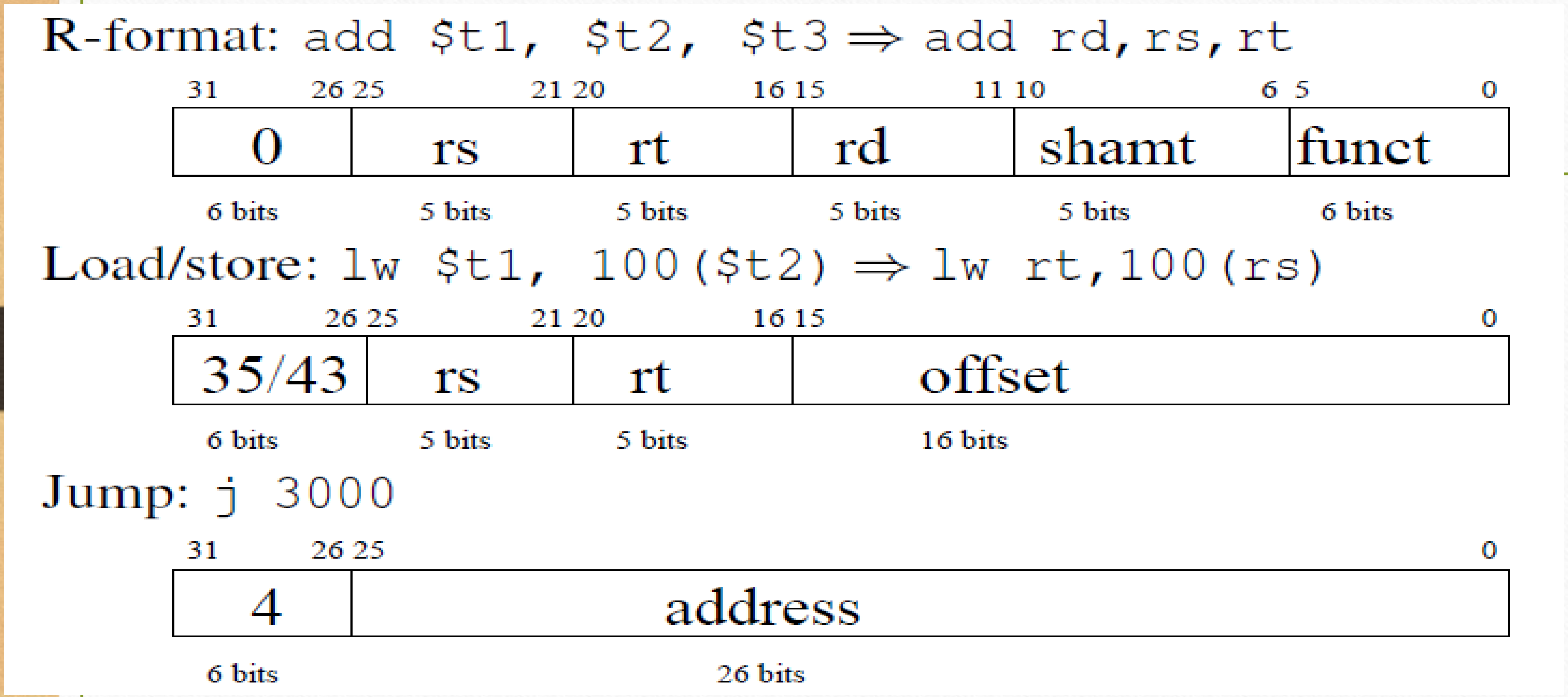
Single Cycle Architecture

Part 3

How the Instruction Bits are Broken up:



R-format: `add $t1, $t2, $t3` ⇒ `add rd,rs,rt`

31 – 26	25 – 21	20 – 16	15 – 11	10 – 6	5 – 0
0	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Load/store: `lw $t1, 100($t2)` ⇒ `lw rt,100(rs)`

31 – 26	25 – 21	20 – 16	15 – 0
35/43	rs	rt	offset
6 bits	5 bits	5 bits	16 bits

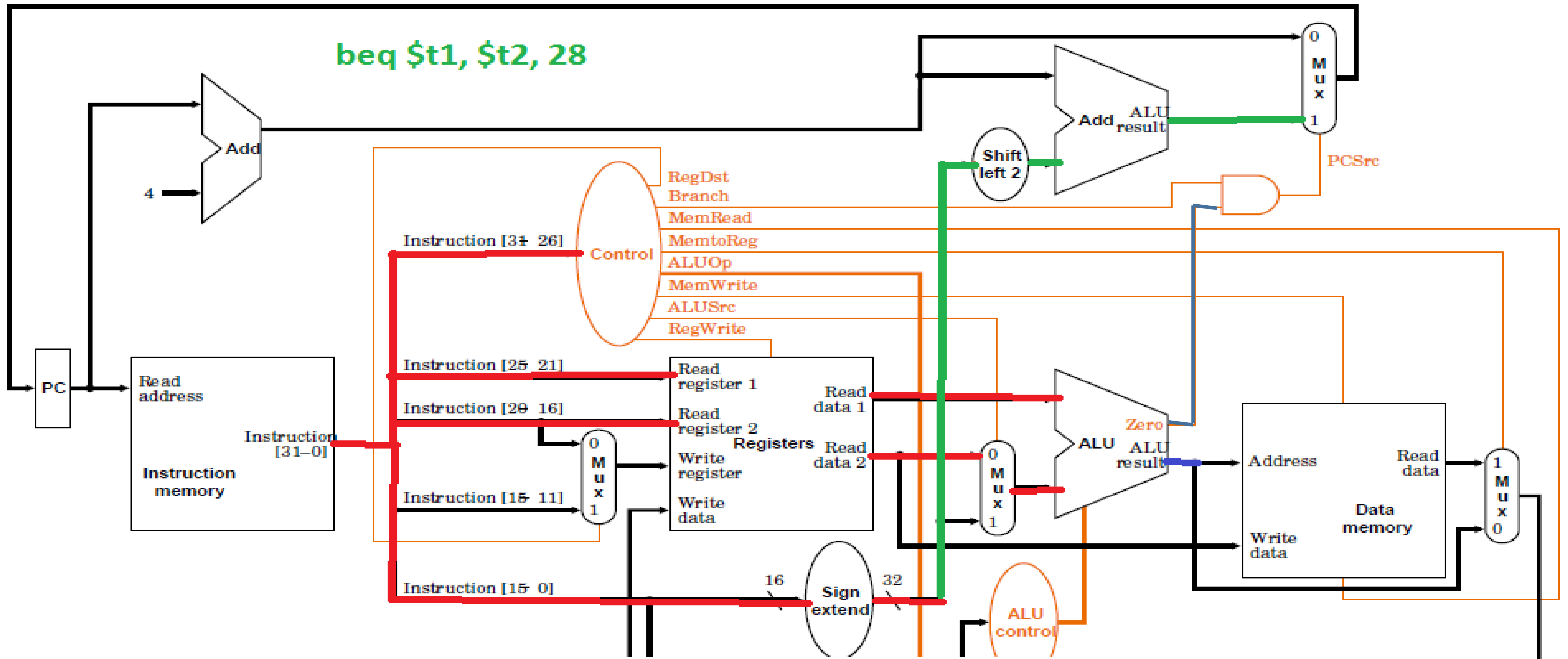
Jump: `j 3000`

31 – 26	25 – 0
4	address
6 bits	26 bits

sll, sub

beq, subi, addi

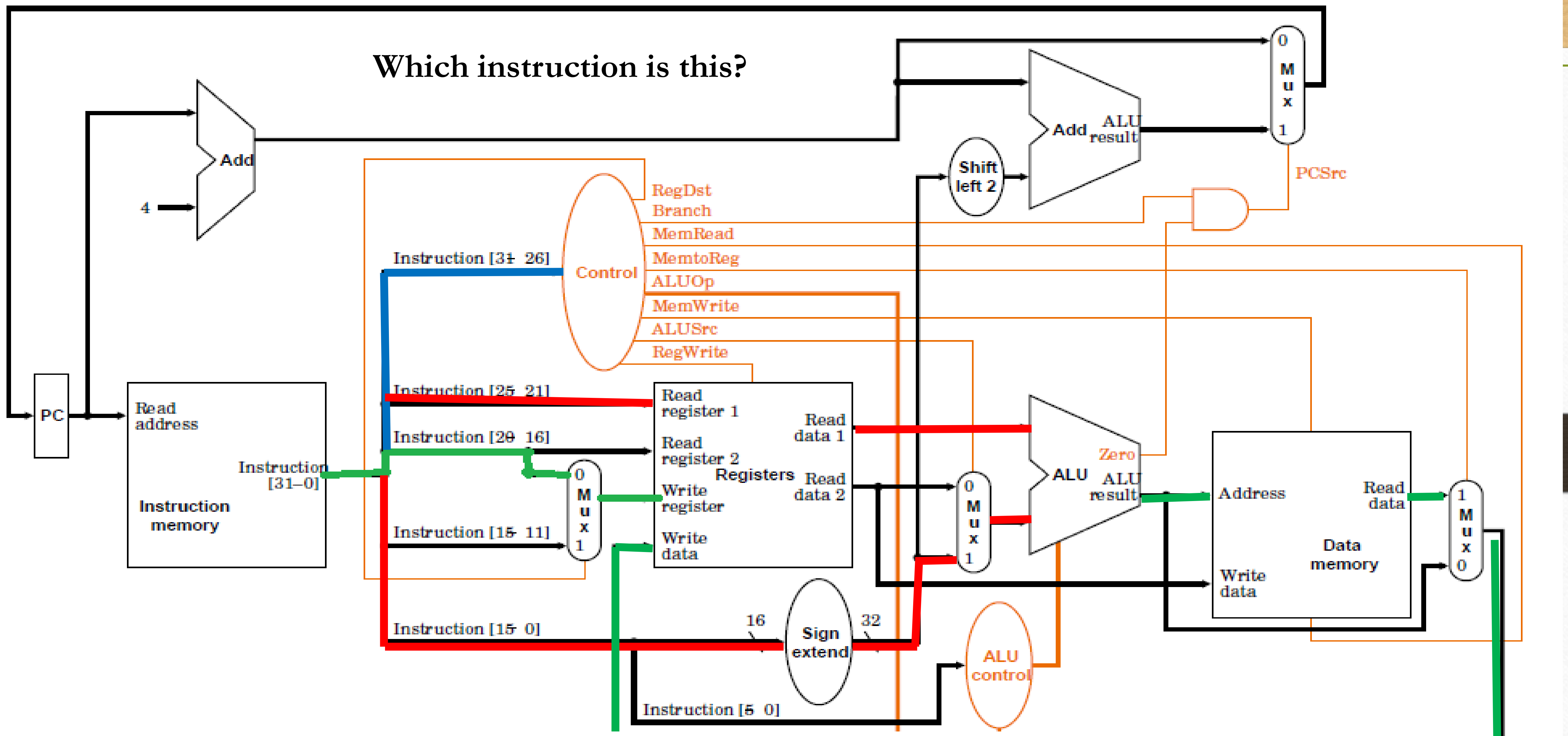
special



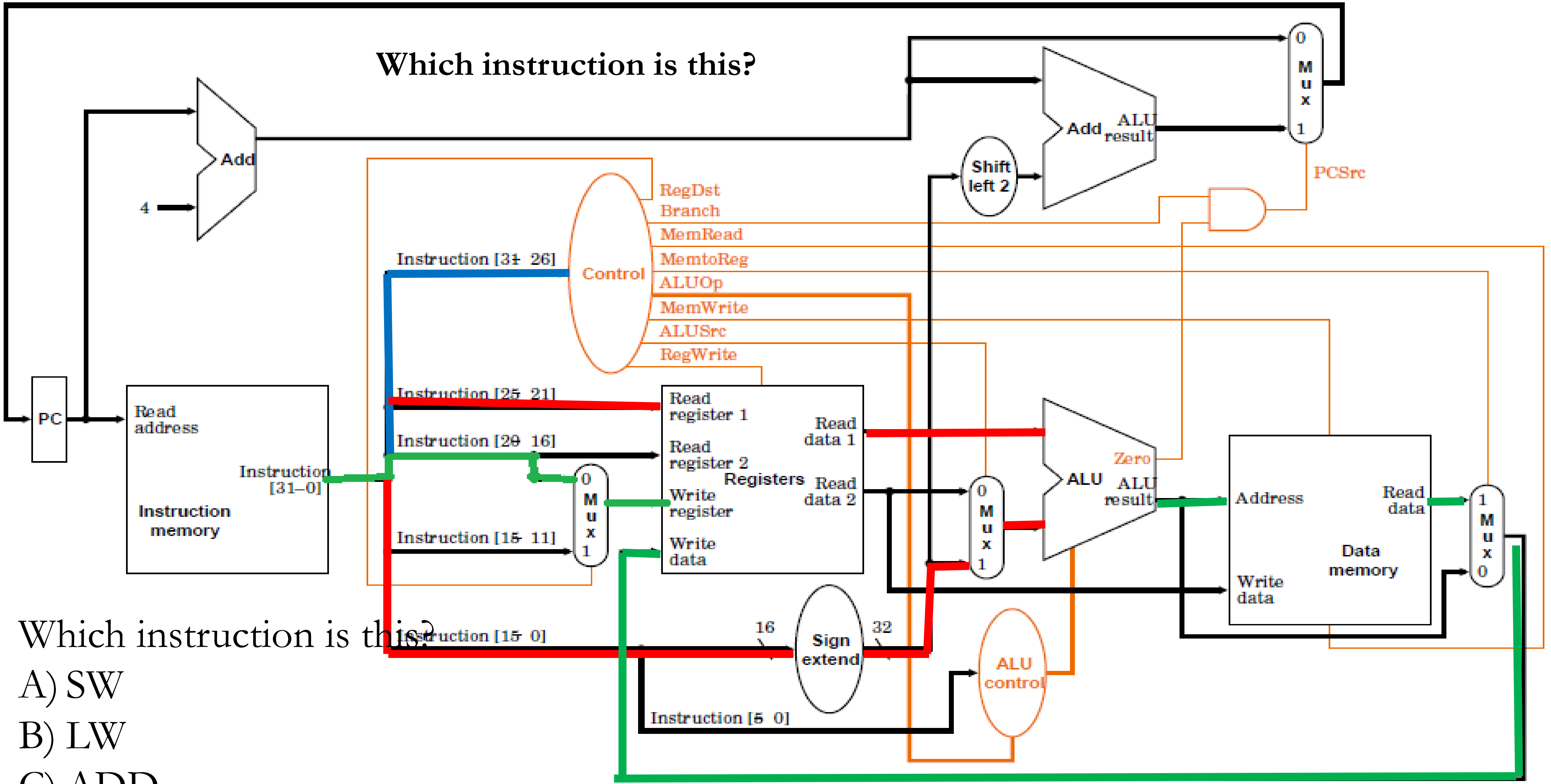
Type	Reg Dst	ALU Src	Mem ToReg	Reg Write	Mem Read	Mem Write	Branch	ALU op1	ALU op0
R-format	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

Tells you exactly what the control lines
Need to be for each instruction

Which instruction is this?



Type	Reg Dst	ALU Src	Mem ToReg	Reg Write	Mem Read	Mem Write	Branch	ALU op1	ALU op0
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



Which instruction is this?

