Single Cycle Architecture

Part 2

I get enough time to answer clicker questions

- A) NO -never
- B) YES -always
- C) Yes and NO but I could use more time

Office hours : Today 1-2
Thursday by appointment(this week)

In class I find the Pace of the lecture…

A) Too fast

B) Too slow

C) Bit too fast

D) Bit too slow

E) Just Right ☺

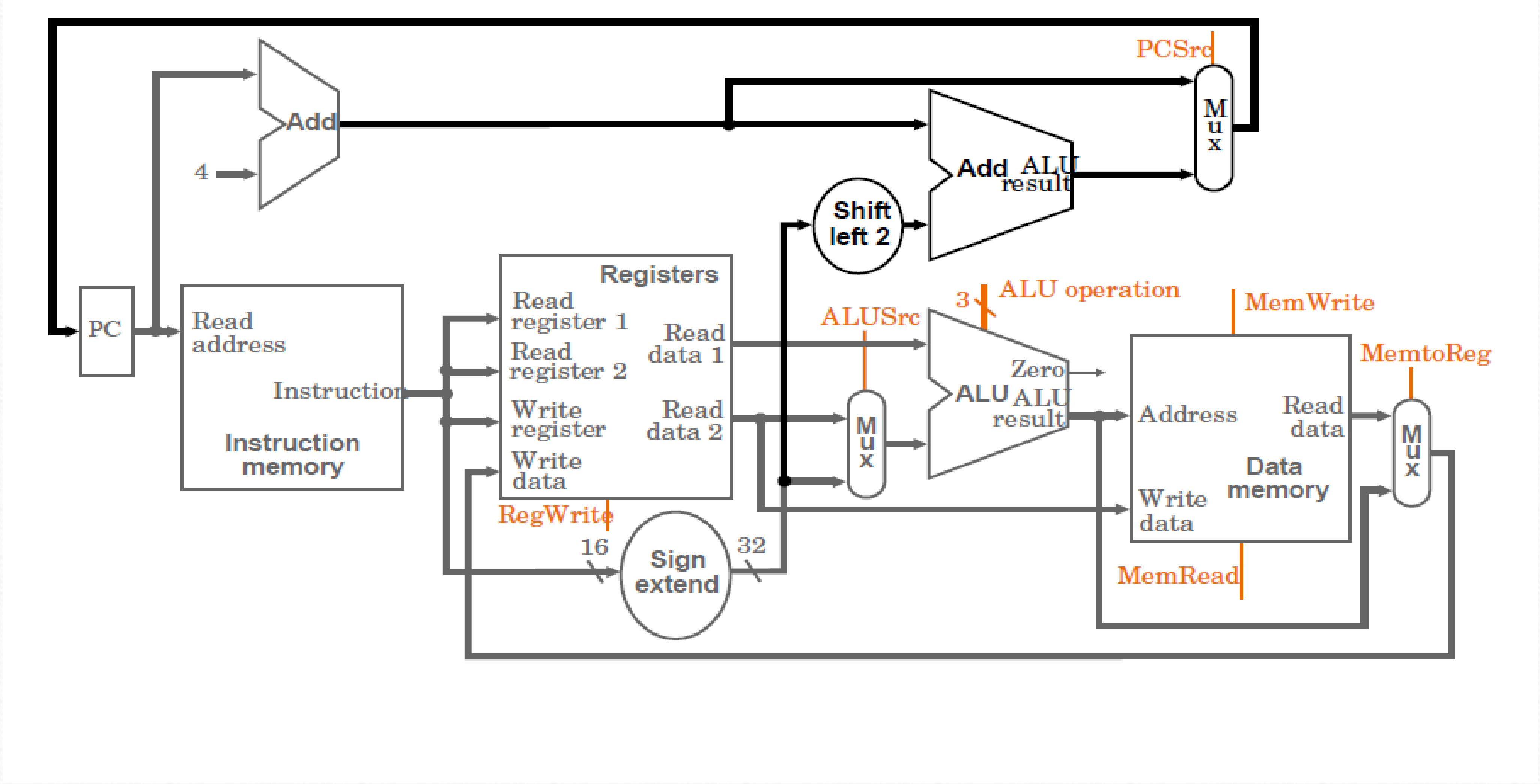
Midterm June 18 Thursday

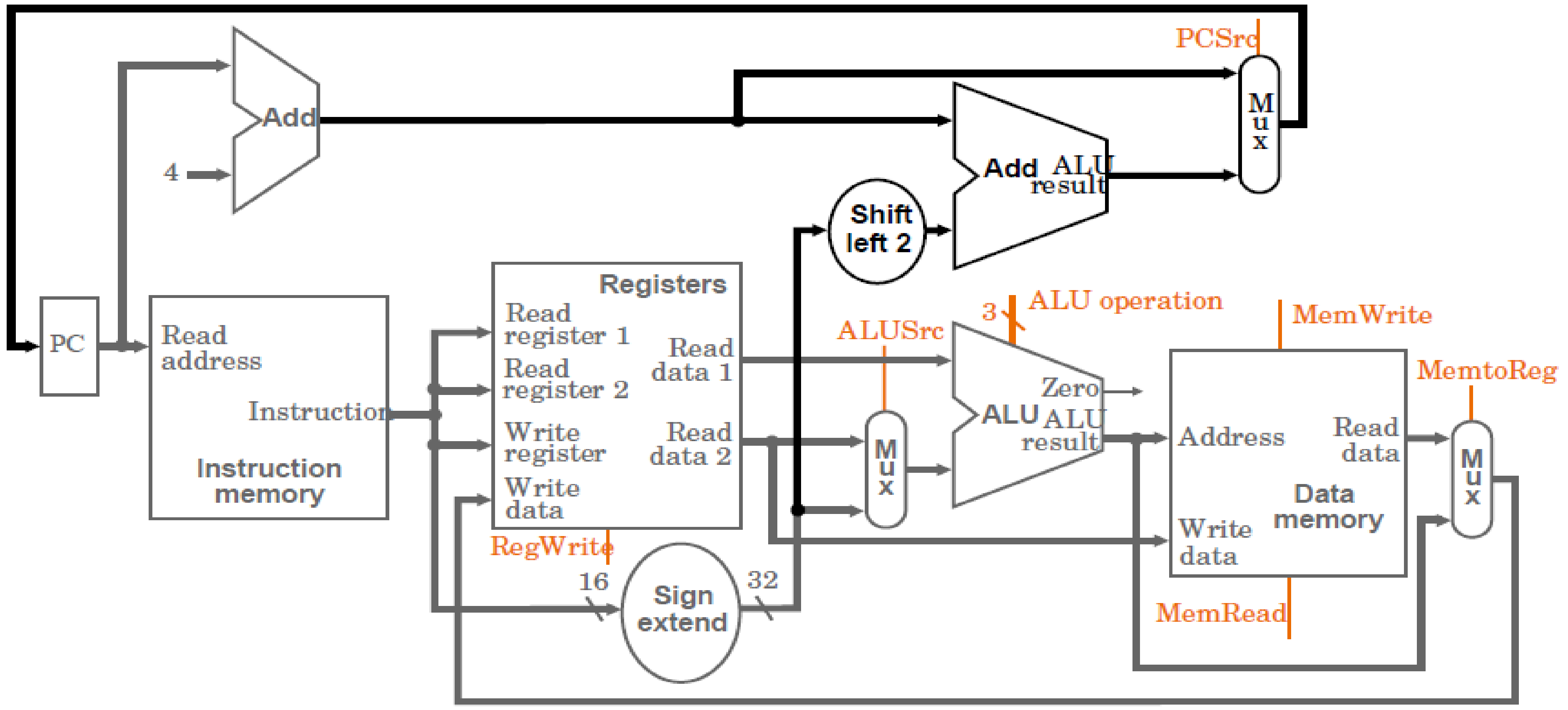
- Will do some in class review on Wed (june 17)
- Covering everything ending at Monday Lecture

