Chapter 4 (Part 1)
Selected Problems

(from the textbook and Exams)

Question 1

We wish to add the instruction `bne` (branch if not equal) to the single-cycle datapath described in this chapter. Add any necessary datapaths and control signals to the single-cycle datapath.

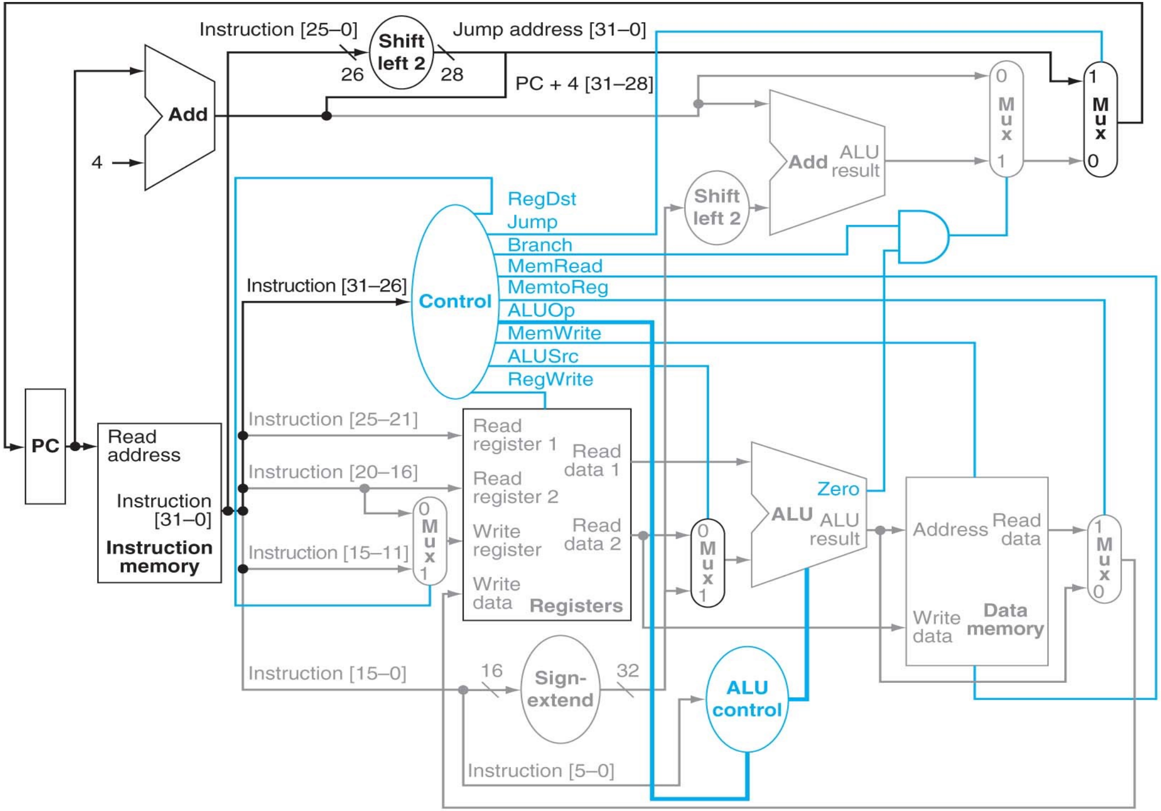
Branch on not equal

```
bne rs, rt, label
```

5	rs	rt	Offset
6	5	5	16

Conditionally branch the number of instructions specified by the offset if register `rs` is not equal to `rt`.

