CS320 Fall 2014
Homework#5
Quiz in Lecture Thursday October 30, 2014
There are no make-up quizzes!

Problem 1: **Datapath timing**

Assuming the following timings for the components of the datapath. All other components have no delay.

- Data/Instruction Memory Read: 200ps
- Data Memory Write: 100ps
- Register File Read: 100ps
- Register File Write: 50ps
- Adders: 80ps
- ALUs: 200ps
- Logical Shifter/Sign Extension: 30ps
- Multiplexors and other gates: 10ps

a. Calculate the delay for each single cycle instruction type (R-type, lw, sw, beq, j). See attached pages
b. Calculate the delay for each clock cycle of the finite state machine (each stage of each instruction, i.e. each bubble). See attached pages
c. What is the minimum clock cycle period at which the multi-cycle datapath can operate properly? 270ps
   What is the length of each instruction type in ps (R-type, lw, sw, beq, j)?
   - lw 270*5 = 1350ps
   - sw/r-type 270*4 = 1080 ps
   - beq/j 270*3 = 810 ps

Problem 2: **MIPS Execution**

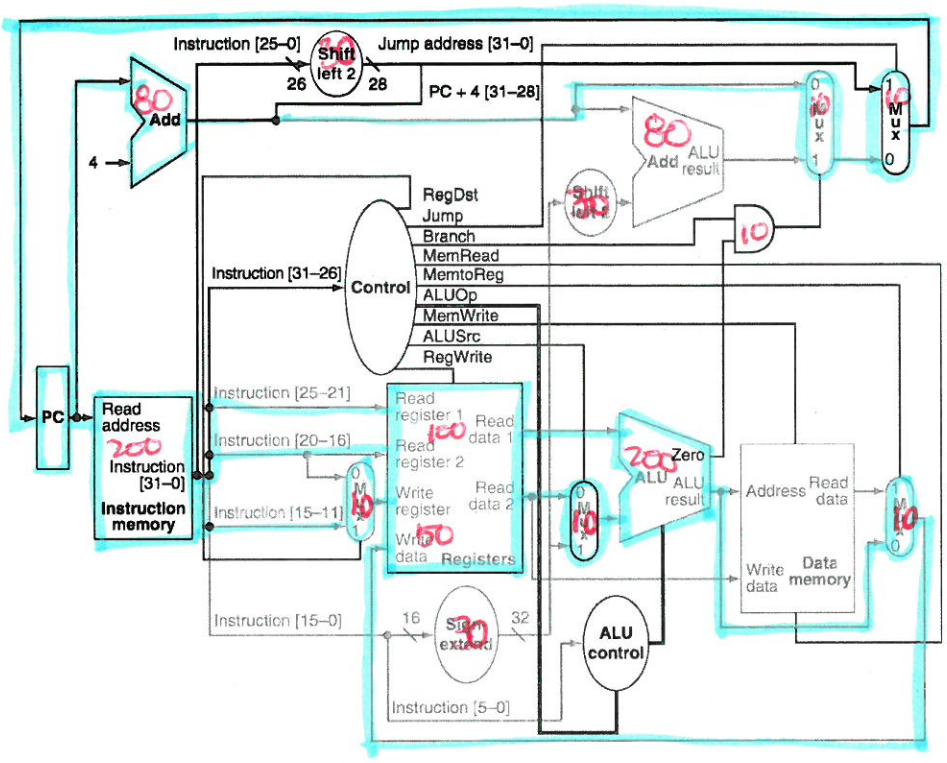
Based on the timings from Question 1, examine the following MIPS program P.

```
        add  $t0, $s4, $s3
        and  $t2, $t0, $s1
        add  $t1, $t1, $t1
        add  $t2, $t2, $t2
        lw   $t3, OPTION
        beq  $s0, $t3, PARSE_WRITE_SP_LE
        and  $t0, $t1, 0xff
        and  $t1, $t1, 0xff00
        or   $t1, $t1, $t0
        and  $t0, $t2, 0xff
        and  $t2, $t2, 0xff00
        or   $t2, $t2, $t0
PARSE_WRITE_SP_LE:
        sw $t1, 0($a1)
        lw $v0, WRITE_FILE
        sw $t2, 0($a1)
```

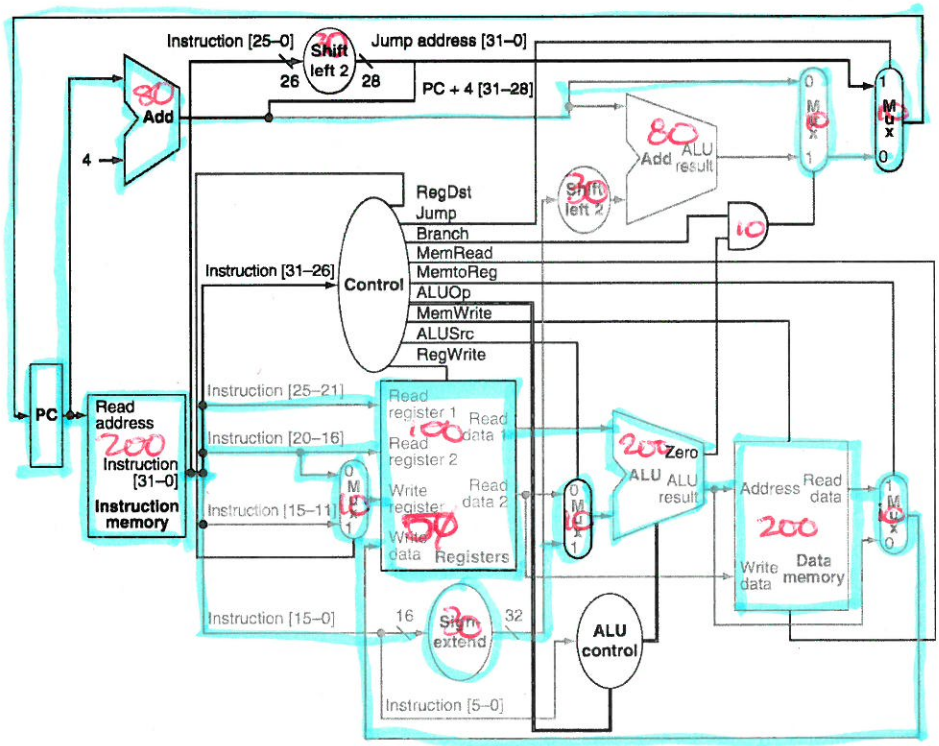
a. In the multicycle datapath, in which clock cycle is the `lw $t3, OPTION` instruction fetched? 17th cycle
b. In the multicycle datapath, how many cycles does it take Program P to execute, assuming the branch is not taken? 61 clock cycles
c. Which values (eg. MEM[$s0], branch address, Instruction[15:0], etc) are stored in the A, B, ALUOut, and MDR registers during the execution cycle of the following instruction: `sw $t1, 0($a1)`
   A: Reg[rs] = Reg[$a1]
   B: Reg[rt] = Reg[$t1]
   ALUOut: Reg[rs] + Instr[15:0] = Reg[$a1] + 0
   MDR: Value read from memory (since Mem Rd = 0, it is the value last time read from the memory), which is the sw instruction itself.

Rtype 570ps

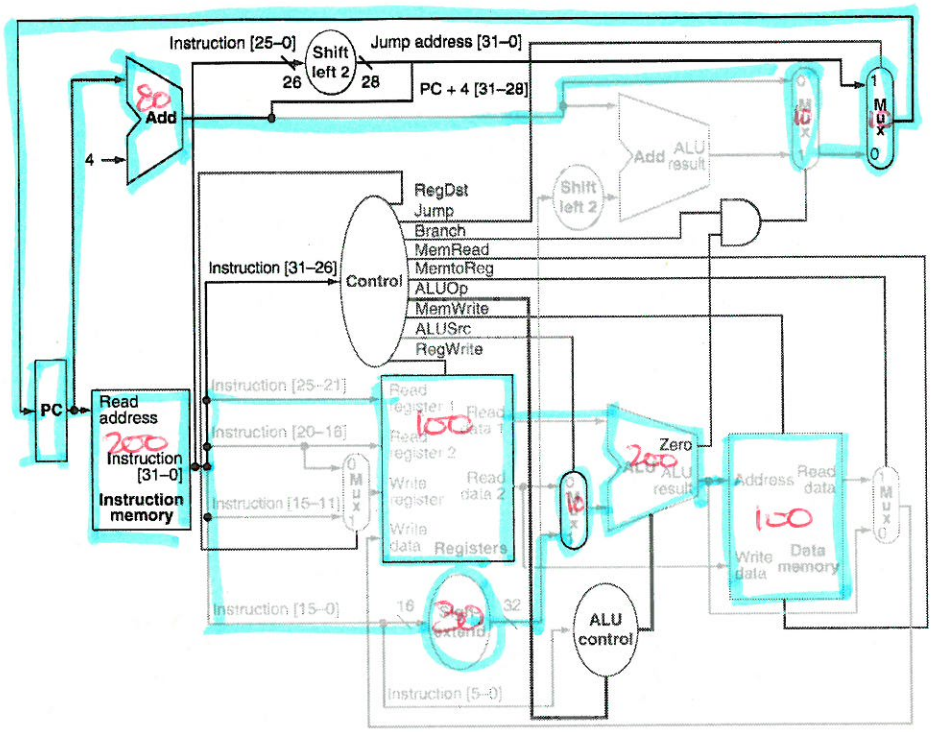
Critical path:
PC → Imem → Reg Rd → ALUSrc → ALU → MemtoReg
Rd
→ RegWrite



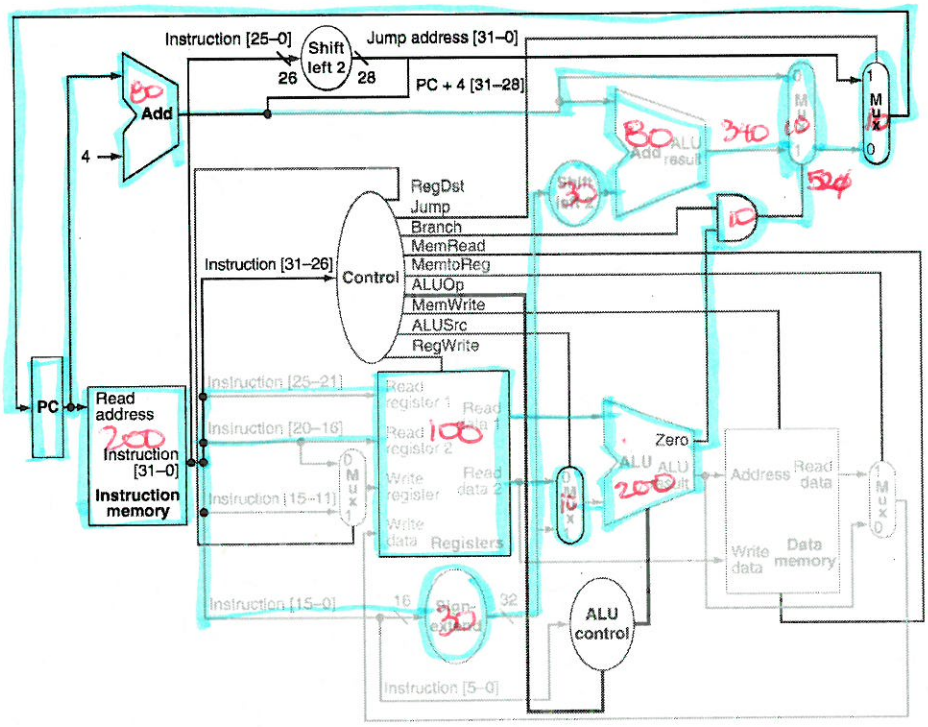
LW 760ps PC → Imem → Reg Rd → ALU → D Mem → MemtoReg → RegWrite
Rd



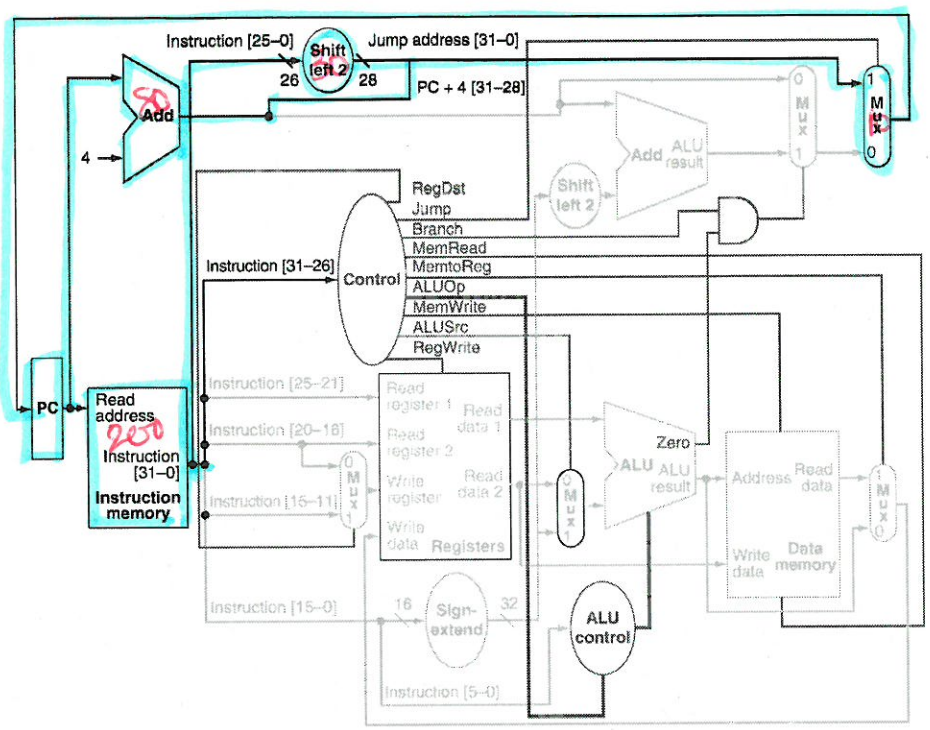
SW 600ps PC → Imem (Rd) → RegRd → ALU → DMem Write

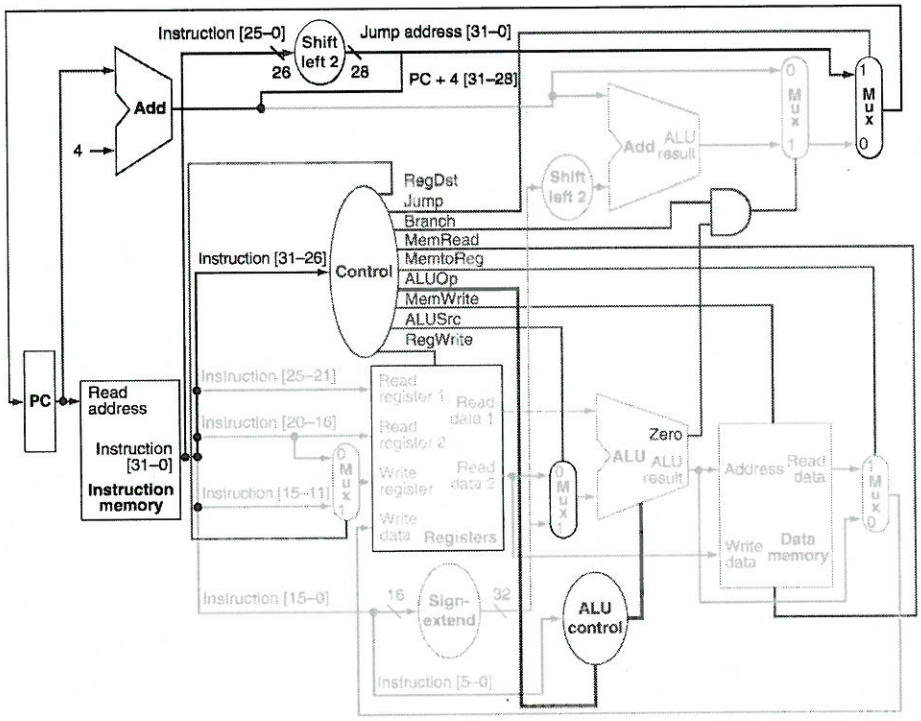


Beq 540ps PC → Imem (Rd) → Reg Rd → ALUSrc → ALU → AND gate → MUX → Jump mux



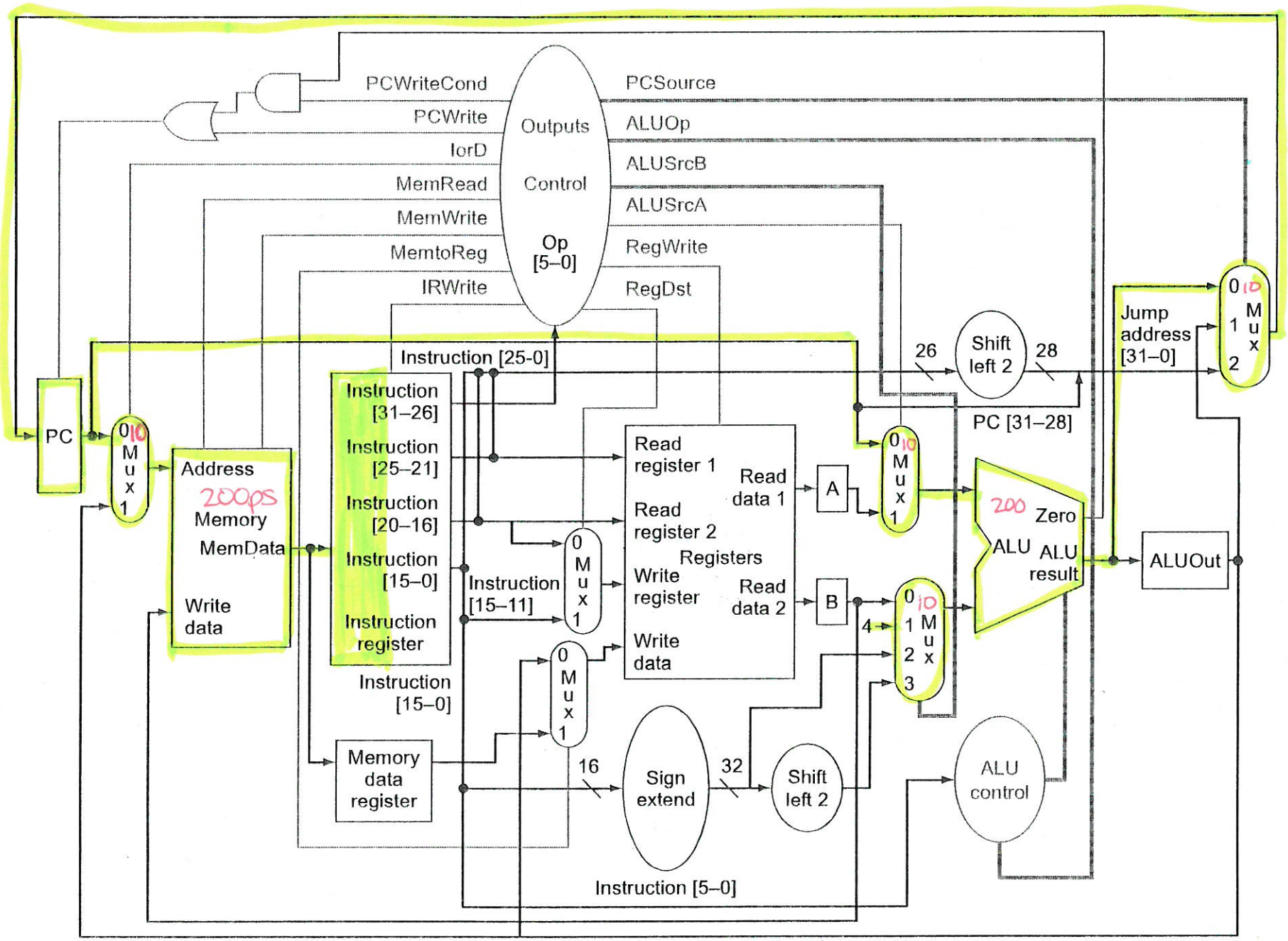
Jump 2 ops PC → IMem Rd → Shift left → Jump