What Statement about memory in the Datapath is NOT true ☺

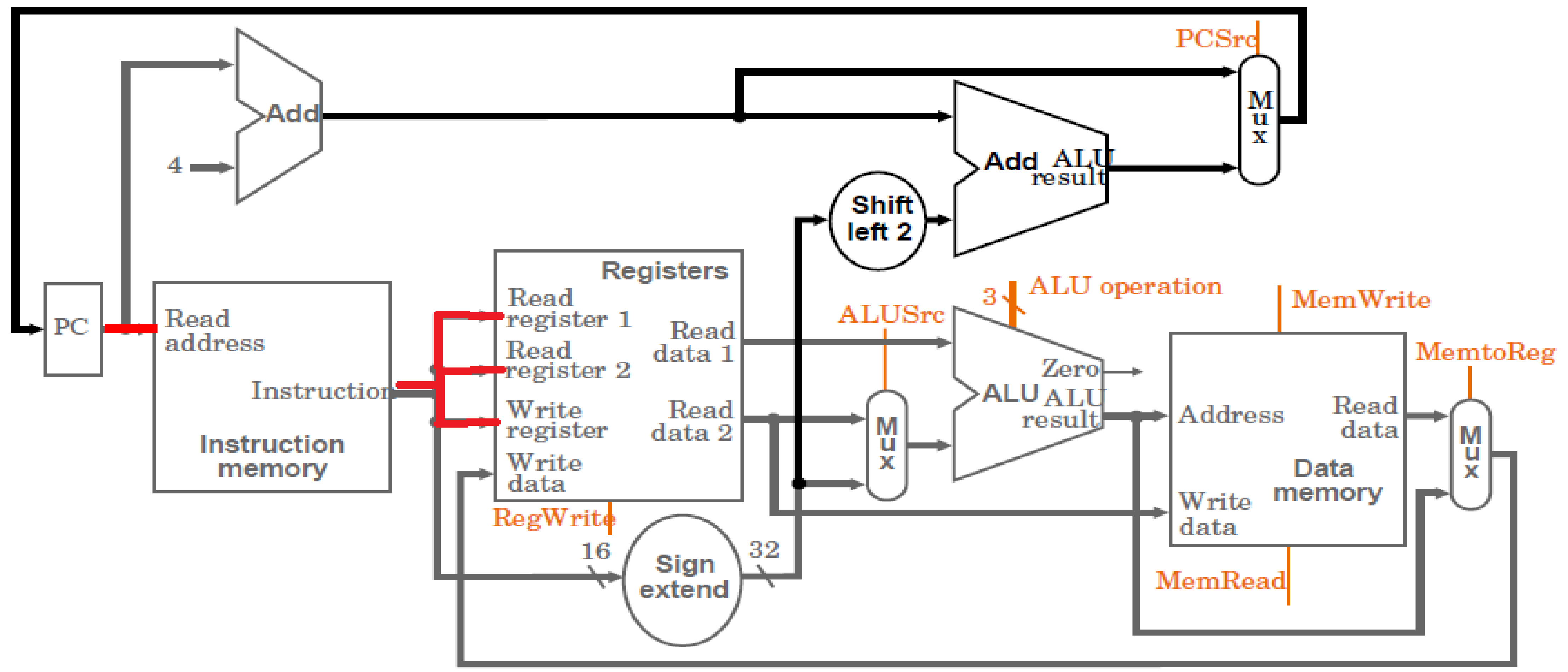
- A) the instruction and data memory are a type of cache
- B) Data memory can be read from and written to
- C) The Registers contain memory in flip flops
- D) The Register file allows reading and writing to the registers
- E) The Instruction memory in the datapath allows reading of instructions and writing new instructions

Assembled Datapath Without Control Unit:




PCSrc
Add
4
M u x
Add ALU result
Shift left 2
PC
Read address
Instruction
Instruction memory
Registers
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
RegWrite
16
Sign extend
32
ALUSrc
M u x
3
ALU operation
Zero
ALU
ALU result
MemWrite
Address
Read data
Write data
Data memory
MemRead
MemtoReg
M u x

add $t1, $t2, $t3

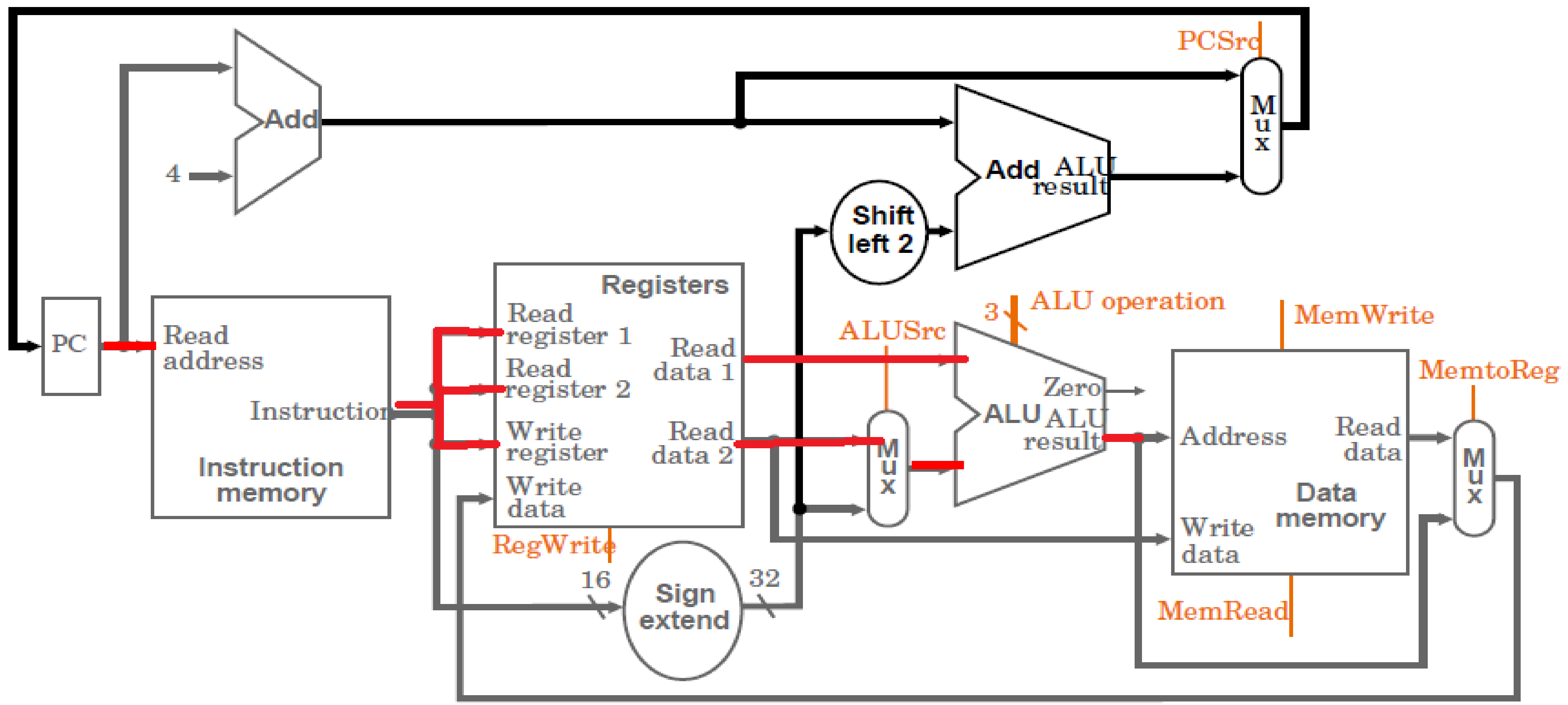


add $t1, $t2, $t3

First Step: Always Fetch Instruction from Instruction Memory

Next: Use all the Instruction Bits

*Access the Register File with the two source register numbers

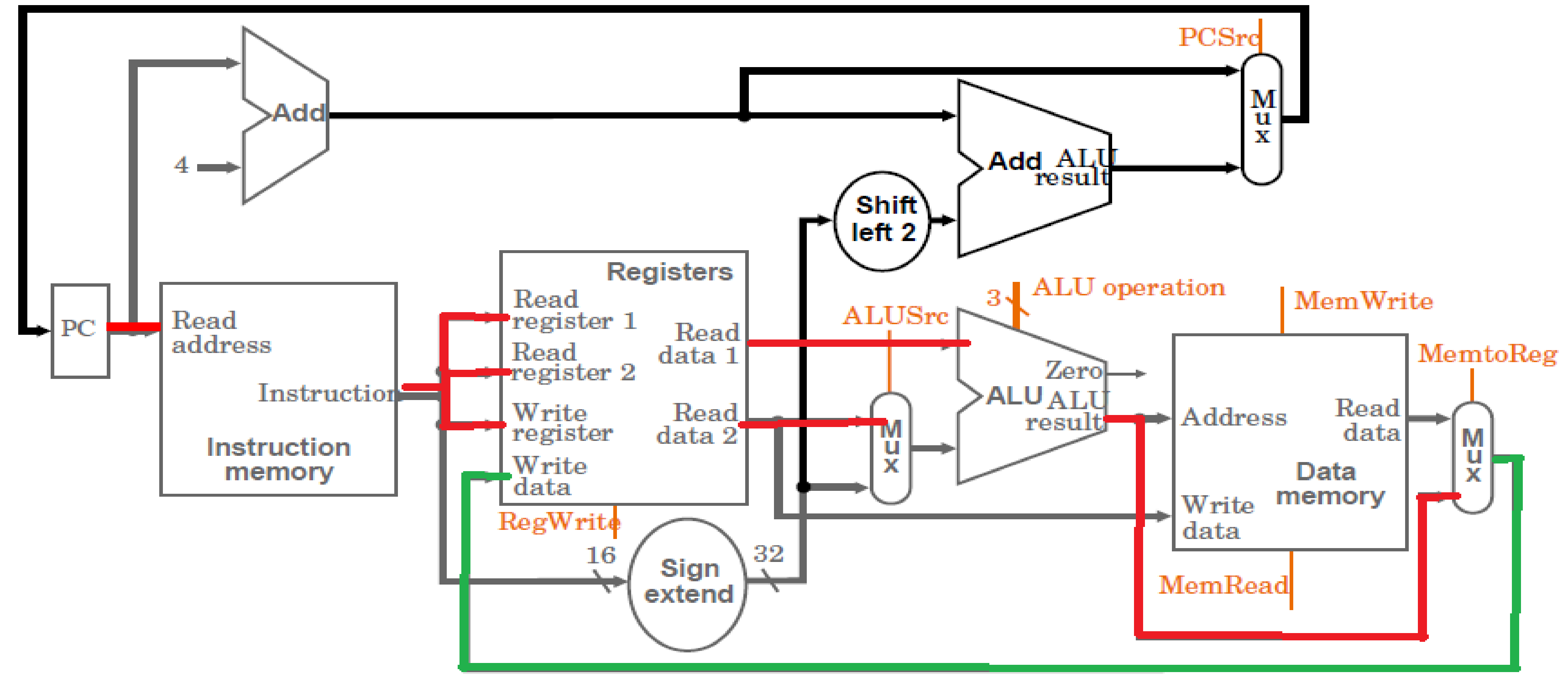


add $t1, $t2, $t3

Next: Take Data of the two source registers

*Send this through the ALU, Perform an operation as given in the instruction (add).

