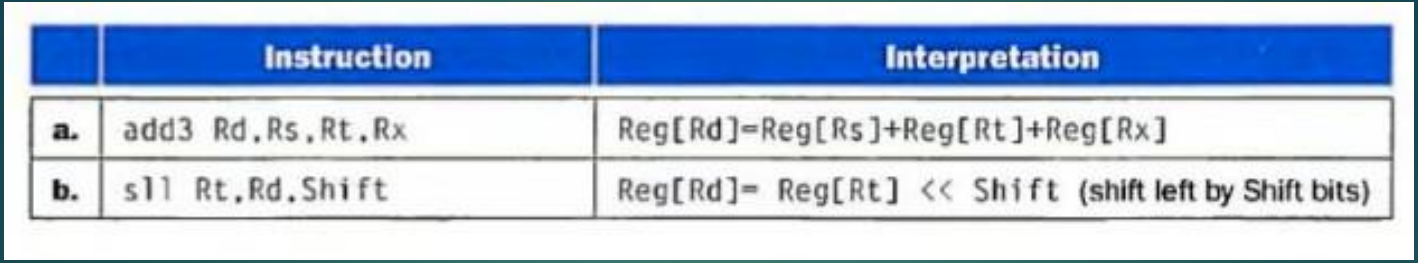
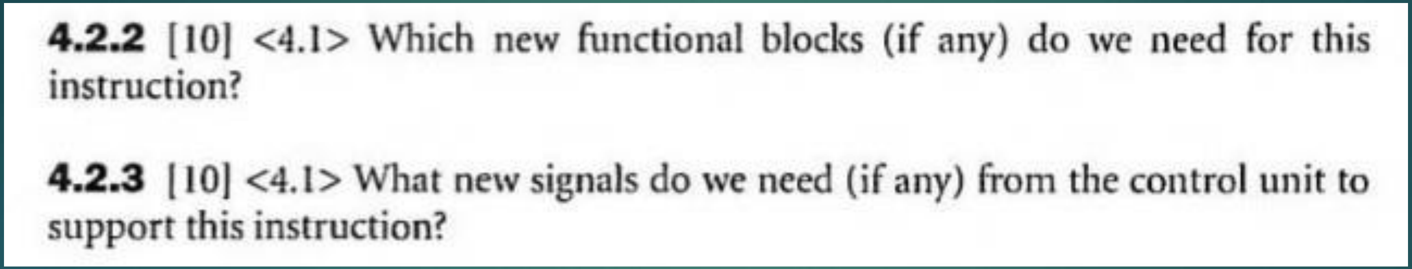
VE370 RC3

HW3 & HW4 SOLUTION

4.2.2



	Instruction	Interpretation
a.	add3 Rd,Rs,Rt,Rx	Reg[Rd]=Reg[Rs]+Reg[Rt]+Reg[Rx]
b.	sll Rt,Rd,Shift	Reg[Rd]= Reg[Rt] << Shift (shift left by Shift bits)

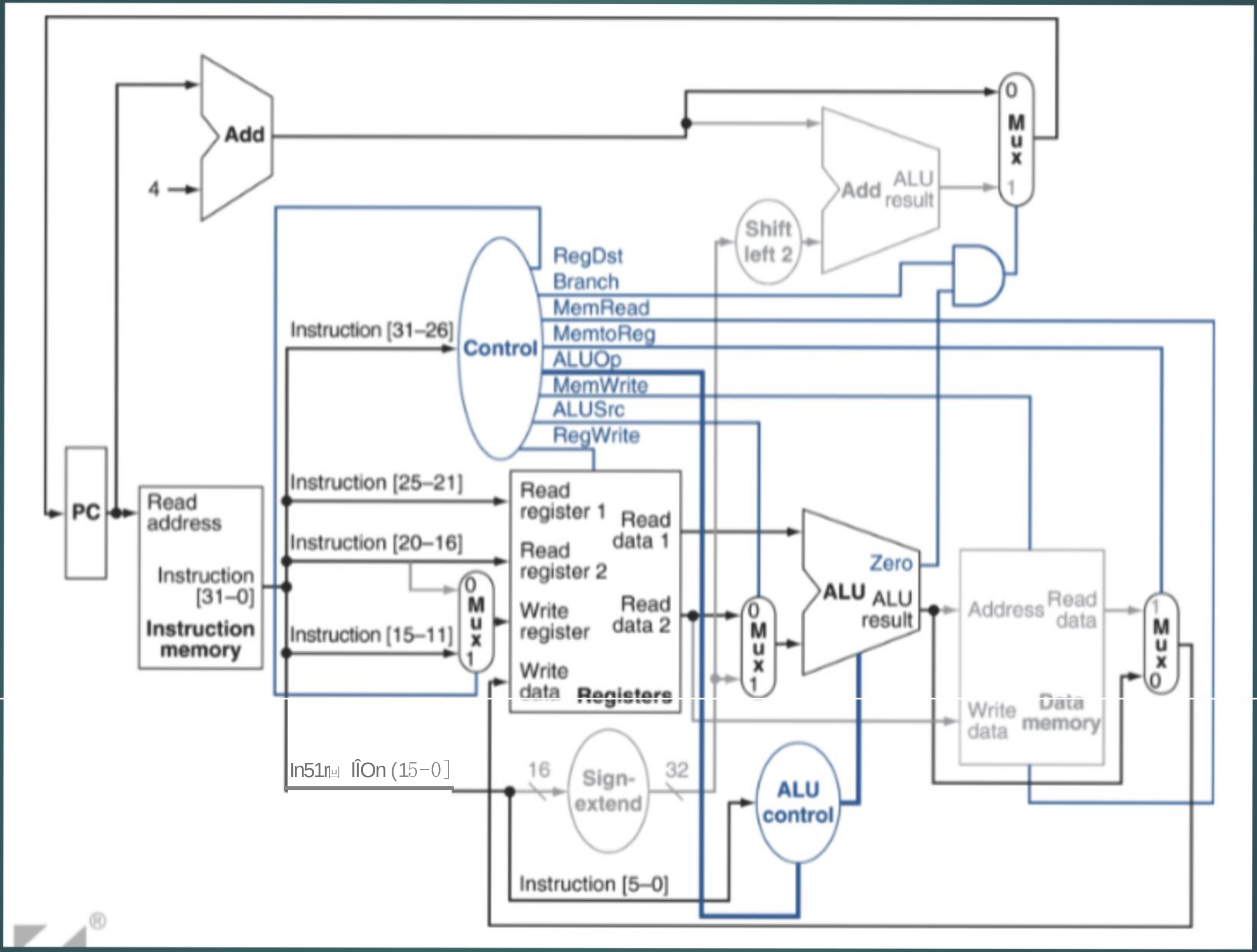


4.2.2 [10] <4.1> Which new functional blocks (if any) do we need for this instruction?