Stage 0: Instruction Fetch

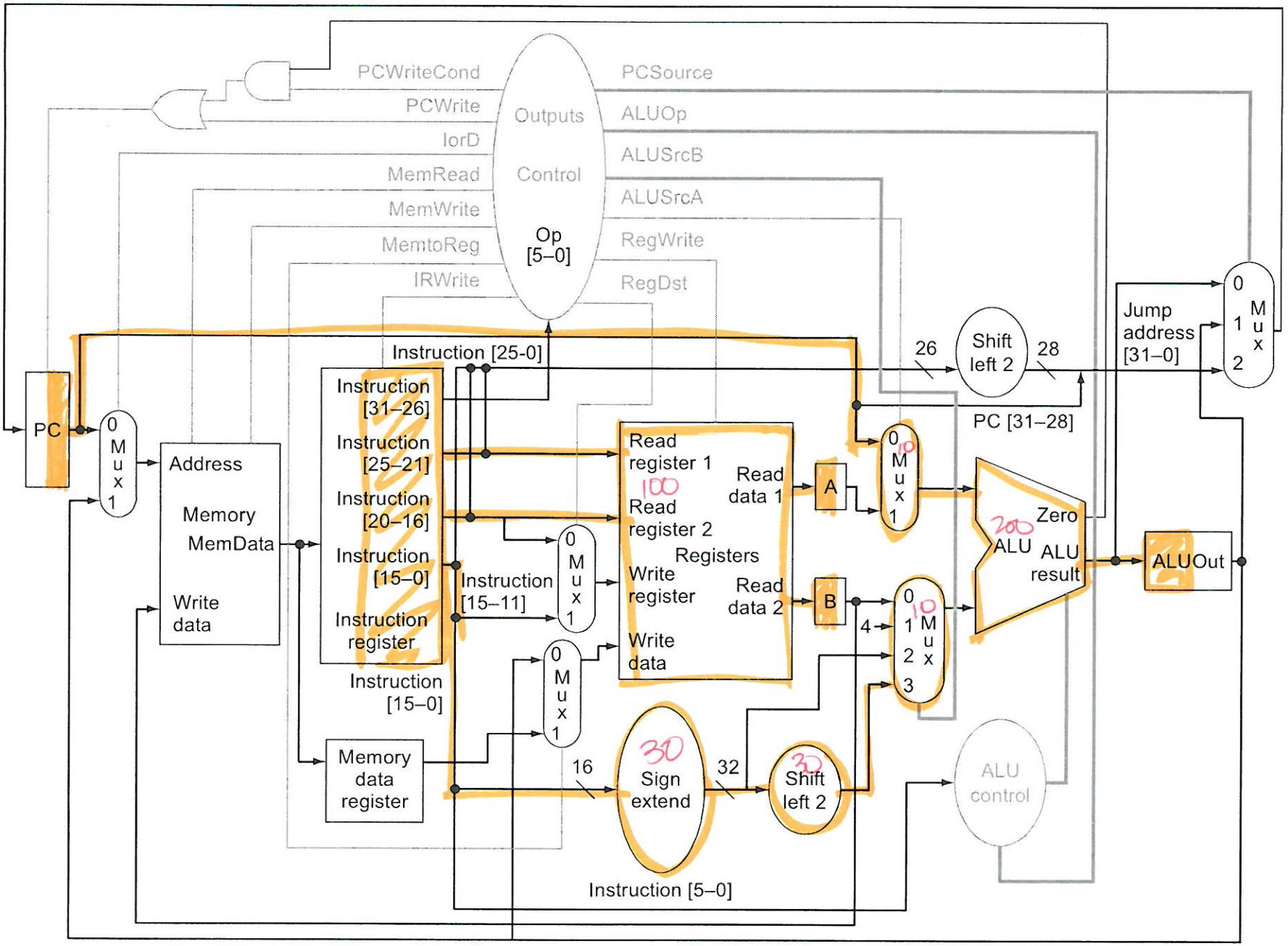
PC → ALUSrcA → ALU → PCSource → PC 210 ps



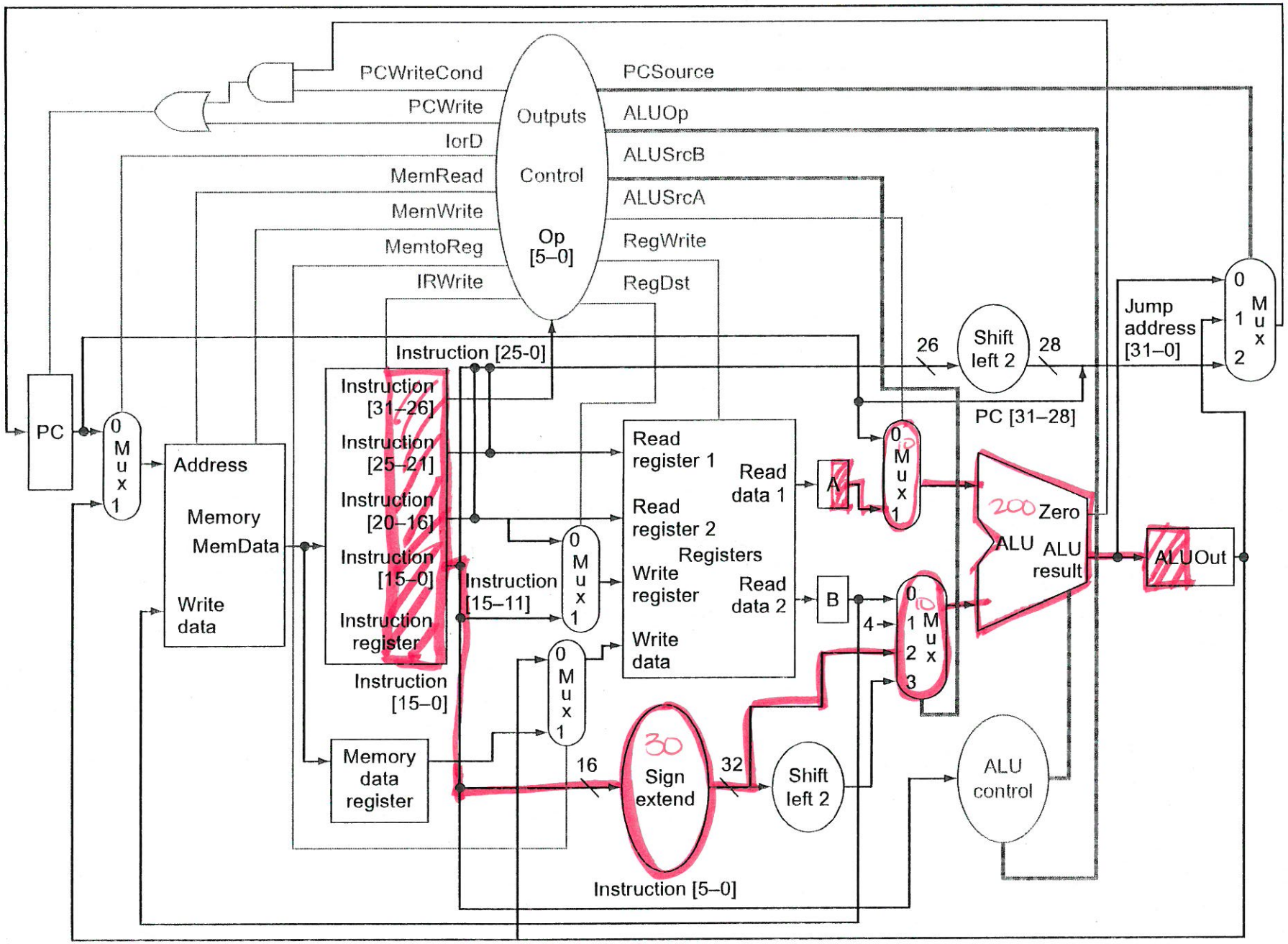
Stage 1: Decode

IR → Sign Extend → Shift left 2 → ALUSrcB → ALU → ALUOut

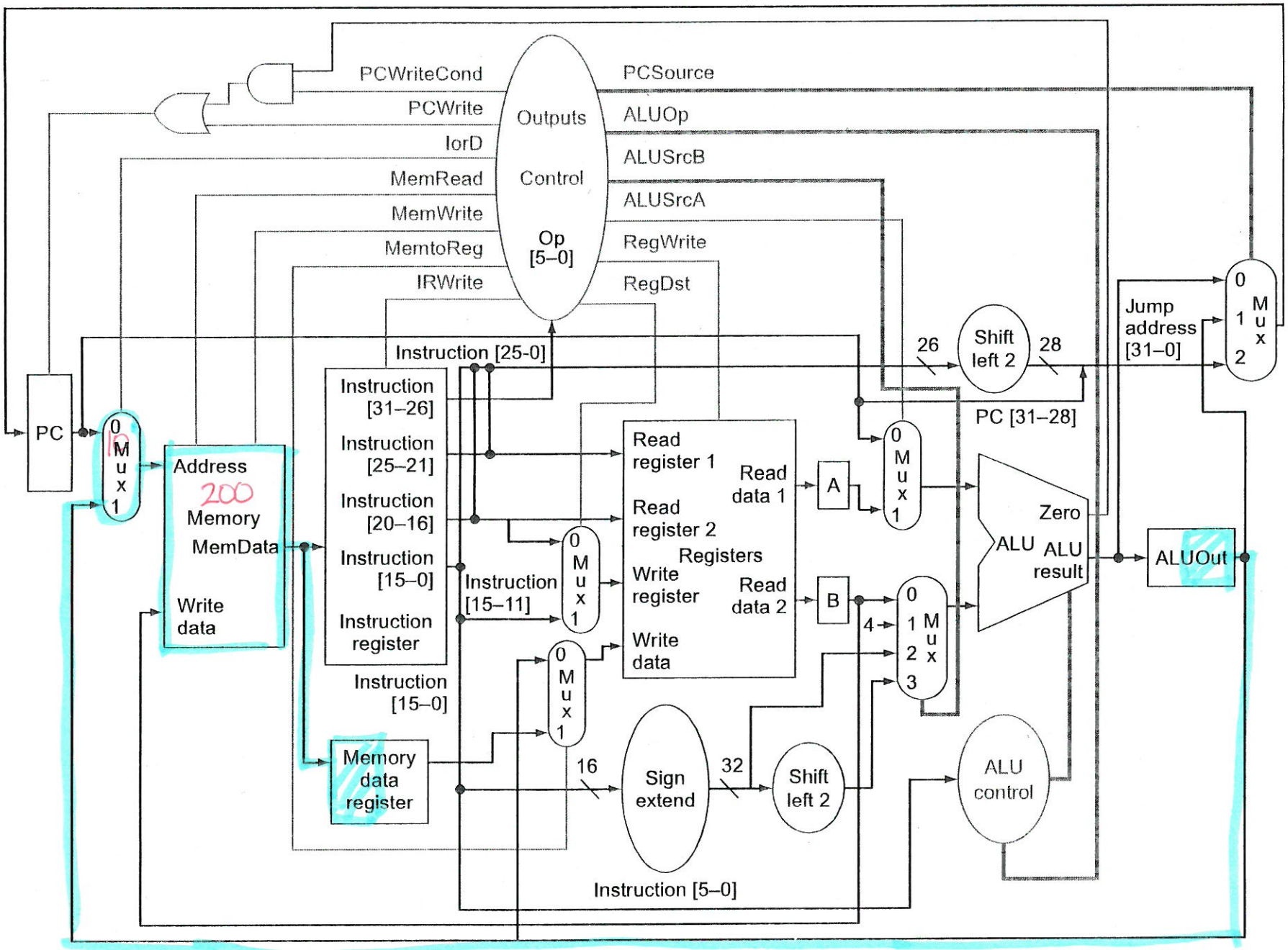
270ps



Stage 2: IR → Sign Extend → ALUSrcB → ALU → ALUOut 240ps

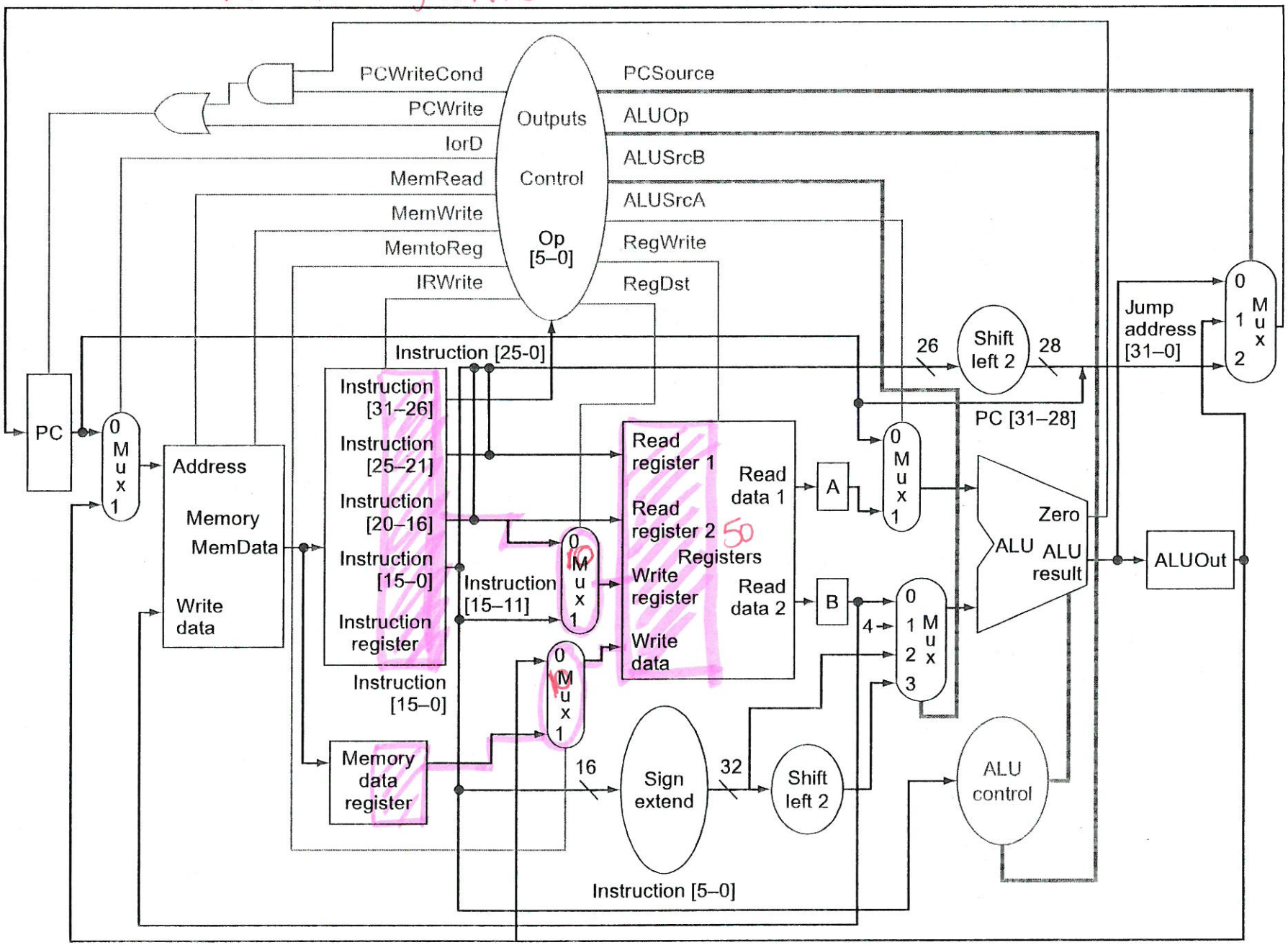


Stage 3: ALUOut → IorD → Memory Rd → MDR 210ps.