Single Cycle Datapath



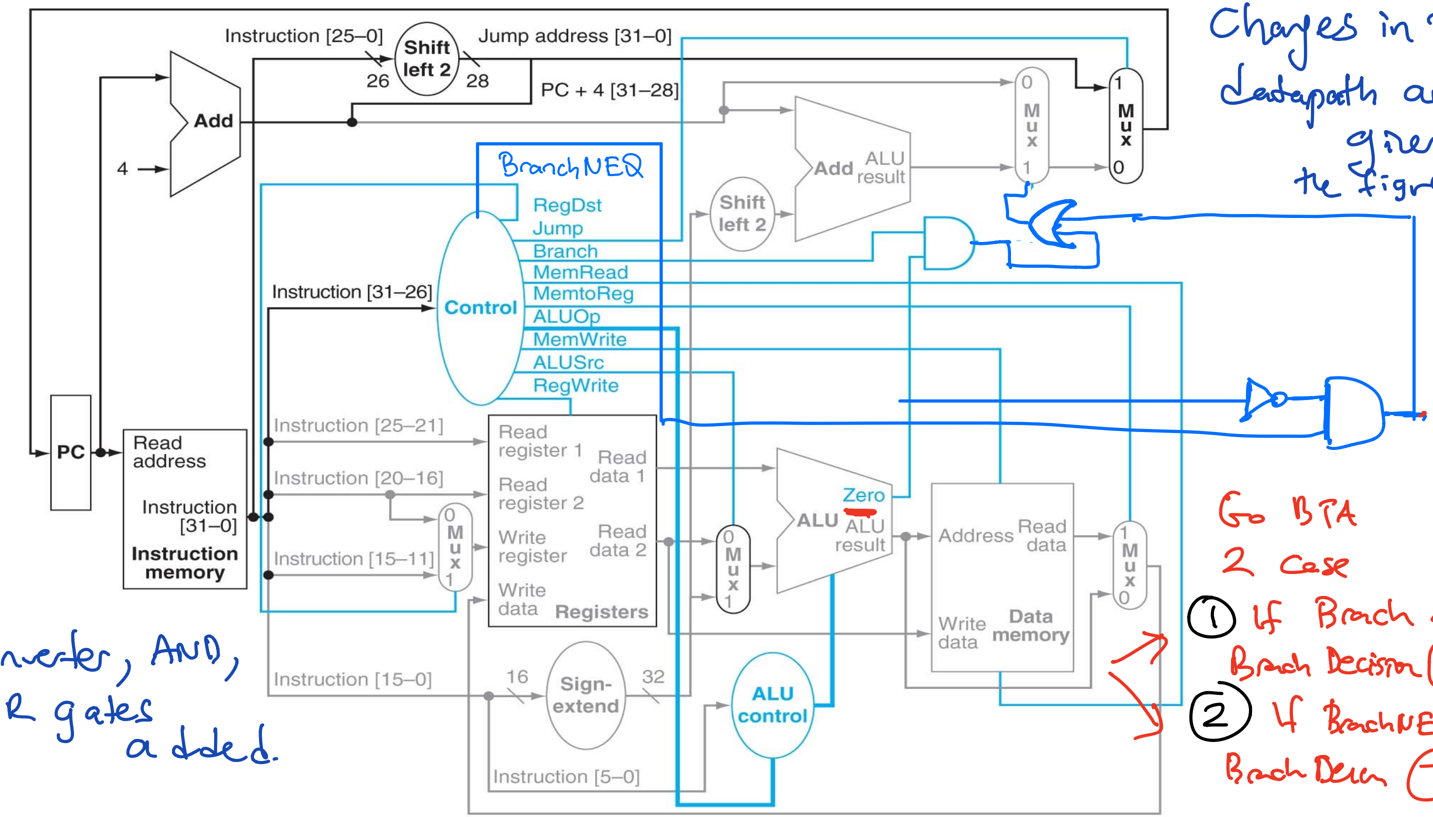
Answer

- Put a new AND gate which has two inputs (inverted Zero from ALU and BranchNEQ signal from control unit).
- Connect the output of new AND gate to a new OR gate. The second input of the OR gate is the output of the original AND gate in the datapath which is right after the ALU.
- Output of the OR gate is the control signal of the mux (which was controlled with the AND gate before).

Single Cycle Datapath

Answer

Changes in the datapath are given in the figure



Go BTA

2 case

① If Branch & Branch Decision (T)

② If BranchNEQ Branch Decision (T)

- Inverter, AND, OR gates added.