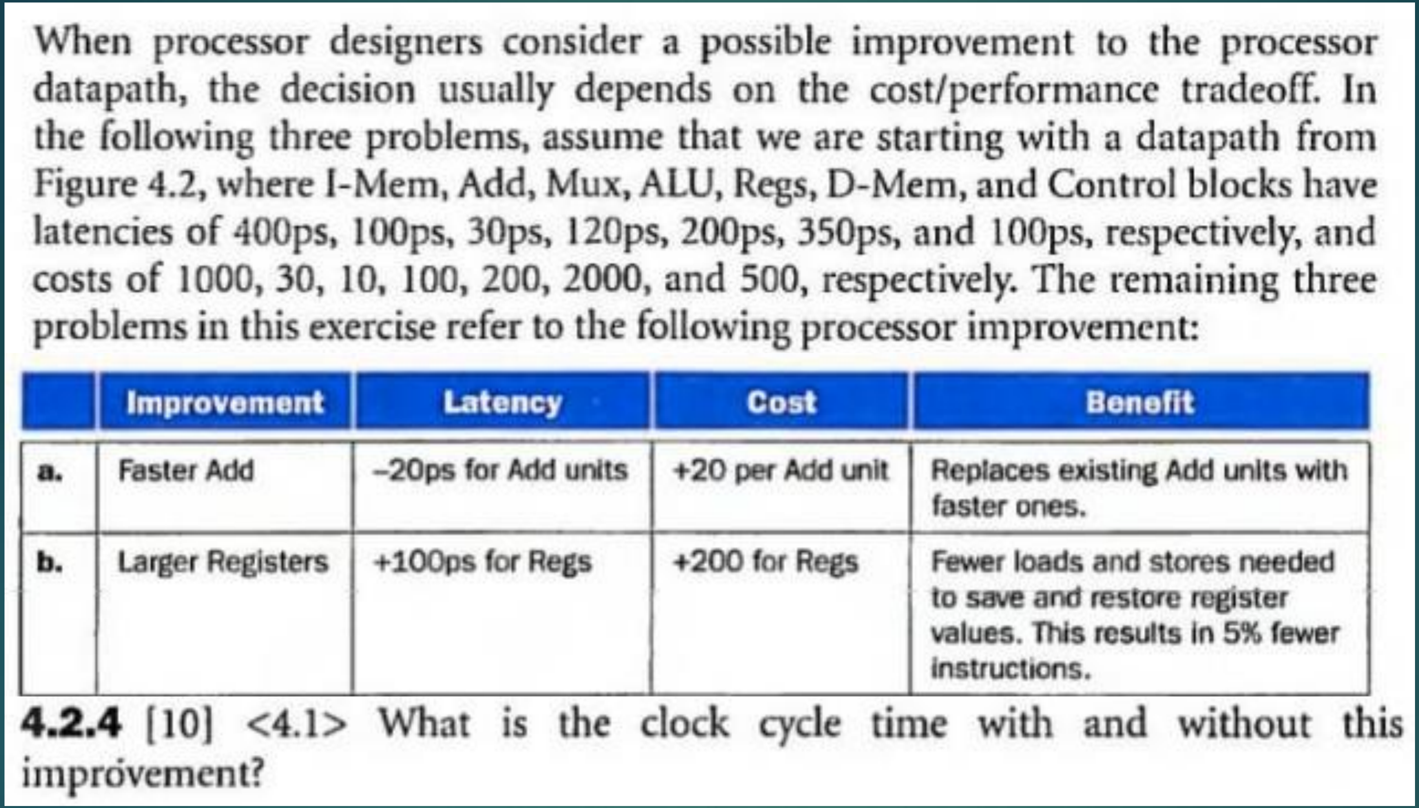
4.2.3 [10] <4.1> What new signals do we need (if any) from the control unit to support this instruction?

2.a) need a ALU and Register that can take three inputs, or another two-input ALU
  b) need to extend the ALU such that the shift operation can be performed

2.a) need a signal that tell the ALU whether to perform operation for 2 or 3 inputs, or the signal that tells the second ALU to work
  b) need a signal to tell ALU to perform shift operation


Add
4
Add
ALU result
Shift left 2
M u x
0
1
RegDst
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Control
Instruction [31–26]
PC
Read address
Instruction [31–0]
Instruction memory
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
Zero
ALU
ALU result
Address
Read data
Write data
Data memory
16
Sign-extend
32
ALU control
Instruction [5–0]
In51r回 lÎOn (15−0]

4.2.4



When processor designers consider a possible improvement to the processor datapath, the decision usually depends on the cost/performance tradeoff. 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:

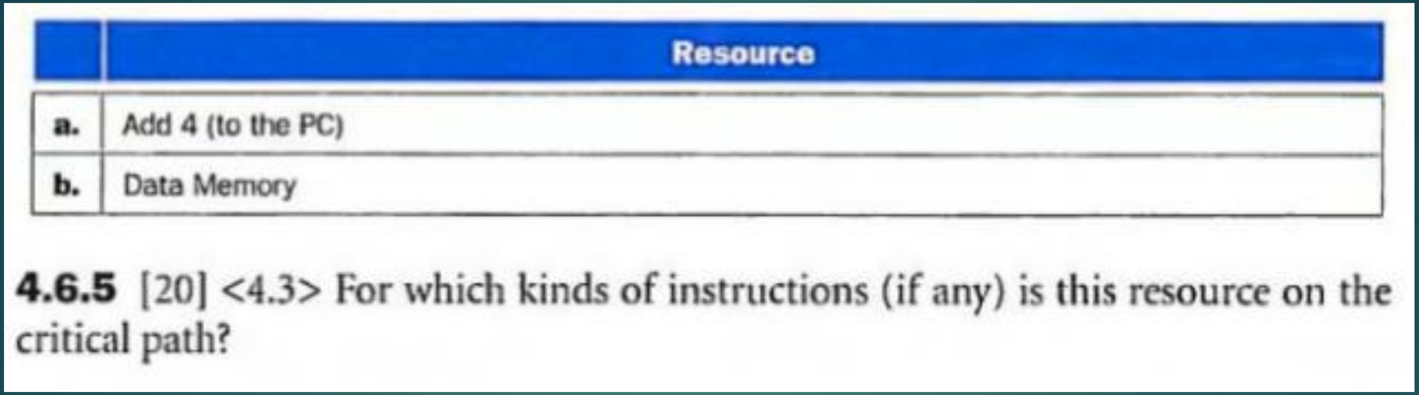
	Improvement	Latency	Cost	Benefit
a.	Faster Add	–20ps for Add units	+20 per Add unit	Replaces existing Add units with faster ones.
b.	Larger Registers	+100ps for Regs	+200 for Regs	Fewer loads and stores needed to save and restore register values. This results in 5% fewer instructions.