A) SW

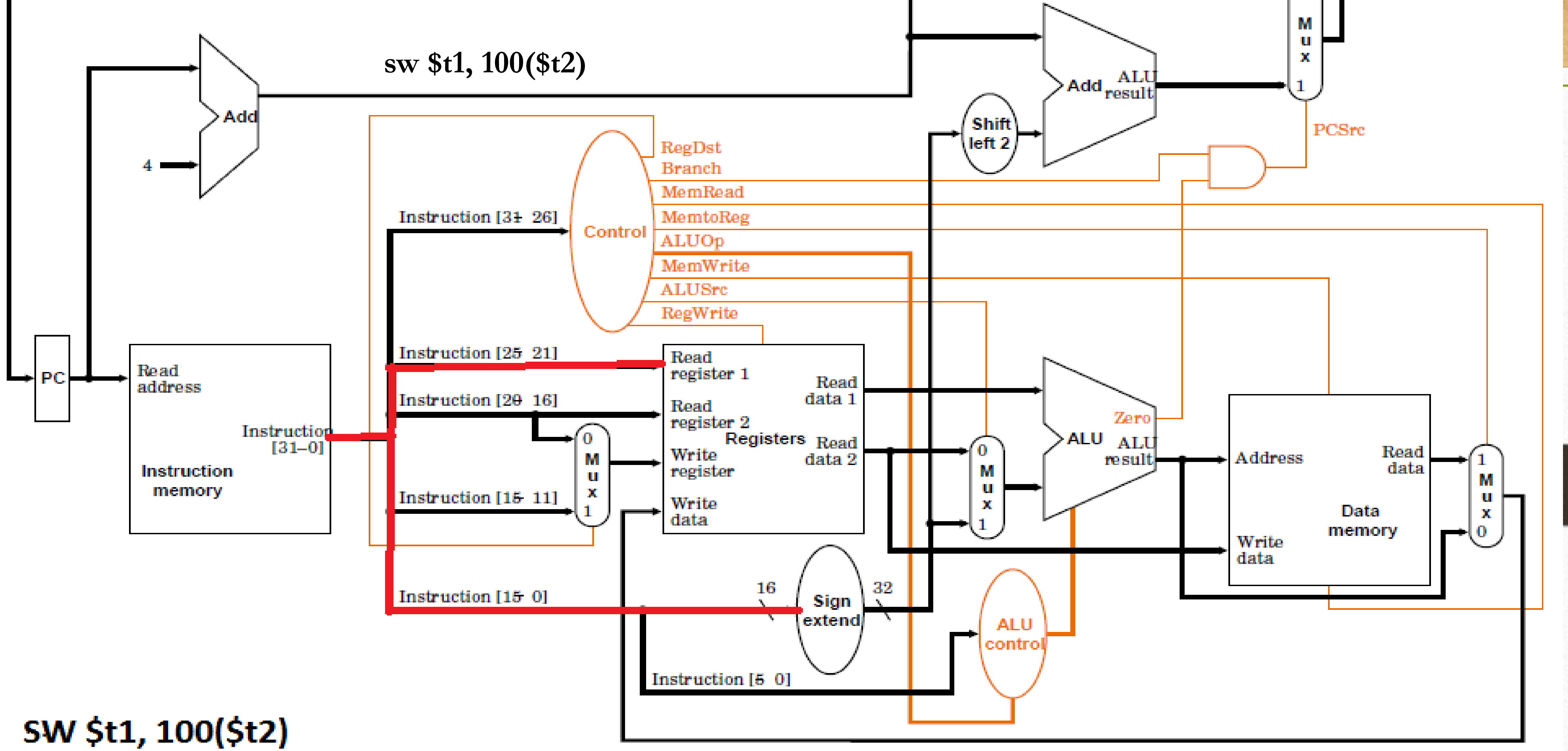
B) LW

C) ADD

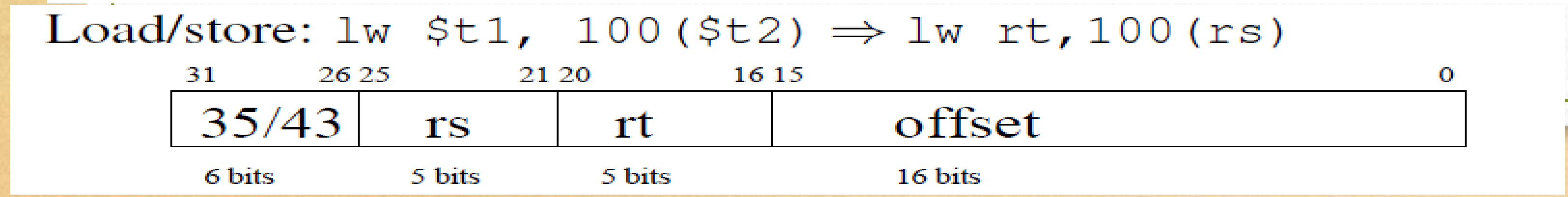
D) SUB

E) ANY

Control Units : Course Notes



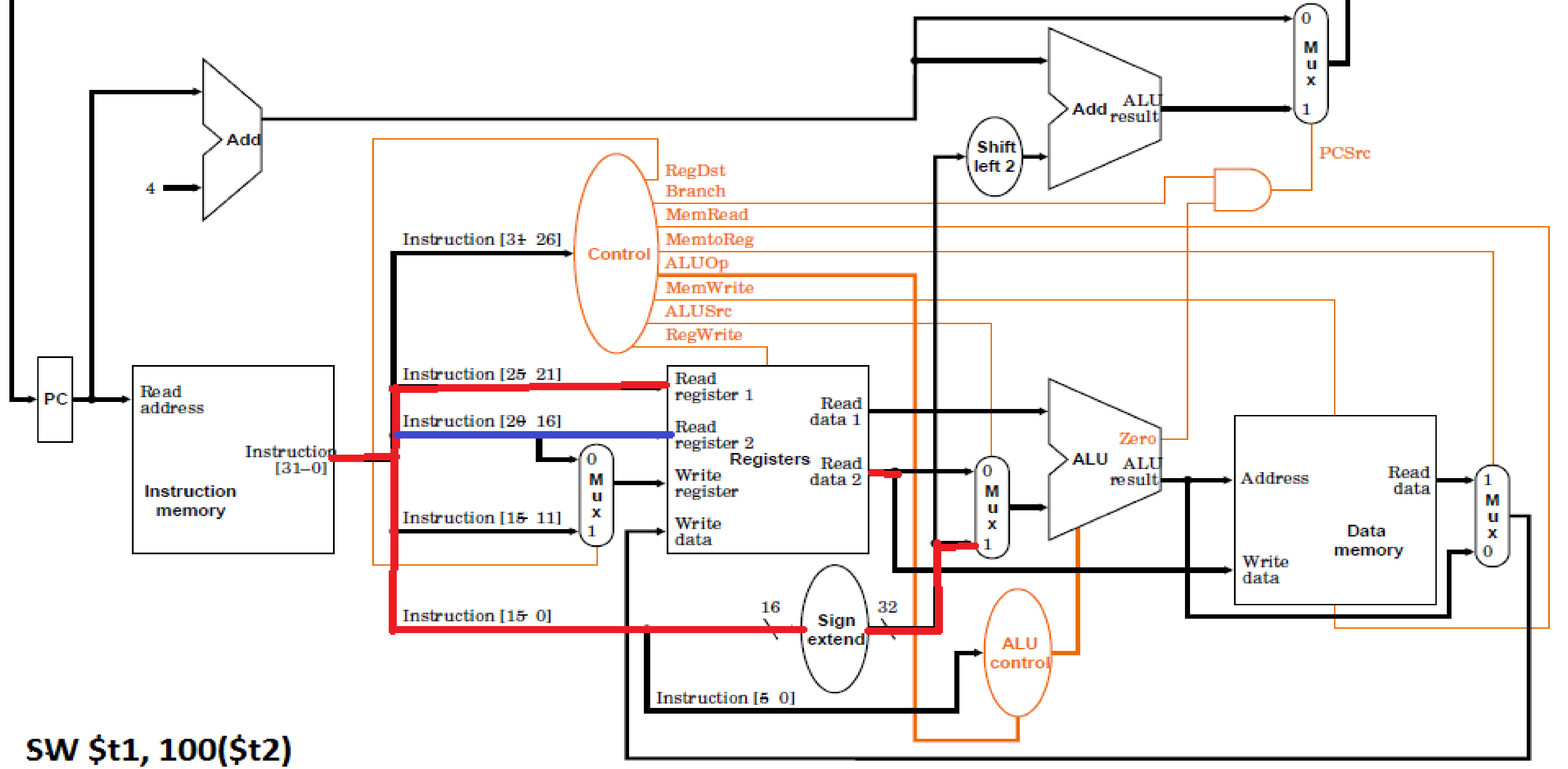
SW $t1, 100($t2)



Load/store: `lw $t1, 100($t2) ⇒ lw rt,100(rs)`

31–26	25–21	20–16	15–0
35/43	rs	rt	offset
6 bits	5 bits	5 bits	16 bits

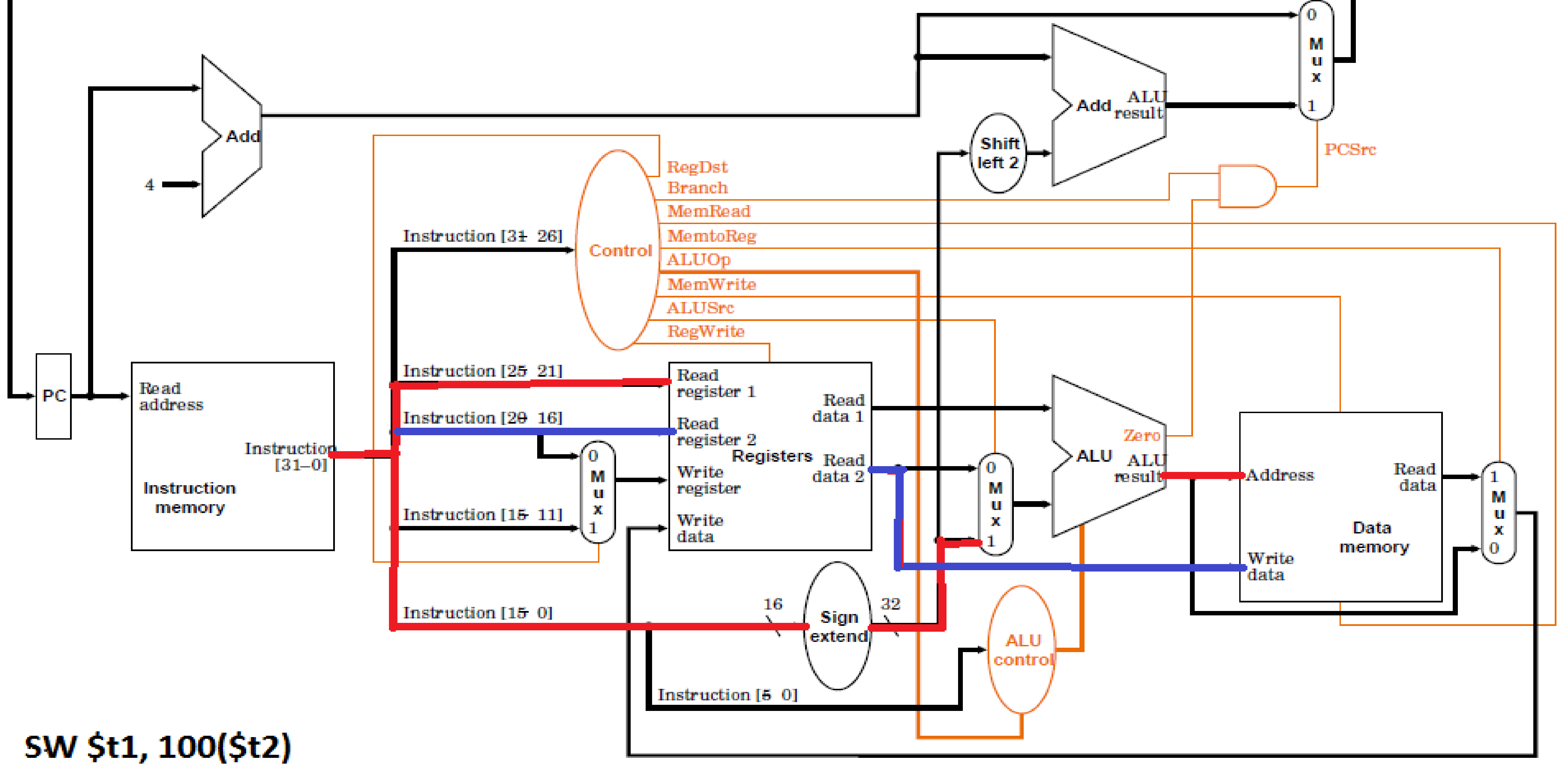
SV



SW $t1, 100($t2)

Load/store: `lw $t1, 100($t2) ⇒ lw rt,100(rs)`

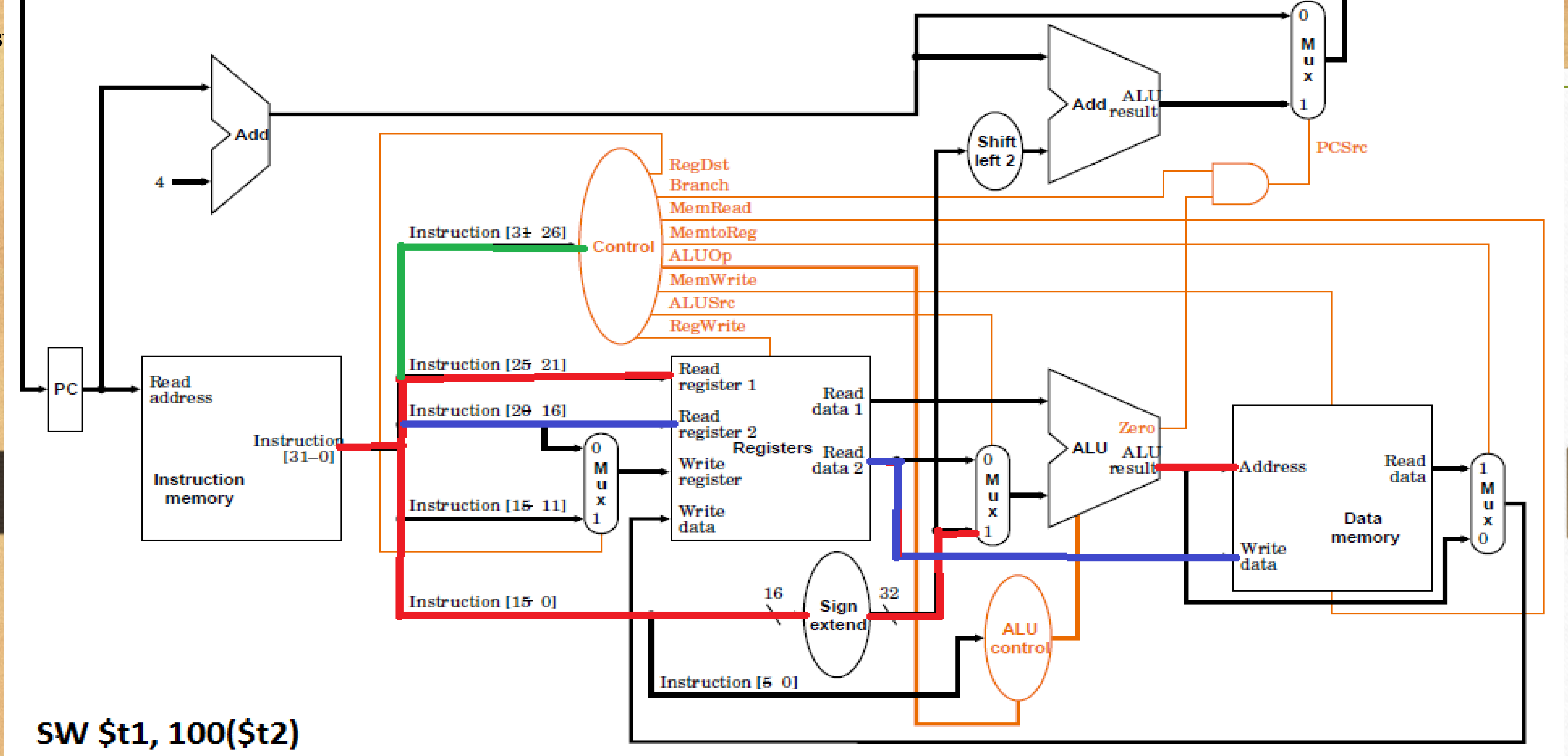
31 – 26	25 – 21	20 – 16	15 – 0
35/43	rs	rt	offset
6 bits	5 bits	5 bits	16 bits



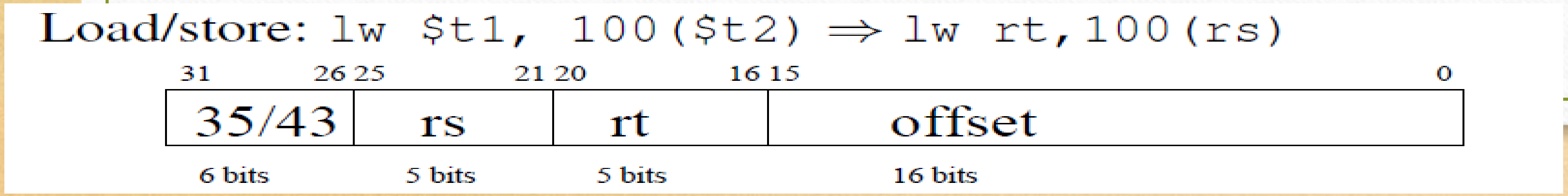
SW $t1, 100($t2)

Load/store: lw $t1, 100($t2) ⇒ lw rt, 100(rs)