Question 2. (past exam question)

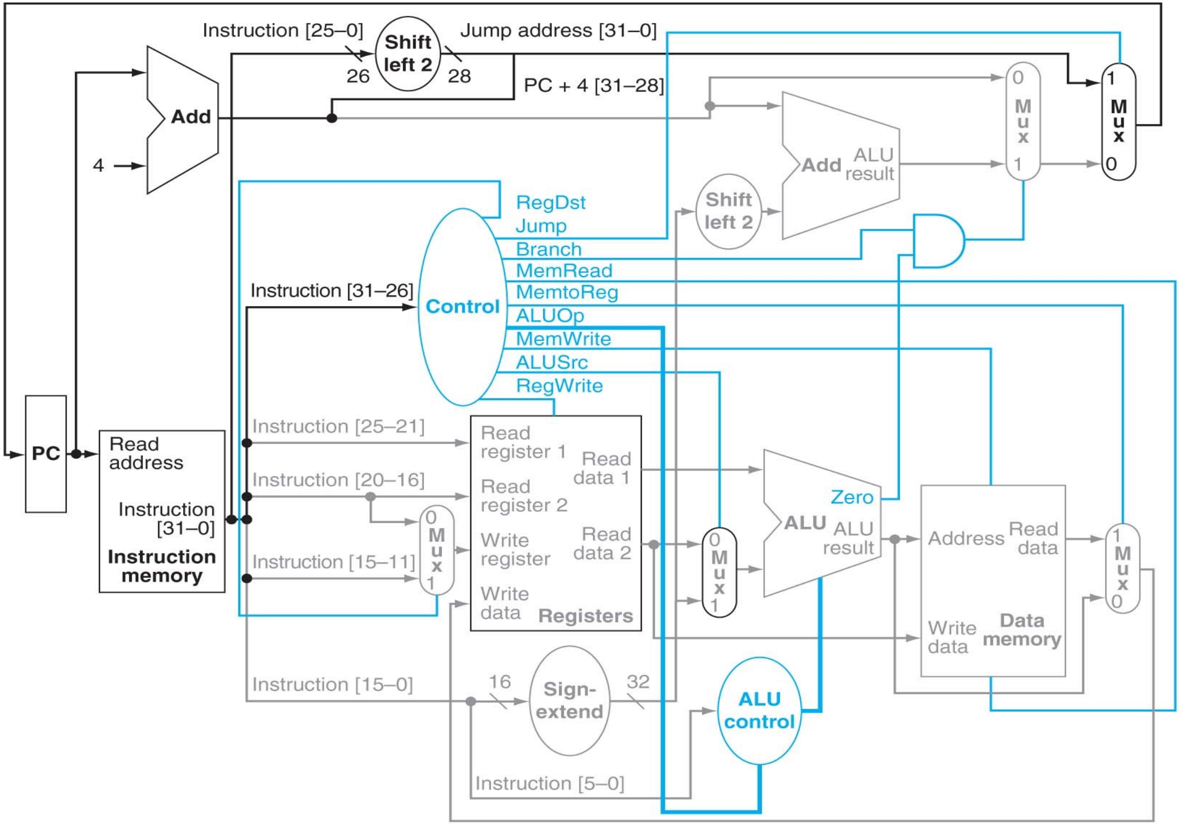
Assume that we extend our single-cycle MIPS implementation so that it handles the “ldnew” instruction, where “ldnew” has R-type instruction format. An example is given below:

Example: ldnew $s1, $s2, $s3 # $s1 = Memory [$s2 + $s3]

# It sums two registers to obtain the effective load address.

a. **Explain** the required changes in the complete single-cycle datapath clearly (including changes in any component of the datapath including ALU, additional wires, muxes and control/selector signals). In case of no change in the datapath, then express it accordingly.

Single Cycle Datapath



Question 2.a

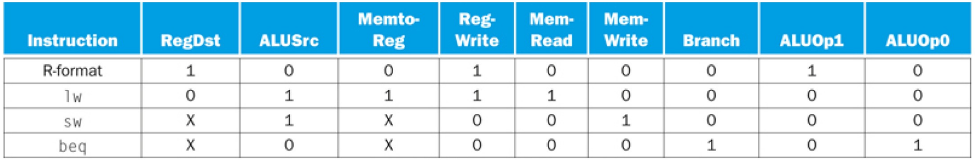
- No change in datapath
- No new components or connections are needed for the datapath.

Question 2.b

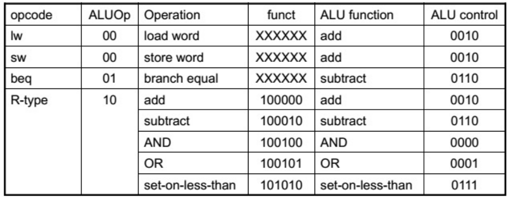
For the ldnew instruction, write the values of control lines given in Figure 4.18 at page 323 in new edition appropriately (i.e., 0, 1 or x for don't care). Include any new control signal(s) if needed.