4.2.4 [10] <4.1> What is the clock cycle time with and without this improvement?

Longest path: I-mem + Reg + ALU + Mux + D-mem + Mux + Reg= 1330

a) No change. (add not in the longest path)
b) Reg + 100 => total +100 = 1430

4.6.5



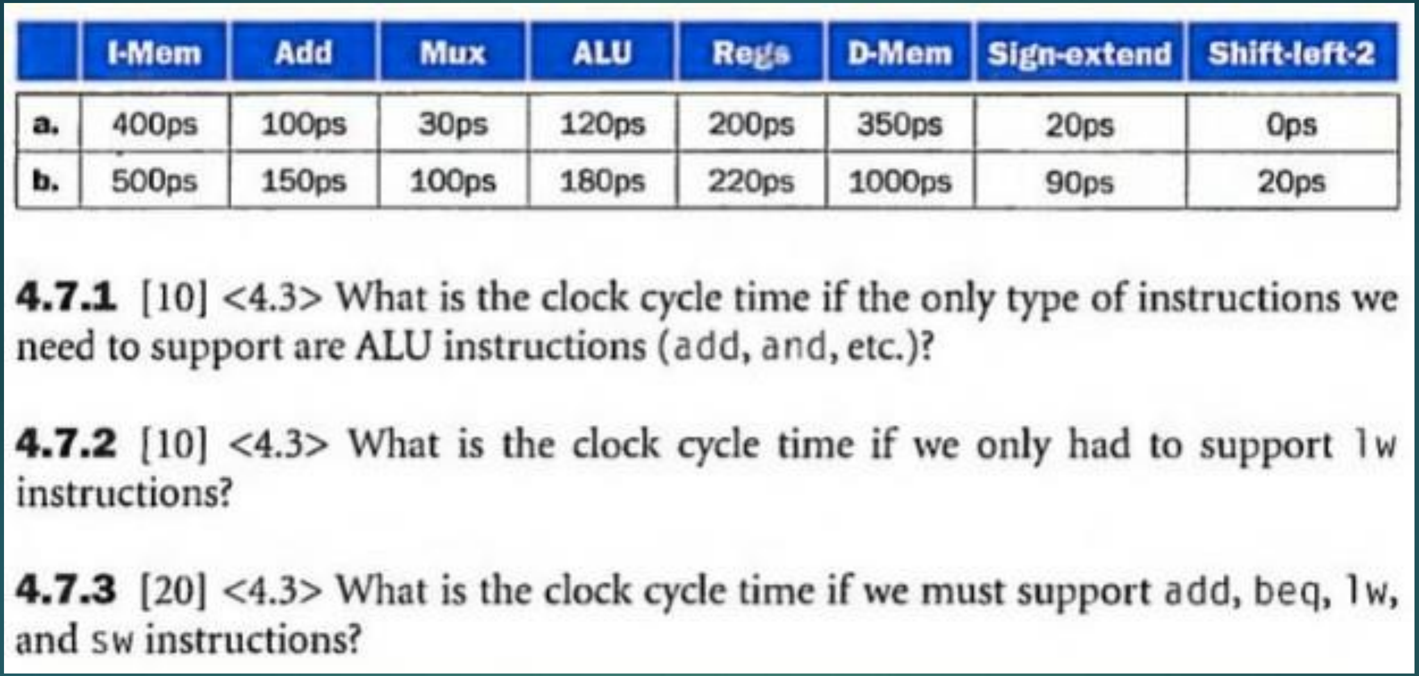
	Resource
a.	Add 4 (to the PC)
b.	Data Memory

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

a) No. Considering the Instruction Memory. For all kinds of instructions, reading from the instruction memory is needed, and it will take longer time

b) Load(lw) and store(sw)

4.7.1~3



	I-Mem	Add	Mux	ALU	Regs	D-Mem	Sign-extend	Shift-left-2
a.	400ps	100ps	30ps	120ps	200ps	350ps	20ps	0ps
b.	500ps	150ps	100ps	180ps	220ps	1000ps	90ps	20ps

4.7.1 [10] <4.3> What is the clock cycle time if the only type of instructions we need to support are ALU instructions (add, and, etc.)?

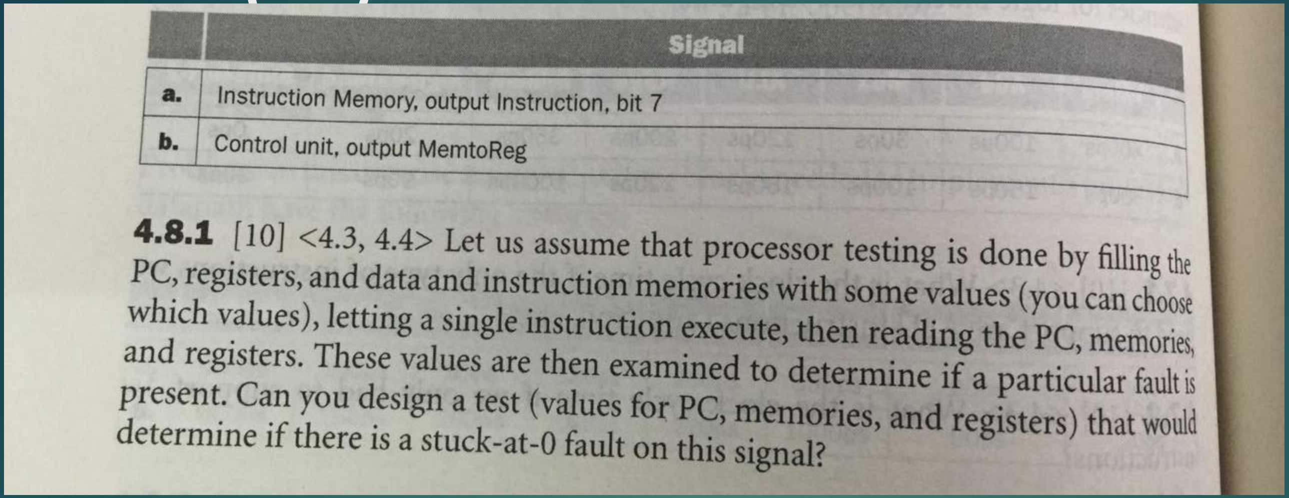
4.7.2 [10] <4.3> What is the clock cycle time if we only had to support `lw` instructions?

4.7.3 [20] <4.3> What is the clock cycle time if we must support `add`, `beq`, `lw`, and `sw` instructions?