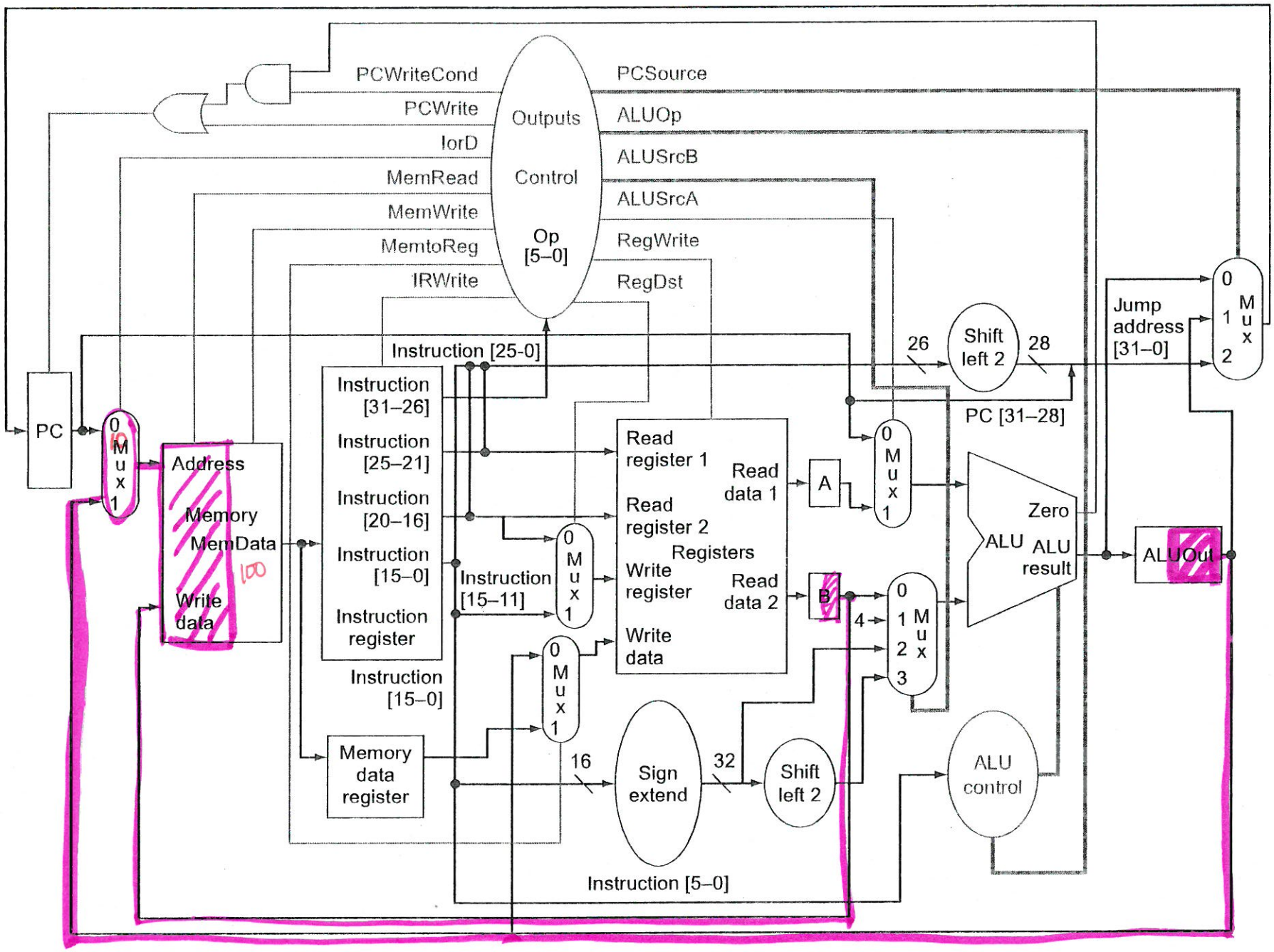


Stage 4: MDR → Reg Write
IR → Reg Write

60ps



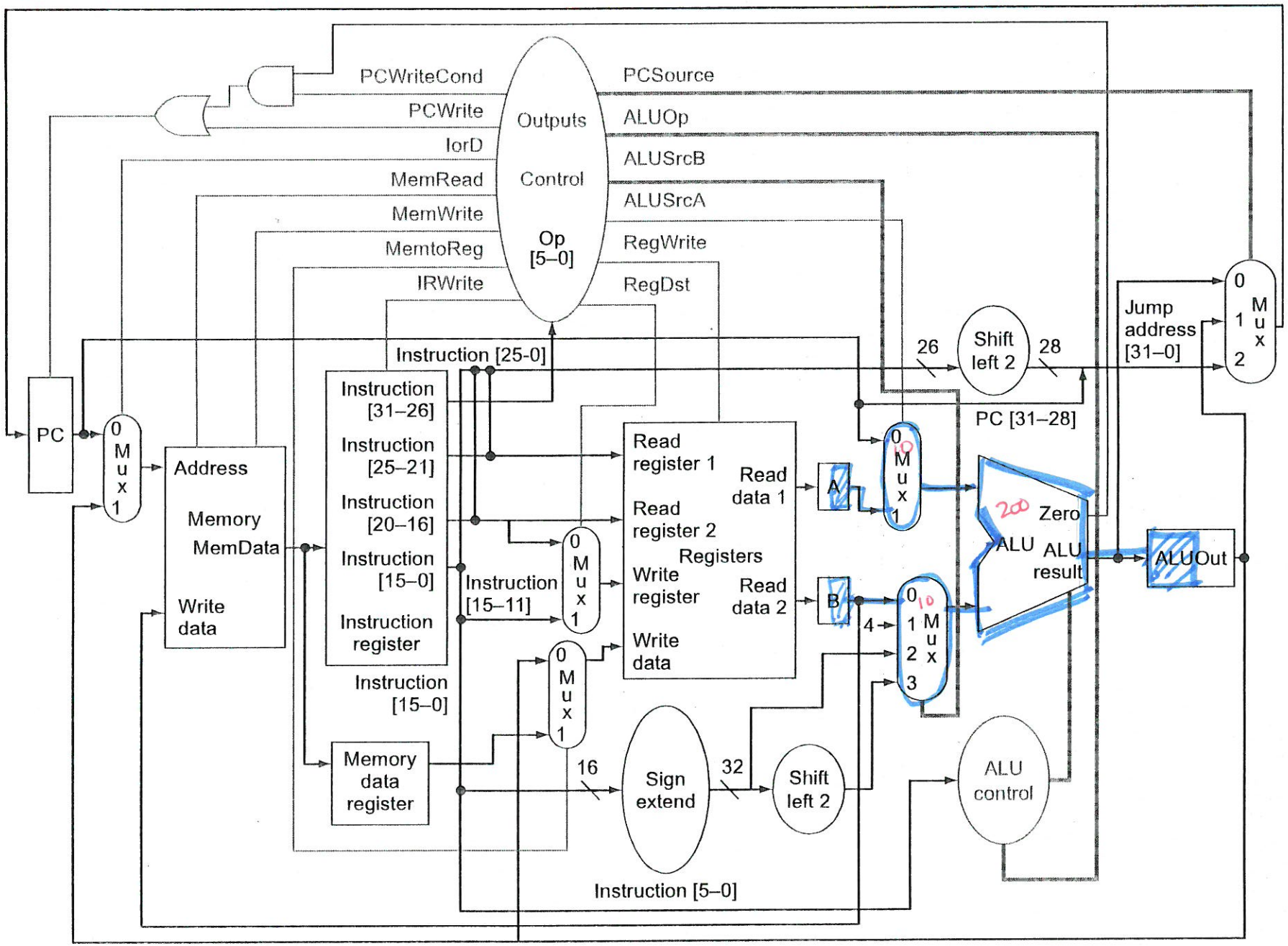
Stage 5: ALUOut → IorD → Mem Write 110ps



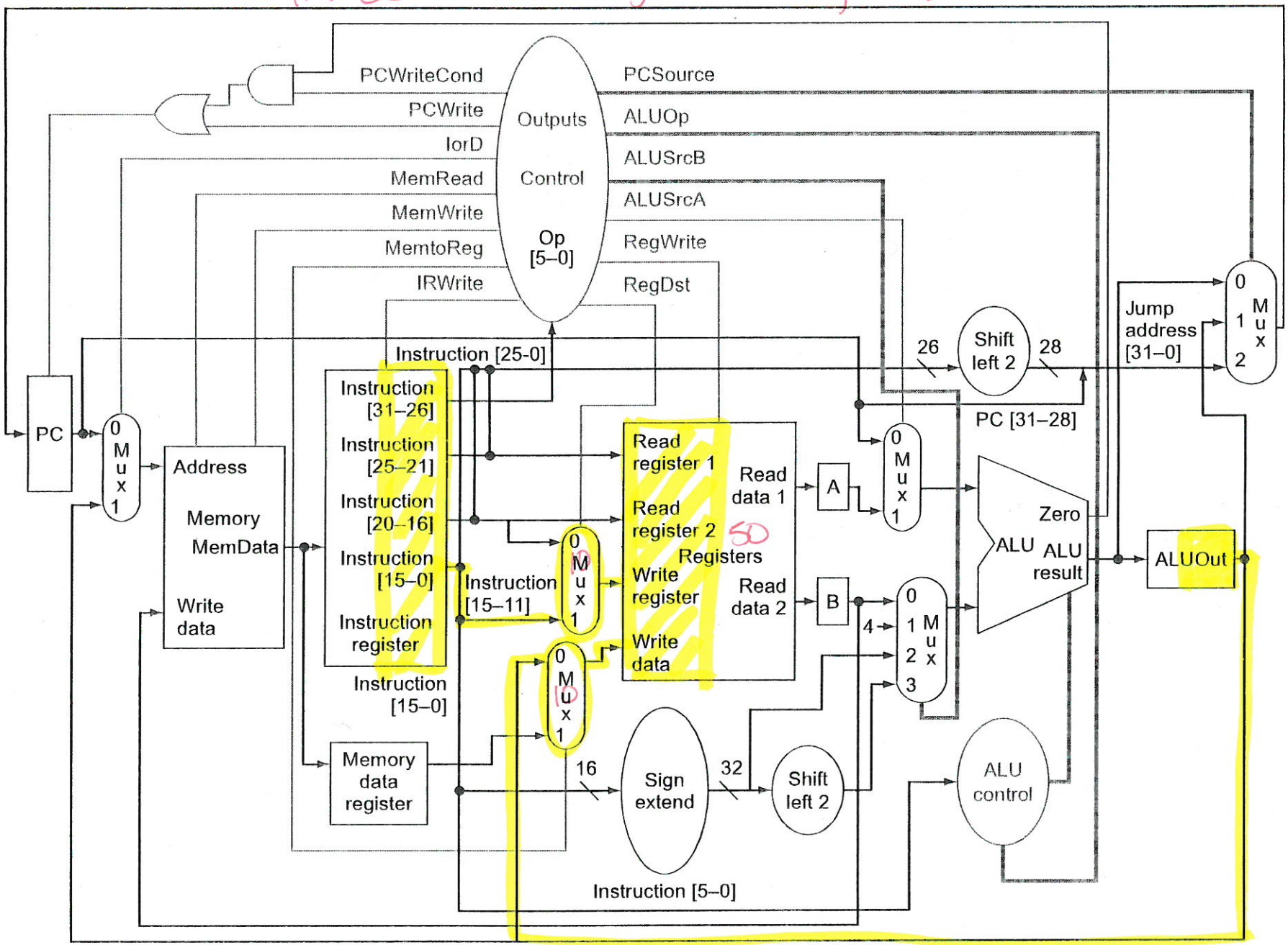
Stage 6:

A → ALU → ALUOut
B → ALU → ALUOut

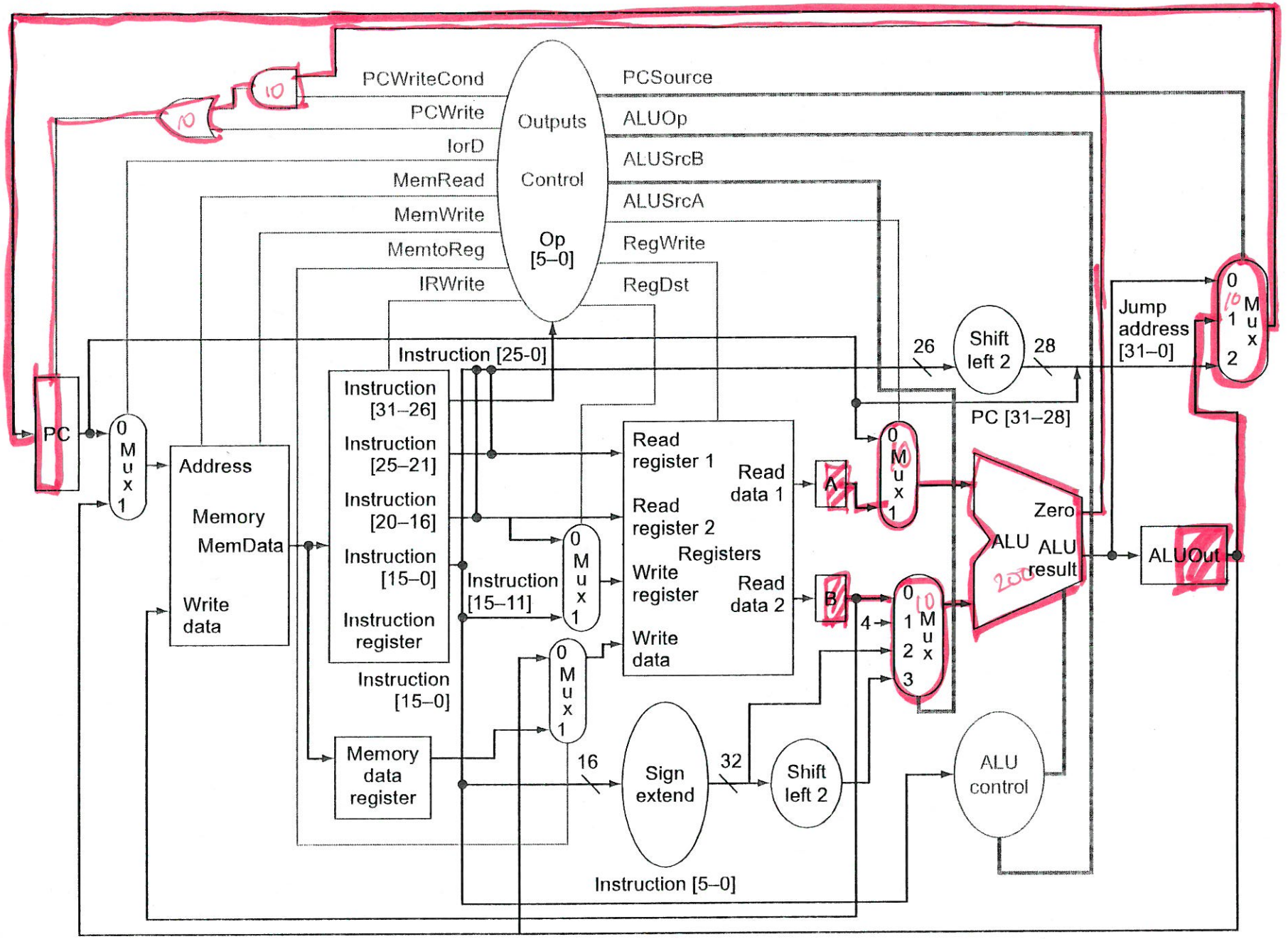
210ps



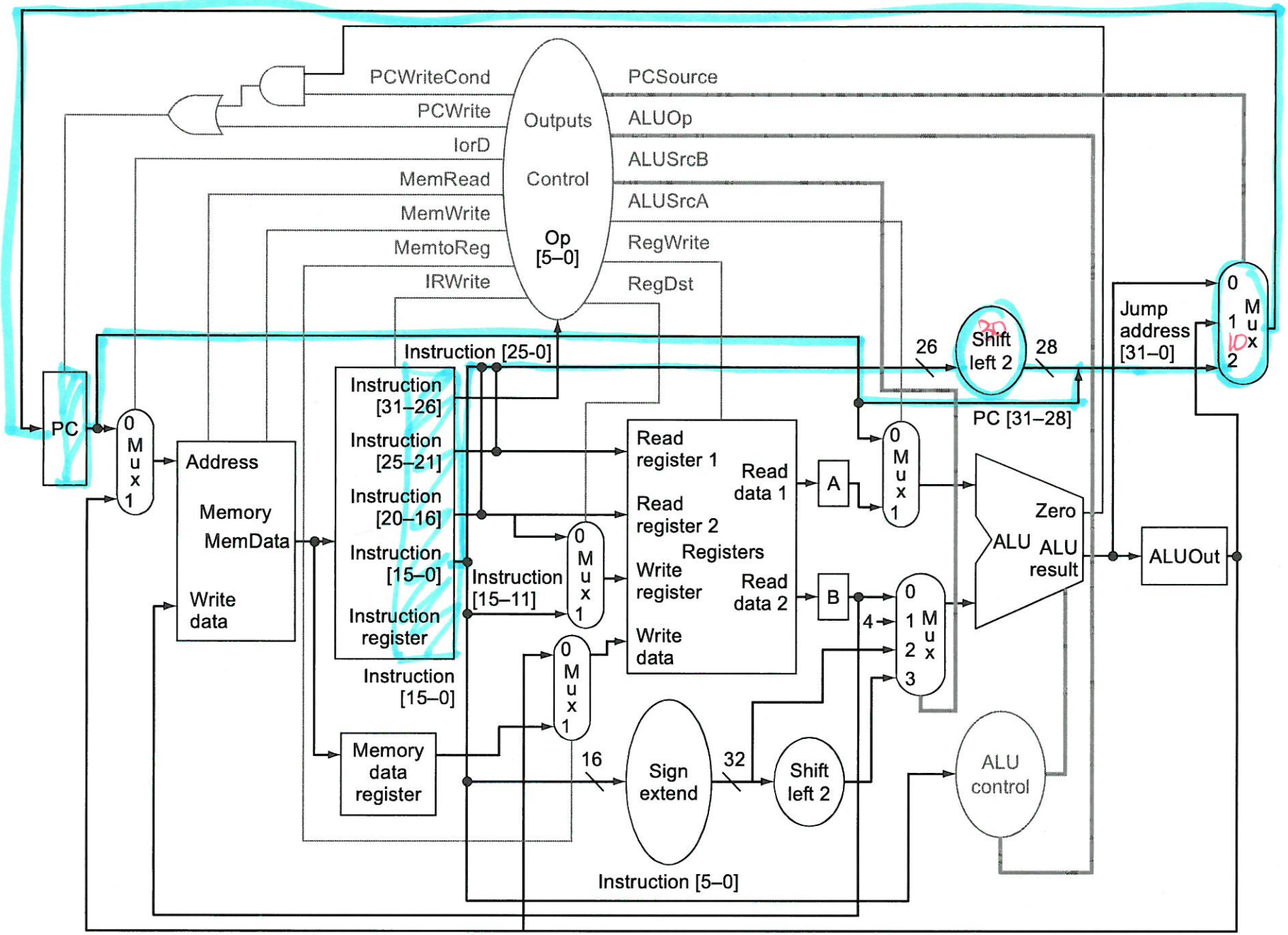
Stage 7: ALUOut → MemtoReg → Reg File Write
IR [illegible] → RegDst → Reg File Write
6ops



Stage 8: A → ALU → ANDgate → OrGate → PC control 230ps



Stage 9: IR → Shift left 2 → PCSource → PC 40ps



Problem 3: **Multi-cycle datapath**

In class we covered the MIPS multi-cycle implementation also which operates for the same basic subset of MIPS instructions as in the single-cycle basic implementation. In this problem you will introduce functionality for additional MIPS instructions. Use this datapath and finite state machine to specify your changes. In all cases, try to find a solution that minimizes the number of clock cycles required for the new instruction. Explicitly state how many cycles it takes to execute the new instruction in your modified finite state machine. Use the Multi-cycle handout posted on PIAZZA.

PLEASE IGNORE THE TIMING VALUES IN THE FOLLOWING SOLUTIONS. THEY ARE NOT CORRECT.

Modify the datapath and the finite state machine to:

a. Implement 'addi' instruction

Reg[rt] ← Reg[rs] + immed[15:0];
register rs is added to immediate value in the instruction and stored in rt

b. Implement 'jal' instruction.

Reg[31] ← PC+4 # $ra register stores the return address
PC ← PC+4[31:28], (Instruction[25:0] << 2)
PC is the top 4 bits of the PC+4 value concatenated with the lowest 26 bits of the jump address
shifted the left by 2 bits

c. Implement a new 'wai' instruction. The instruction, `wai rt`, stands for "where am i" and puts the instruction address into a register specified by the rt field. (Immediate format)

Reg[rt] ← PC # $ra register stores the return address

d. Implement 'bne'

if(Reg[rs] != Reg[rt]) PC ← PC+4 + Instruction[15:0] << 2

e. Implement 'beqi' (Assume the rs field is a 5-bit 2's complement value)

if(Reg[rt] == 2's complement value in rs field)
PC ← PC+4 + Instruction[15:0] << 2

f. Implement 'bgtz' (Assume the rt register is set to $0)

if(Reg[rs] >= 0) PC ← PC+4 + Instruction[15:0] << 2

g. Implement a new 'incbeq' instruction. The instruction, `incbeq rs, rt, label`, stands for "increment and branch on equal". Increment always occurs!

if(Reg[rs] == Reg[rt]) PC ← PC+4 + Instruction[15:0] << 2
Reg[rs] ← Reg[rs] +4 #the increment always occurs

h. Implement a new 'sneg' instruction. The instruction, `sneg rs`, stands for "set on negative". (Immediate format)

if (Reg[rs] < 0) Reg[rt] = 1 , else Reg[rt] ← 0

i. Implement a new 'lwdec' instructon. The instruction `lwdec rs, offset(rt)`, loads a value and decrements the rt value by one memory word.