31–26	25–21	20–16	15–0
35/43	rs	rt	offset
6 bits	5 bits	5 bits	16 bits



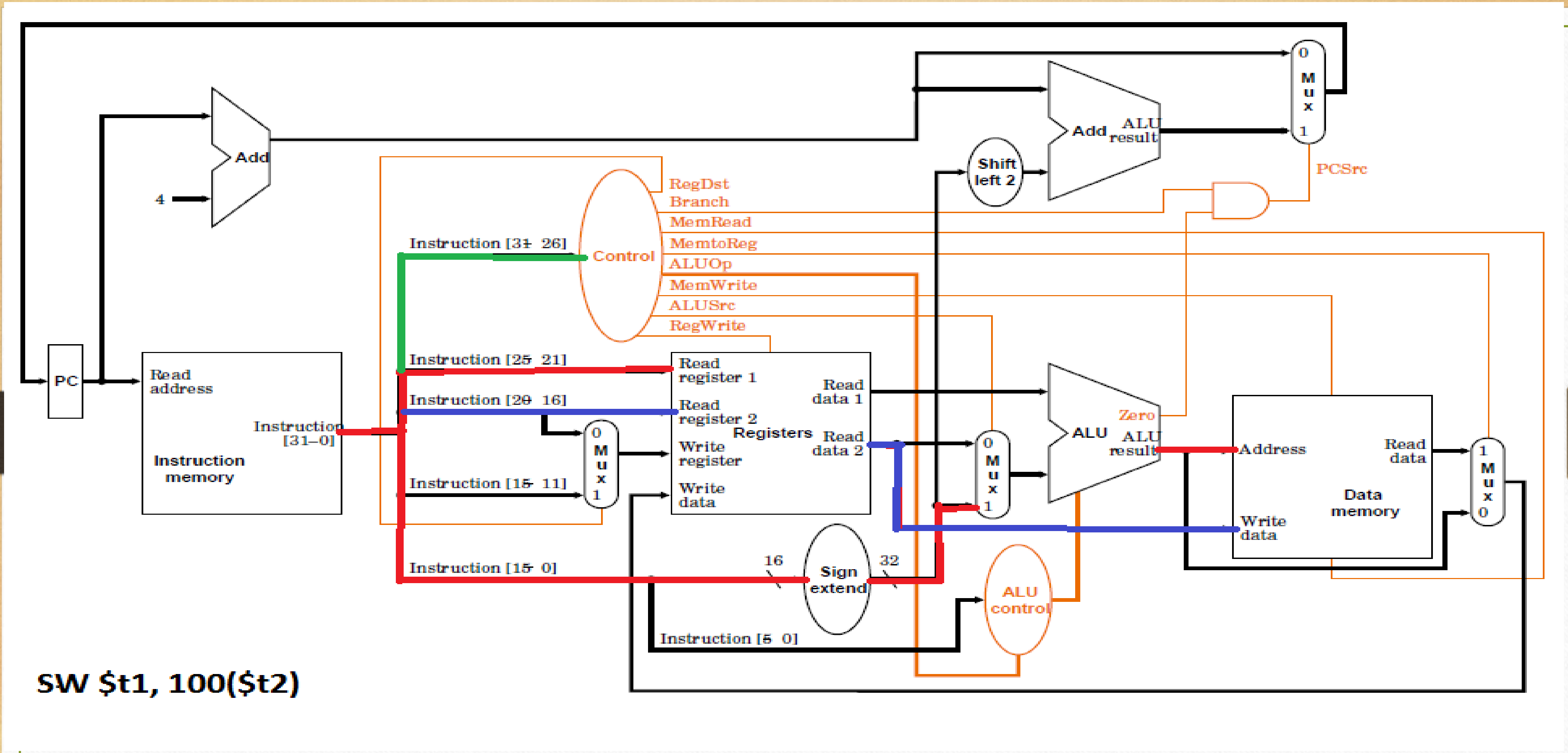
SW $t1, 100($t2)

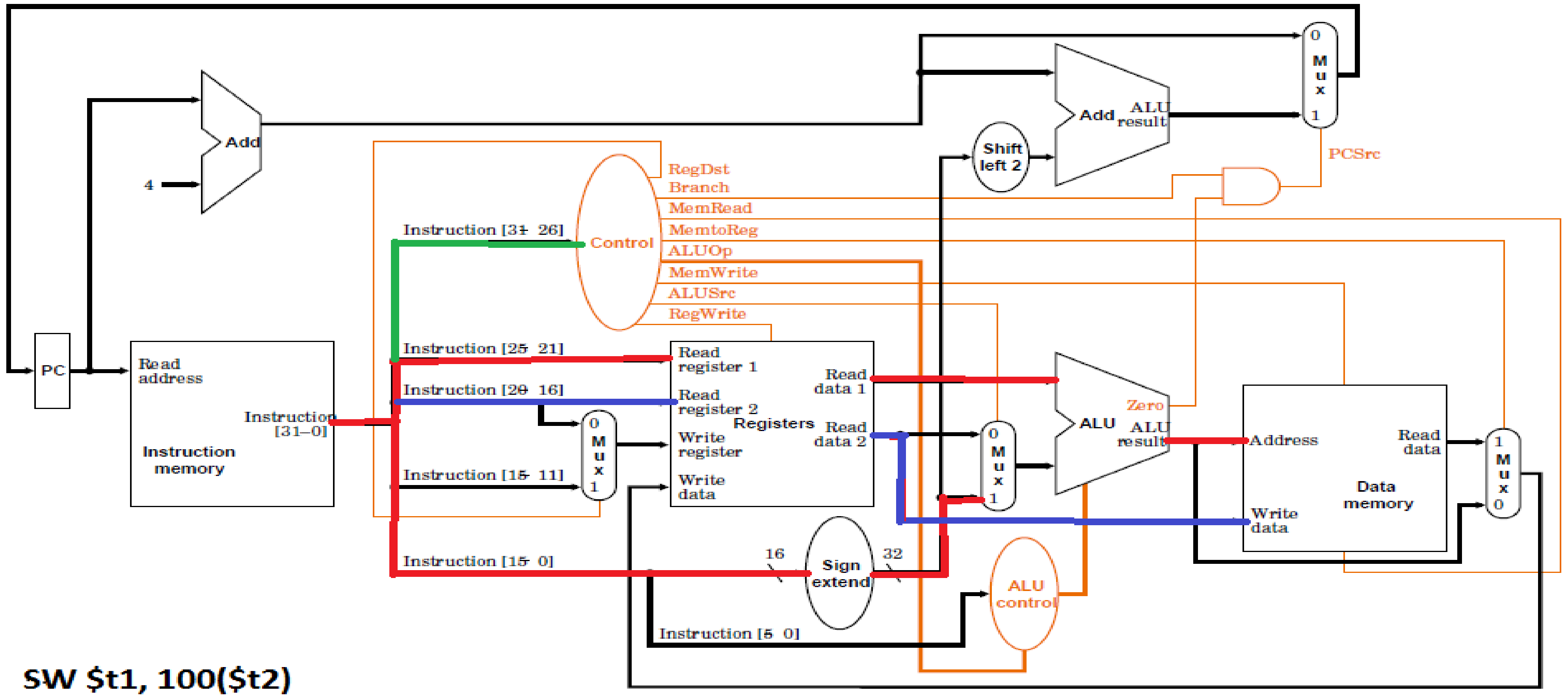


Load/store: `lw $t1, 100($t2)` ⇒ `lw rt,100(rs)`

31 26	25 21	20 16	15 0
35/43	rs	rt	offset
6 bits	5 bits	5 bits	16 bits

sw $t1, 100($t2)


Add
4
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]
Read register 1
Read register 2
Write register
Write data
Registers
Read data 1
Read data 2
16
32
Sign extend
ALU control
Shift left 2
Add
ALU result
PCSrc
0 Mux 1
ALU
Zero
ALU result
Address
Read data
Write data
Data memory
1 Mux 0
SW $t1, 100($t2)



SW $t1, 100($t2)

WHICH ONE OF THE FOLLOWING MUST BE TRUE TO COMPLETE SW instruc

A) MemWrite = OFF (Low) , MemtoReg = 1

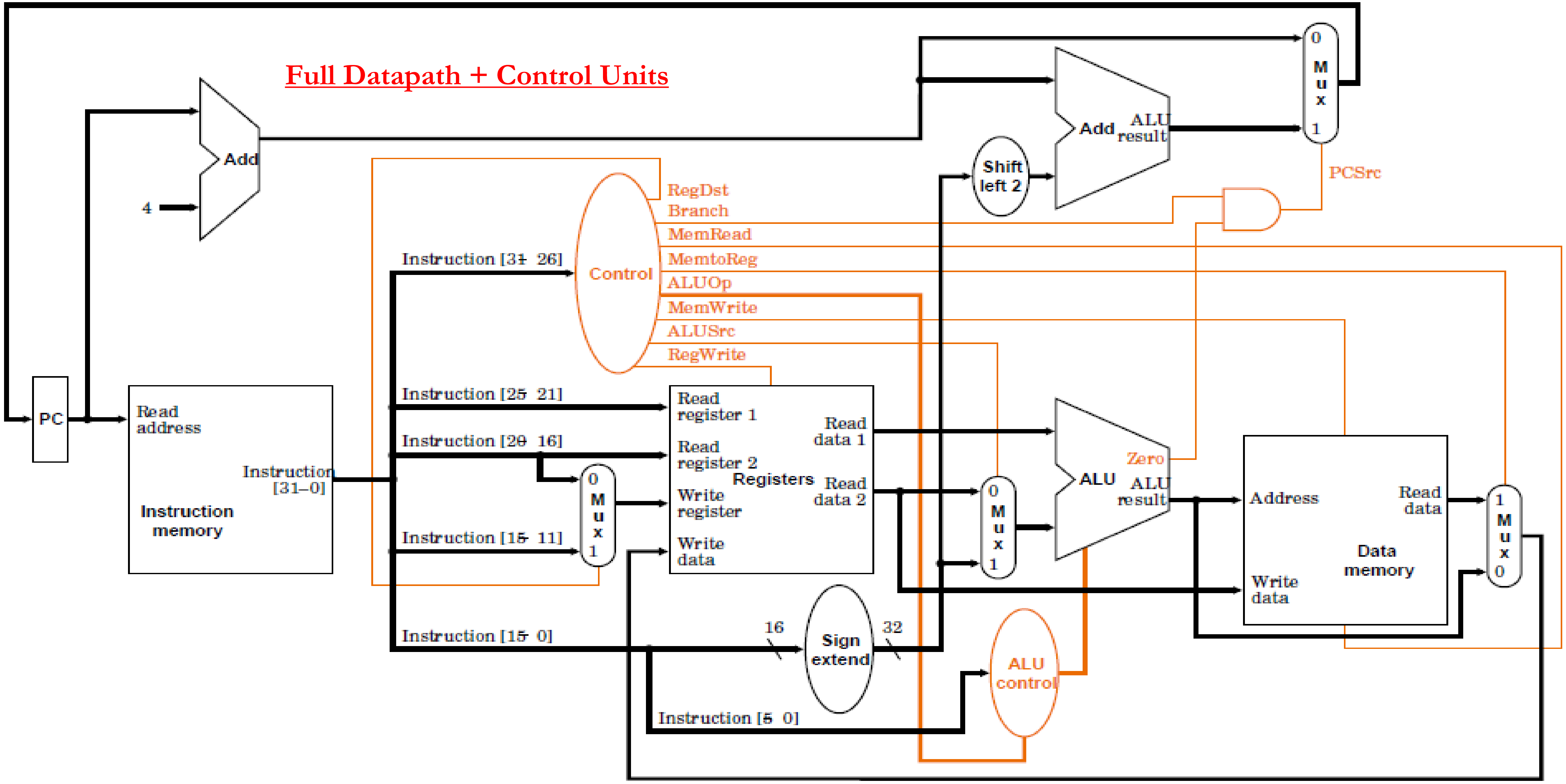
B) ALUSrc = 1 RegDst =1

C) ALUSrc =1 , MemWrite =1

D) MemtoReg =1 , MemWrite =1

E) MemtoReg = X, ALUSrc =0

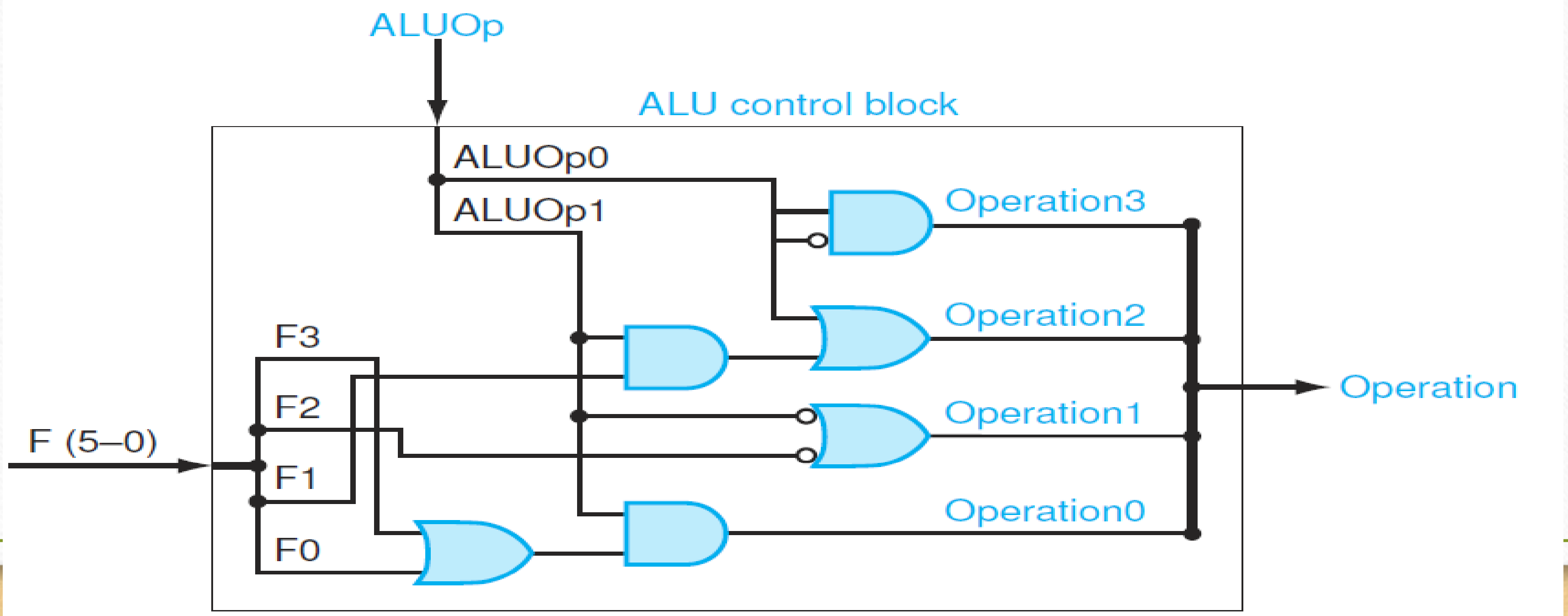
Full Datapath + Control Units

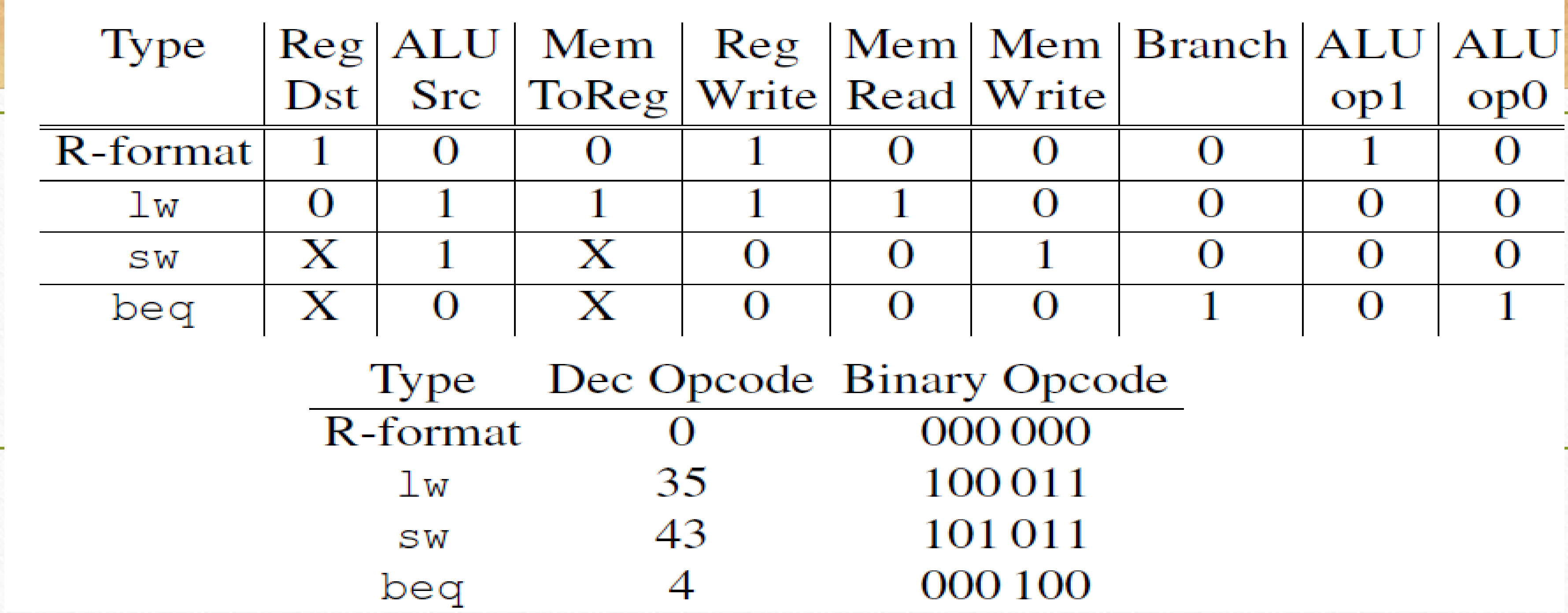


apping of operation to ALU control input:

Opcode	Operation	ALUop	Funct	ALU action	ALU ctrl input
35	lw	00	XXXXXX	add	0010
43	sw	00	XXXXXX	add	0010
4	beq	01	XXXXXX	subtract	0110
0	add	10	100000	add	0010
0	sub	10	100010	subtract	0110
0	and	10	100100	AND	0000
0	or	10	100101	OR	0001
0	slt	10	101010	slt	0111