1. longest path: I-mem, Reg, Mux, ALU, Mux, Reg
   a) 400 + 200 + 30 + 120 + 30 + 200= 980
   b) 500 + 220 + 100 + 180 + 100 + 220 = 1320
2. longest path: I-mem, Reg, Mux, ALU, D-mem, Mux, Reg
   a) 980 + 350 = 1330
   b) 1320 + 1000 = 2320

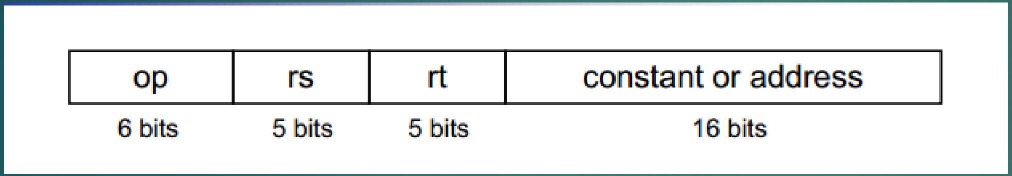
4.7.3 same as 4.7.2, as the path having the longest delay will be that of lw

4.8.1(a)



	Signal
a.	Instruction Memory, output Instruction, bit 7
b.	Control unit, output MemtoReg

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?

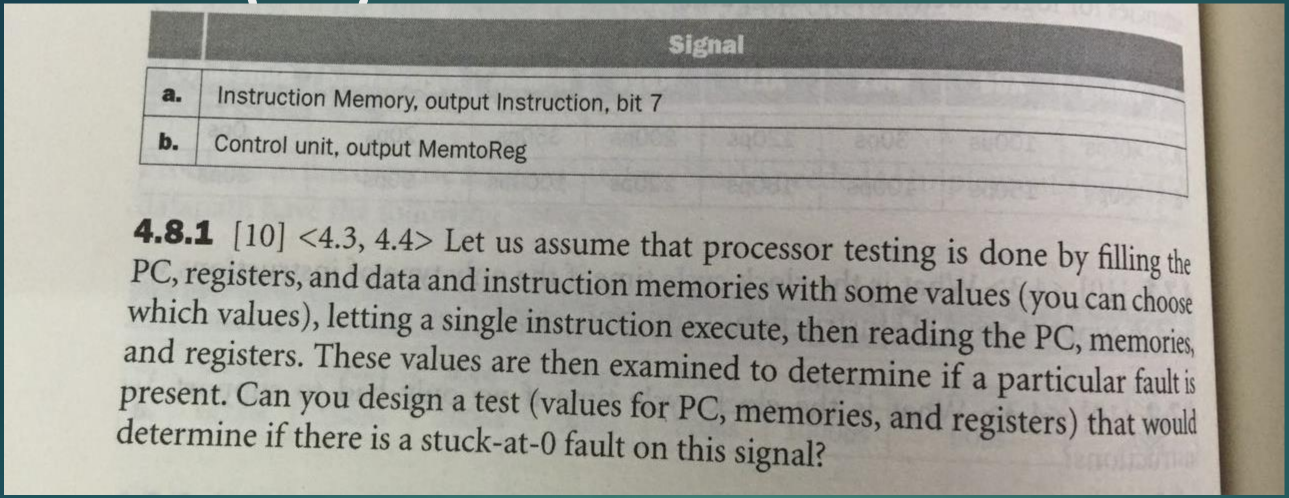


Consider in what type of instructions can we test this signal

In the I-type, we can conveniently test this bit by writing an immediate value to a register

Set the value of registers to be 0, and use instructions like
addi $t0, $t0, 128
(128 is 10000000)
If the value of $t0 is still 0, the there is a stuck-at-0 fault

4.8.1(b)

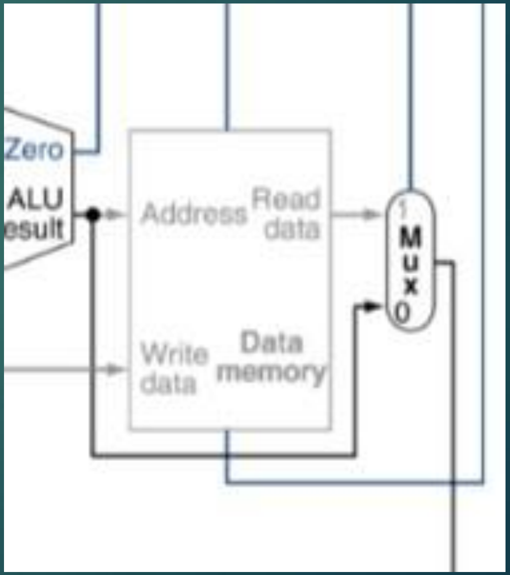


	Signal
a.	Instruction Memory, output Instruction, bit 7
b.	Control unit, output MemtoReg

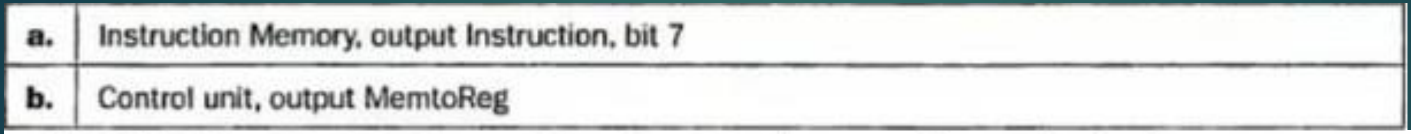
4.8.1 [10] <4.3, 4.4> Let us assume that processor testing is done by filling the PC, registers, and data and instruction memories with some values (you can choose which values), letting a single instruction execute, then reading the PC, memories, and registers. These values are then examined to determine 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 set MemtoReg = 1 is lw instruction.

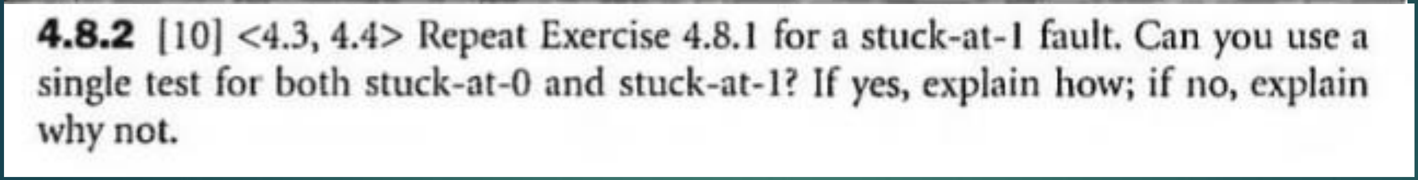
So we first fill the D-mem with a constant value (e.g 1) then set a register to anthoer value (e.g s0 = 5). The excecute the instruction load a value from D-mem to the register (e.g lw $s0 0($t0)). If the value of s0 is not 1then it means we have the stuck-at-0 fault.