*Produce a Result – out of the ALU



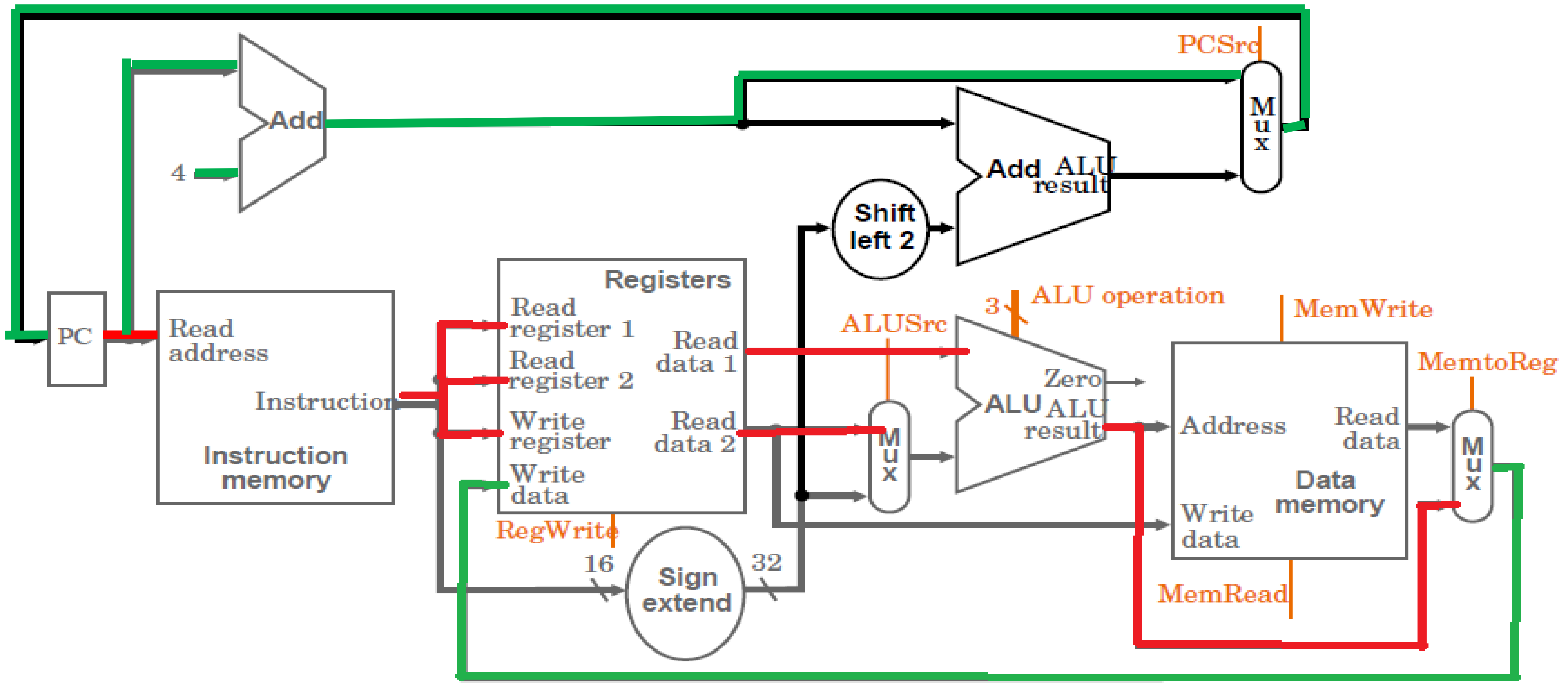
add $t1, $t2, $t3

Next: This result needs to be Written to the Destination register.

*provide the result as Write Data to register file

*Writing to the Register file occurs at END of the clock cycle.