	RegDst	ALUsrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	ALUop	Jump
ldnew	1	0	1	1	1	0	0	10	0

Figure 4.18



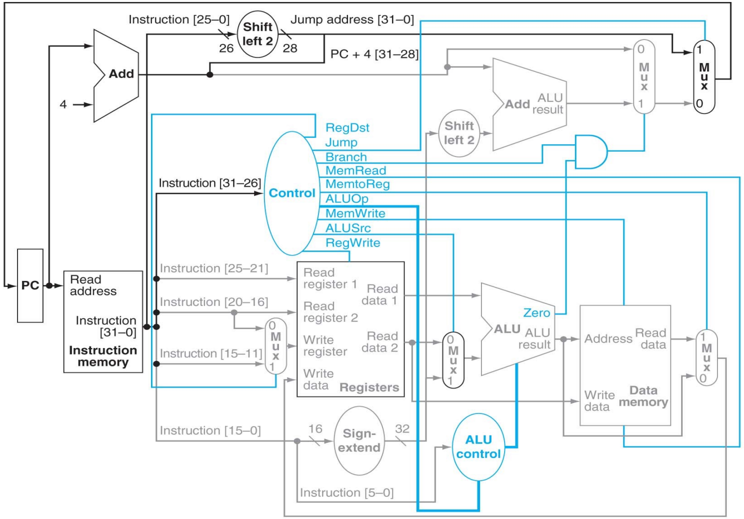
Instruction	RegDst	ALUSrc	Memto-Reg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
R-format	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1



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

Question 3 (Q.4.6.1, 4.6.2, 4.6.3 from 5th Edition)

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 stuckat-1 fault, and the affected signal always has a logical value of 0 or 1, respectively. The following problems refer to bit 0 of the Write Register input on the register file in Figure 4.24.



Question 3 (Q.4.6.1)

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?

Answer:

To test for a stuck-at-0 fault on a wire, we need an instruction that puts that wire to a value of 1 and has a different result if the value on the wire is stuck at zero:

If this signal is stuck at zero, an instruction that writes to an odd-numbered register will end up writing to the even-numbered register. So if we place a value of zero in R30 and a value of 1 in R31, and then execute:

ADD R31, R30, R30

the value of R31 is supposed to be zero. If bit 0 of the Write Register input to the Registers unit is stuck at zero, the value is written to R30 instead and R31 will be 1.

Question 3 (Q.4.6.2)

Repeat 4.6.1 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.

Answer:

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.

The test for stuck-at-1 is analogous to the stuck-at-0 test:

We can place a value of zero in R31 and a value of 1 in R30, then use

ADD R30,R31,R31

which is supposed to place 0 in R30. If this signal is stuck-at-1, the write goes to R31 instead, so the value in R30 remains 1.

Question 3 (Q.4.6.3)

If we know that the processor has a stuck-at-1 fault on this signal, is the processor still usable? To be usable, we must be able to convert any program that executes on a normal MIPS processor into a program that works on this processor. You can assume that there is enough free instruction memory and data memory to let you make the program longer and store additional data. Hint: the processor is usable if every instruction “broken” by this fault can be replaced with a sequence of “working” instructions that achieve the same effect.

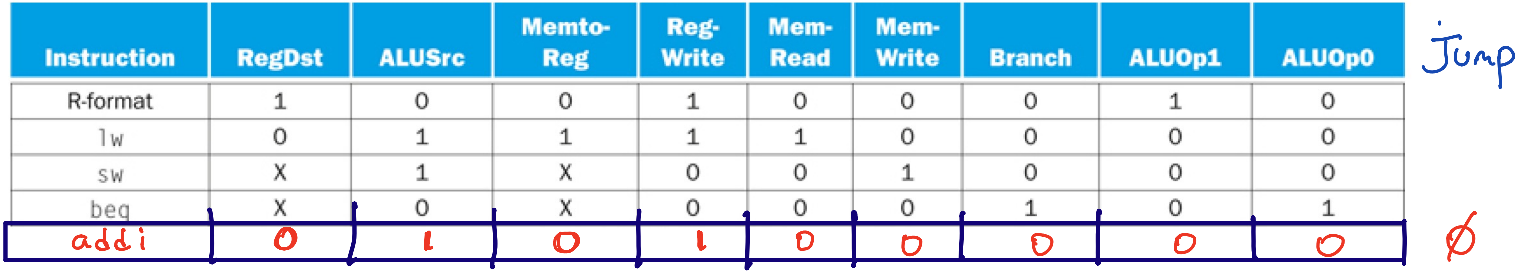
Answer:

We need to rewrite the program to use only odd-numbered registers

Question 4

We wish to add the instruction **addi (add immediate)** to the single-cycle datapath described in the chapter. Add any necessary datapaths and control signals to the single-cycle datapath of the Figure 4.24 which represents the datapath and show the necessary additions (if any) to the Figure 4.18 which represents the setting of the control lines.