4.8.2



a.	Instruction Memory, output Instruction, bit 7
b.	Control unit, output MemtoReg



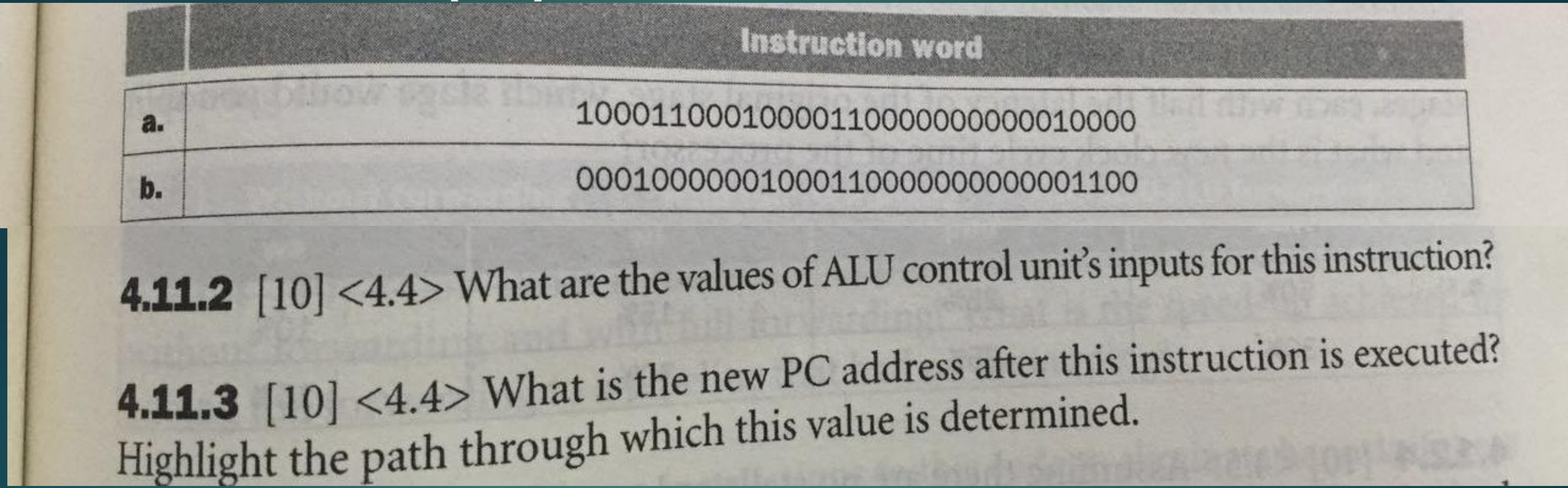
4.8.2 [10] <4.3, 4.4> Repeat Exercise 4.8.1 for a stuck-at-1 fault. Can you use a single test for both stuck-at-0 and stuck-at-1? If yes, explain how; if no, explain why not.

(a) Set $t0 as 128, use addi $t0, zero, 0. If the register is still 128, there is a stuck-at-1 fault

(b) Test in this case not reliable.
Instructions that have a 0-value MemtoReg:
ReadMem also being 0: might be noise at the output of the data memory, and it may be the same as the original value in the register

Can not perfrom the test for both error at the same time using a single instruction. The signal cannot be 0 and 1 at the same time

4.11.2-3(a)



	Instruction word
a.	10001100010000110000000000010000
b.	00010000001000110000000000001100

4.11.2 [10] <4.4> What are the values of 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.

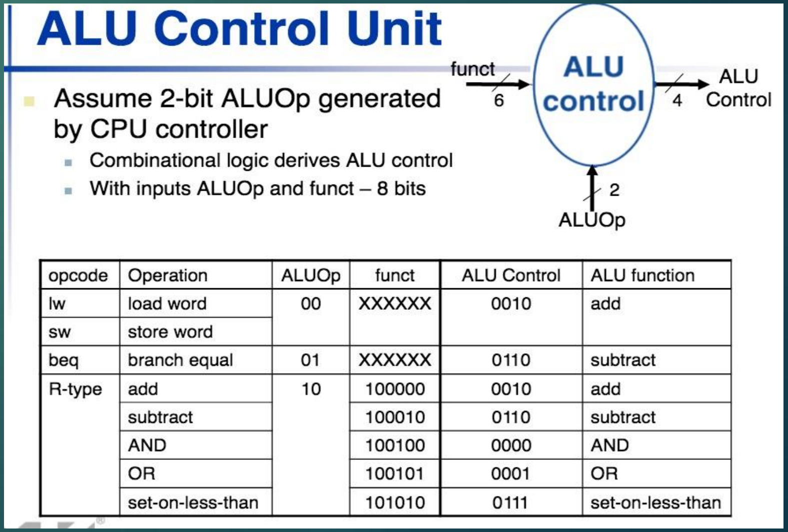
lw: ALUOp should be 00

4.11.2 (a)

ALUOp	Ins[5-0]
00	XXXXXX

4.11.3 (a)

The new PC should be PC+4



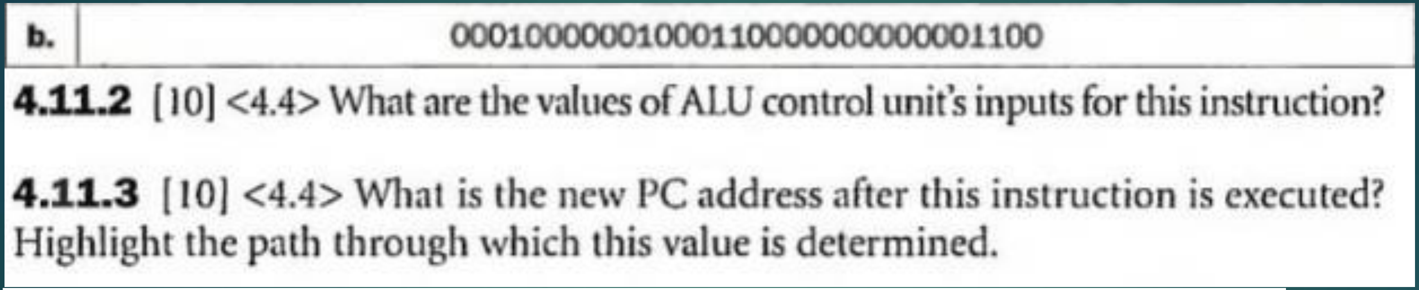
ALU Control Unit

- Assume 2-bit ALUOp generated by CPU controller
  - Combinational logic derives ALU control
  - With inputs ALUOp and funct – 8 bits

funct
6
ALU control
4
ALU Control
2
ALUOp

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

4.11.2-3(b)