All writes will occur at the end of the clock cycle: Including PC update



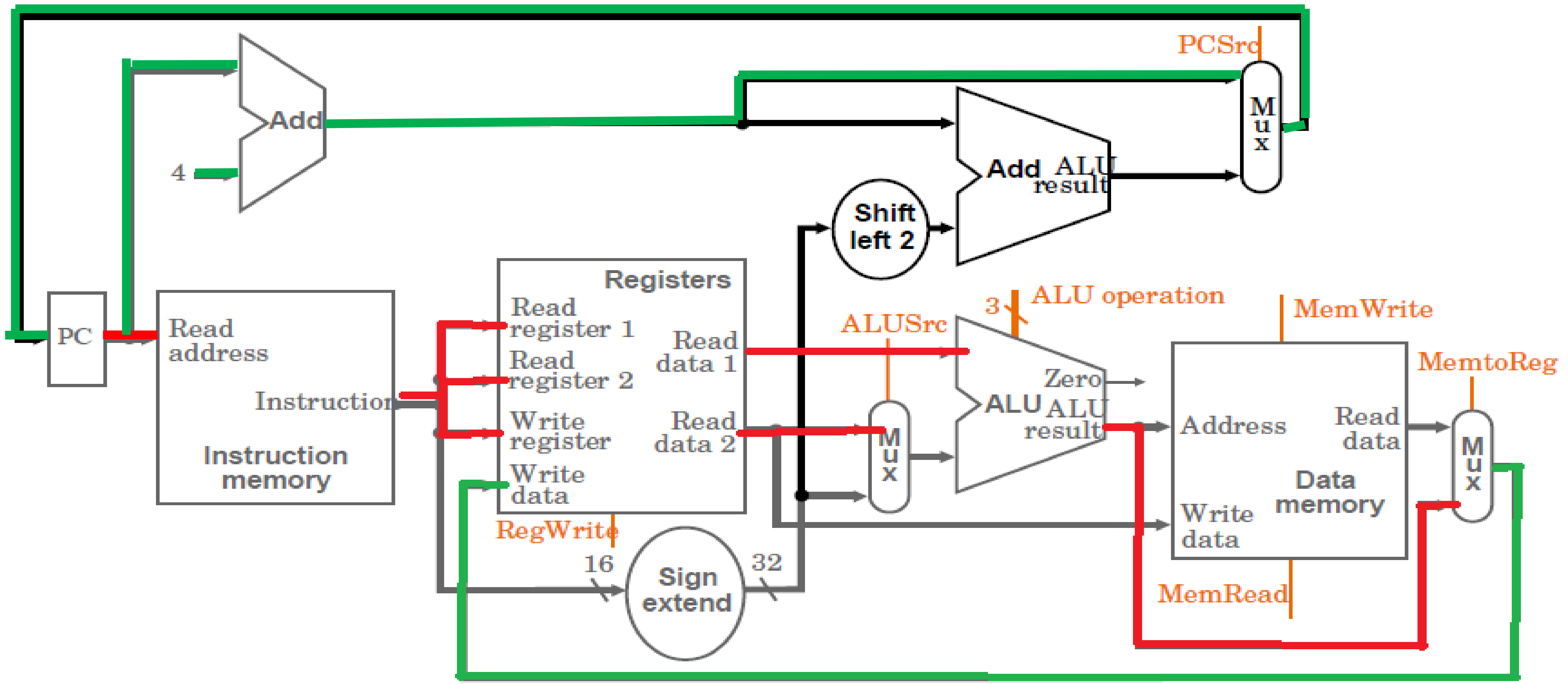
add $t1, $t2, $t3

Next: This result needs to be Written to the Destination register.

*provide the result as Write Data to register file

*Writing to the Register file occurs at END of the clock cycle.

All writes will occur at the end of the clock cycle: Including PC update

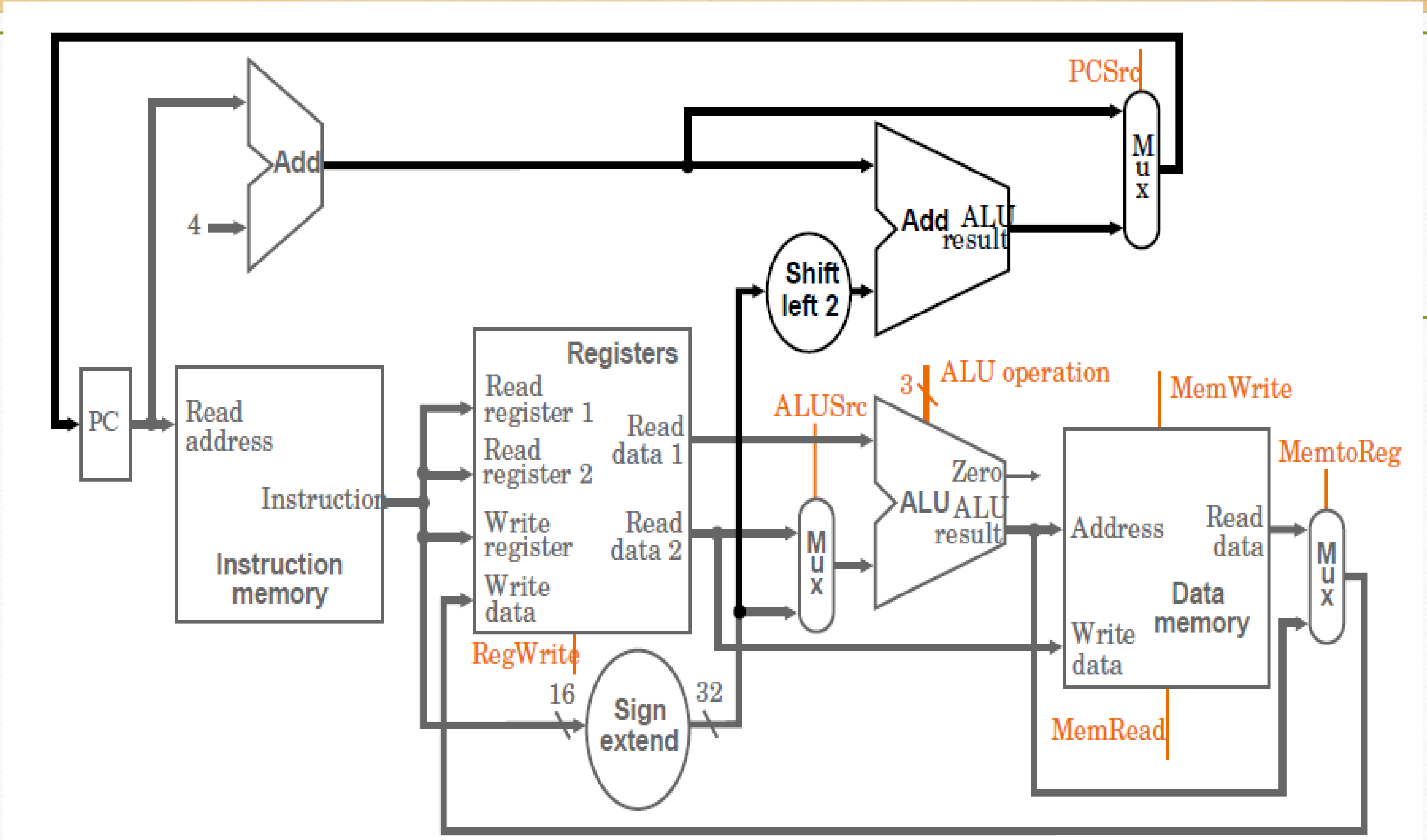


add $t1, $t2, $t3

Important: Control Line Bits (Select lines to Multiplexors, and other control lines) are essential in this process

