

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?

We can test this bit by writing an immediate value to the register with bit 0 as 1.

addi $t1, $0,128

$t1:01001 $t0:01000

If the value of $t0 is 128 while $t1 is not, there is a stuck-at-0 fault.

Exercise 4.8.1(b)

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?

The only instruction that use Add unit in upper right corner is branch. Bit 0 of ALU result relates to PC value. The ALU result represents the address of an instruction, whose bit 0 is always 0. Hence, there is no way to design a test for a stuck-at-0 fault.

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

4.8.2 [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.

The test for stuck-at-zero requires an instruction that sets the signal to 1 and the test for stuck-at-1 requires an instruction that sets the signal to 0. Because the signal cannot be both 0 and 1 in the same cycle, we cannot test the same signal simultaneously for stuck-at-0 and stuck-at-1 using only one instruction.

(a). addi $t0, $0,128

If $t1 becomes 128 while $t0 doesn‘t, there is a stuck-at-1 fault

(b). beq $s0, $t0, LOOP

If it doesn’t jump to LOOP, there is a stuck-at-1 fault.

Exercise 4.11.2-4.11.3

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

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

4.11.3 [10] <4.4> What is the new PC address after this instruction is executed? Highlight the path through which this value is determined.

4.11.2

(a). sw

ALUOp: 00 Funct: XXXXXX

(b). set-on-less-than

ALUOp: 10 Funct: 101010

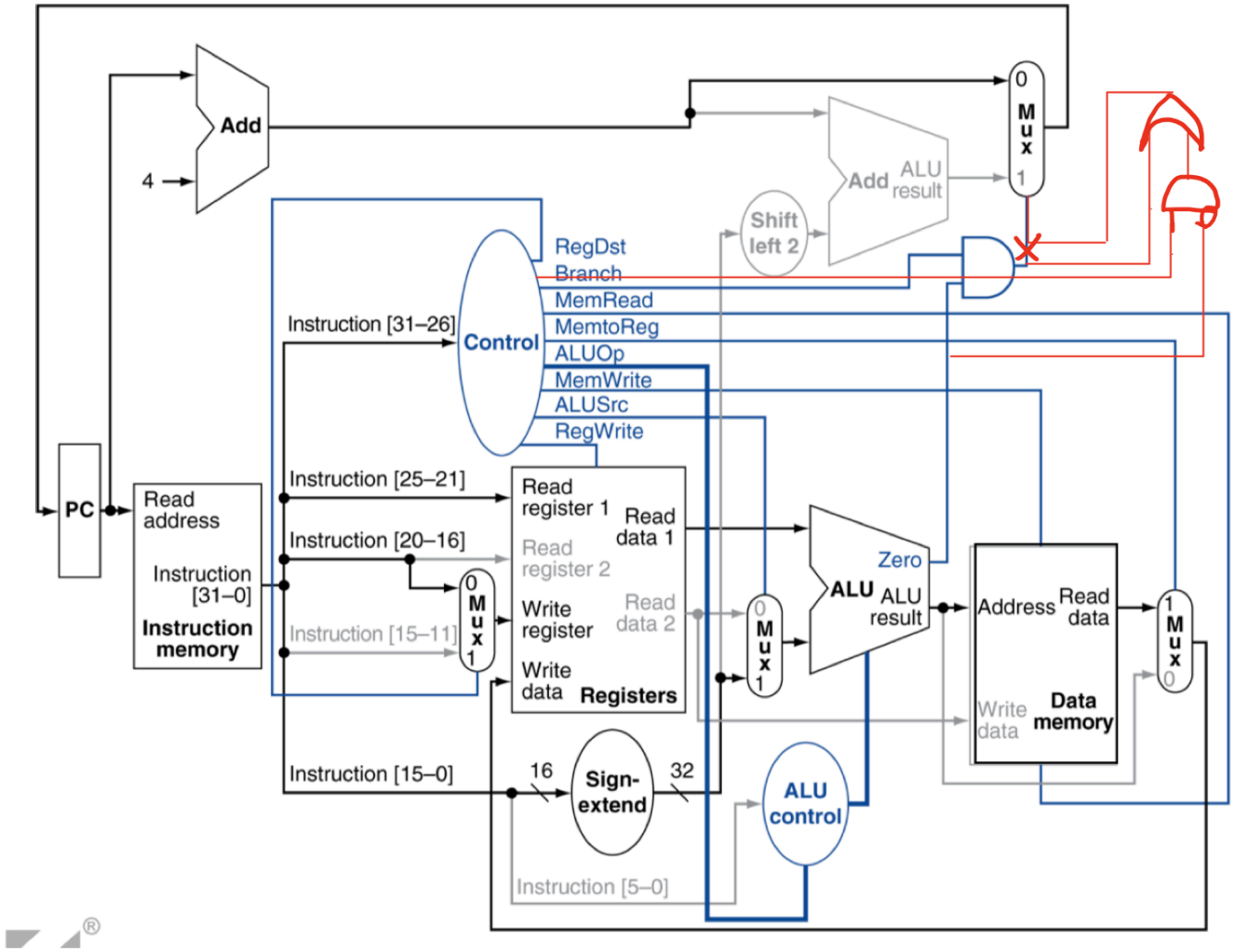
4.11.3

New PC is PC+4

opcode	Operation	ALUOp	funct	ALU Control	ALU function
lw	load word	00	XXXXXX	0010	add
sw	store word				
beq	branch equal	01	XXXXXX	0110	subtract
R-type	add	10	100000	0010	add
	subtract		100010	0110	subtract
	AND		100100	0000	AND
	OR		100101	0001	OR
	set-on-less-than		101010	0111	set-on-less-than

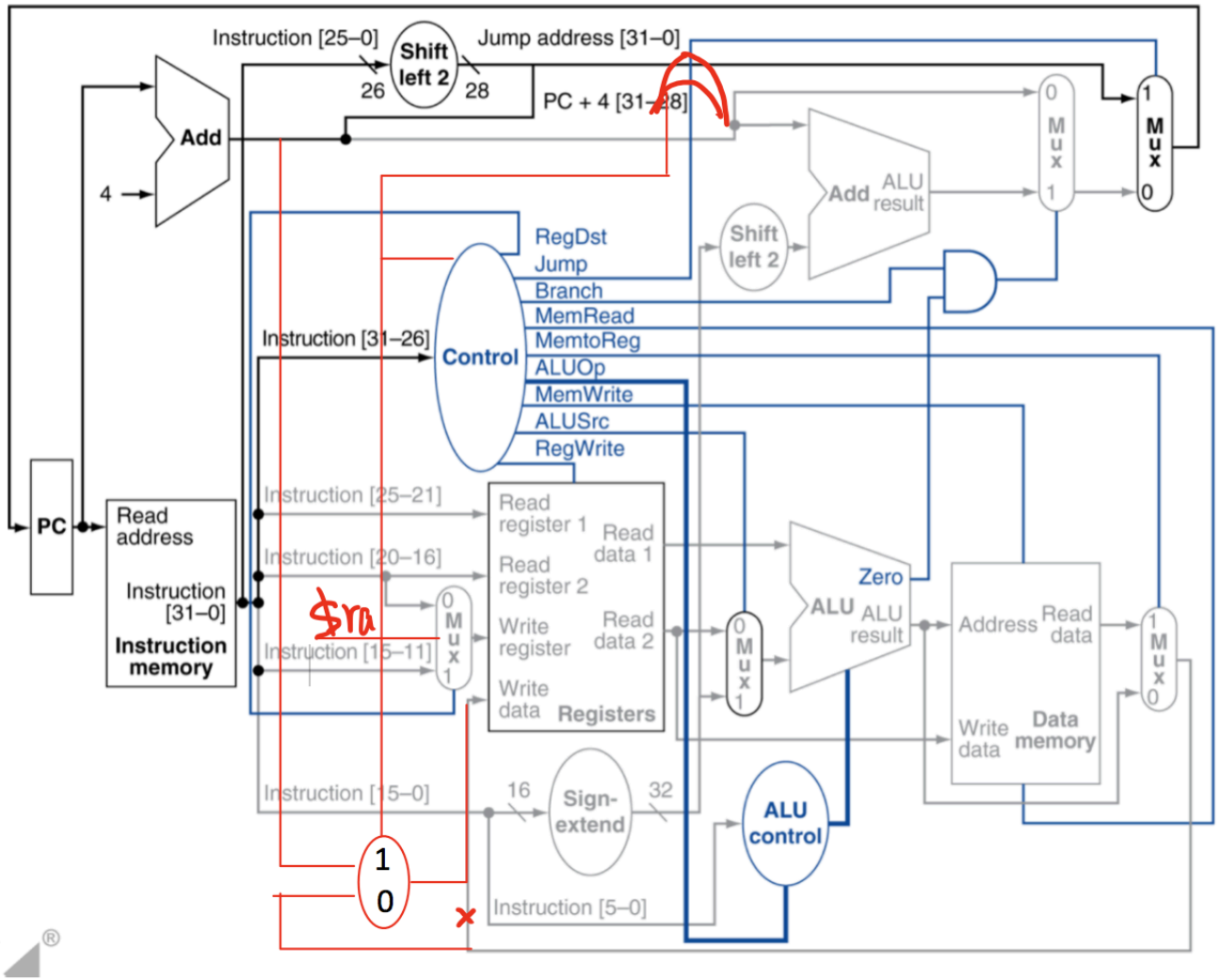
12.

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.