b.	00010000001000110000000000001100

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

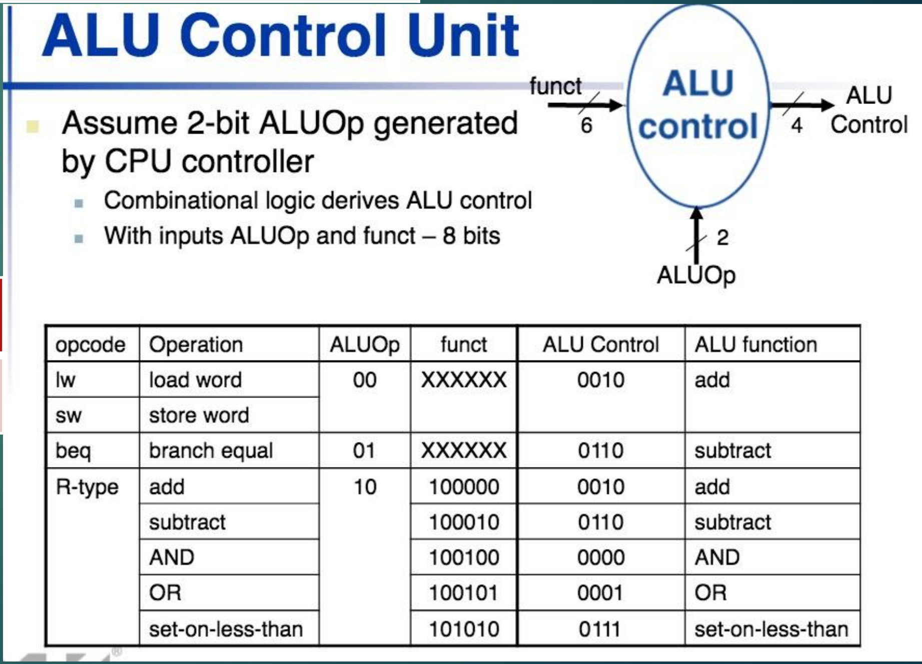
4.11.3 [10] <4.4> What is the new PC address after this instruction is executed? Highlight the path through which this value is determined.

beq	1	3	12
000100	00001	00011	0000000000001100

4.11.2 (b)

ALUOp	Ins[5-0]
01	001100

4.11.3 (b)
If $1=$3, the new PC is PC+4+12*4

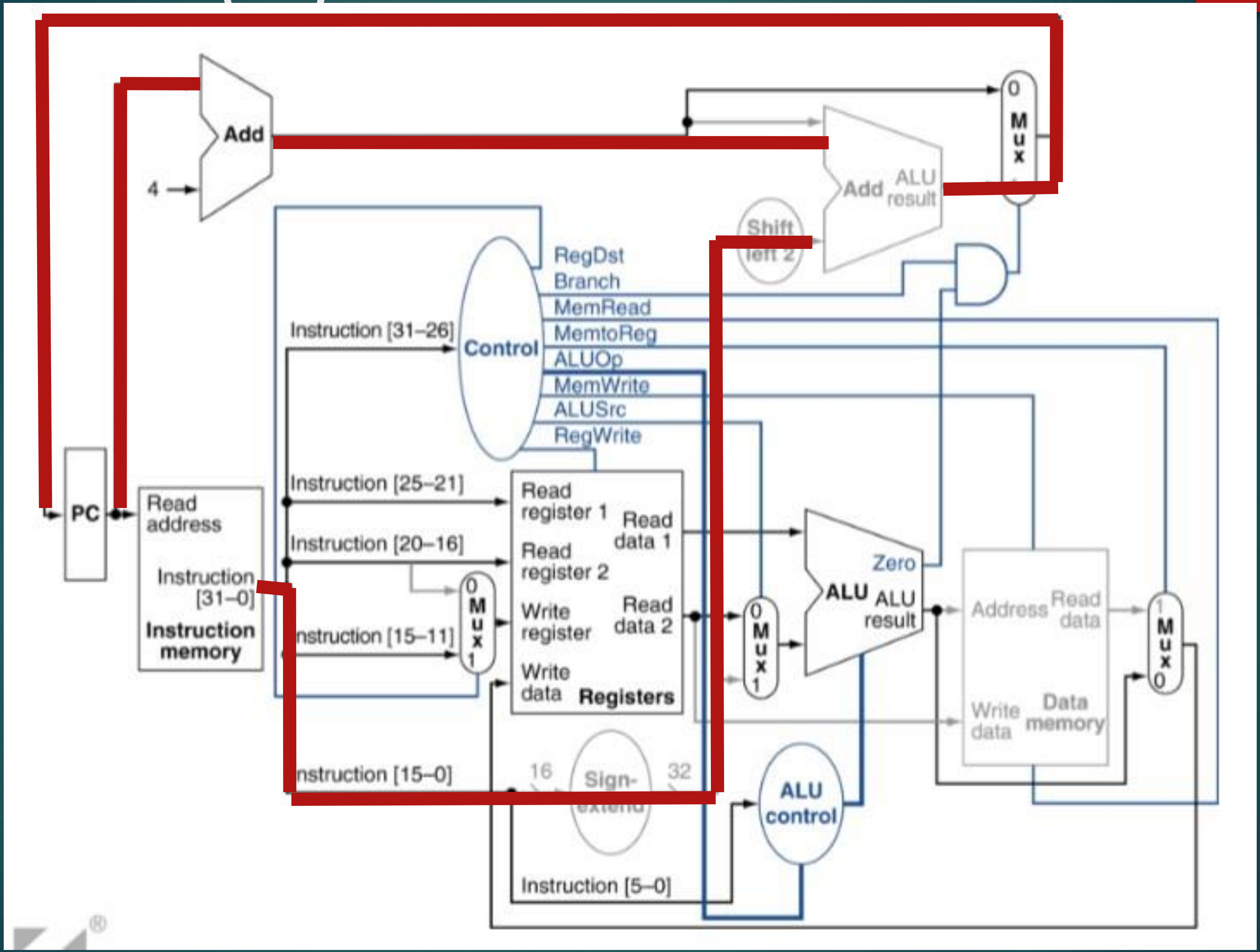


ALU Control Unit

- Assume 2-bit ALUOp generated by CPU controller
  - Combinational logic derives ALU control
  - With inputs ALUOp and funct – 8 bits