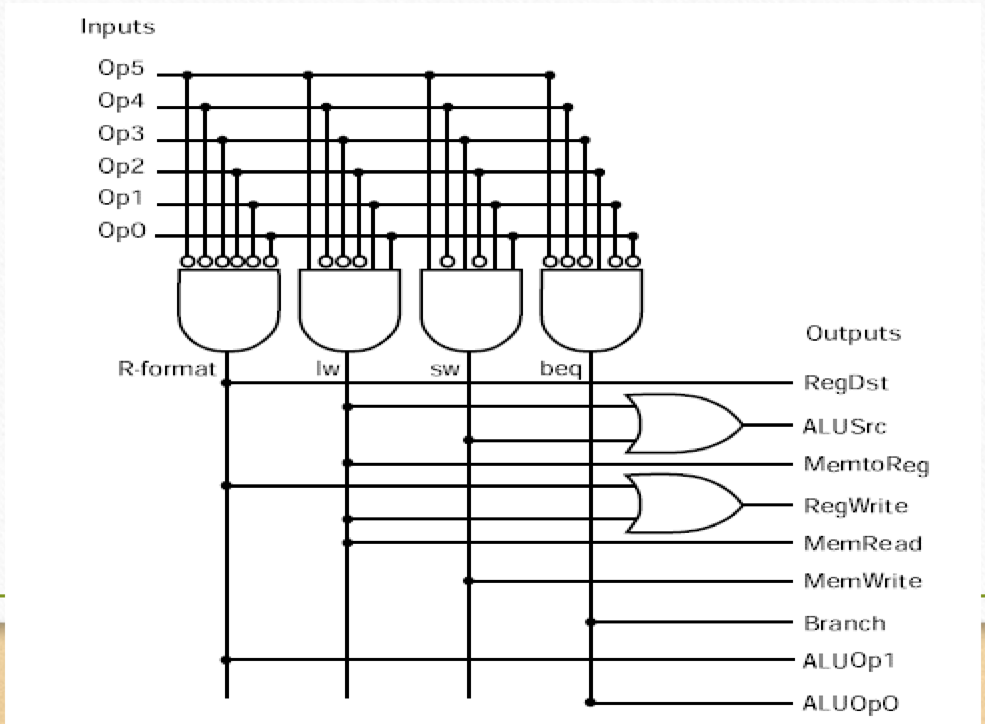
ALUop		Funct field						Operation
ALUop1	ALUop0	F5	F4	F3	F2	F1	F0	3210
0	0	X	X	X	X	X	X	0010
X(0)	1	X	X	X	X	X	X	0110
1	X(0)	X	X	0	0	0	0	0010
1	X(0)	X	X	0	0	1	0	0110
1	X(0)	X	X	0	1	0	0	0000
1	X(0)	X	X	0	1	0	1	0001
1	X(0)	X	X	1	0	1	0	0111

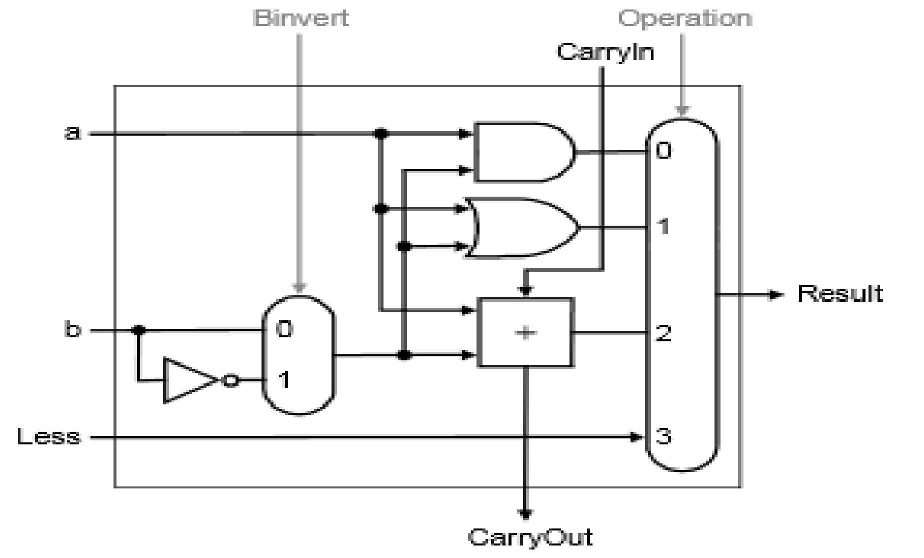


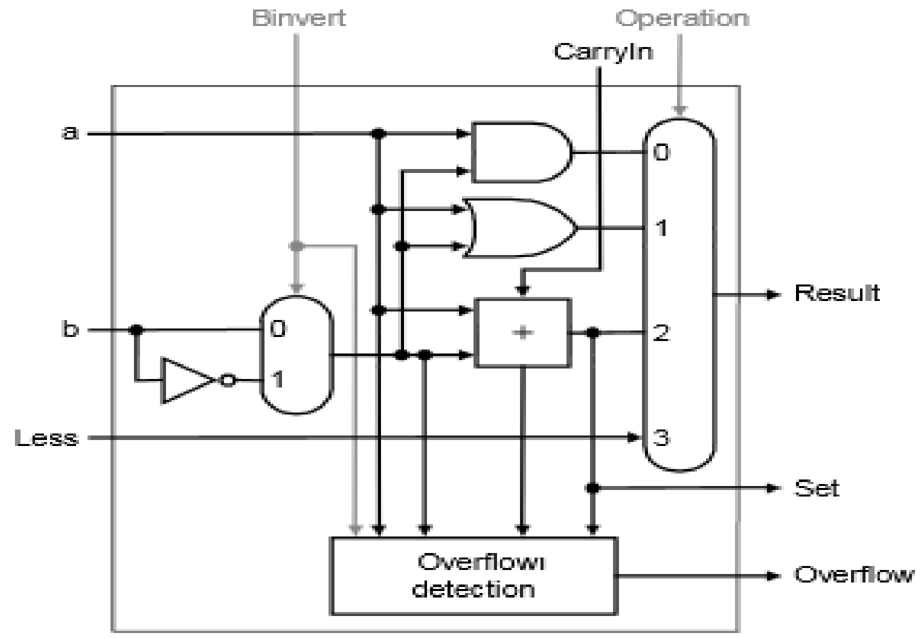


Type	Reg Dst	ALU Src	Mem ToReg	Reg Write	Mem Read	Mem Write	Branch	ALU op1	ALU op0
R-format	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

Type	Dec Opcode	Binary Opcode
R-format	0	000 000
lw	35	100 011
sw	43	101 011
beq	4	000 100







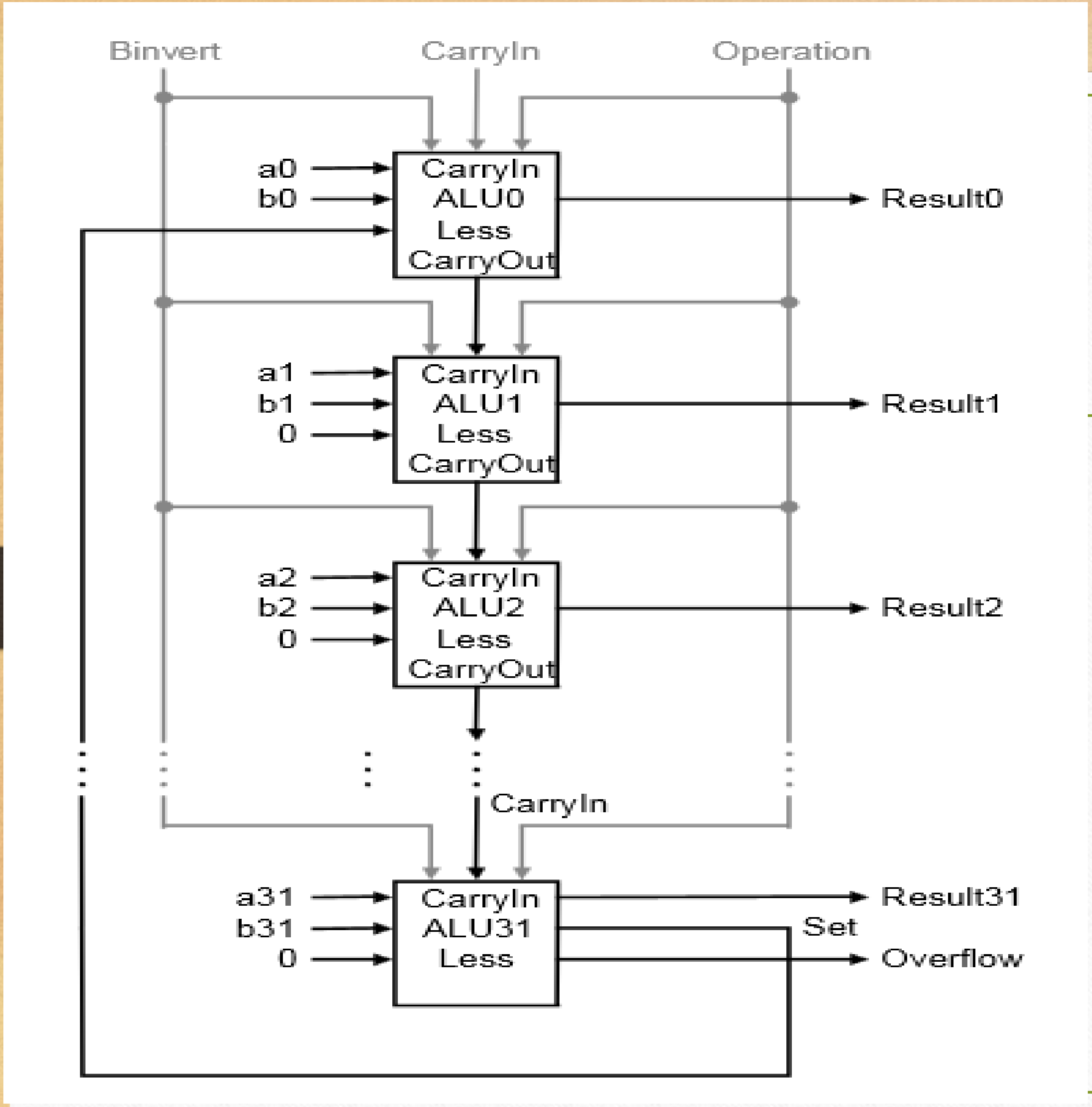
SET ON LESS THAN INSTRUCTION

slt $d, $a, $b R-Format

Description: If $a is less than $b, $d is set to one. It gets zero otherwise.

Operation: if $s < $t → $d = 1;
else $d = 0; Advance PC +4

- Recall that a 32bit ALU is actually a series of 1-bit ALUs that are all
- Performing the same operation.



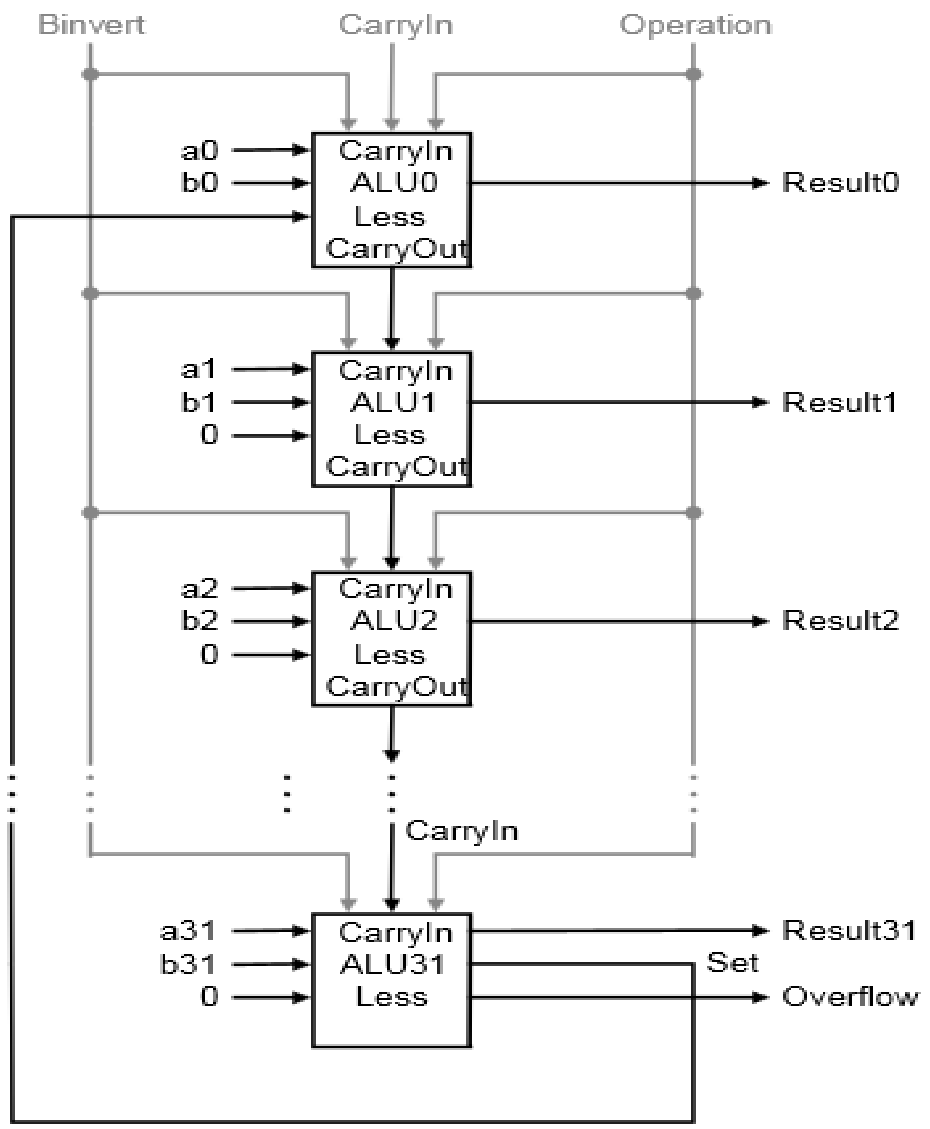
Hardware for SLT: Internal using ALU

Zero sent to every bit
Except first:
This is dependent on Sign of the difference
Between a, and b.

IS a< b, if so a-b, and a is smaller
Answer will be negative

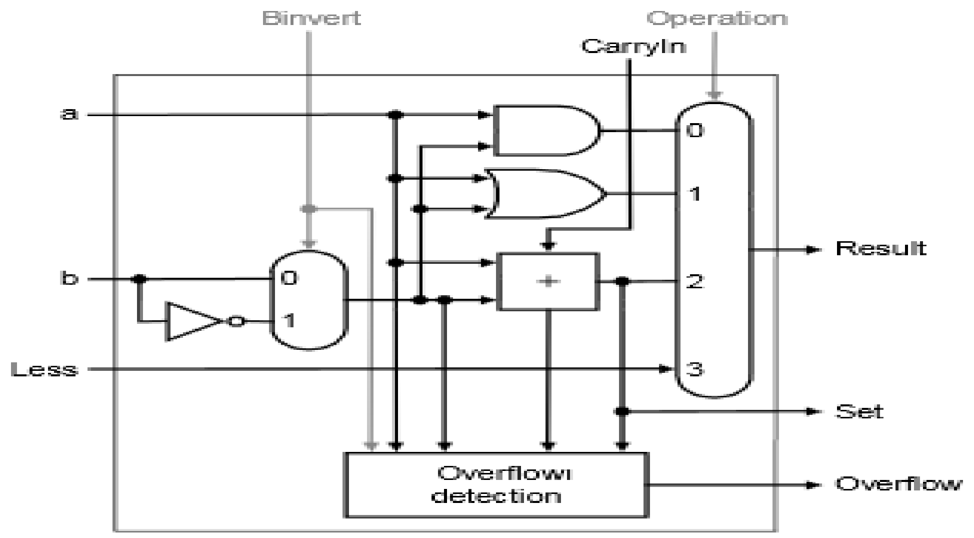
If a = -3 , b =-4 answer will be positive
If a = -5, b =-4 answer will be negative also