Control Lines must guarantee that Add instruction will get executed in the datapath

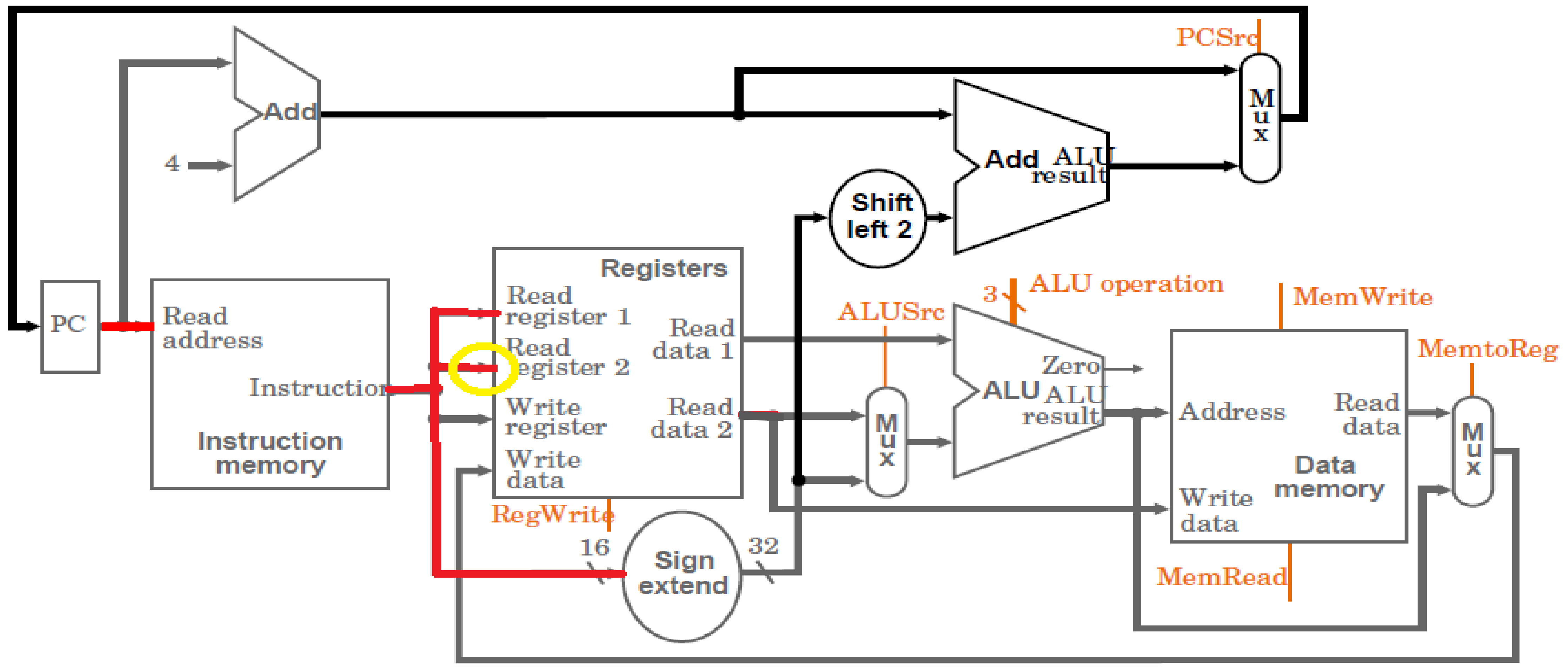
lw $t1, 100($t2)



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

lw $t1, 100($t2)

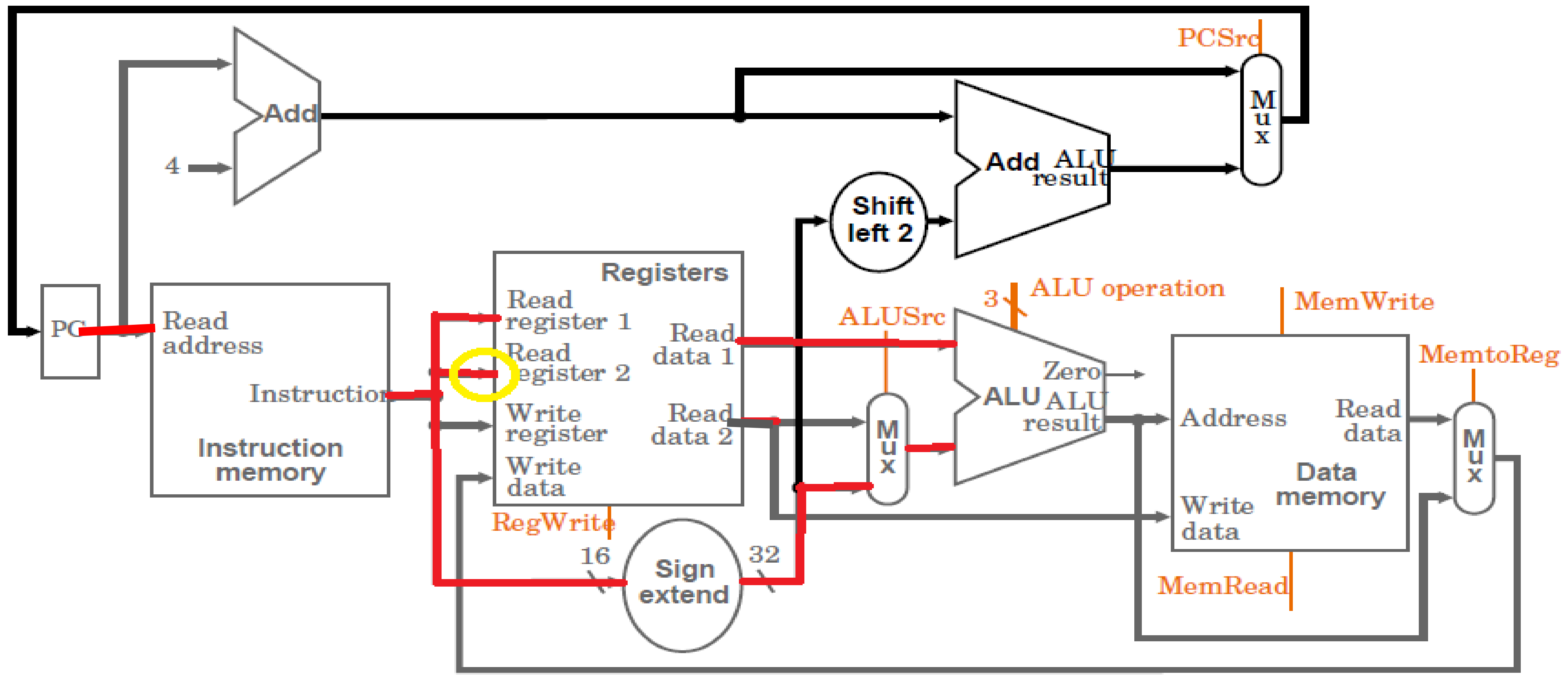


Now the **rt** register is the destination _Not a source

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

lw $t1, 100($t2)