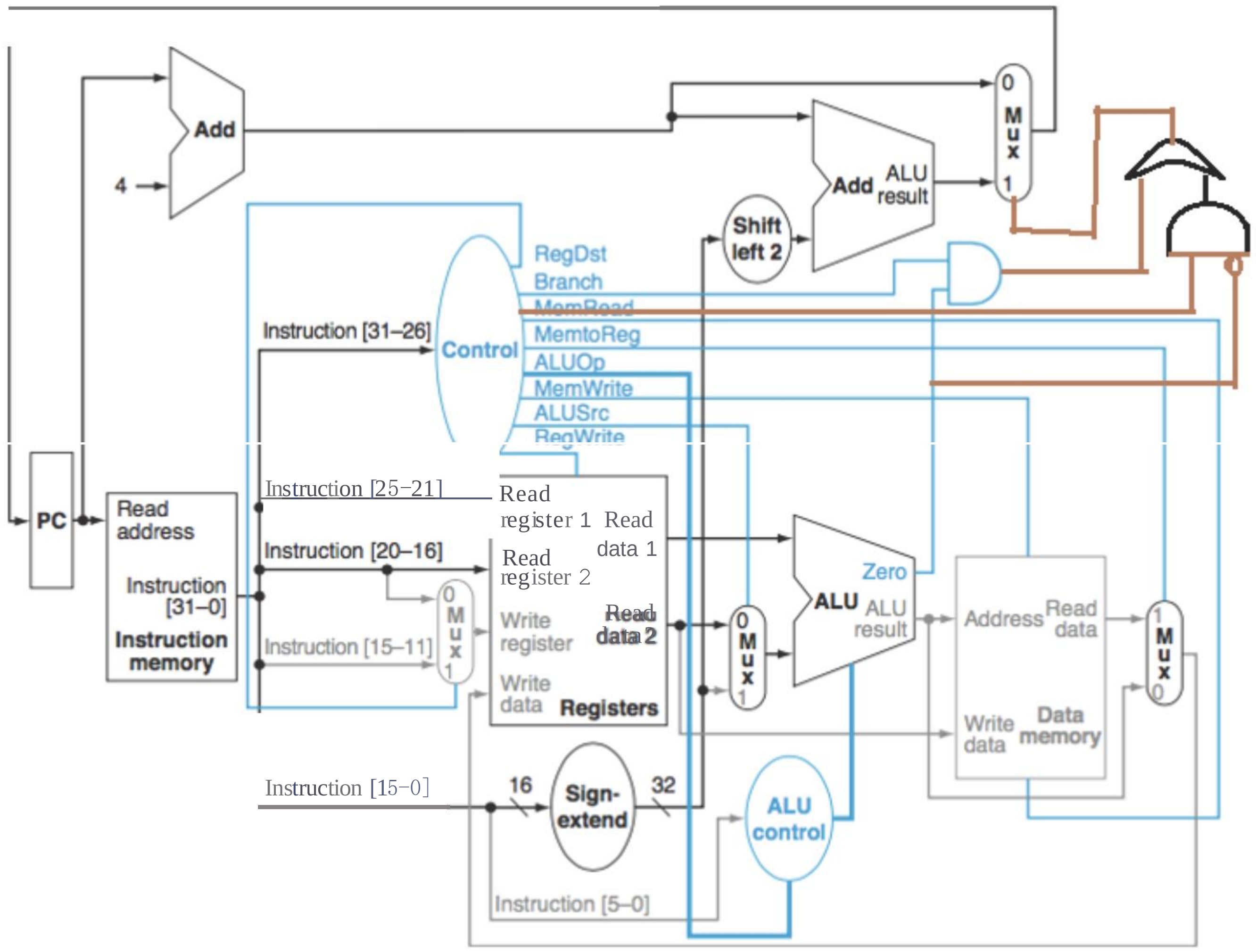
opcode	Operation	ALUOp	funct	ALU Control	ALU function
lw	load word	00	XXXXXX	0010	add
sw	store word				
beq	branch equal	01	XXXXXX	0110	subtract
R-type	add	10	100000	0010	add
	subtract		100010	0110	subtract
	AND		100100	0000	AND
	OR		100101	0001	OR
	set-on-less-than		101010	0111	set-on-less-than

4.11.3(b)



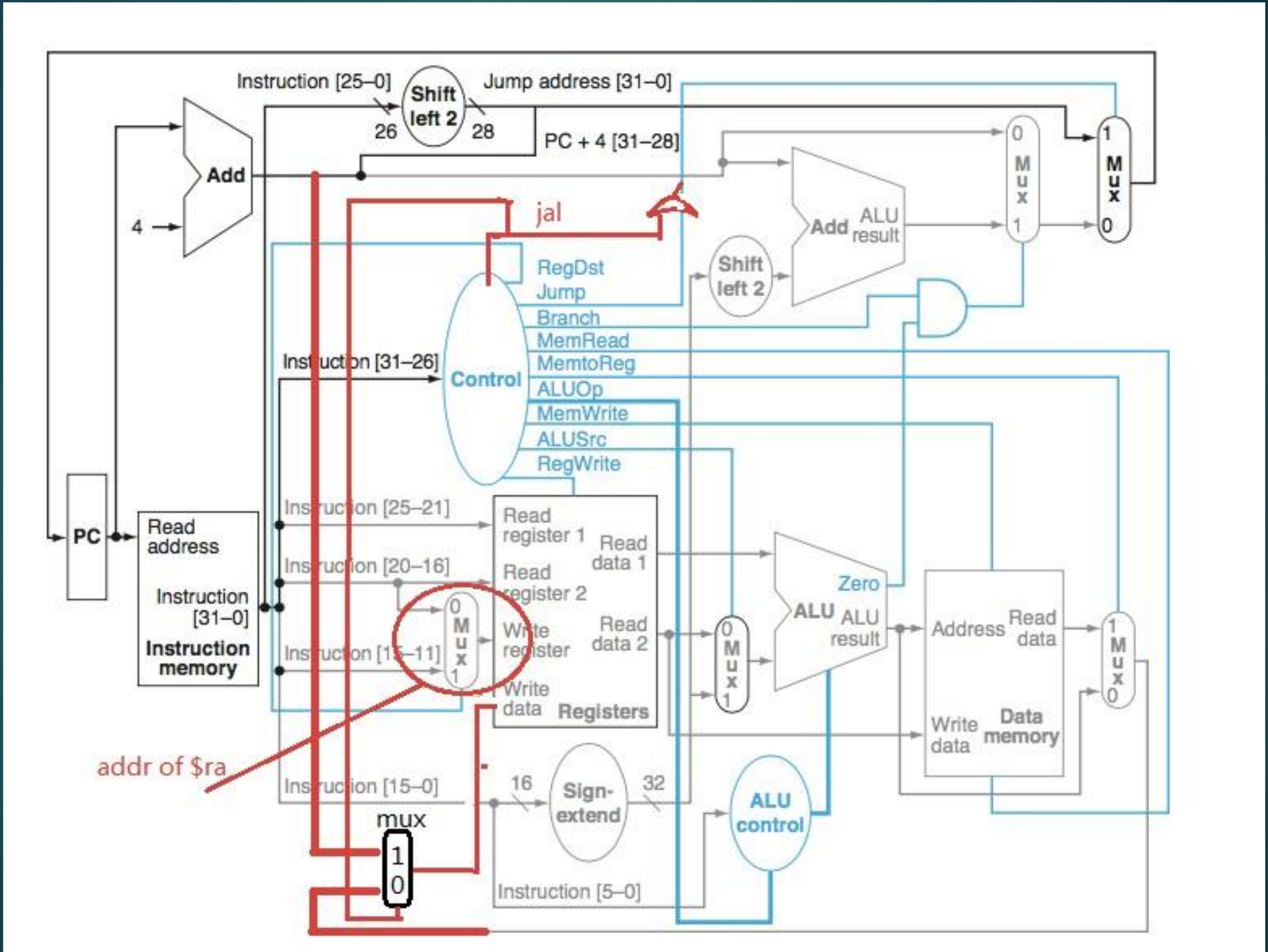
Add
4
Add
ALU result
Shift left 2
0 Mux 1
PC
Read address
Instruction [31–0]
Instruction memory
Instruction [31–26]
Control
RegDst
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Instruction [15–0]
Instruction [5–0]
0 Mux 1
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
16
Sign-extend
32
0 Mux 1
Zero
ALU
ALU result
ALU control
Address
Read data
Write data
Data memory
1 Mux 0