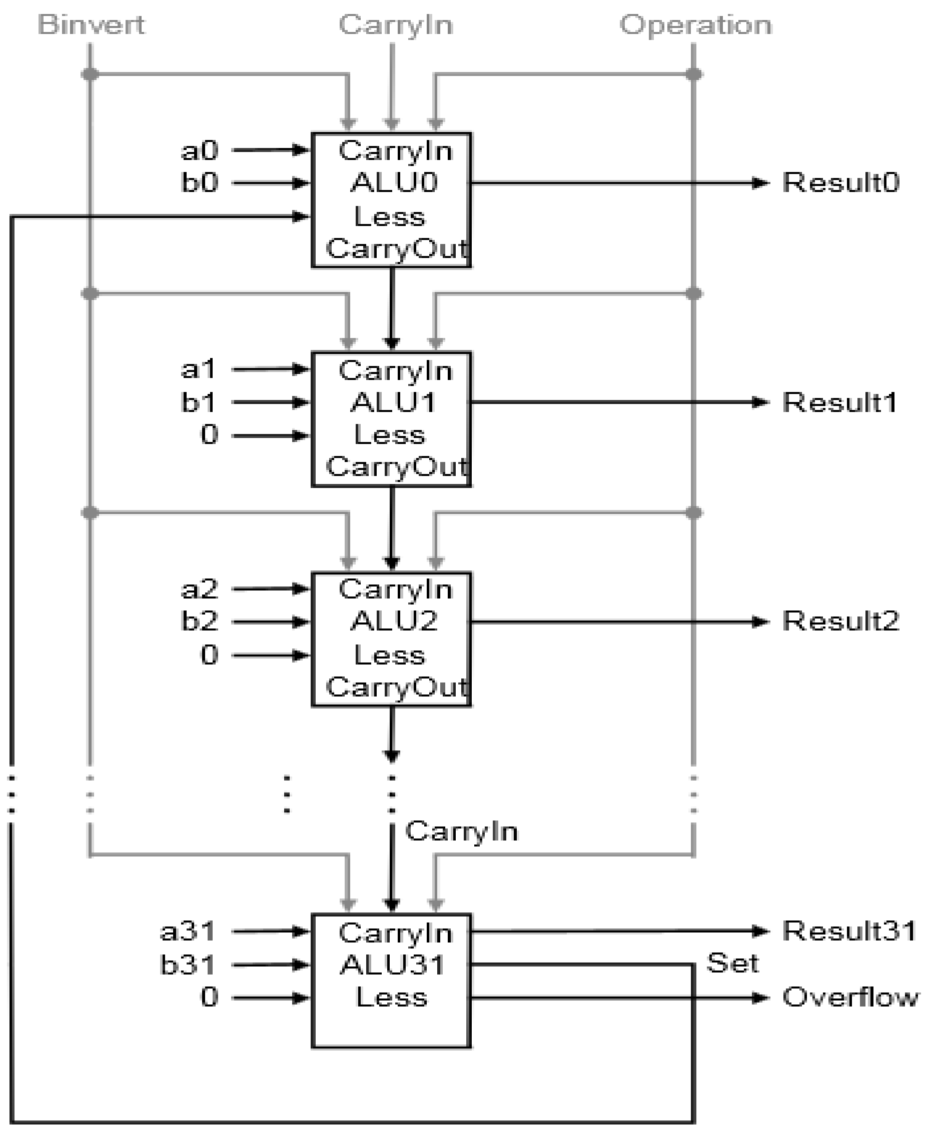
We need to examine the very Last Bit
Is this big indicating a negative answer:
This bit is used to SET first bit of answer



Mapping of operation to ALU control input:

Opcode	Operation	ALUop	Funct	ALU action	ALU ctrl input
35	lw	00	XXXXXX	add	0010
43	sw	00	XXXXXX	add	0010
4	beq	01	XXXXXX	subtract	0110
0	add	10	100000	add	0010
0	sub	10	100010	subtract	0110
0	and	10	100100	AND	0000
0	or	10	100101	OR	0001
0	slt	10	101010	slt	0111

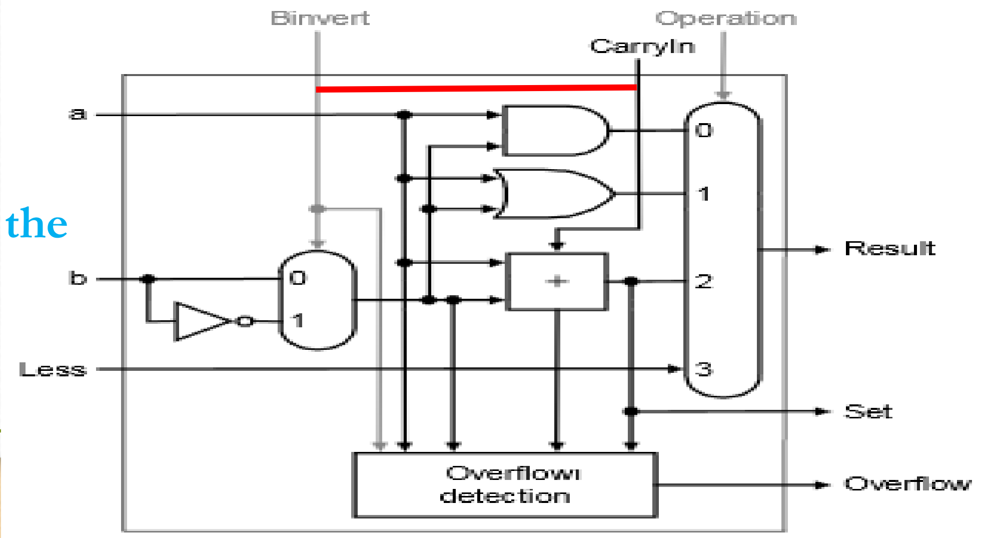




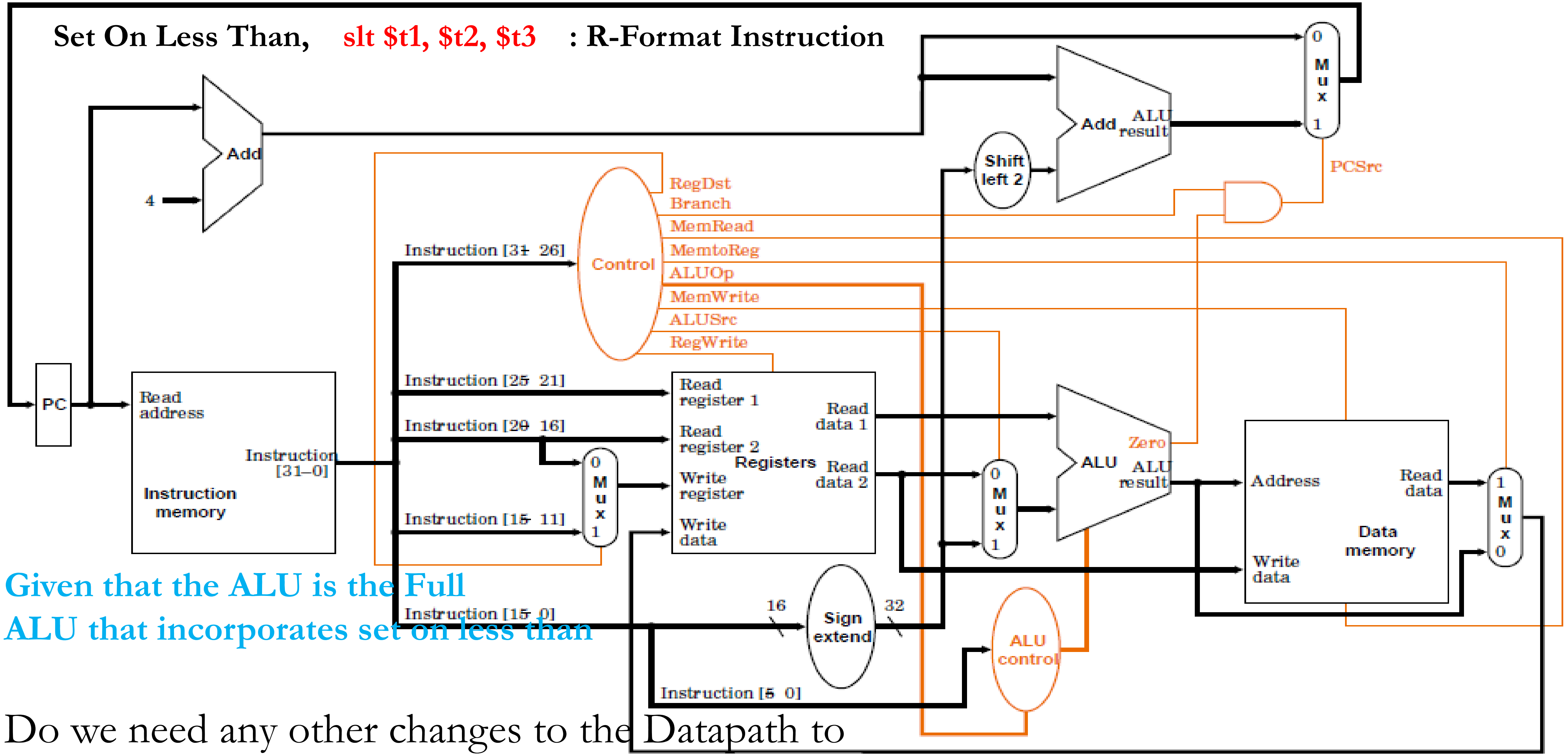
Mapping of operation to ALU control input:

Opcode	Operation	ALUop	Funct	ALU action	ALU ctrl input
35	lw	00	XXXXXX	add	0010
43	sw	00	XXXXXX	add	0010
4	beq	01	XXXXXX	subtract	0110
0	add	10	100000	add	0010
0	sub	10	100010	subtract	0110
0	and	10	100100	AND	0000
0	or	10	100101	OR	0001
0	slt	10	101010	slt	0111

Internally
B-invert
Would be tied to the
First Carry-In



Set On Less Than, slt $t1, $t2, $t3 : R-Format Instruction



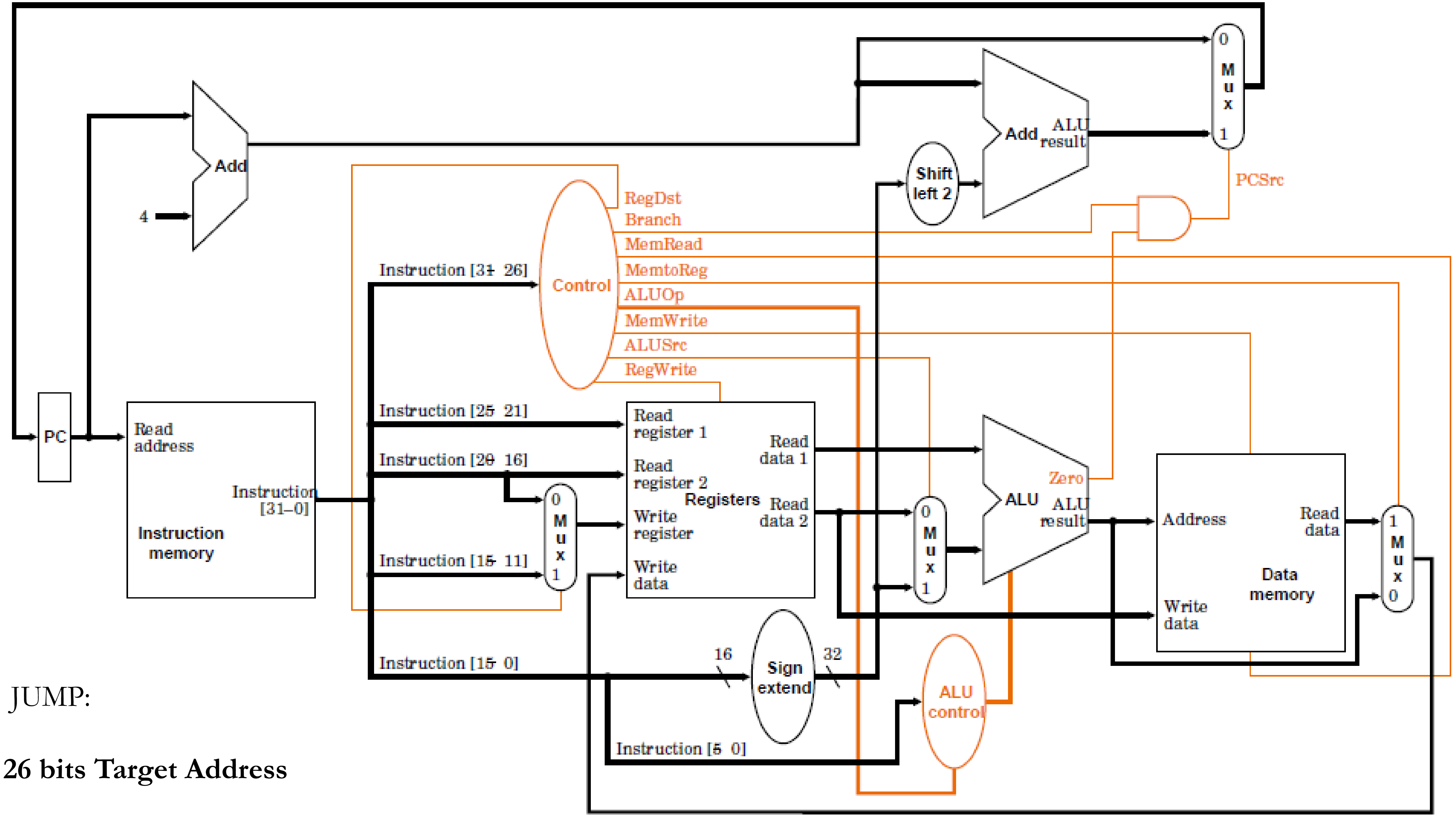
Given that the ALU is the Full ALU that incorporates set on less than

Do we need any other changes to the Datapath to Perform the slt instruction:

A) Yes : need to be able to WRITE to the Destination Register $t1

B) NO, Datapath has everything it needs

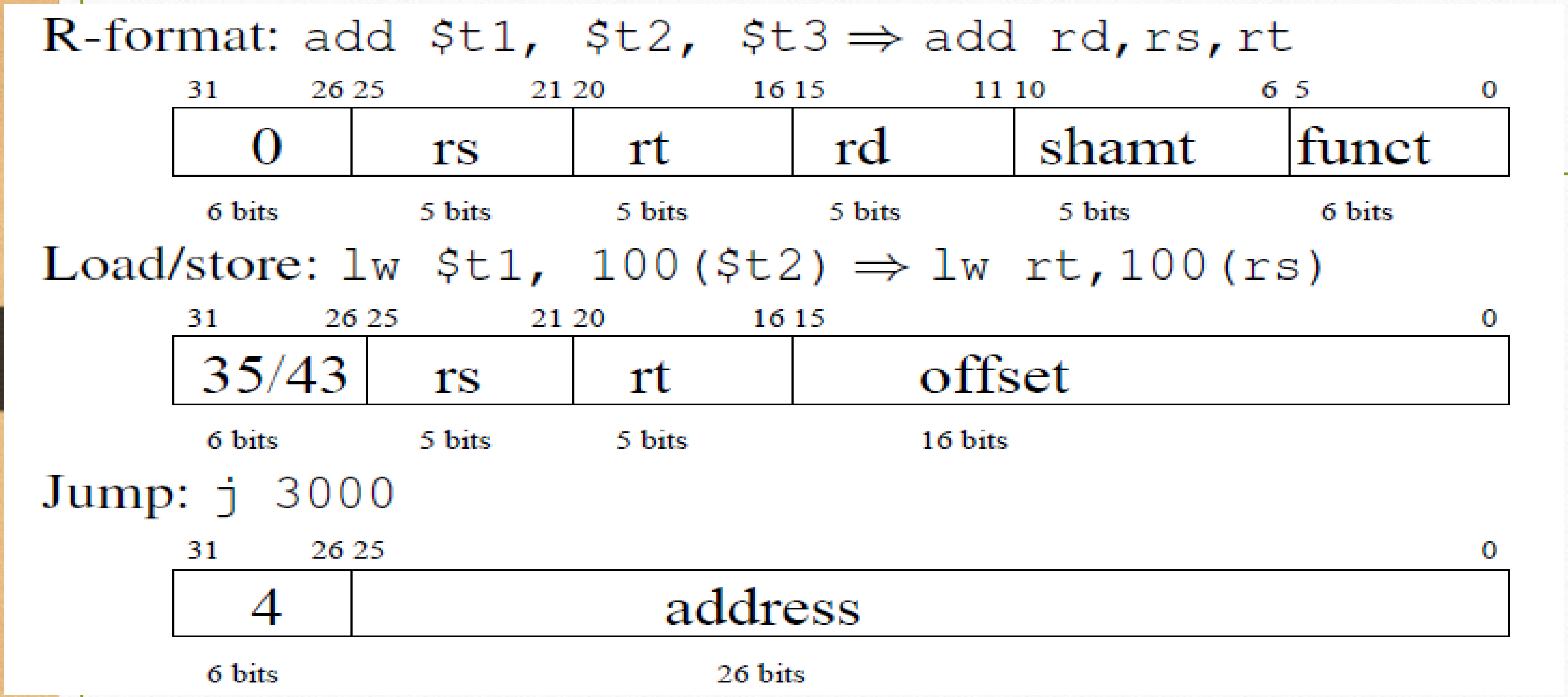
C) Yes: need to be able to compare two registers, $t2 and $t3



JUMP:

26 bits Target Address

How the Instruction Bits are Broken up:



R-format: `add $t1, $t2, $t3` ⇒ `add rd,rs,rt`

31 – 26	25 – 21	20 – 16	15 – 11	10 – 6	5 – 0
0	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Load/store: `lw $t1, 100($t2)` ⇒ `lw rt,100(rs)`