Reg[rt] ← Mem[Reg[rs] + sign-extended offset]
Reg[rs] ← Reg[rs] – 4

j. Implement a new 'swr' instruction. The instruction `swr rd, rt(rs)`, uses a register value for the offset when calculating the memory address to store data to.

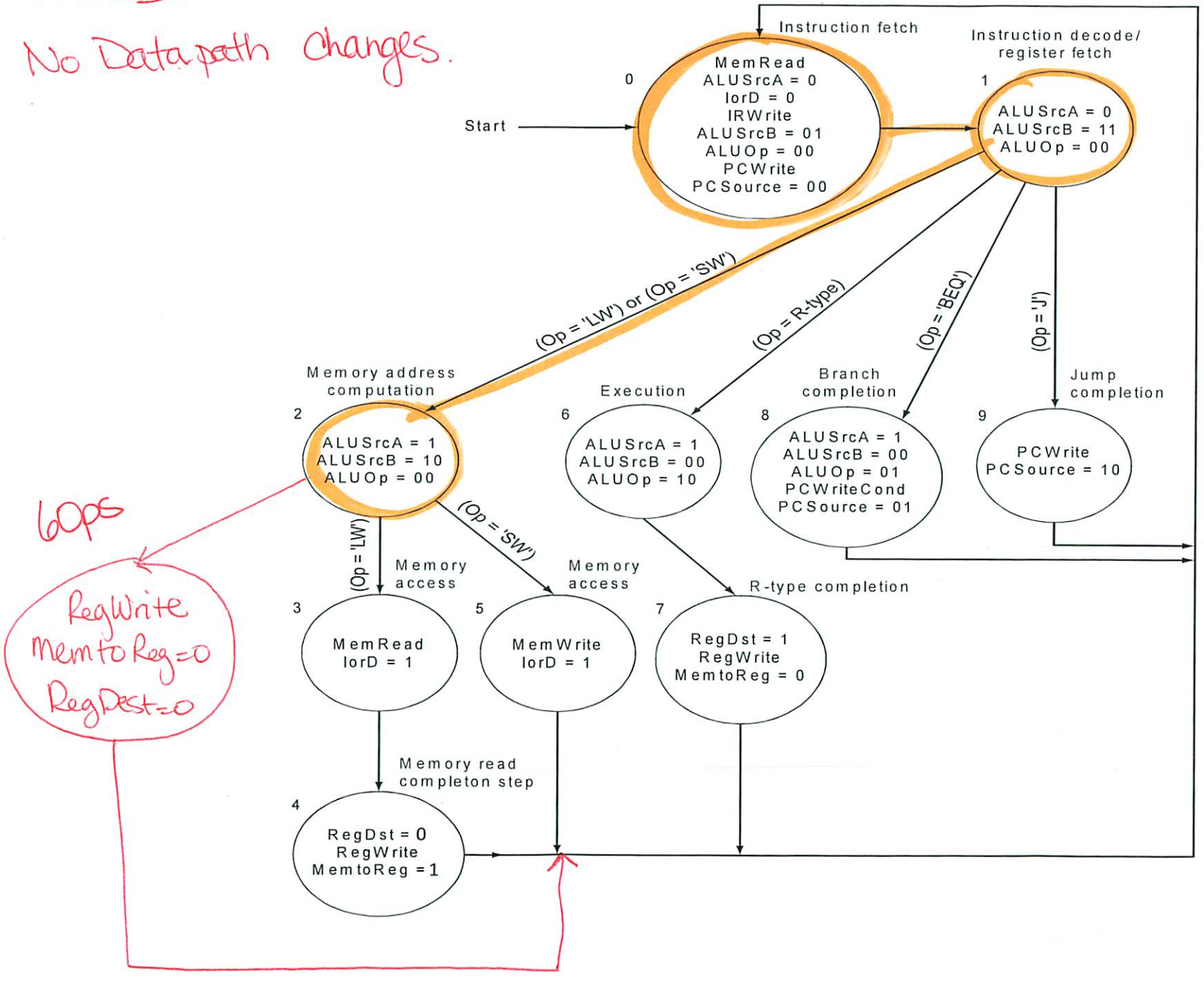
Mem[Reg[rs] + Reg[rt]] = Reg[rd]

k. Implement a new 'slti' instruction. The instruction sets `slti rs, rt, immediate` sets the rs register to the immediate value if the value in register rs is less than register rt.

if(Reg[rs] < Reg[rt])
Reg[rs] ← 2's complement immediate value

ADDI:

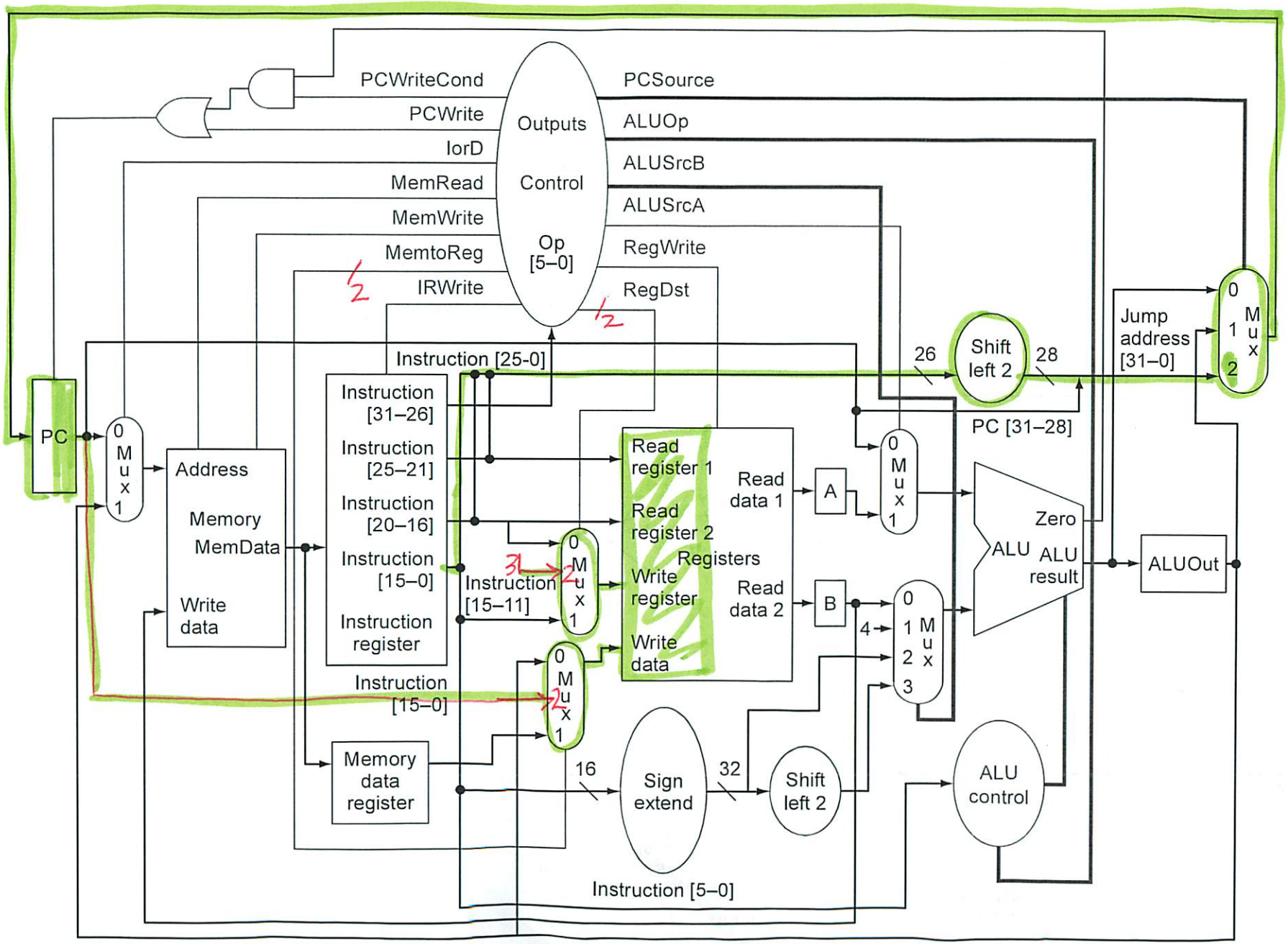
No Datapath changes.

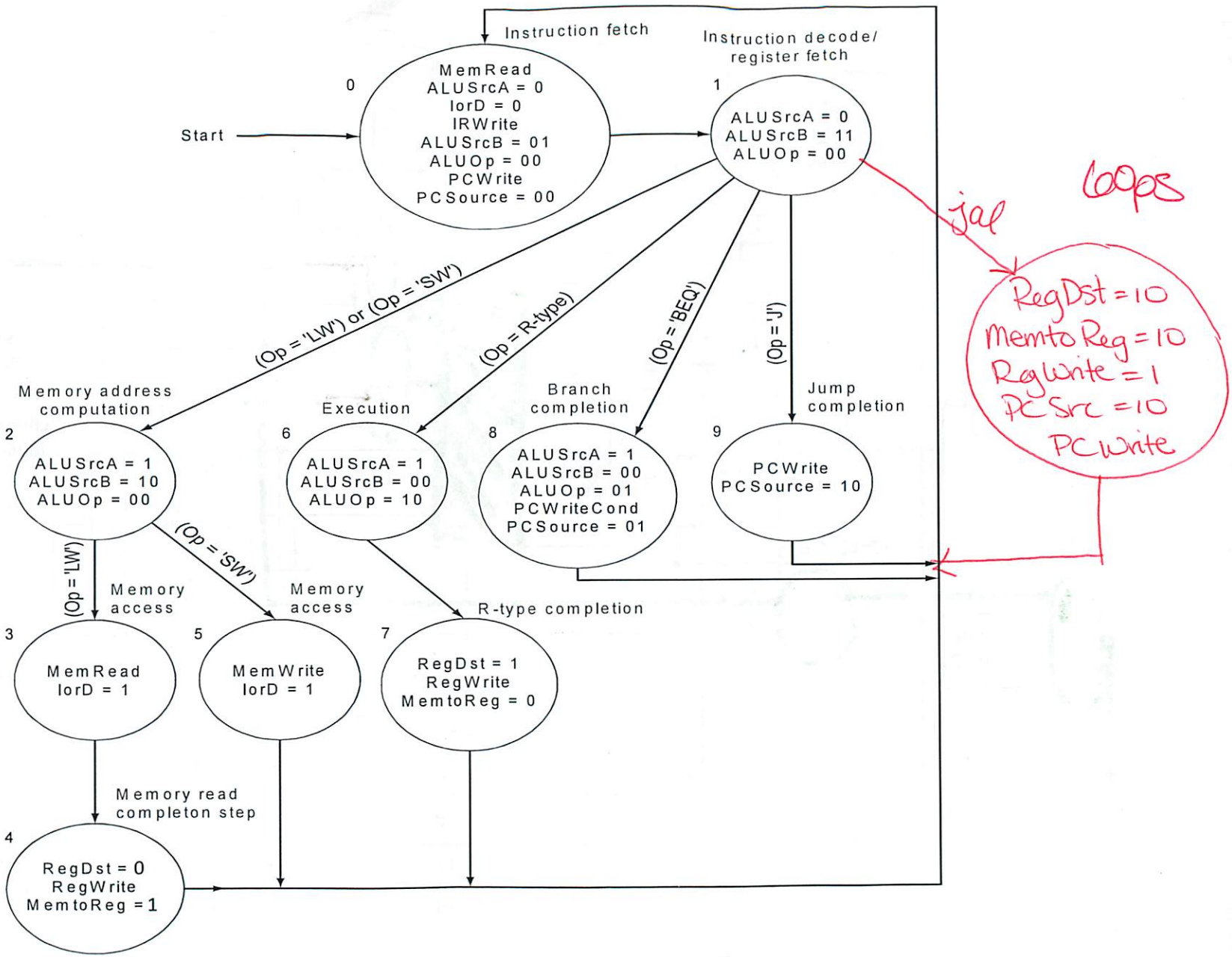


6 Ops

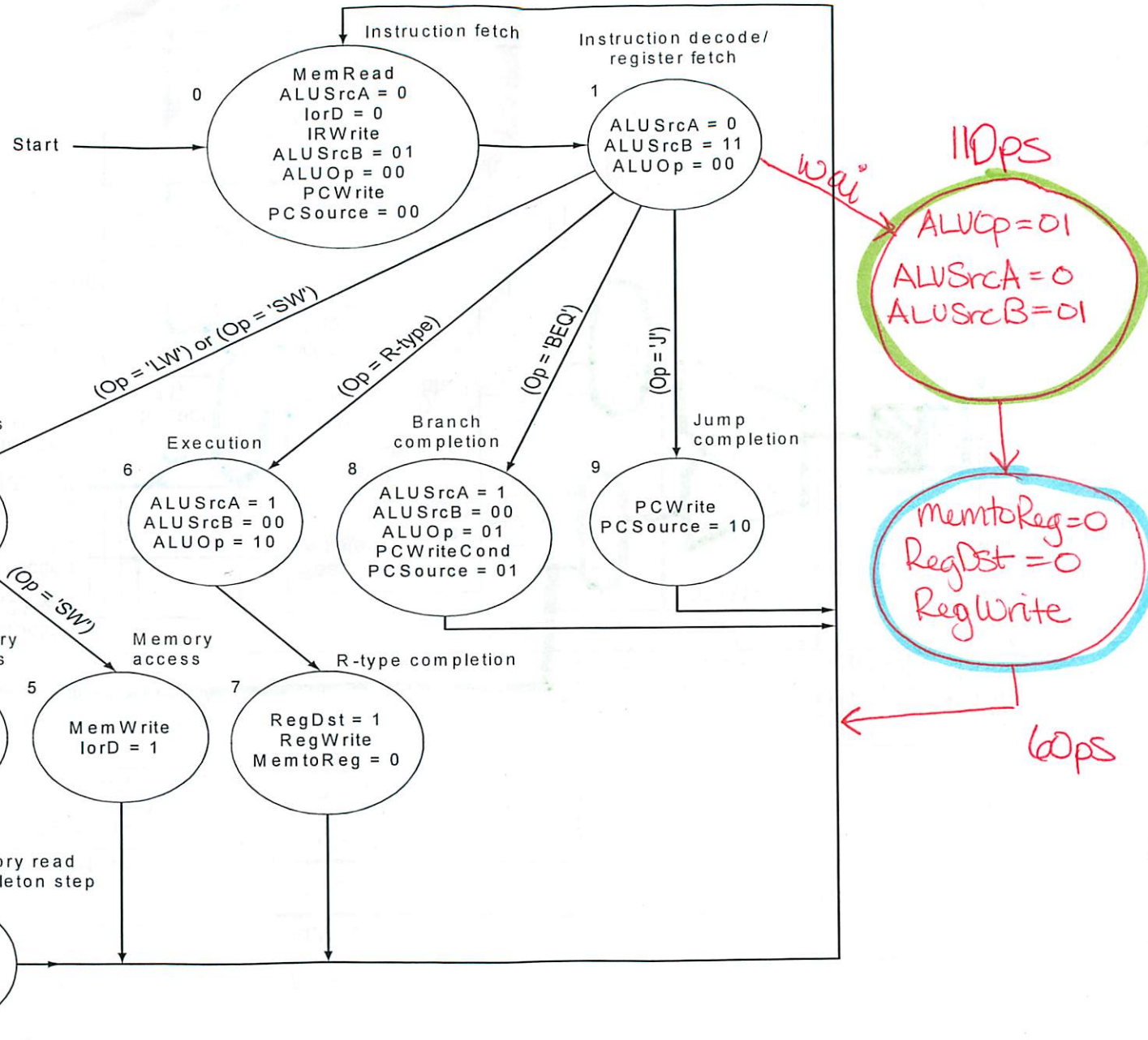
RegWrite
MemtoReg=0
RegDest=0

jal

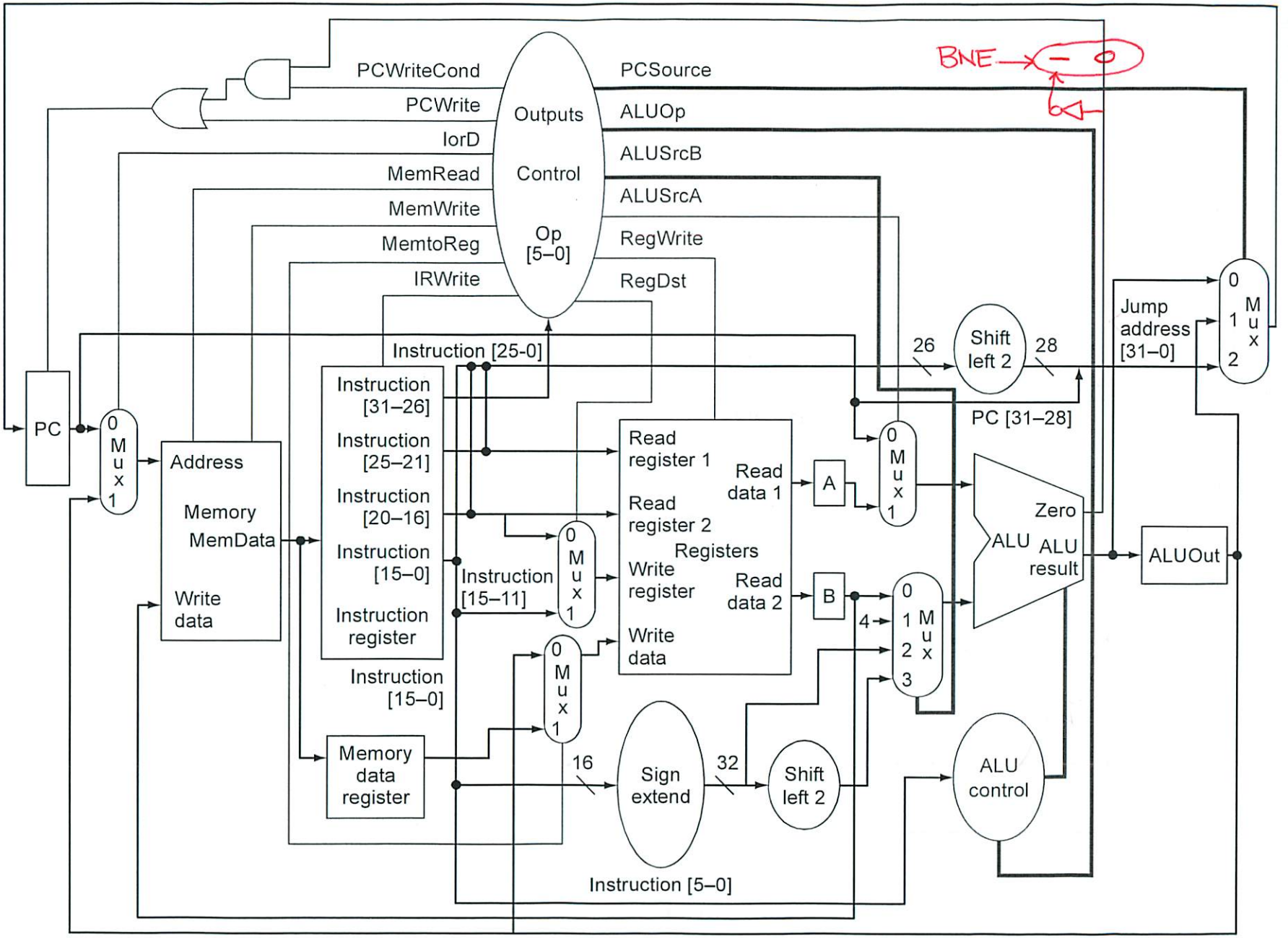

PCWriteCond
PCWrite
IorD
MemRead
MemWrite
MemtoReg
IRWrite
Outputs
Control
Op
[5–0]
PCSource
ALUOp
ALUSrcB
ALUSrcA
RegWrite
RegDst
1/2
1/2
Instruction [25-0]
26
Shift
left 2
28
Jump
address
[31–0]
0
1
2
M
u
x
PC [31–28]
PC
0
M
u
x
1
Address
Memory
MemData
Write
data
Instruction
[31–26]
Instruction
[25–21]
Instruction
[20–16]
Instruction
[15–0]
Instruction
register
Instruction
[15–11]
31
Instruction
[15–0]
Memory
data
register
Read
register 1
Read
register 2
Registers
Write
register
Write
data
Read
data 1
Read
data 2
A
B
4
ALU
Zero
ALU
result
ALUOut
16
Sign
extend
32
Shift
left 2
ALU
control
Instruction [5–0]