12. Modify MIPS single cycle implementation based on Figure 4.24 to support the instruction BNE.+J

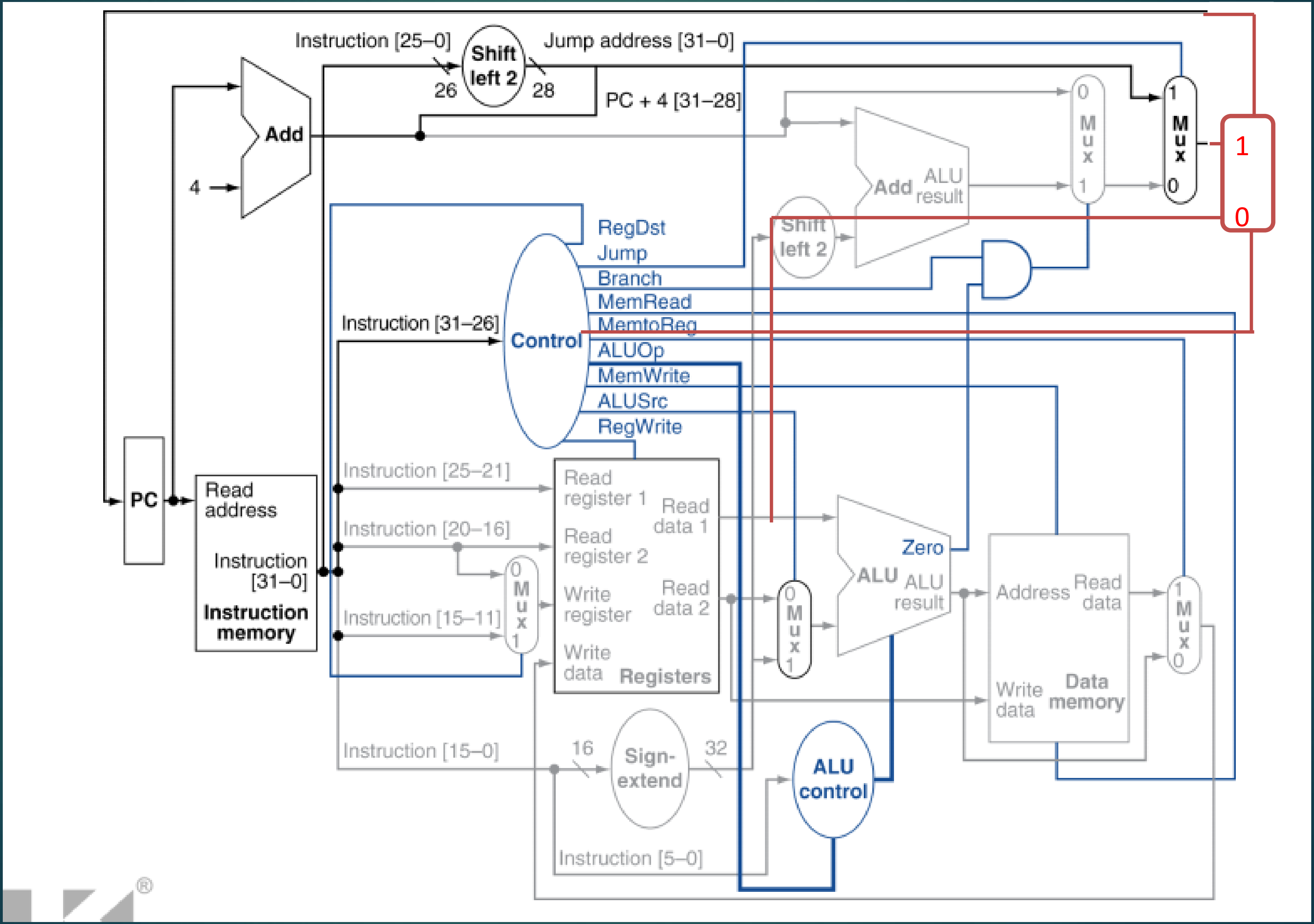


F1GURE 4.21 The . . 缉 path 1ft o......tlon for • br8ft份。仲 • 刷. . lnatructlon. The ∞ntrollines, datapa山 山 巾 and connections that are active are h剧 院 。 After using 由e register 缸e and ALU to perform 由e ∞mpare, 由Zero output is used to sel剧 由 e n 创 program ∞unter from between 由e t柄。candidates.

Support Jal


Instruction [25–0]
Shift left 2
26
28
Jump address [31–0]
PC + 4 [31–28]
Add
4
jal
Add ALU result
Shift left 2
0 Mux 1
1 Mux 0
RegDst
Jump
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Control
Instruction [31–26]
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Instruction [15–0]
Instruction [5–0]
PC
Read address
Instruction [31–0]
Instruction memory
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
0 Mux 1
16
Sign-extend
32
Zero
ALU
ALU result
ALU control
Address
Read data
Write data
Data memory
1 Mux 0
addr of $ra
mux
1
0

Support jr



Instruction [25–0]
Shift left 2
26
28
Jump address [31–0]
PC + 4 [31–28]
Add
4
ALU
Add result
Shift left 2
M u x
0
1
1
0
RegDst
Jump
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Instruction [31–26]
Control
PC
Read address
Instruction [31–0]
Instruction memory
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
Zero
ALU
ALU result
Address
Read data
Write data
Data memory
Instruction [15–0]
16
32
Sign-extend
ALU control
Instruction [5–0]