31 – 26	25 – 21	20 – 16	15 – 0
35/43	rs	rt	offset
6 bits	5 bits	5 bits	16 bits

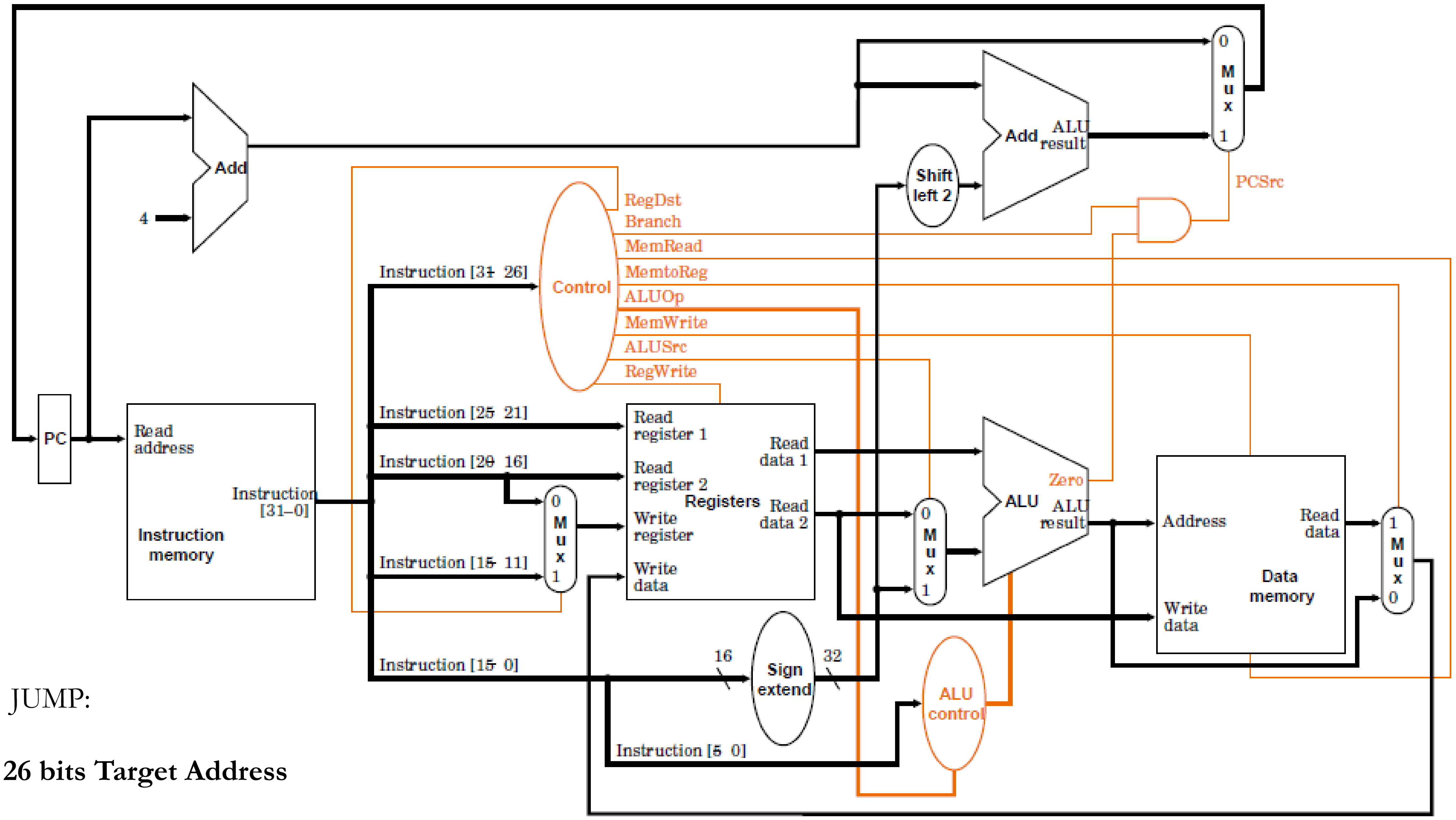
Jump: `j 3000`

31 – 26	25 – 0
4	address
6 bits	26 bits

sll, sub

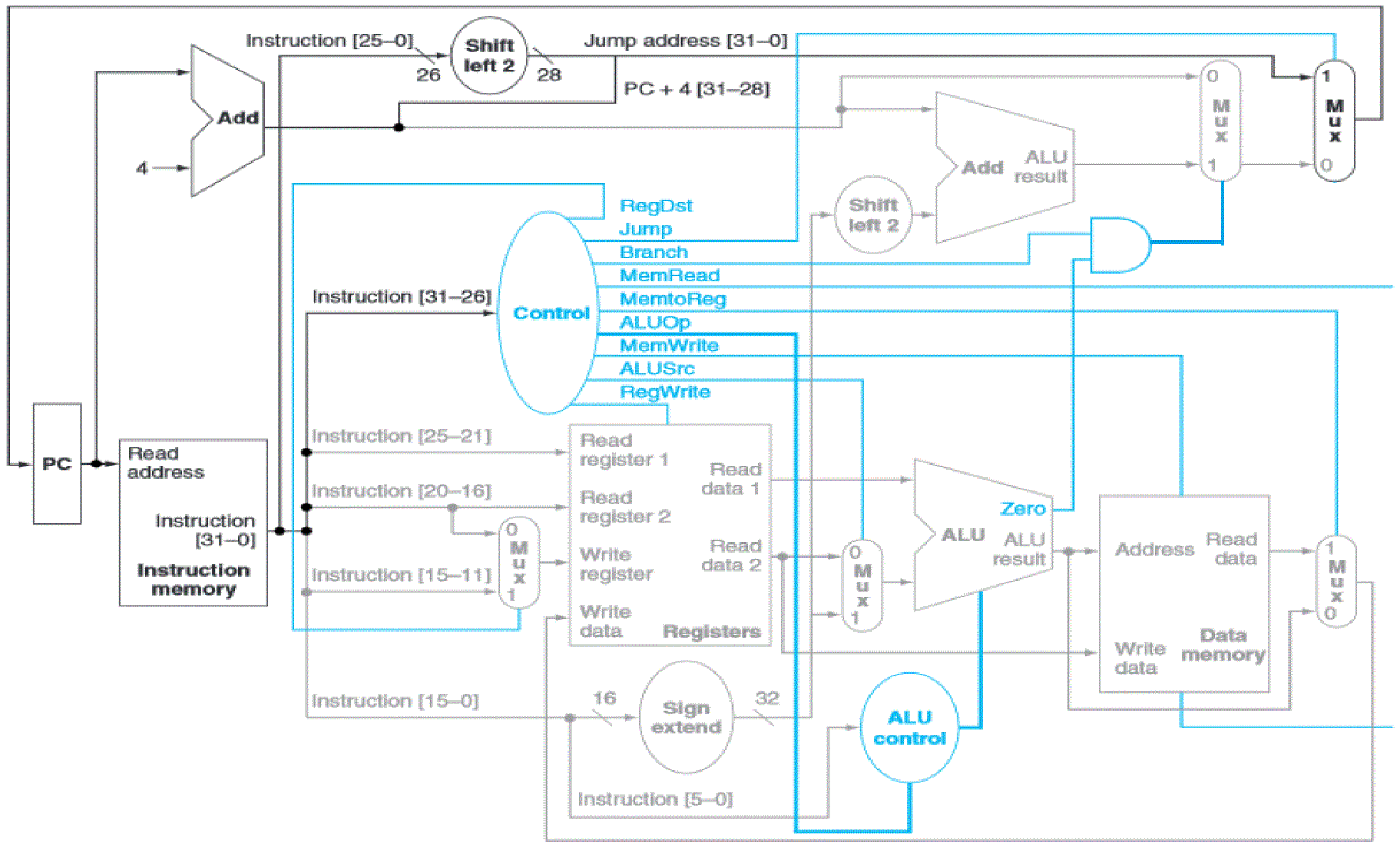
beq, subi, addi

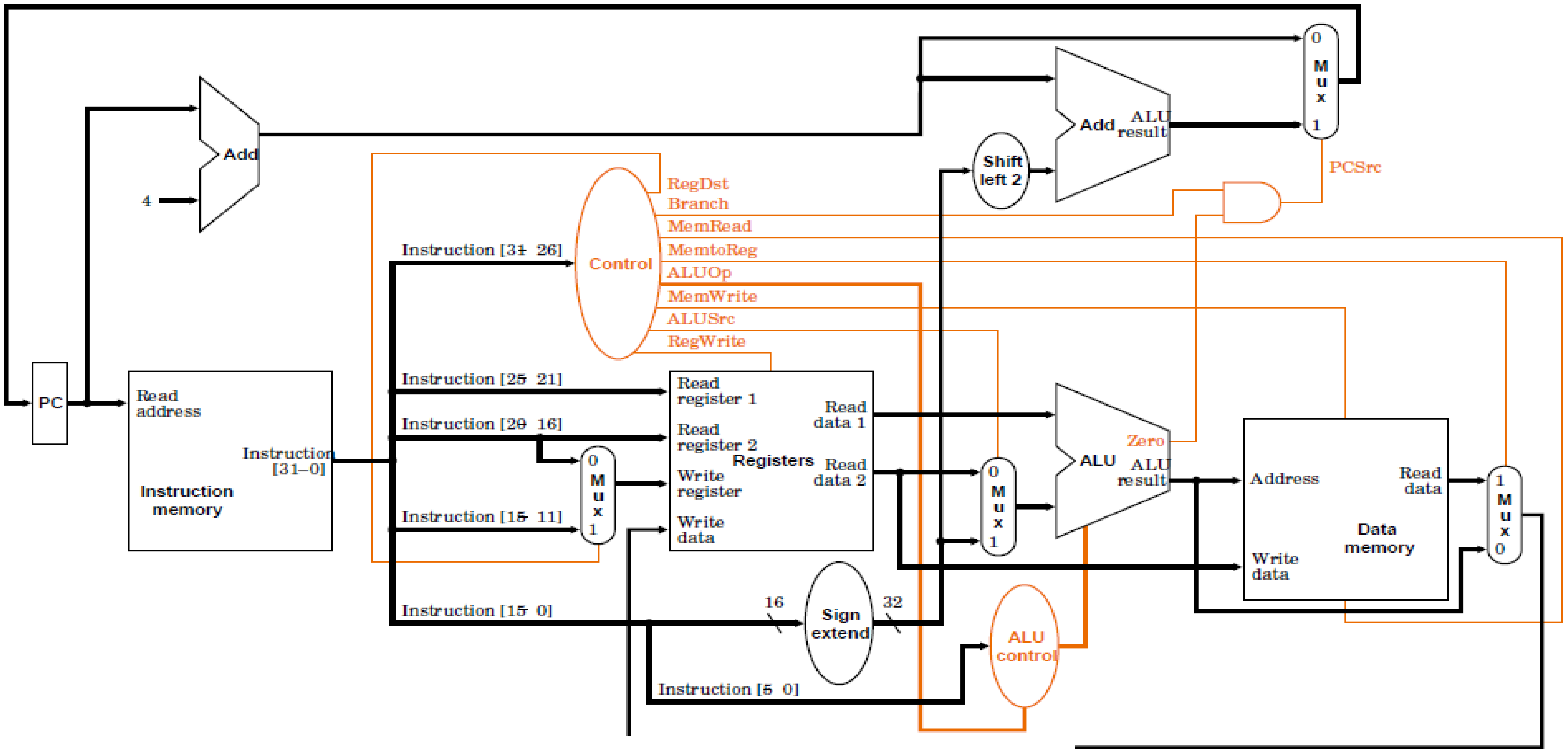
special



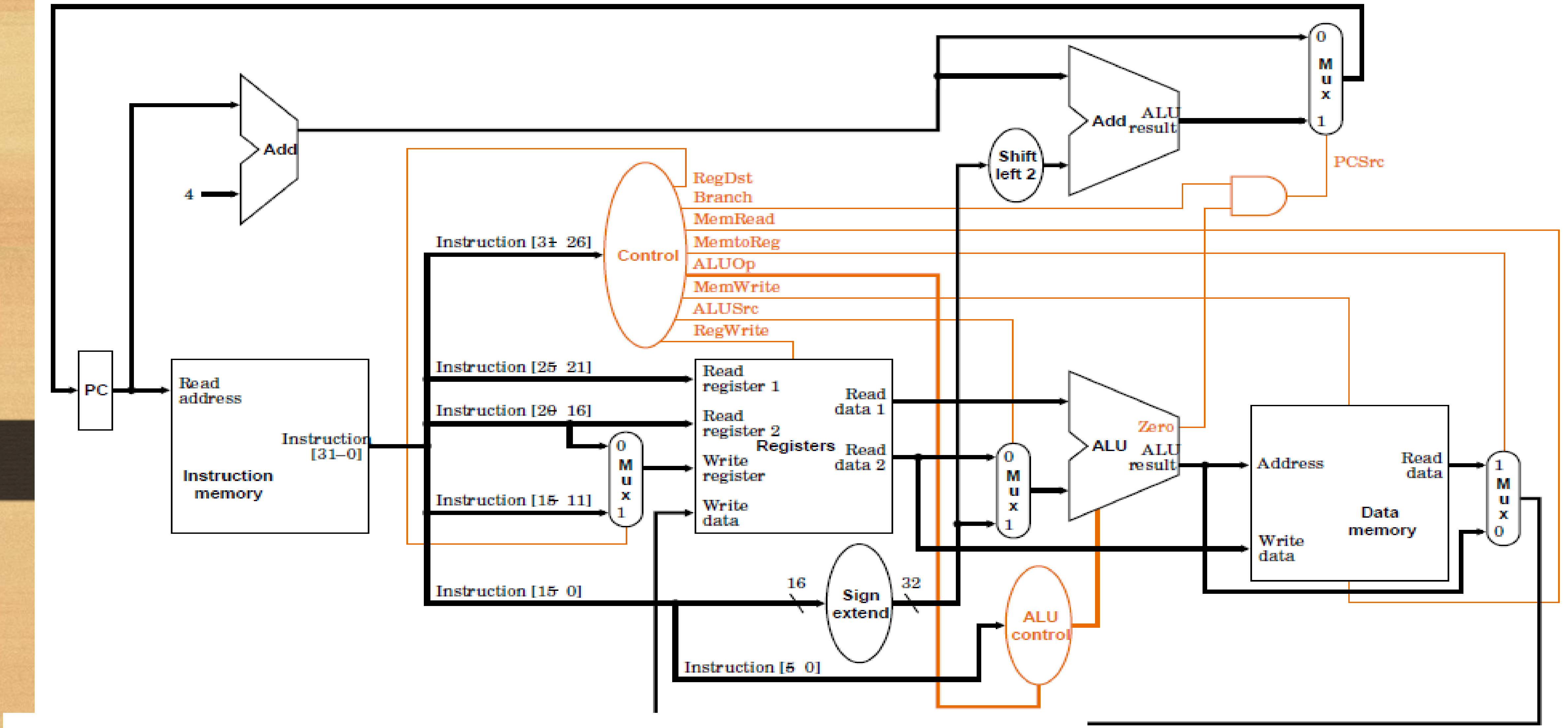
JUMP:

26 bits Target Address


Instruction [25–0]
Shift left 2
26
28
Jump address [31–0]
PC + 4 [31–28]
Add
4
Add
ALU result
0
Mux
1
1
Mux
0
Shift left 2
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]
0
Mux
1
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
0
Mux
1
ALU
Zero
ALU result
Address
Read data
Write data
Data memory
1
Mux
0
Instruction [15–0]
16
Sign extend
32
ALU control
Instruction [5–0]



- Suppose memory units take 200 ps (picoseconds), ALUs 100 ps, register files 50 ps, no delay on other units