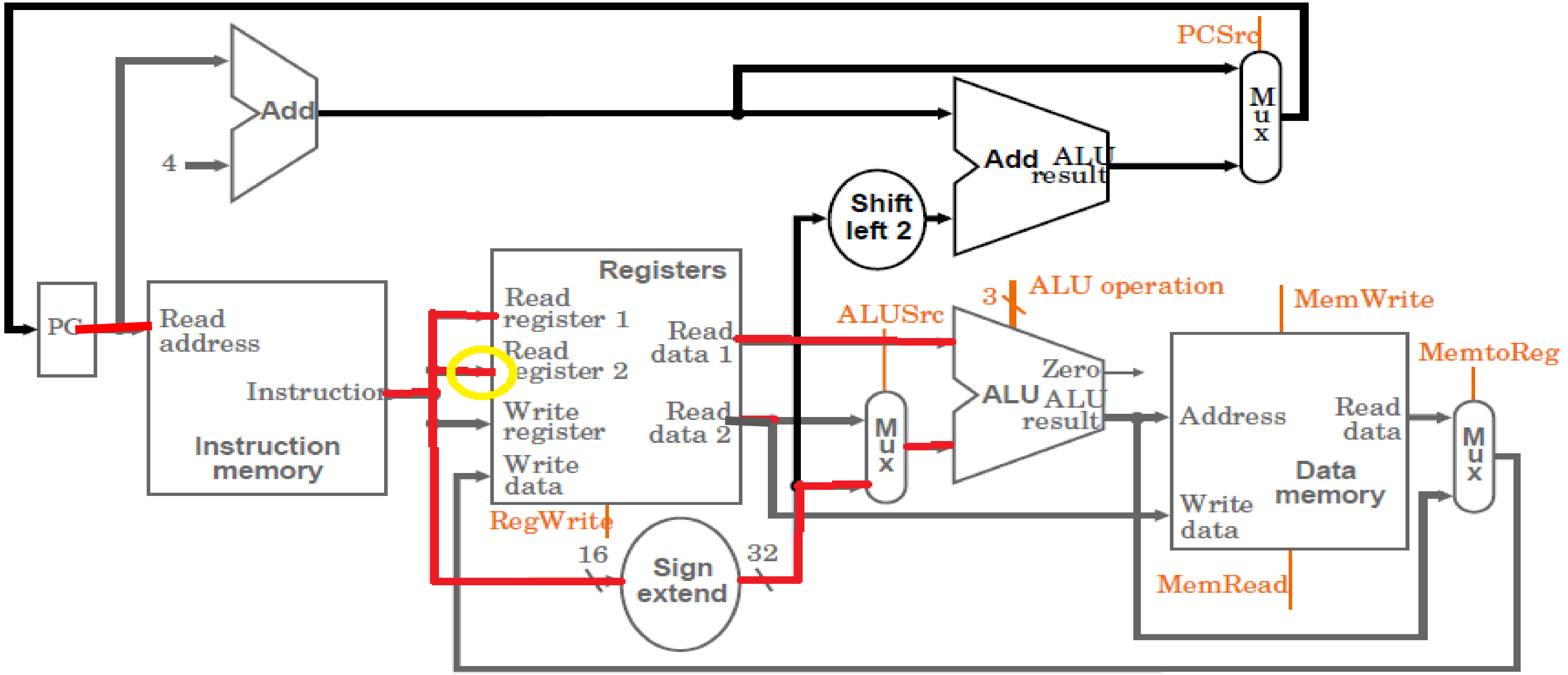
ALU: Will perform rs + sign extended offset

*Sign Extend ?

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

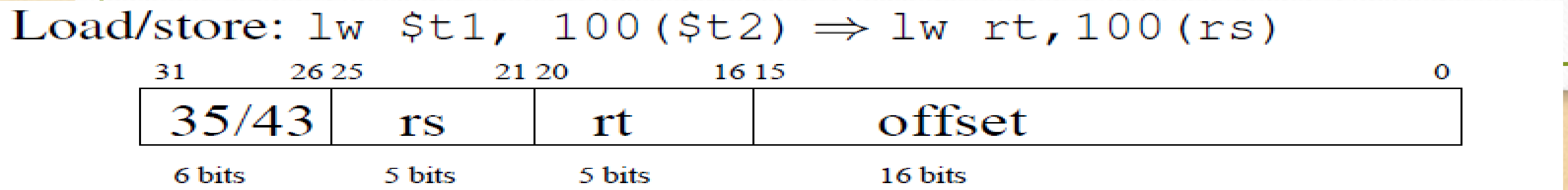
31 26	25 21	20 16	15 0
35/43	rs	rt	offset
6 bits	5 bits	5 bits	16 bits

lw $t1, 100($t2)