Instruction fetch
Instruction decode/
register fetch
0
MemRead
ALUSrcA = 0
IorD = 0
IRWrite
ALUSrcB = 01
ALUOp = 00
PCWrite
PCSource = 00
Start
1
ALUSrcA = 0
ALUSrcB = 11
ALUOp = 00
(Op = 'LW') or (Op = 'SW')
(Op = R-type)
(Op = 'BEQ')
(Op = 'J')
Memory address
computation
Execution
Branch
completion
Jump
completion
2
ALUSrcA = 1
ALUSrcB = 10
ALUOp = 00
6
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 10
8
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 01
PCWriteCond
PCSource = 01
9
PCWrite
PCSource = 10
(Op = 'LW')
(Op = 'SW')
Memory
access
Memory
access
R-type completion
3
MemRead
IorD = 1
5
MemWrite
IorD = 1
7
RegDst = 1
RegWrite
MemtoReg = 0
Memory read
completon step
4
RegDst = 0
RegWrite
MemtoReg = 1
jal
loops
RegDst = 10
MemtoReg = 10
RegWrite = 1
PCSrc = 10
PCWrite

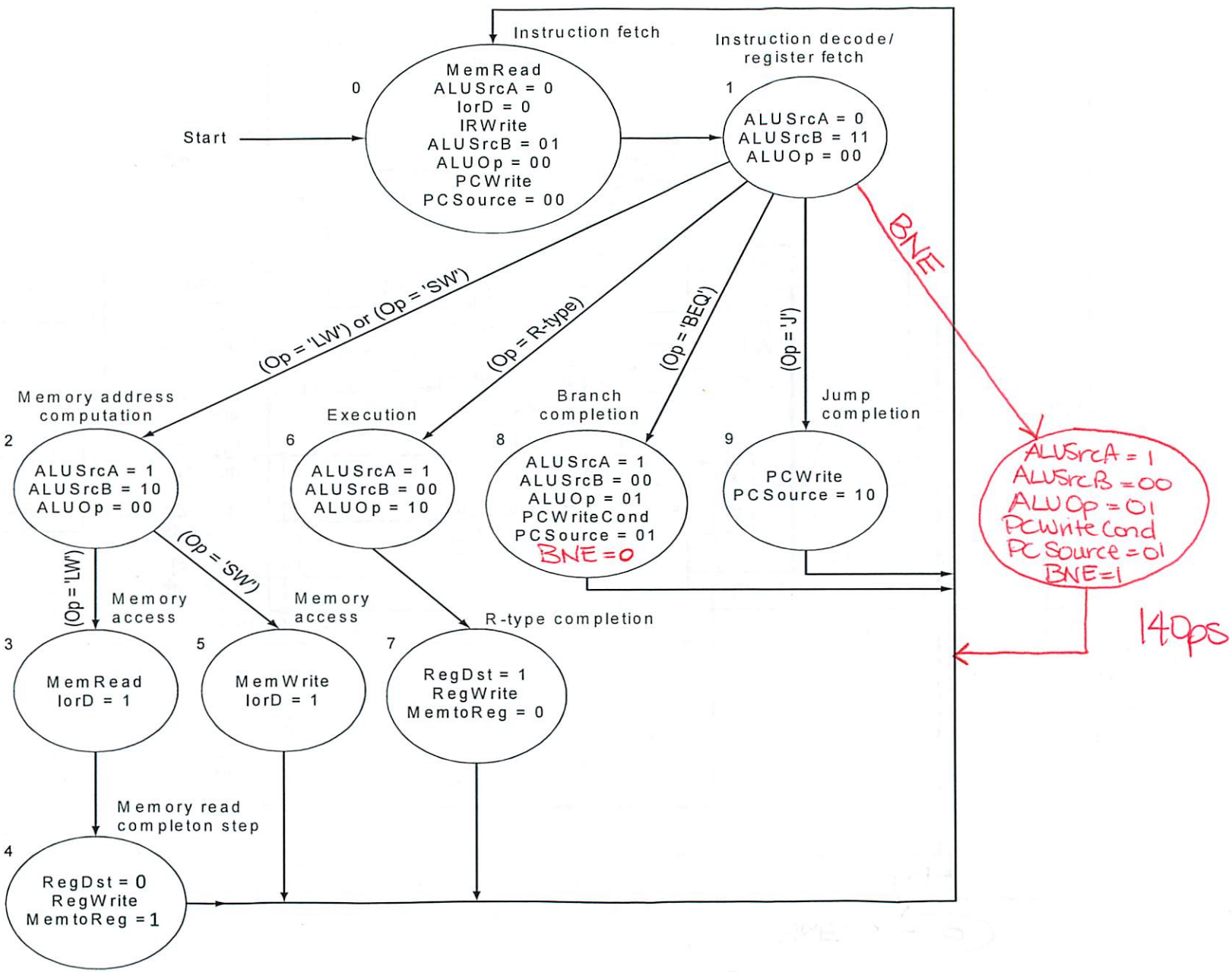
wai: R-type format
No Datapath Chages



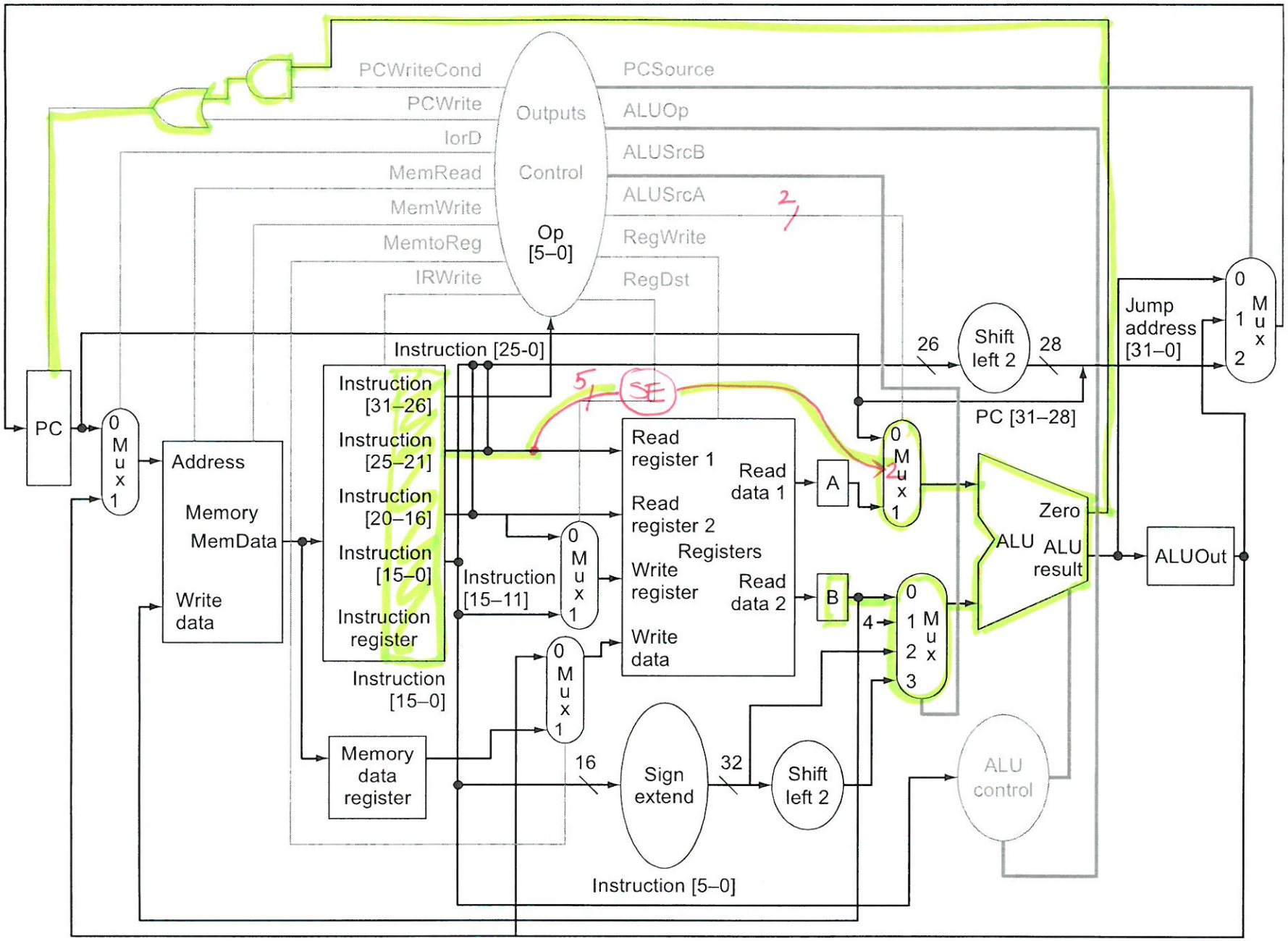
bne:

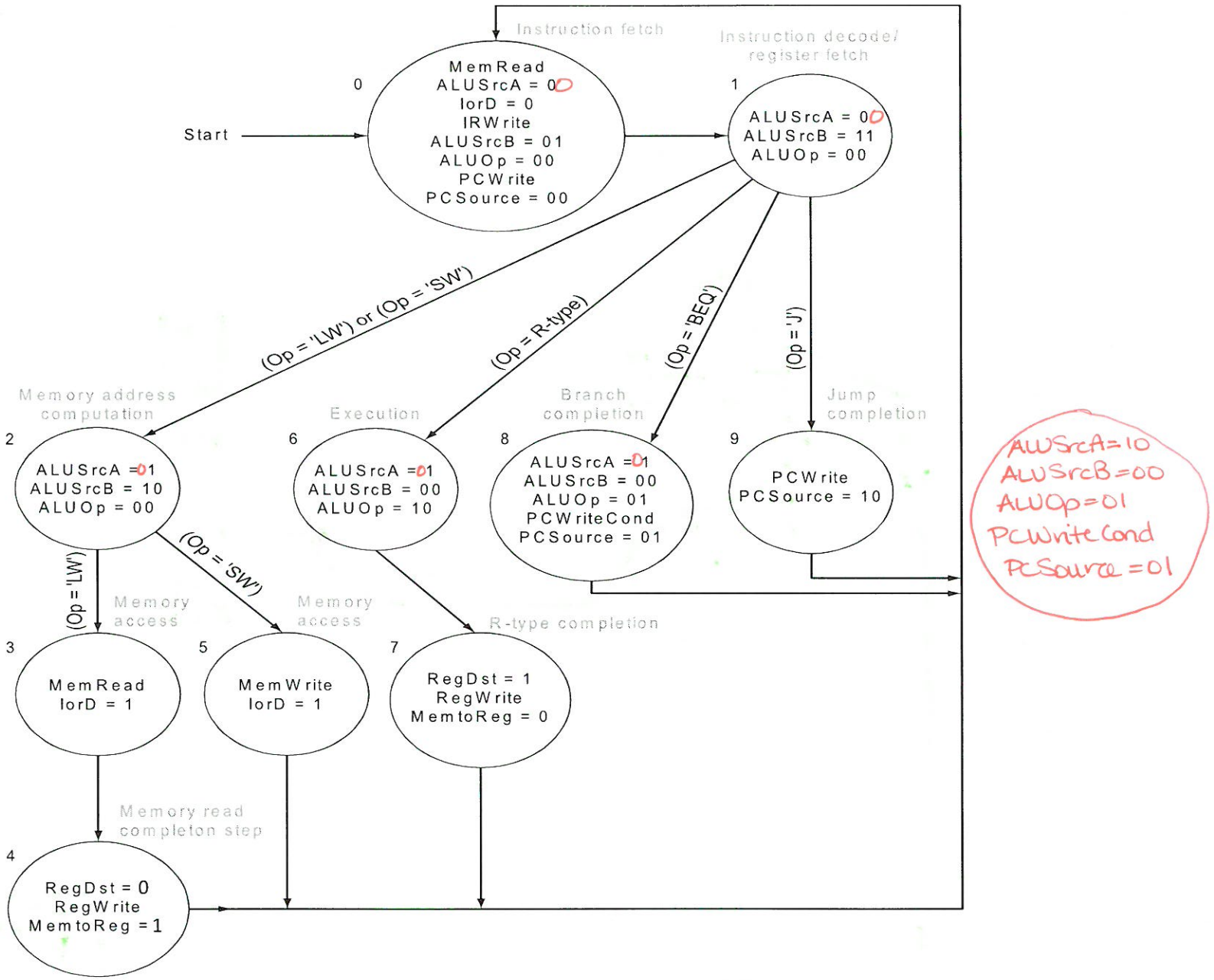

BNE
PCWriteCond
PCWrite
IorD
MemRead
MemWrite
MemtoReg
IRWrite
Outputs
Control
Op
[5–0]
PCSource
ALUOp
ALUSrcB
ALUSrcA
RegWrite
RegDst
Instruction [25-0]
26
Shift
left 2
28
PC [31–28]
Jump
address
[31–0]
M
u
x
PC
Address
Memory
MemData
Write
data
Instruction
[31–26]
Instruction
[25–21]
Instruction
[20–16]
Instruction
[15–0]
Instruction
register
Instruction
[15–0]
Instruction
[15–11]
Read
register 1
Read
register 2
Registers
Write
register
Write
data
Read
data 1
Read
data 2
A
B
4
Zero
ALU
ALU
result
ALUOut
Memory
data
register
16
Sign
extend
32
Shift
left 2
ALU
control
Instruction [5–0]

BNE:


Instruction fetch
Instruction decode/
register fetch
0
MemRead
ALUSrcA = 0
IorD = 0
IRWrite
ALUSrcB = 01
ALUOp = 00
PCWrite
PCSource = 00
Start
1
ALUSrcA = 0
ALUSrcB = 11
ALUOp = 00
BNE
(Op = 'LW') or (Op = 'SW')
(Op = R-type)
(Op = 'BEQ')
(Op = 'J')
Memory address
computation
Execution
Branch
completion
Jump
completion
2
ALUSrcA = 1
ALUSrcB = 10
ALUOp = 00
6
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 10
8
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 01
PCWriteCond
PCSource = 01
BNE=0
9
PCWrite
PCSource = 10
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 01
PCWriteCond
PCSource = 01
BNE=1
140ps
(Op = 'LW')
(Op = 'SW')
Memory
access
Memory
access
R-type completion
3
MemRead
IorD = 1
5
MemWrite
IorD = 1
7
RegDst = 1
RegWrite
MemtoReg = 0
Memory read
completon step
4
RegDst = 0
RegWrite
MemtoReg = 1