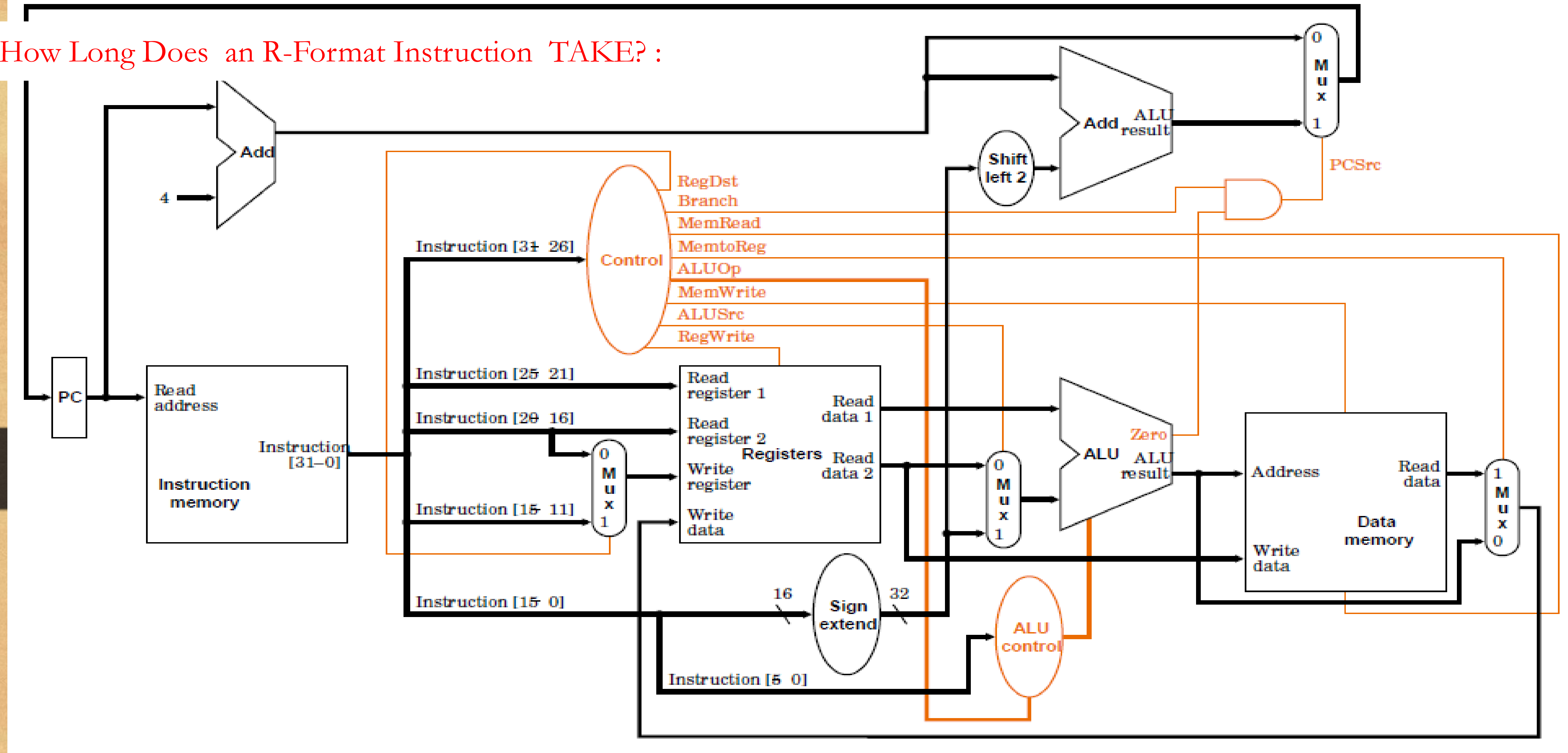
- Suppose memory units take 200 ps (picoseconds), ALUs 100 ps, register files 50 ps, no delay on other units

How Long Does JUMP TAKE? :

- Jump: Read from Instruction memory
- Shift Left x2 : Minimal Time
- Write to PC: Minimal Time

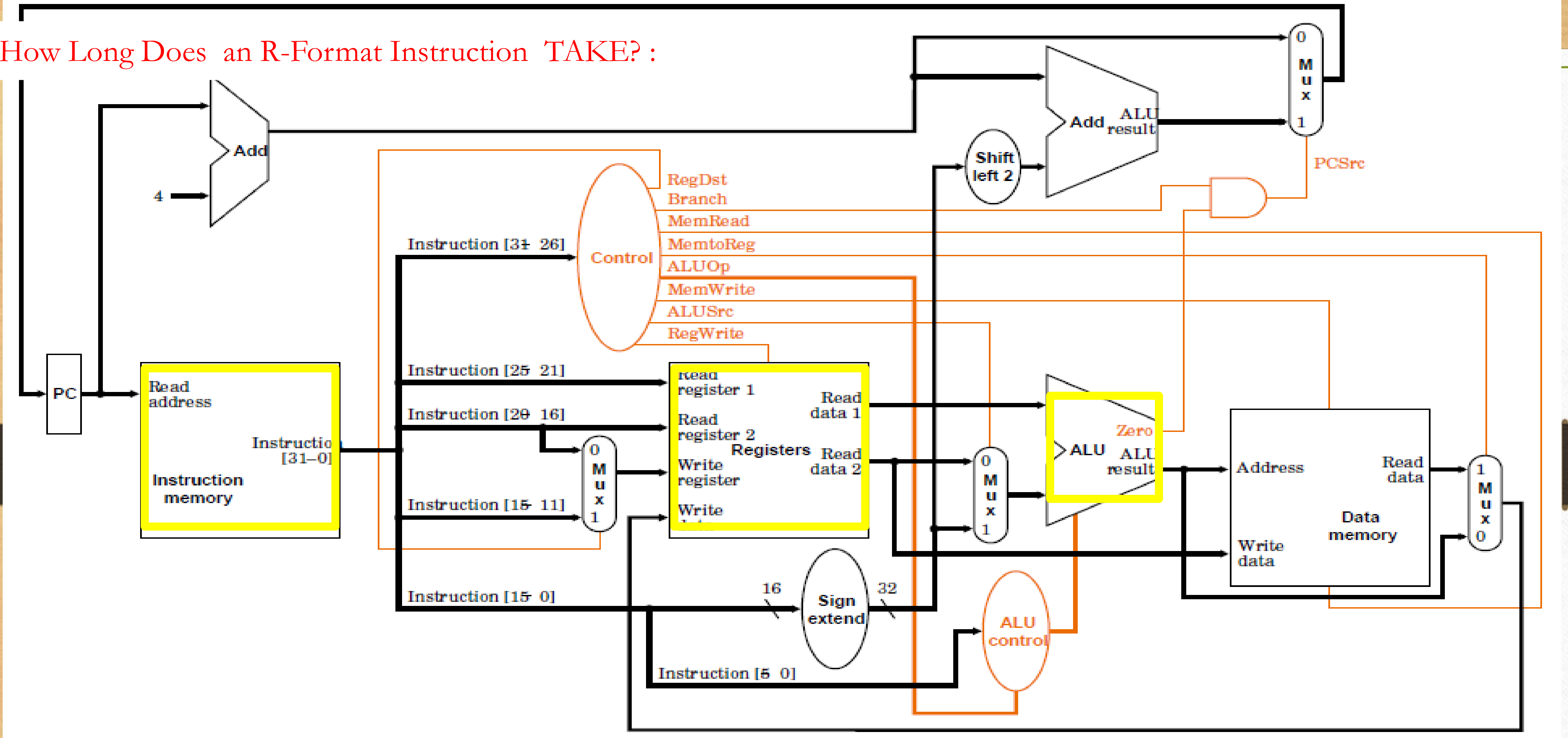
- Jump: Read from Instruction memory **(200ps)**
- Shift Left x2 : Minimal Time **(0)**
- Write to PC: Minimal Time **(0)**

How Long Does an R-Format Instruction TAKE? :



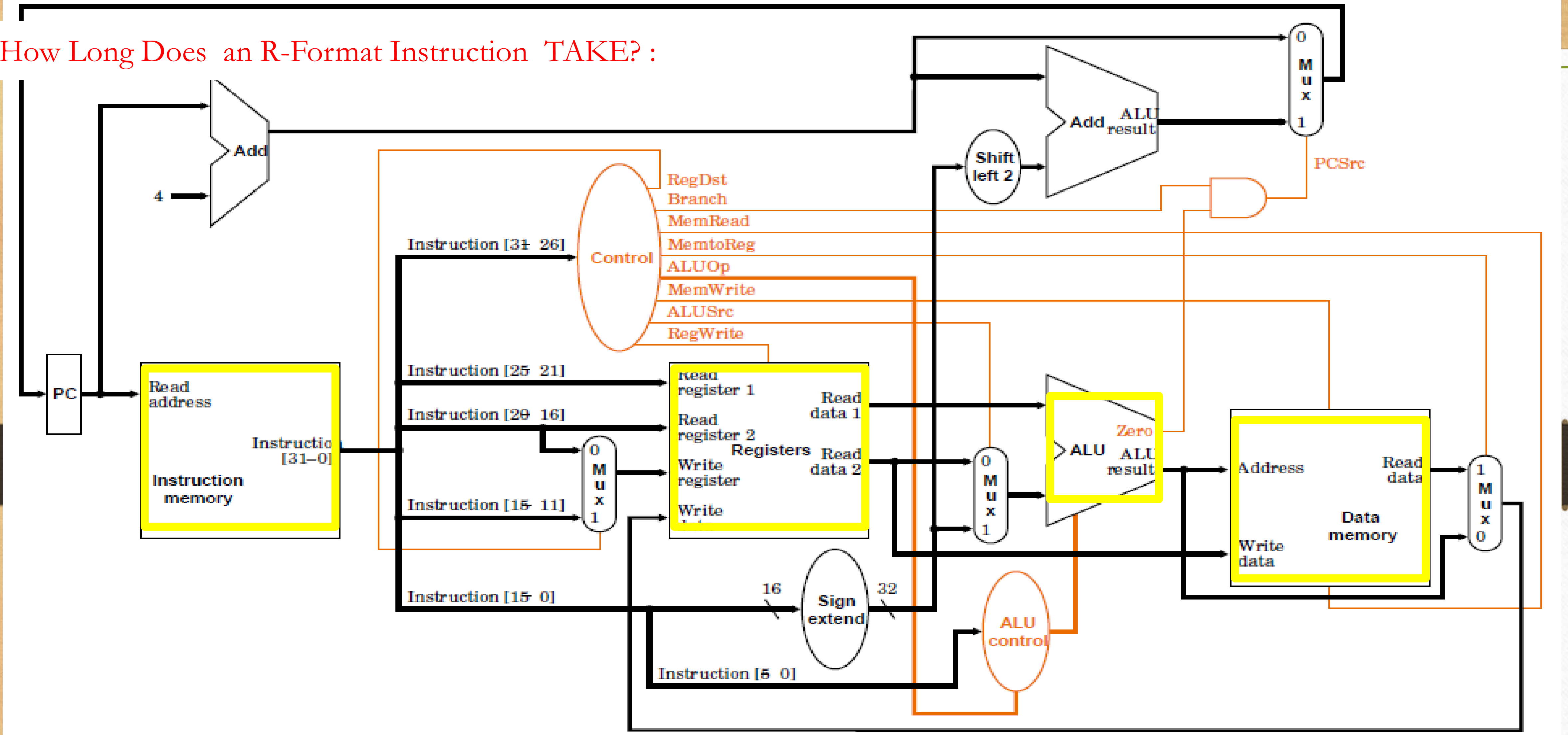
- Suppose memory units take 200 ps (picoseconds), ALUs 100 ps, register files 50 ps, no delay on other units

How Long Does an R-Format Instruction TAKE? :



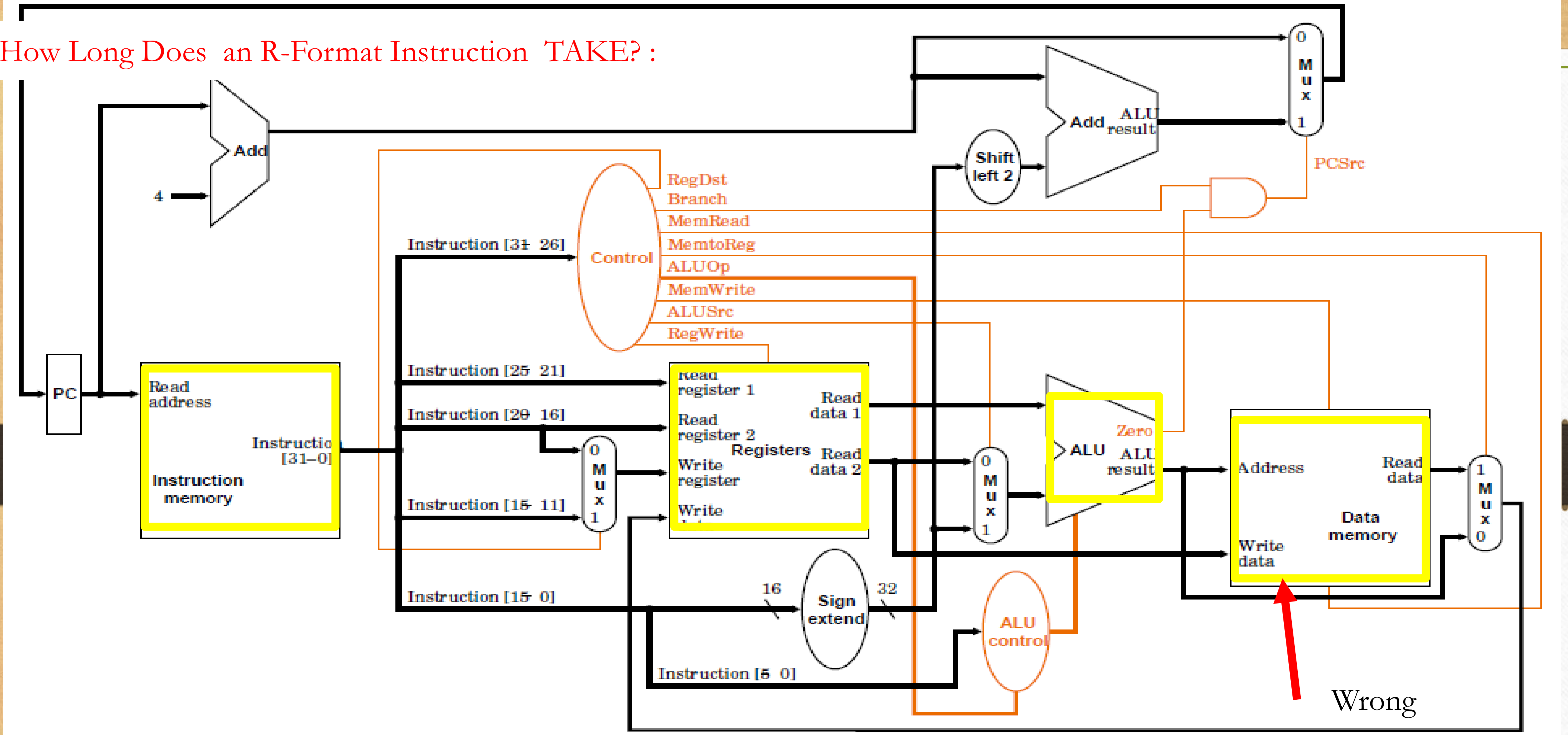
- Suppose memory units take 200 ps (picoseconds), ALUs 100 ps, register files 50 ps, no delay on other units

How Long Does an R-Format Instruction TAKE? :



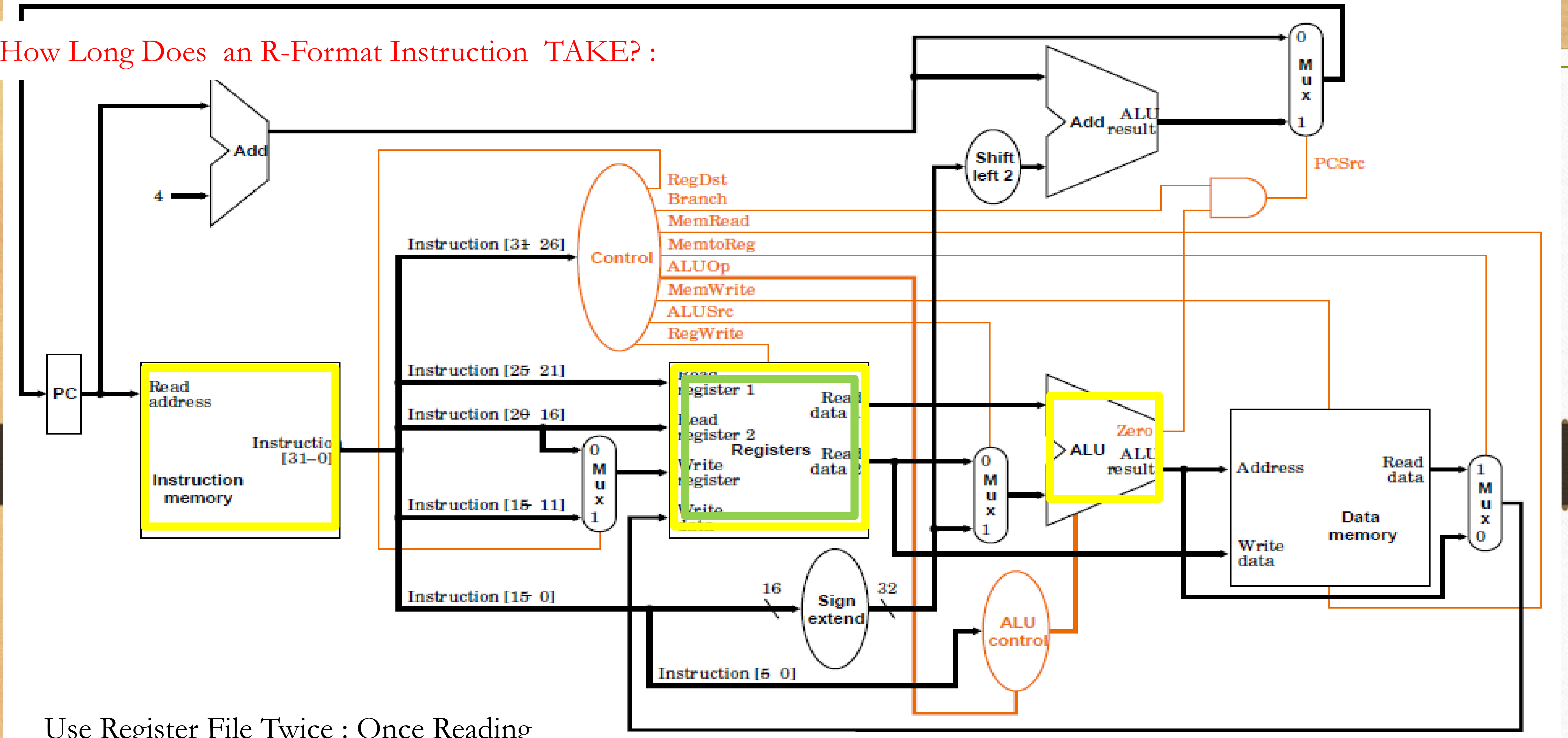
- Suppose memory units take 200 ps (picoseconds), ALUs 100 ps, register files 50 ps, no delay on other units