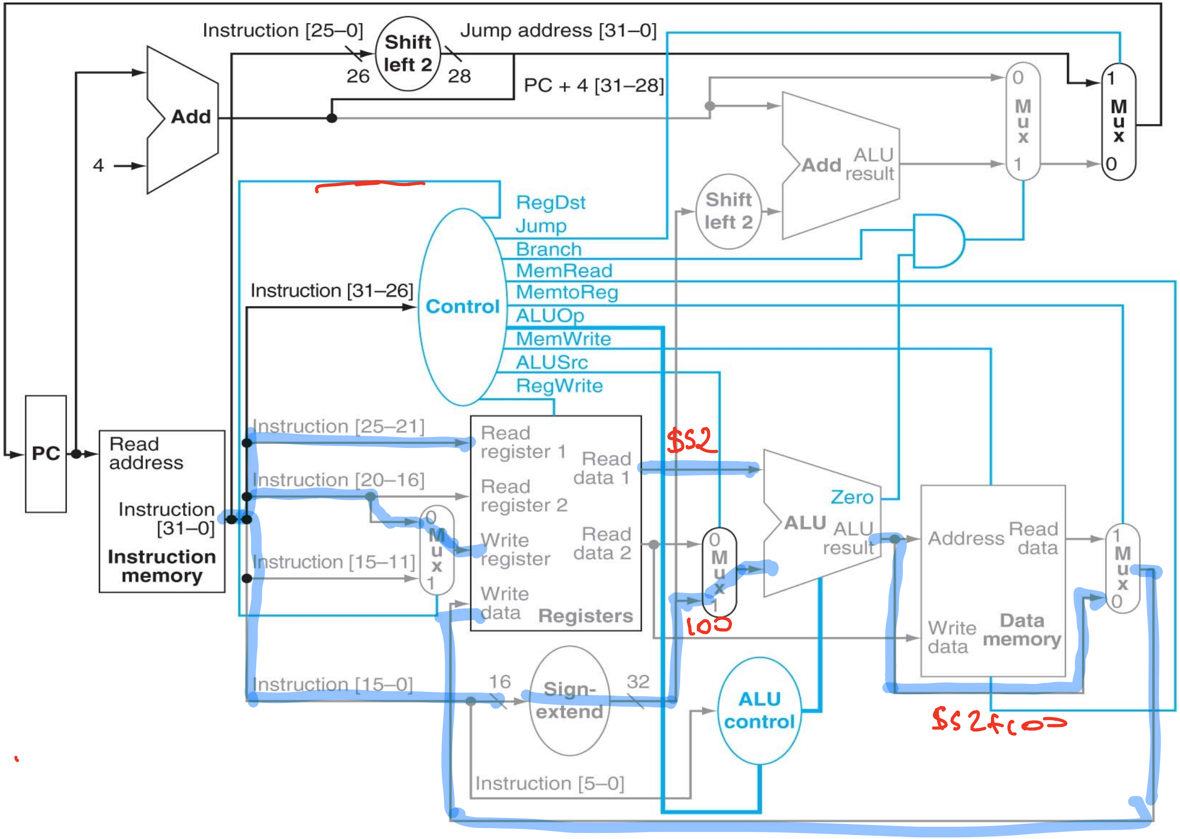
Figure 4.18



Instruction	RegDst	ALUSrc	Memto-Reg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0	Jump
R-format	1	0	0	1	0	0	0	1	0	
lw	0	1	1	1	1	0	0	0	0	
sw	X	1	X	0	0	1	0	0	0	
beq	X	0	X	0	0	0	1	0	1	
addi	0	1	0	1	0	0	0	0	0	Ø

Single Cycle Datapath

addi $s1, $s2, 100



Answer

No datapath chages are required.