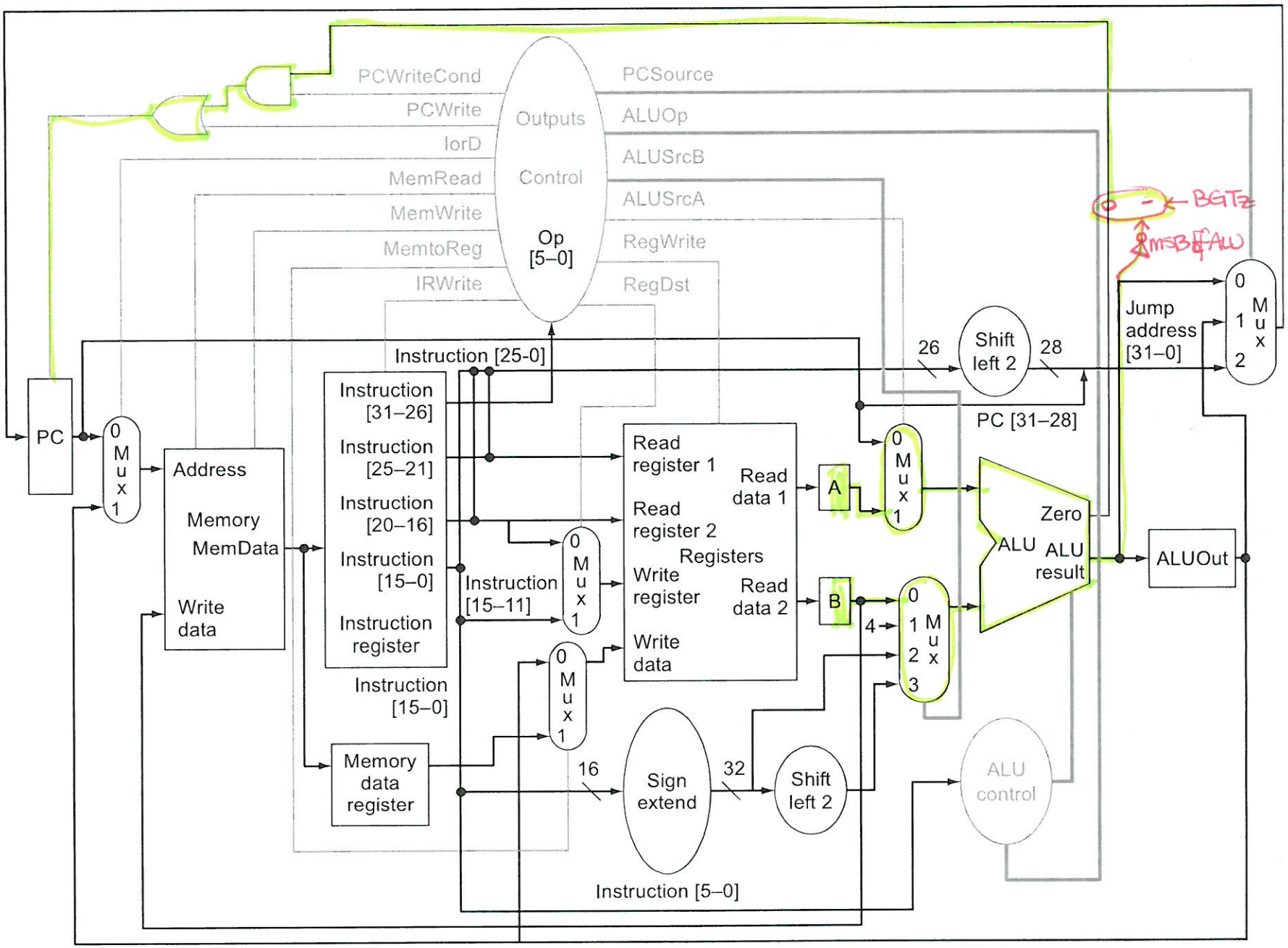
beqi

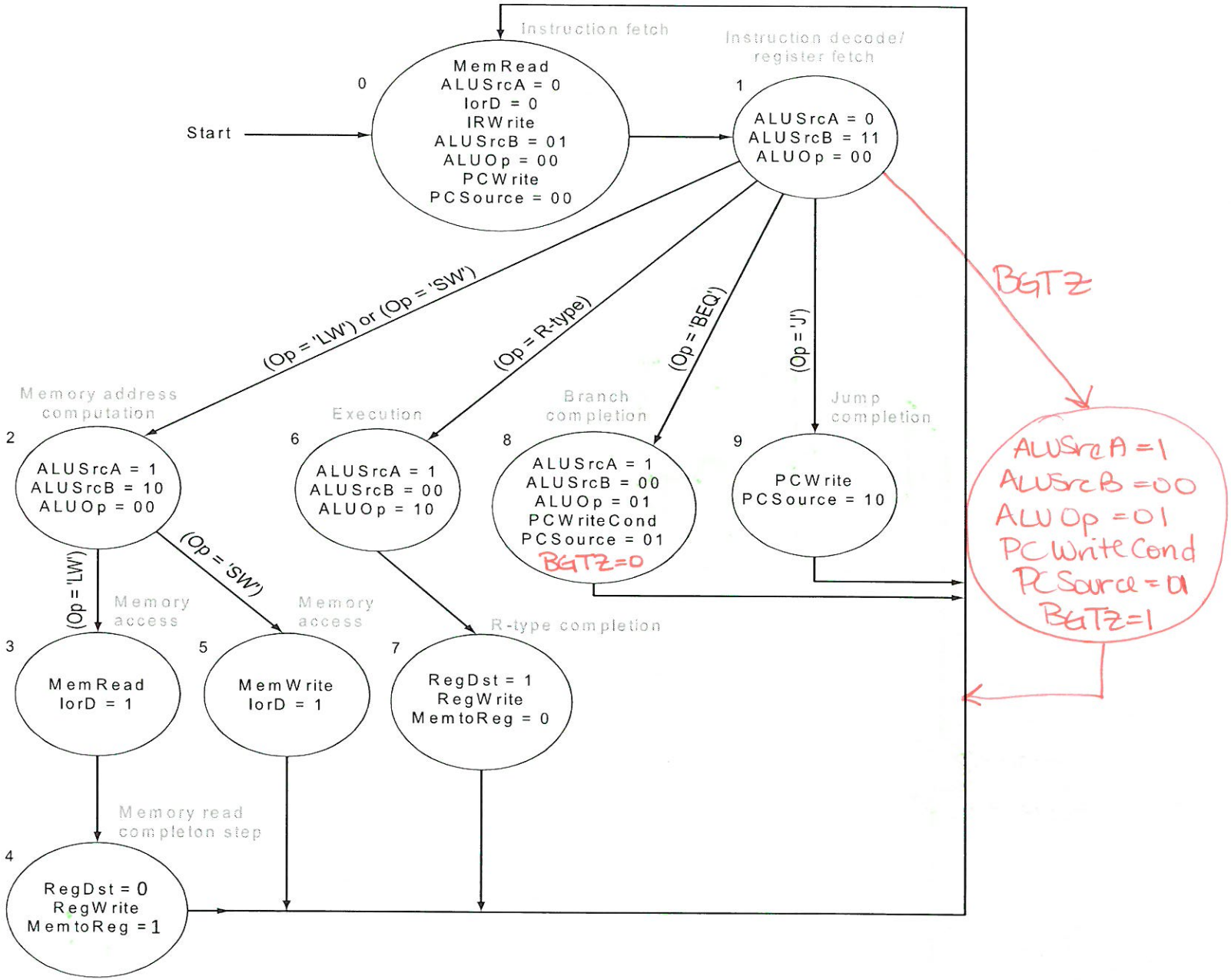

PCWriteCond
PCWrite
IorD
MemRead
MemWrite
MemtoReg
IRWrite
Outputs
Control
Op
[5–0]
PCSource
ALUOp
ALUSrcB
ALUSrcA
RegWrite
RegDst
Instruction [25-0]
26
Shift
left 2
28
PC [31–28]
Jump
address
[31–0]
M
u
x
0
1
2
PC
Address
Memory
MemData
Write
data
Instruction
[31–26]
Instruction
[25–21]
Instruction
[20–16]
Instruction
[15–0]
Instruction
register
Instruction
[15–0]
Instruction
[15–11]
Memory
data
register
Read
register 1
Read
register 2
Registers
Write
register
Write
data
Read
data 1
Read
data 2
A
B
4
Zero
ALU
ALU
result
ALUOut
16
Sign
extend
32
Shift
left 2
ALU
control
Instruction [5–0]
5
SE



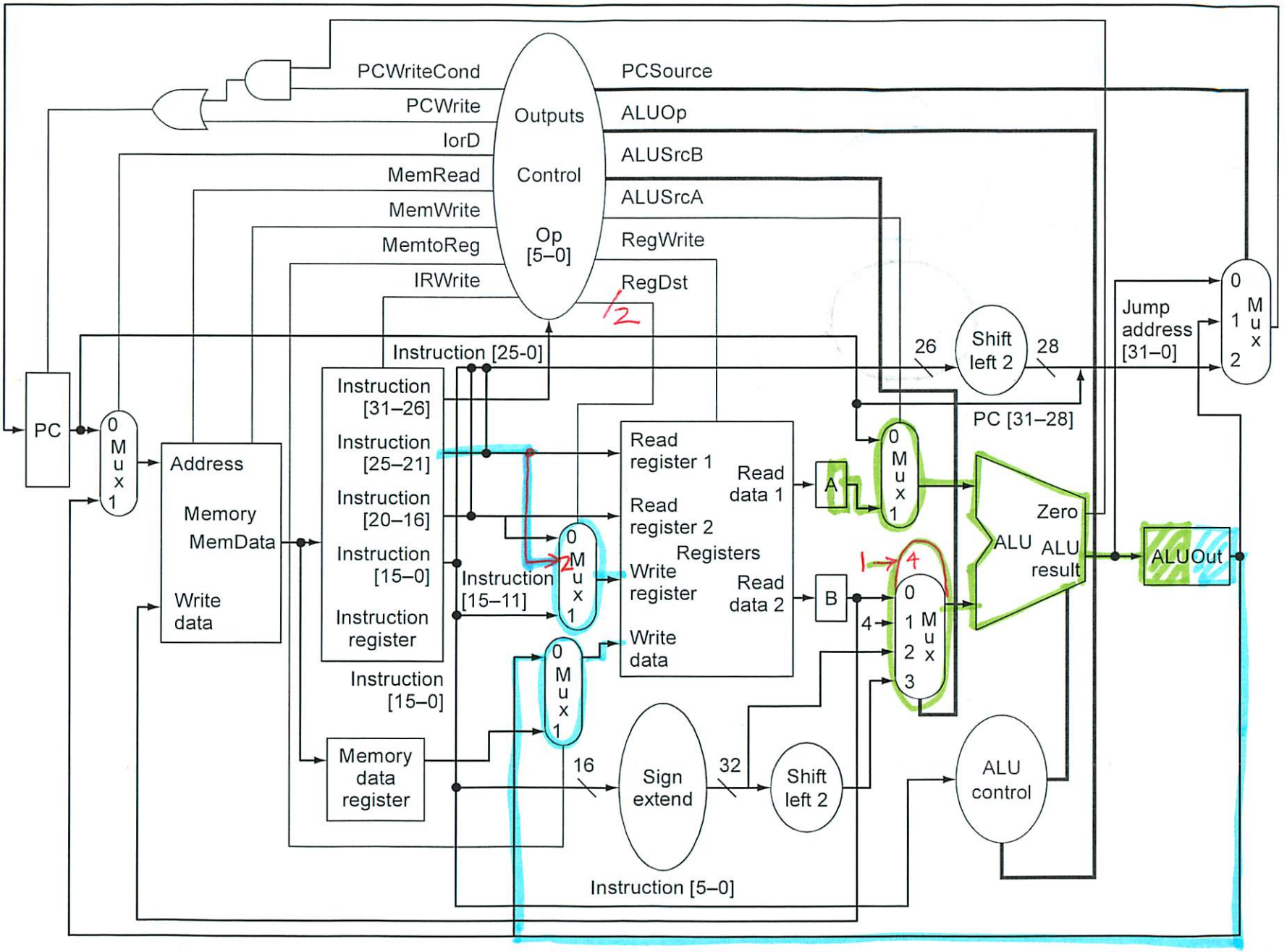
ALUSrcA=10
ALUSrcB=00
ALUOp=01
PCWriteCond
PCSource=01

bgtz #Assume rt = $0

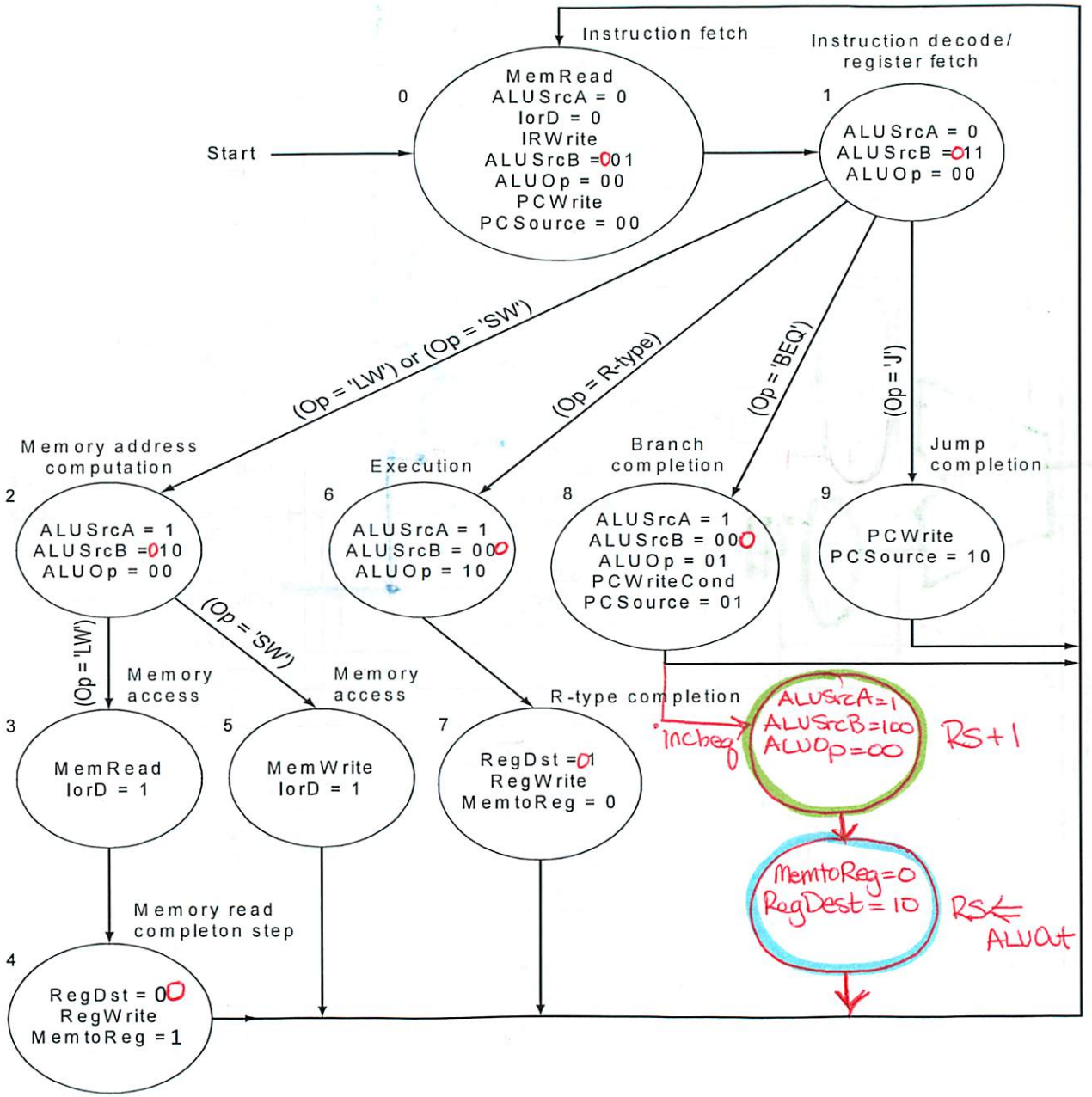



Instruction fetch
Instruction decode/
register fetch
0
MemRead
ALUSrcA = 0
IorD = 0
IRWrite
ALUSrcB = 01
ALUOp = 00
PCWrite
PCSource = 00
Start
1
ALUSrcA = 0
ALUSrcB = 11
ALUOp = 00
BGTZ
(Op = 'LW') or (Op = 'SW')
(Op = R-type)
(Op = 'BEQ')
(Op = 'J')
Memory address
computation
Execution
Branch
completion
Jump
completion
2
ALUSrcA = 1
ALUSrcB = 10
ALUOp = 00
6
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 10
8
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 01
PCWriteCond
PCSource = 01
BGTZ=0
9
PCWrite
PCSource = 10
ALUSrcA =1
ALUSrcB =00
ALUOp =01
PCWriteCond
PCSource=01
BGTZ=1
(Op = 'LW')
(Op = 'SW')
Memory
access
Memory
access
R-type completion
3
MemRead
IorD = 1
5
MemWrite
IorD = 1
7
RegDst = 1
RegWrite
MemtoReg = 0
Memory read
completon step
4
RegDst = 0
RegWrite
MemtoReg =1

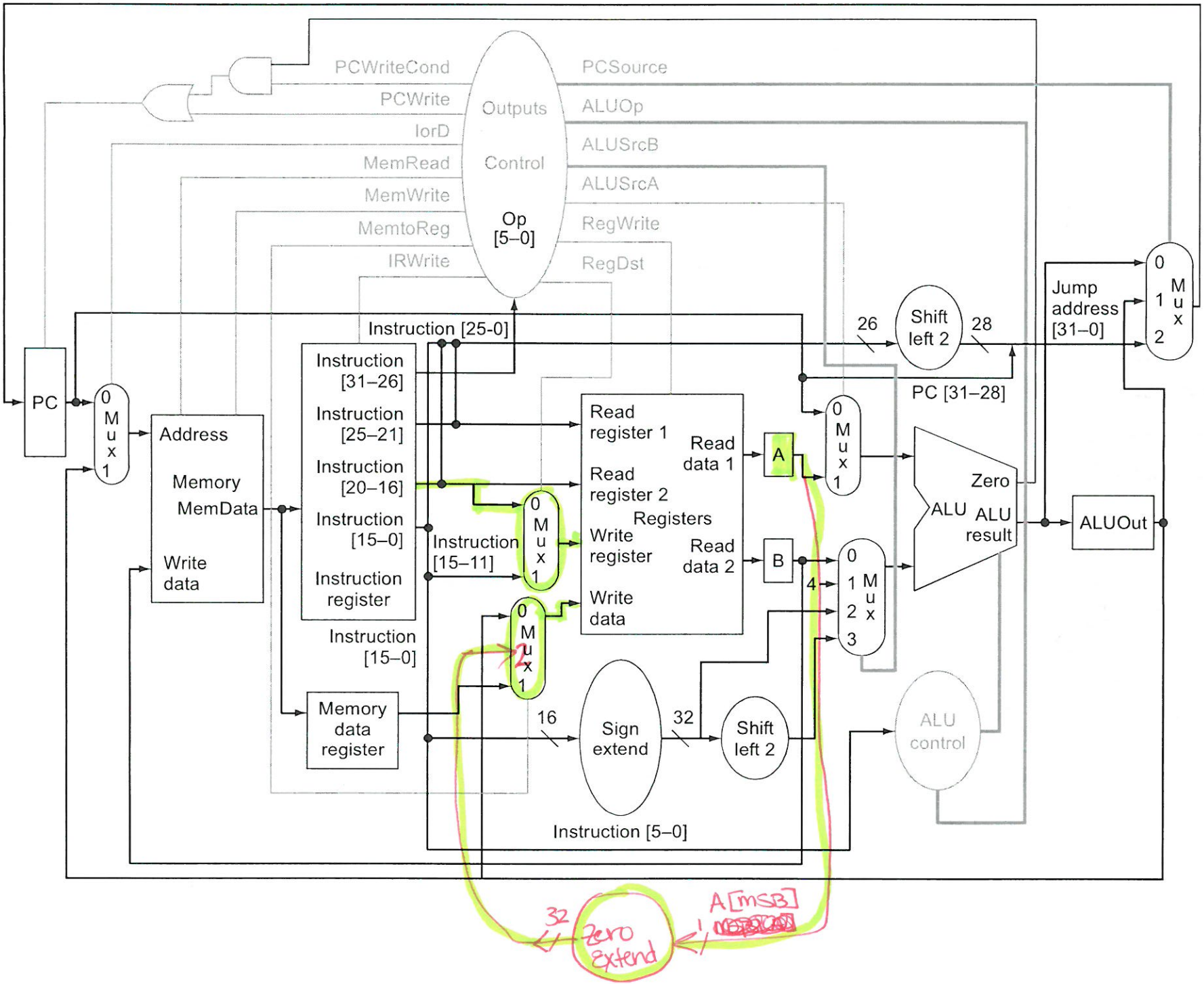
incbeq rs, rt, label

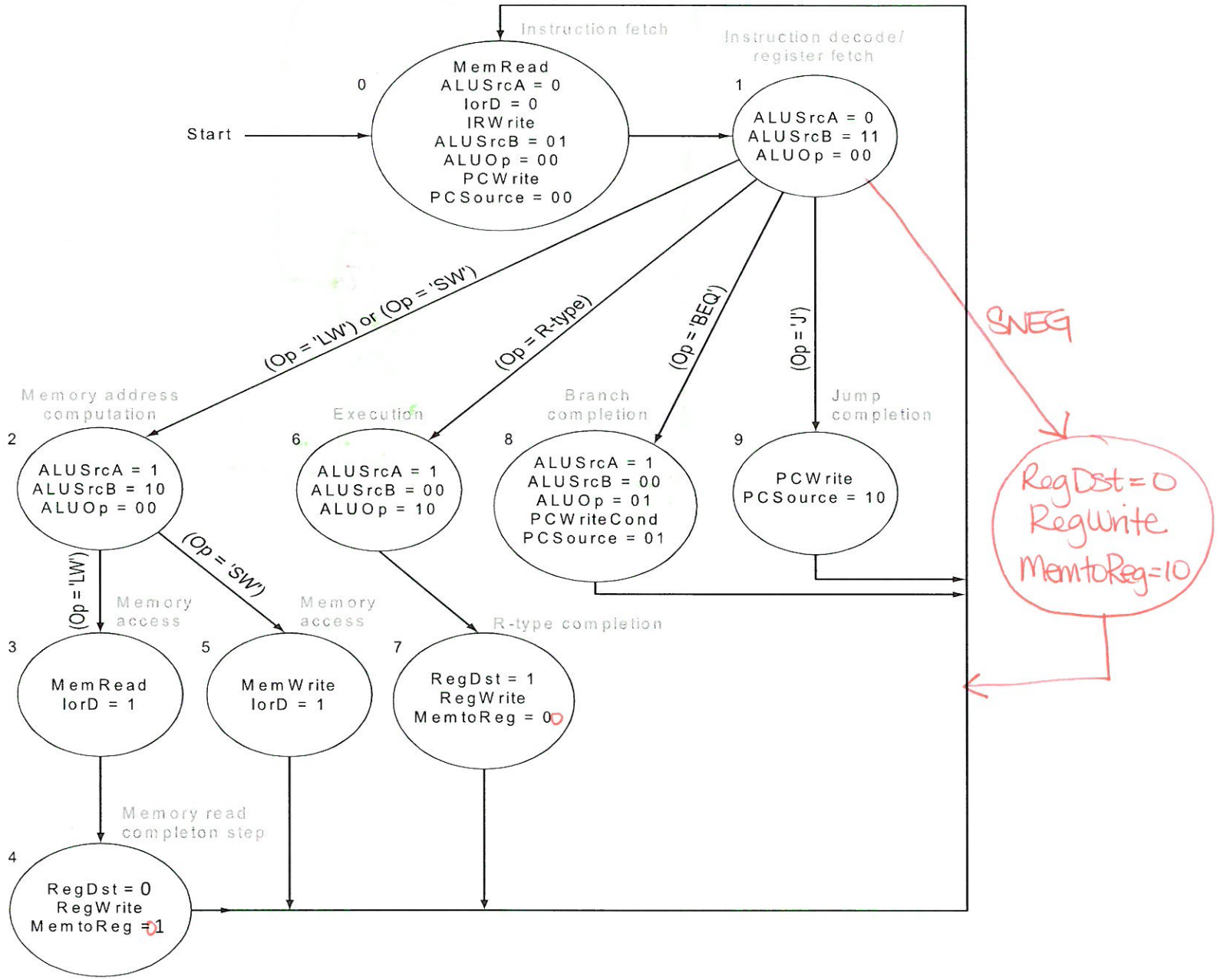


incbeq:

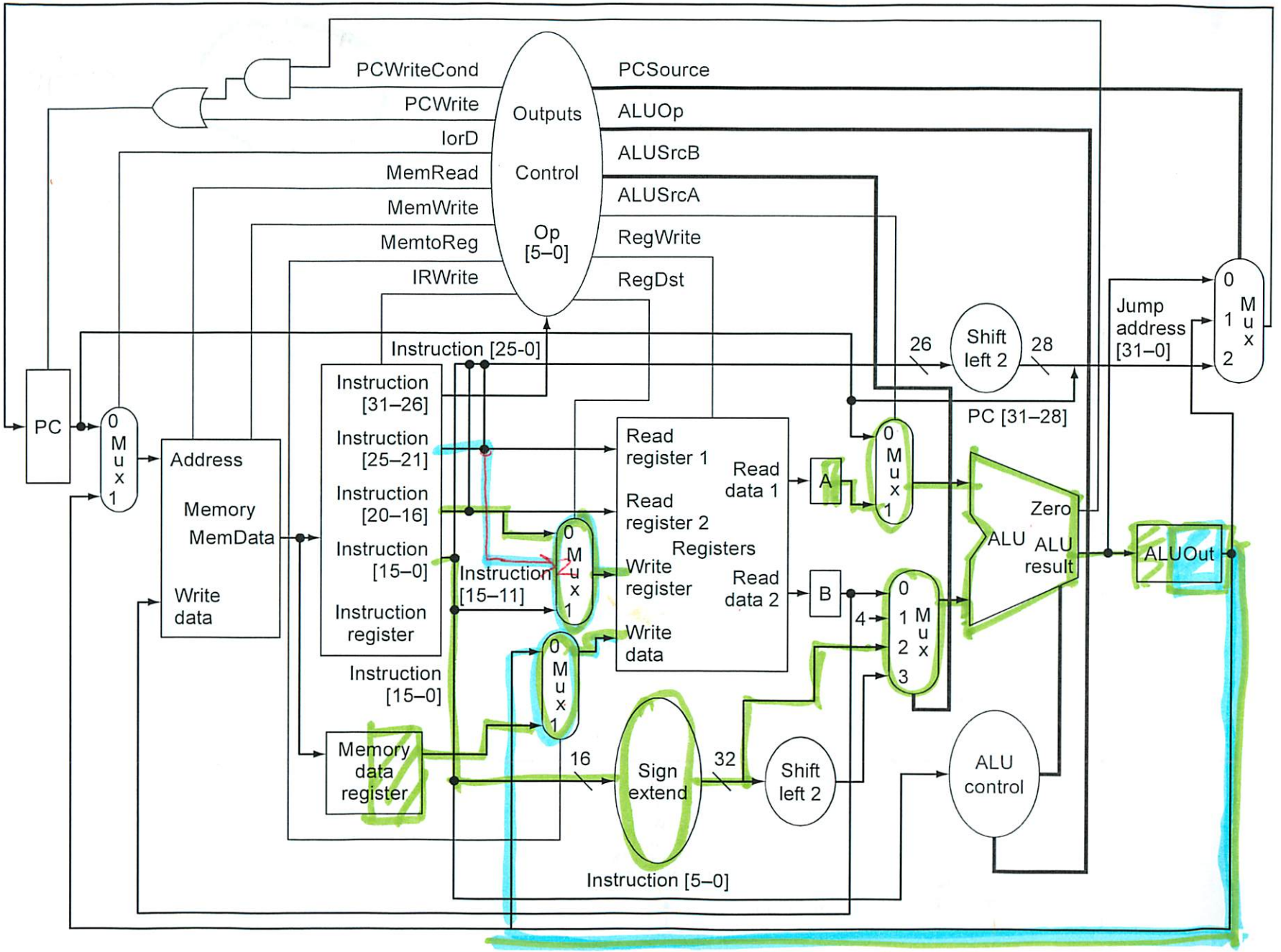

Instruction fetch
Instruction decode/
register fetch
0
MemRead
ALUSrcA = 0
IorD = 0
IRWrite
ALUSrcB =001
ALUOp = 00
PCWrite
PCSource = 00
Start
1
ALUSrcA = 0
ALUSrcB =011
ALUOp = 00
(Op = 'LW') or (Op = 'SW')
(Op = R-type)
(Op = 'BEQ')
(Op = 'J')
Memory address
computation
Execution
Branch
completion
Jump
completion
2
ALUSrcA = 1
ALUSrcB =010
ALUOp = 00
6
ALUSrcA = 1
ALUSrcB = 000
ALUOp = 10
8
ALUSrcA = 1
ALUSrcB = 000
ALUOp = 01
PCWriteCond
PCSource = 01
9
PCWrite
PCSource = 10
(Op = 'LW')
(Op = 'SW')
Memory
access
Memory
access
R-type completion
3
MemRead
IorD = 1
5
MemWrite
IorD = 1
7
RegDst =01
RegWrite
MemtoReg = 0
Incbeq
ALUSrcA=1
ALUSrcB=100
ALUOp=00
RS+1
110ps
MemtoReg=0
RegDest=10
RS←
ALUout
60ps
Memory read
completon step
4
RegDst = 00
RegWrite
MemtoReg =1

sneg

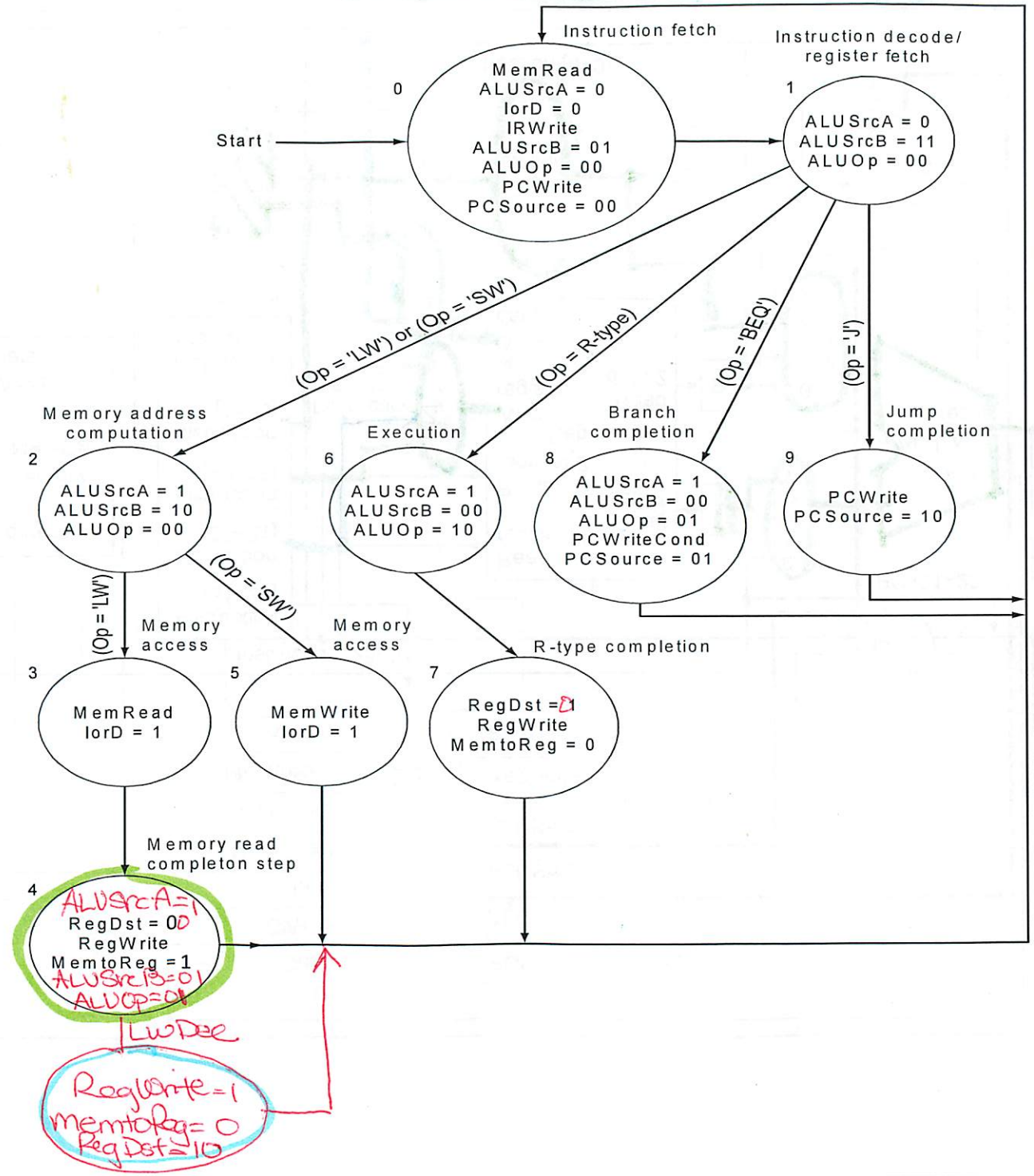

PCWriteCond
PCWrite
IorD
MemRead
MemWrite
MemtoReg
IRWrite
Outputs
Control
Op
[5–0]
PCSource
ALUOp
ALUSrcB
ALUSrcA
RegWrite
RegDst
Instruction [25-0]
26
Shift
left 2
28
Jump
address
[31–0]
PC [31–28]
PC
Address
Memory
MemData
Write
data
Instruction
[31–26]
Instruction
[25–21]
Instruction
[20–16]
Instruction
[15–0]
Instruction
register
Instruction
[15–11]
Instruction
[15–0]
Read
register 1
Read
register 2
Registers
Write
register
Write
data
Read
data 1
Read
data 2
A
B
4
Zero
ALU
ALU
result
ALUOut
Memory
data
register
16
Sign
extend
32
Shift
left 2
ALU
control
Instruction [5–0]
32
Zero
extend
A[msb]


Instruction fetch
Instruction decode/
register fetch
0
MemRead
ALUSrcA = 0
IorD = 0
IRWrite
ALUSrcB = 01
ALUOp = 00
PCWrite
PCSource = 00
Start
1
ALUSrcA = 0
ALUSrcB = 11
ALUOp = 00
SNEG
(Op = 'LW') or (Op = 'SW')
(Op = R-type)
(Op = 'BEQ')
(Op = 'J')
Memory address
computation
Execution
Branch
completion
Jump
completion
2
ALUSrcA = 1
ALUSrcB = 10
ALUOp = 00
6
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 10
8
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 01
PCWriteCond
PCSource = 01
9
PCWrite
PCSource = 10
RegDst=0
RegWrite
MemtoReg=10
(Op = 'LW')
(Op = 'SW')
Memory
access
Memory
access
R-type completion
3
MemRead
IorD = 1
5
MemWrite
IorD = 1
7
RegDst = 1
RegWrite
MemtoReg = 0
Memory read
completon step
4
RegDst = 0
RegWrite
MemtoReg = 1

LDDec.


PCWriteCond
PCWrite
IorD
MemRead
MemWrite
MemtoReg
IRWrite
Outputs
Control
Op
[5–0]
PCSource
ALUOp
ALUSrcB
ALUSrcA
RegWrite
RegDst
Instruction [25-0]
26
Shift
left 2
28
PC [31–28]
Jump
address
[31–0]
M
u
x
0
1
2
PC
Address
Memory
MemData
Write
data
Instruction
[31–26]
Instruction
[25–21]
Instruction
[20–16]
Instruction
[15–0]
Instruction
register
Instruction
[15–11]
Instruction
[15–0]
Memory
data
register
Read
register 1
Read
register 2
Registers
Write
register
Write
data
Read
data 1
Read
data 2
A
B
4
Zero
ALU
ALU
result
ALUOut
16
Sign
extend
32
Shift
left 2
ALU
control
Instruction [5–0]

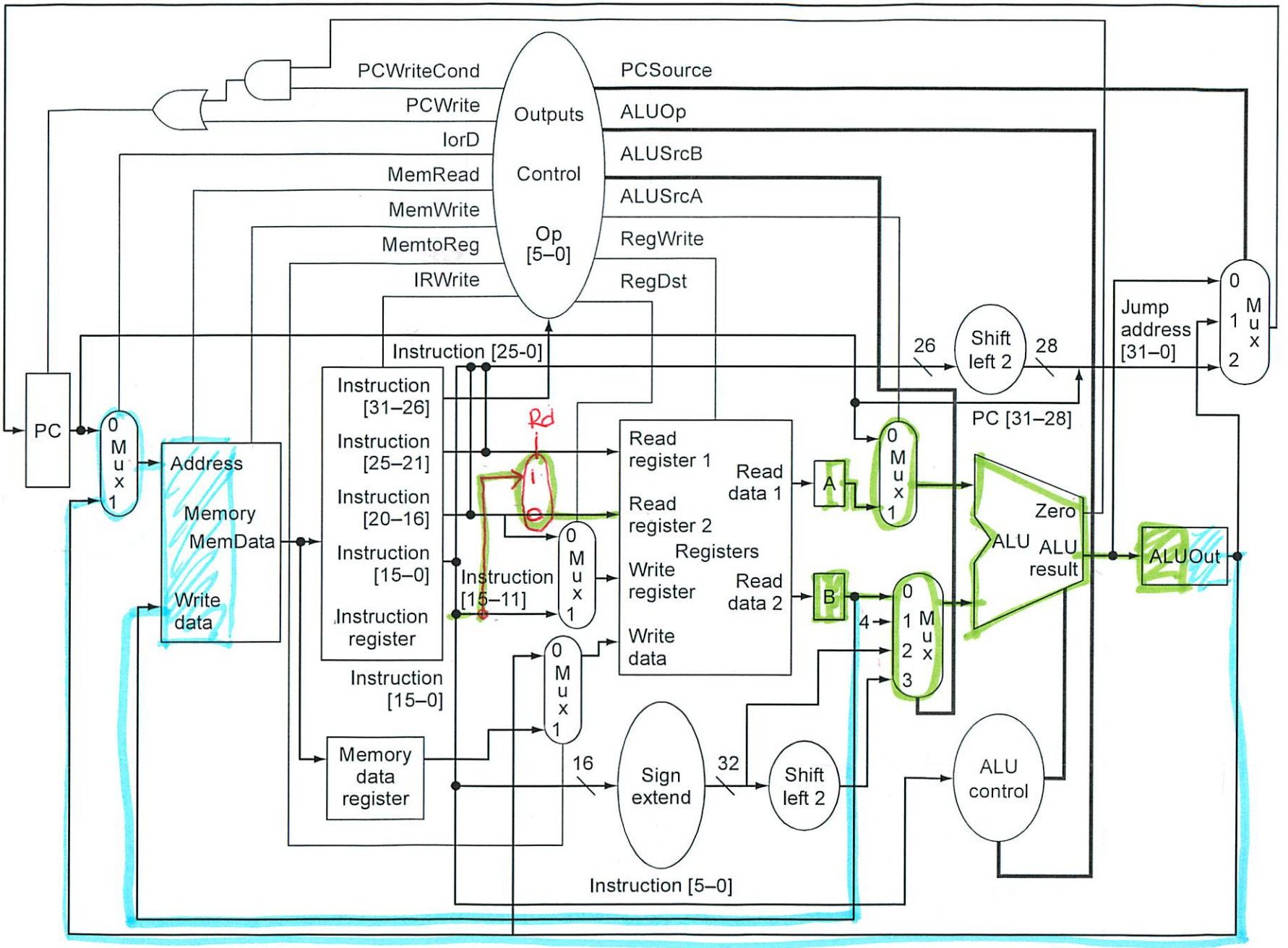
LD Dec.