How Long Does an R-Format Instruction TAKE? :



- Suppose memory units take 200 ps (picoseconds), ALUs 100 ps, register files 50 ps, no delay on other units

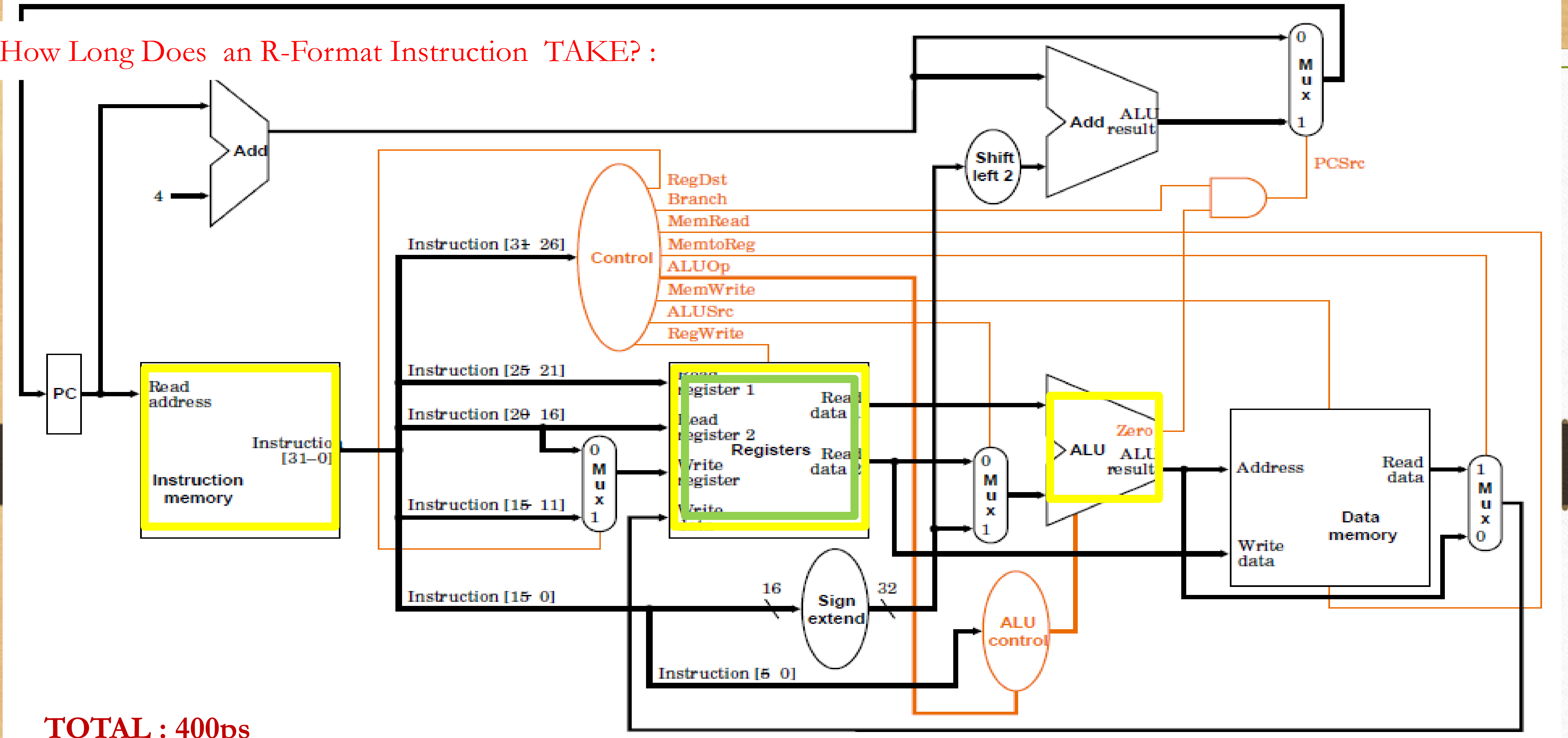
How Long Does an R-Format Instruction TAKE? :



Use Register File Twice : Once Reading
Once Writing

- Suppose memory units take 200 ps (picoseconds), ALUs 100 ps, register files 50 ps, no delay on other units

How Long Does an R-Format Instruction TAKE? :

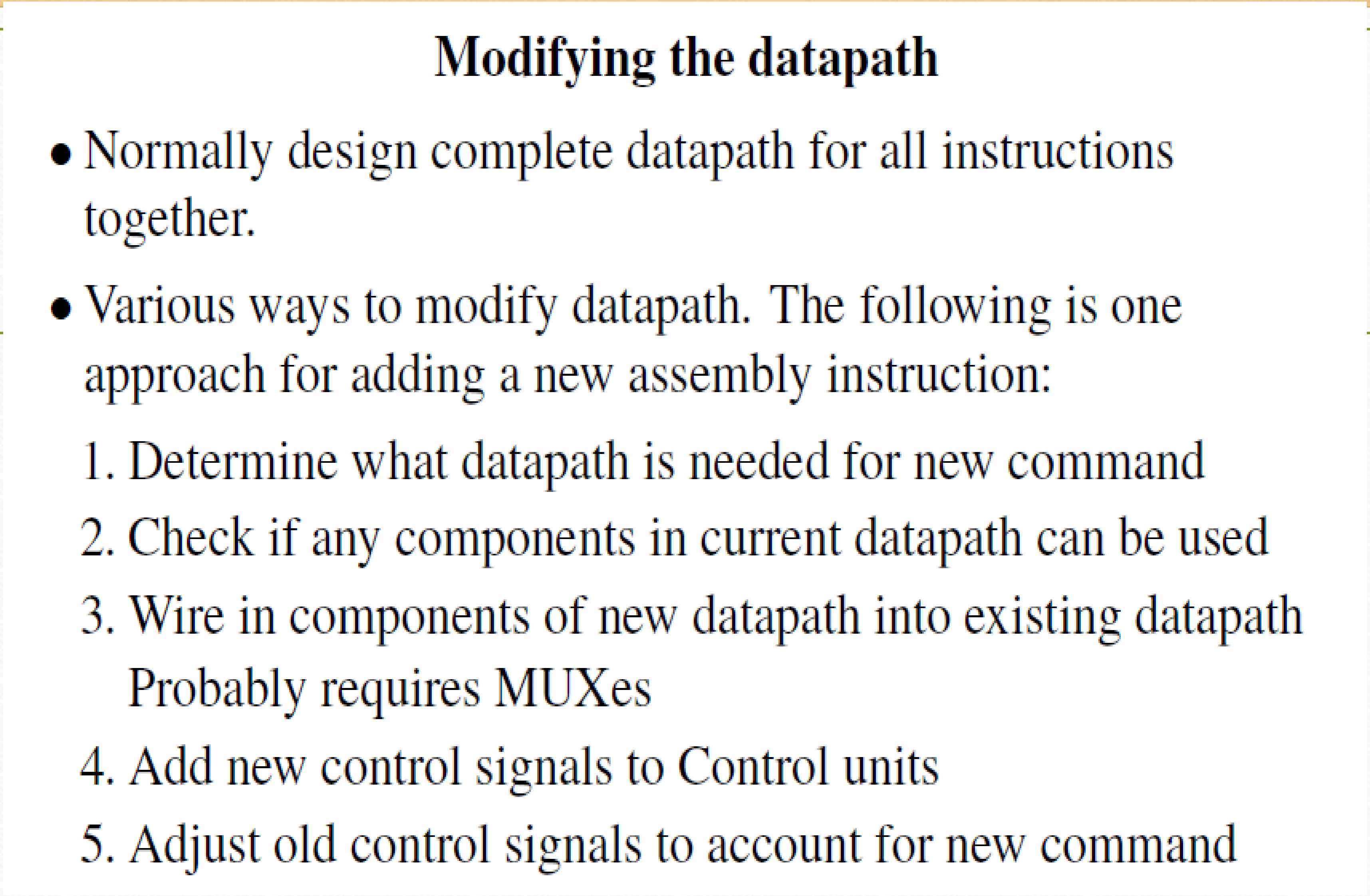


TOTAL : 400ps

- Suppose memory units take 200 ps (picoseconds), ALUs 100 ps, register files 50 ps, no delay on other units

Performance of Single Cycle Machines

- Suppose memory units take 200 ps (picoseconds), ALUs 100 ps, register files 50 ps, no delay on other units
- Jumps take 200 ps, branches take 350 ps, R-format instructions 400 ps, stores 550 ps, loads 600 ps.
- Clock period must be increased to 600 ps or more
- Even worse when floating-point instructions are implemented
- Idea: use multicycle implementation and R format



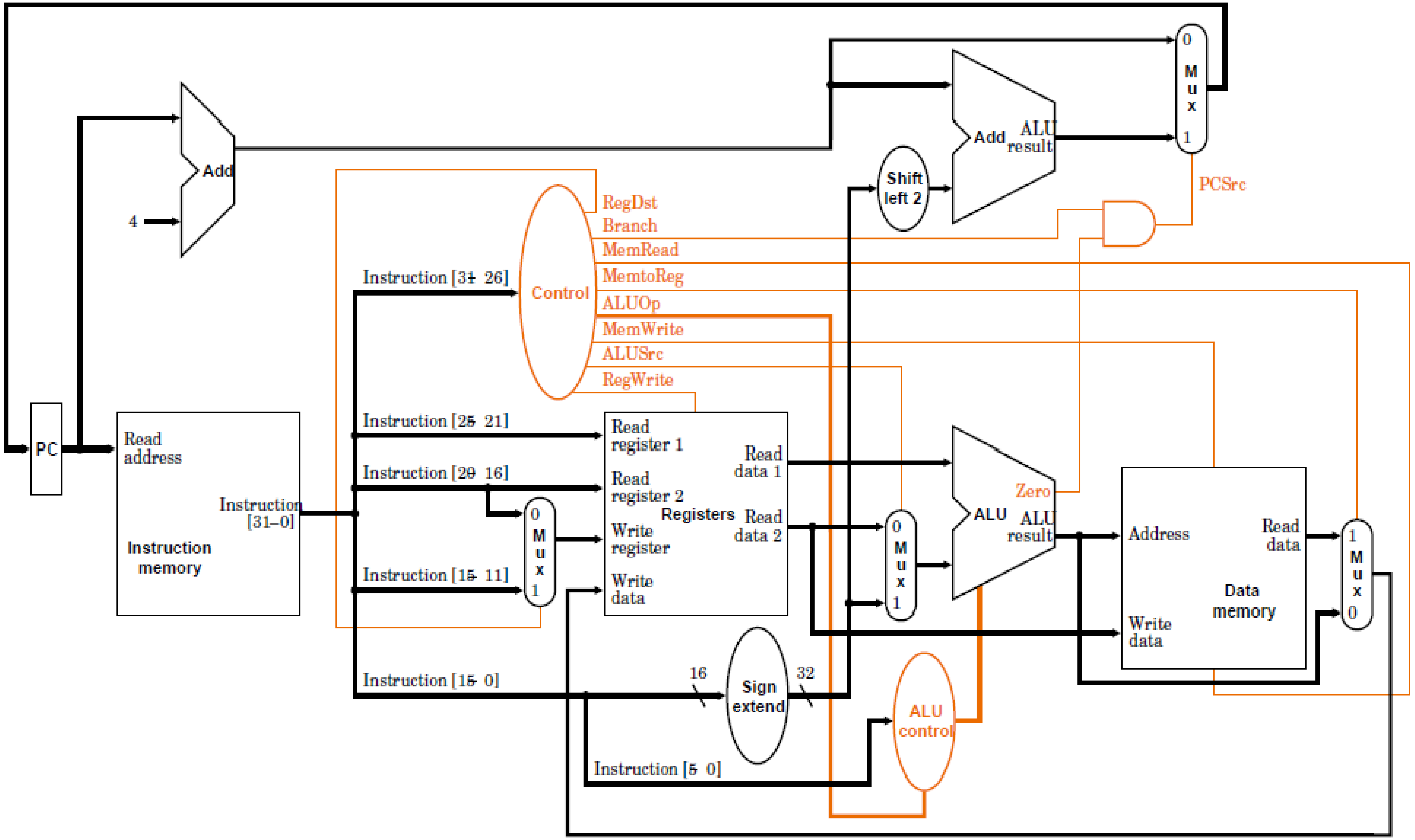
Modifying the datapath

- Normally design complete datapath for all instructions together.
- Various ways to modify datapath. The following is one approach for adding a new assembly instruction:

1. Determine what datapath is needed for new command
2. Check if any components in current datapath can be used
3. Wire in components of new datapath into existing datapath
   Probably requires MUXes
4. Add new control signals to Control units
5. Adjust old control signals to account for new command

- Add `jrel $ra` which performs

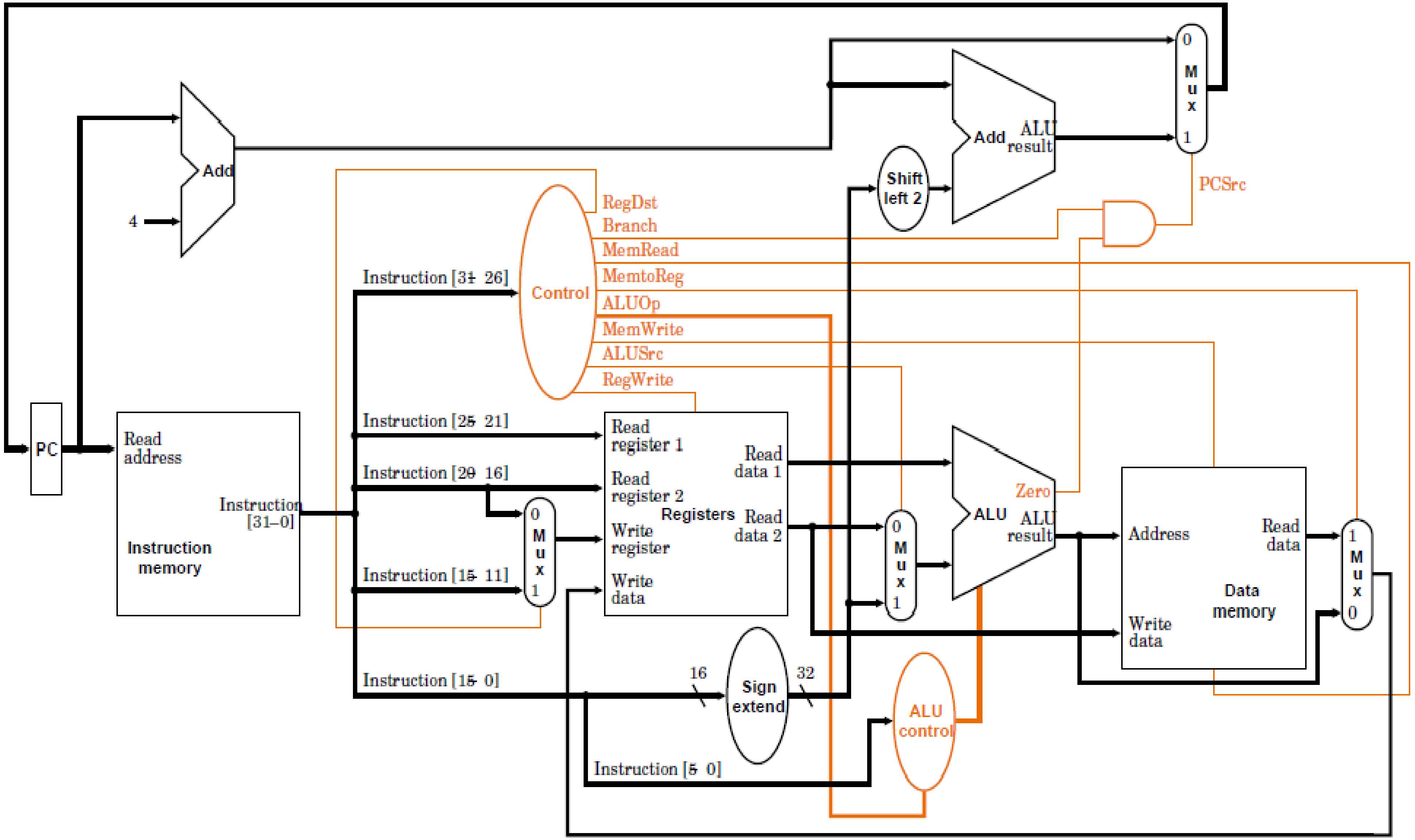
  PC ← PC + 4 + 4*$ra



- Add `jrel $ra` which performs

  PC ← PC + 4 + 4*$ra

Note : $ra is the $rs Register



PC
Add
4
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
M
u
x
1
Read register 1
Read register 2
Write register
Write data
Registers
Read data 1
Read data 2
16
Sign extend
32
Shift left 2
Add
ALU result
PCSrc
ALU
Zero
ALU result
ALU control
Address
Read data
Write data
Data memory