Related now for Figure 4.18 is given in previous page.

Question 5

Assume that you are required to extend our single-cycle MIPS implementation so that it handles the “jnew” instruction, where “jnew” has R-type instruction format where rd field is 0. An example is given below:

```
Example:  jnew   $s3, $s4     #  unconditionally jump to the address given
in  Memory[$s3+$s4]
                                        #   PC← Memory[$s3+$s4]
```

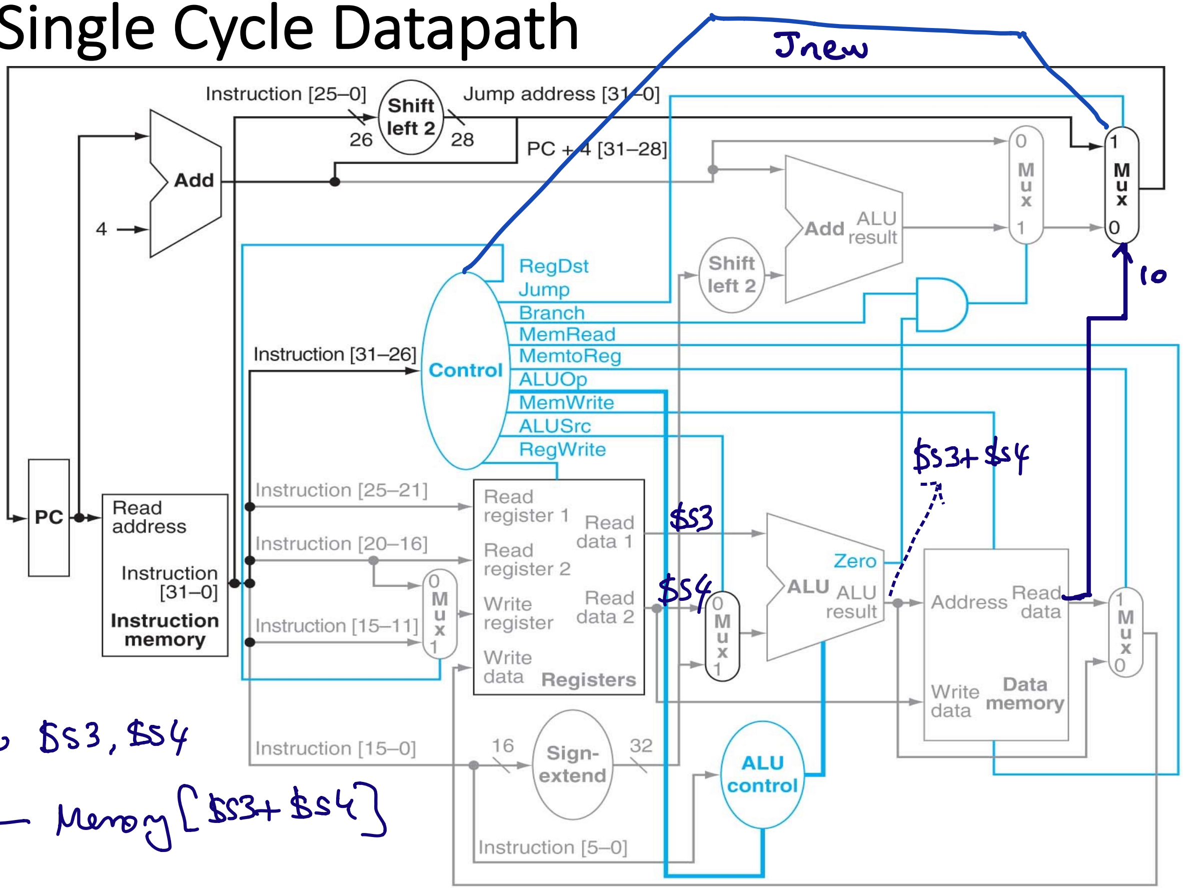
Explain your design by listing the required changes in the complete single-cycle datapath clearly given in Figure 4.24 (additional wires, muxes and control and selector signals if necessary).

Answer

Connect “Read Data” output of data memory to the multiplexor controlled with “Jump” control signal.

Now the multiplexor (control with jump) will have 2 control inputs
(Jump0)->original one
(Jump1)->new one

Single Cycle Datapath



Jnew $s3, $s4

PC ← Memory[$s3+$s4]