110ps

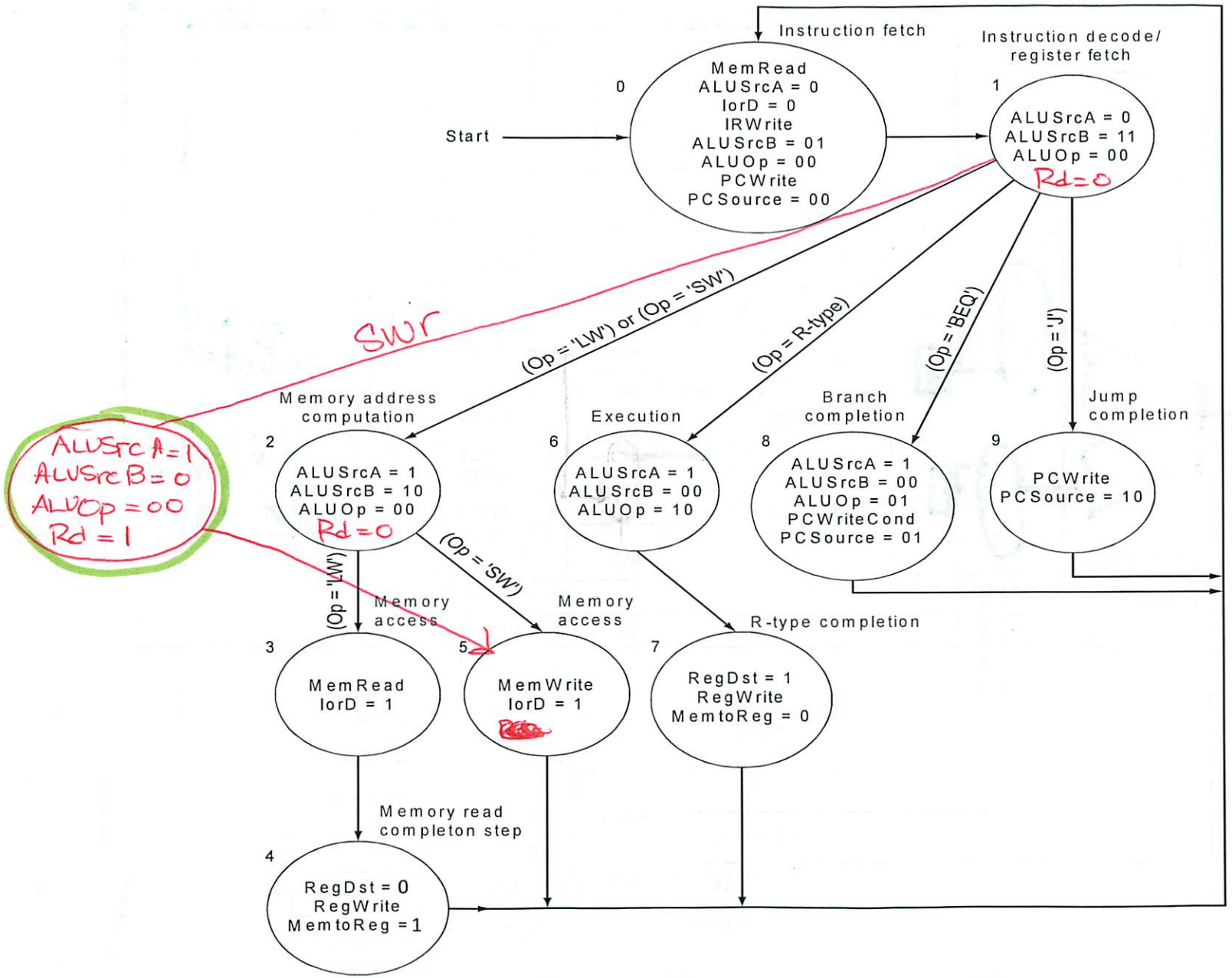
60ps

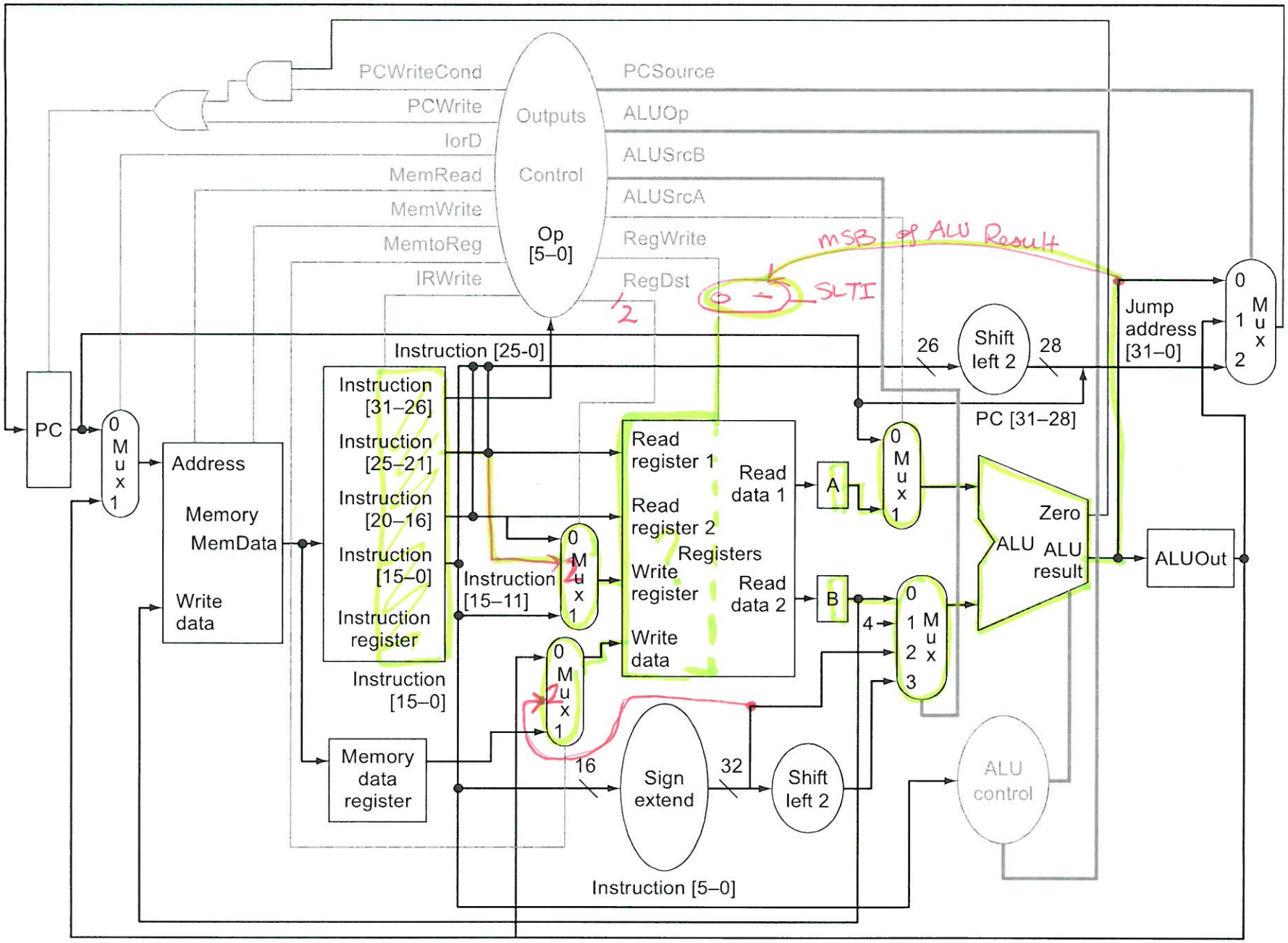
swr rd, rt(rs) Mem[Reg[rs] + Reg[Rt]] = Reg[rd]

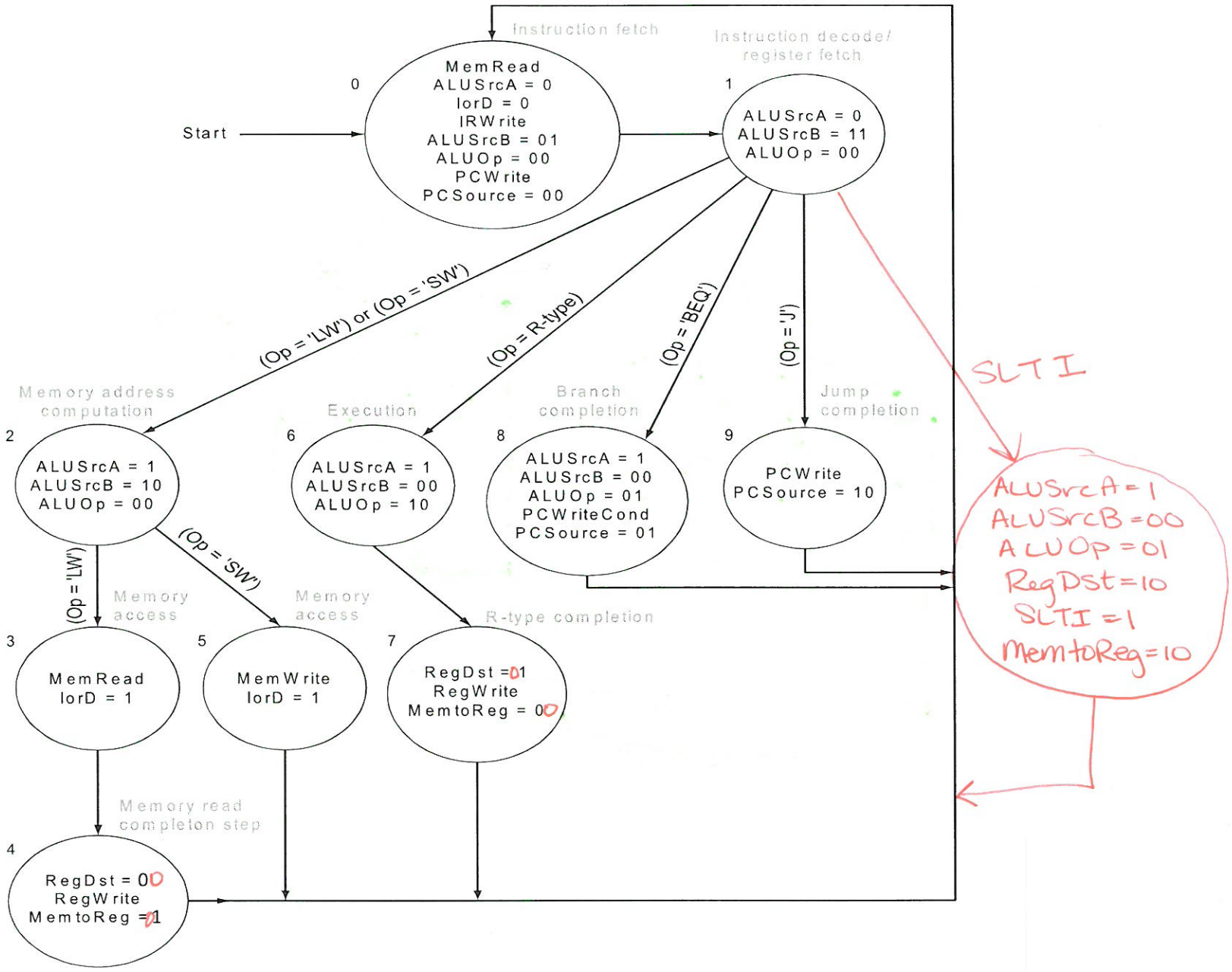


new Stage 110 ps

Swr

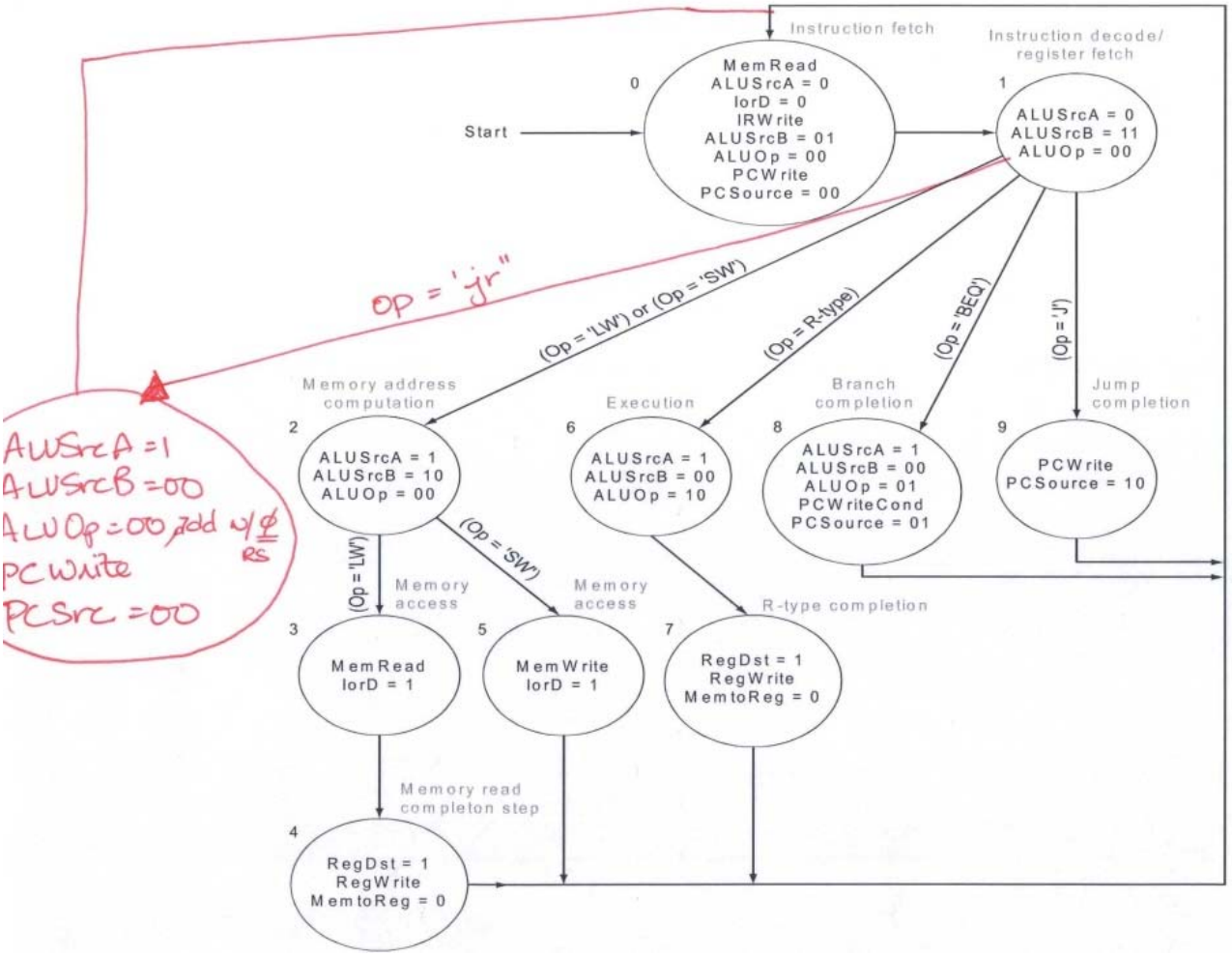

Instruction fetch
Instruction decode/
register fetch
0
MemRead
ALUSrcA = 0
IorD = 0
IRWrite
ALUSrcB = 01
ALUOp = 00
PCWrite
PCSource = 00
Start
1
ALUSrcA = 0
ALUSrcB = 11
ALUOp = 00
Rd=0
Swr
(Op = 'LW') or (Op = 'SW')
(Op = R-type)
(Op = 'BEQ')
(Op = 'J')
Memory address
computation
Execution
Branch
completion
Jump
completion
2
ALUSrcA = 1
ALUSrcB = 10
ALUOp = 00
Rd=0
6
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 10
8
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 01
PCWriteCond
PCSource = 01
9
PCWrite
PCSource = 10
ALUSrcA=1
ALUSrcB=0
ALUOp=00
Rd=1
(Op = 'LW')
(Op = 'SW')
Memory
access
Memory
access
R-type completion
3
MemRead
IorD = 1
5
MemWrite
IorD = 1
7
RegDst = 1
RegWrite
MemtoReg = 0
Memory read
completon step
4
RegDst = 0
RegWrite
MemtoReg =1


PCWriteCond
PCWrite
IorD
MemRead
MemWrite
MemtoReg
IRWrite
Outputs
Control
Op
[5–0]
PCSource
ALUOp
ALUSrcB
ALUSrcA
RegWrite
RegDst
MSB of ALU Result
SLTI
1/2
Jump
address
[31–0]
Instruction [25-0]
26
Shift
left 2
28
PC [31–28]
PC
Address
Memory
MemData
Write
data
Instruction
[31–26]
Instruction
[25–21]
Instruction
[20–16]
Instruction
[15–0]
Instruction
register
Instruction
[15–0]
Instruction
[15–11]
Read
register 1
Read
register 2
Registers
Write
register
Write
data
Read
data 1
Read
data 2
A
B
4
Zero
ALU
ALU
result
ALUOut
Memory
data
register
16
Sign
extend
32
Shift
left 2
ALU
control
Instruction [5–0]
M
u
x


Instruction fetch
Instruction decode/
register fetch
0
MemRead
ALUSrcA = 0
IorD = 0
IRWrite
ALUSrcB = 01
ALUOp = 00
PCWrite
PCSource = 00
Start
1
ALUSrcA = 0
ALUSrcB = 11
ALUOp = 00
(Op = 'LW') or (Op = 'SW')
(Op = R-type)
(Op = 'BEQ')
(Op = 'J')
Memory address
computation
Execution
Branch
completion
Jump
completion
2
ALUSrcA = 1
ALUSrcB = 10
ALUOp = 00
6
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 10
8
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 01
PCWriteCond
PCSource = 01
9
PCWrite
PCSource = 10
(Op = 'LW')
(Op = 'SW')
Memory
access
Memory
access
R-type completion
3
MemRead
IorD = 1
5
MemWrite
IorD = 1
7
RegDst = 01
RegWrite
MemtoReg = 00
Memory read
completon step
4
RegDst = 00
RegWrite
MemtoReg = 01
SLTI
ALUSrcA=1
ALUSrcB=00
ALUOp=01
RegDst=10
SLTI=1
MemtoReg=10

Problem 4: Show how the jump register instruction 'jr' can be implemented to the multi-cycle datapath by simply making changes to the finite state machine. (Hint: $0 = $zero = 0)

rt field of the instruction should be register $0, which always holds the value 0.



Problem 5: Calculate the delay in the datapath after the addition of ALL instructions in Problem 1. Assume the same delays. What is the minimum clock cycle period at which this design can operate properly?
The answer will depend on how you implemented each instruction

Problem 6: Consider changes to the original multi-cycle datapath that alters the register file so that it has only one read port. Describe any changes that will need to be made to the datapath in order to support this modification. Use the datapath diagram to illustrate the changes. Modify the finite state machine to indicate how the instructions will work given your new datapath.

You will need to add control lines to Write registers A and B and a multiplexer (with a control line) to select between the Rs and Rt fields of the instruction.

State 1 is modified with select Rs and WriteA asserted. A new state has to be added before states 6 and 8 to select Rt and WriteB. After the new state control branches to either state 6 or state 8 based upon the op-code. Other states are not affected as on R-Format and branch instruction use input from two registers. Additionally, State 2 must be modified to additionally read the Rt register and store the contents in B. While, lw does not require to read Rt, the store word does. By making the modification to State 2, where the Register file is not

being used, we do not increase the number of clock cycles required for a sw instruction. Control signals to select Rt and WriteB must be specified.

Problem 7: Consider eliminating the two shift left by 2 units in the multicycle datapath. Instead of these units, the ALU will be used for shifting the values instead. What impact does this have on the original 5 instructions (lw, sw, R-type, beq, j)?

If the shift left by 2 is eliminated, then any time this happens will require that the ALU is used.
First, since the Decode stage adds the immediate value shifted left by 2 to calculate the branch address, now a stage in between the Fetch and Decode stages would need to be added to do the shifting of the sign extended value. The value would be in ALUOut and the input 3 of ALUSrcB would need to modified to be the valued from the ALUOut register.

The second Shift left unit is used in satge 9 for the Jump instruction. If this was removed, the datapath would need to go through the ALU and the PCSource mus could eliminate input 2, and use the value out of the ALU instead.

Problem 8: In the single cycle datapath control unit for the original 5 instructions (lw, sw, R-type, beq, j), the MemtoReg control signal can be eliminated and the MemRead or ALUSrc control signals can be used to control the multiplexor instead. This reduces the number of control signals required (less logic gates to implement).

a. Which other signals in the single cycle datapath can be eliminated and replaced by another existing control signal, or the inverse (NOT) of the signal.

   RegDst and ALUOp1
   RegDst & MemRead'
   Branch & AluOp0

b. Are there any signals in the multi-cycle datapath (basic 5 instructions only) which can be eliminated and replaced?

   Yes, MemtoReg is only used ni Stage 4 &7. It can be replaced with RegDst' (which is also only used in Stage 4 & 7)