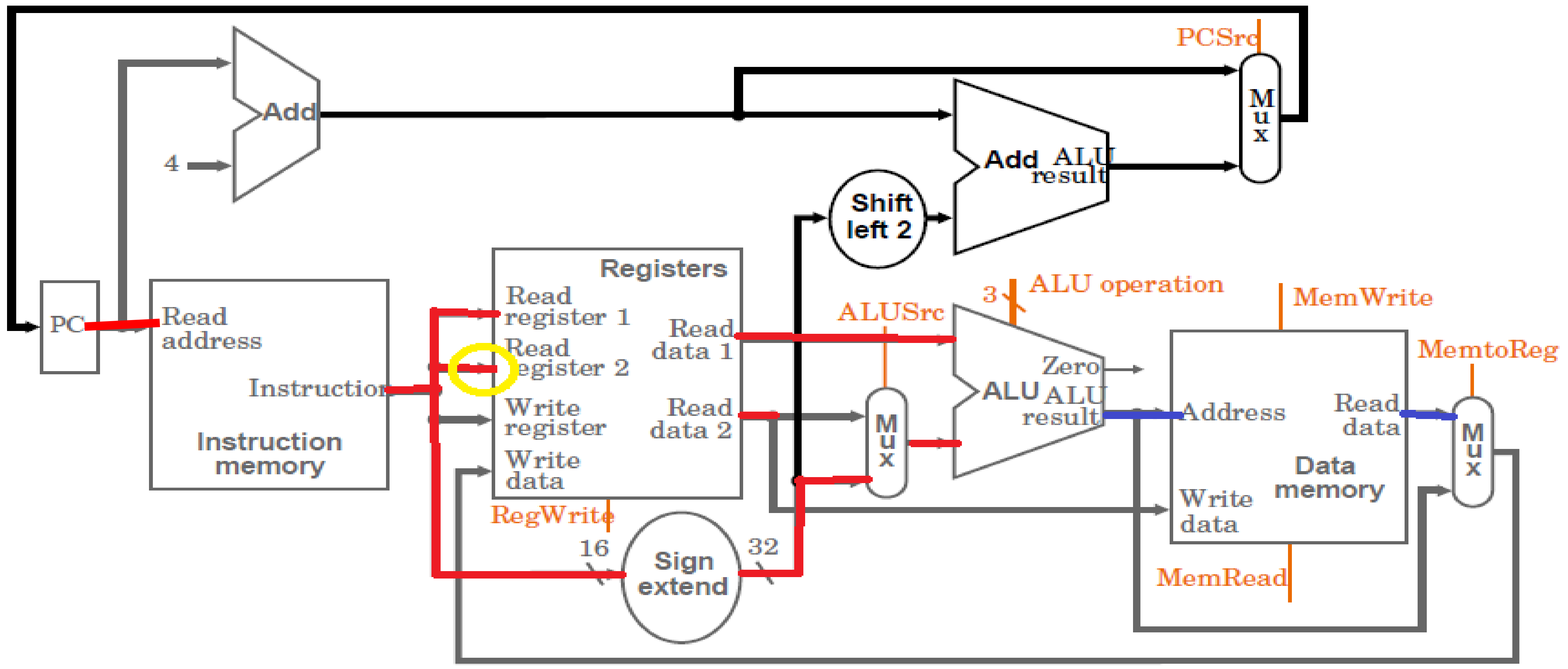
ALU: Will perform rs + sign extended offset

*Sign Extend ➔ALU needs two inputs that are 32 bits each


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

lw $t1, 100($t2)

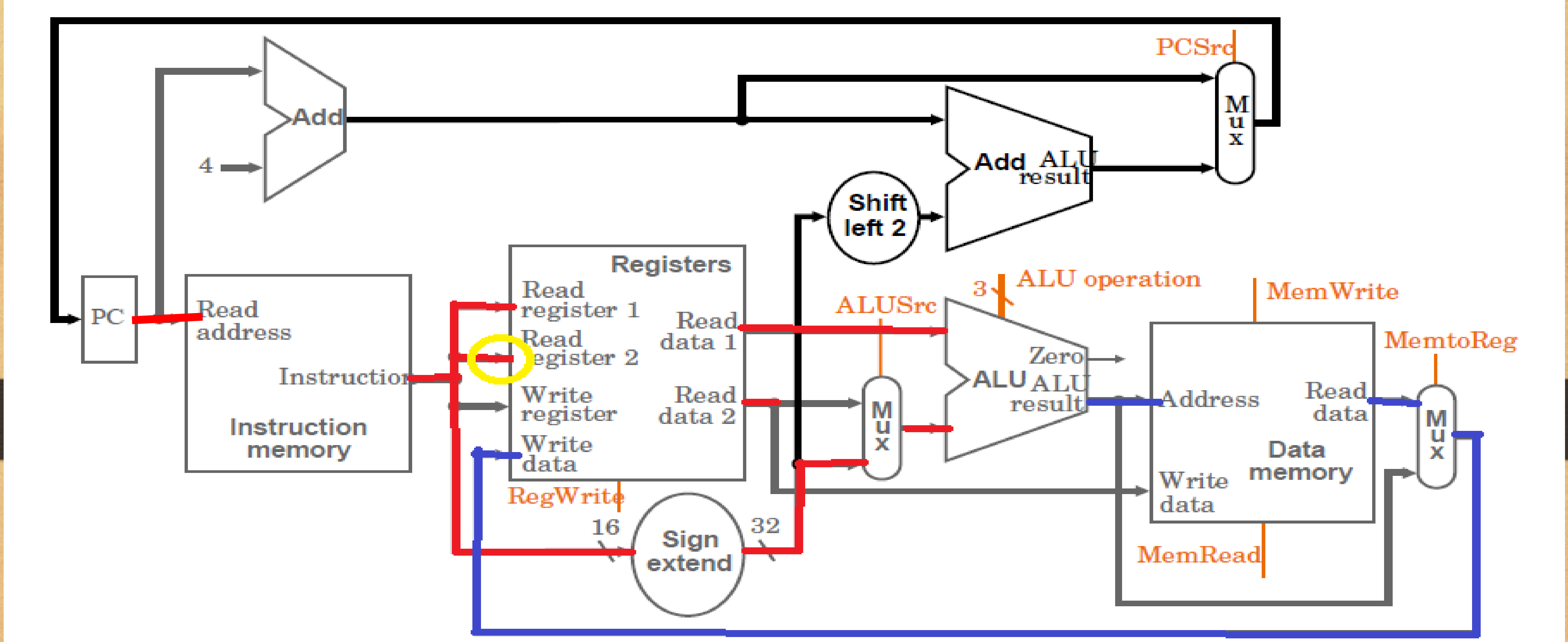


Result of ALU is an Address: Use this address to Access Data Memory

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

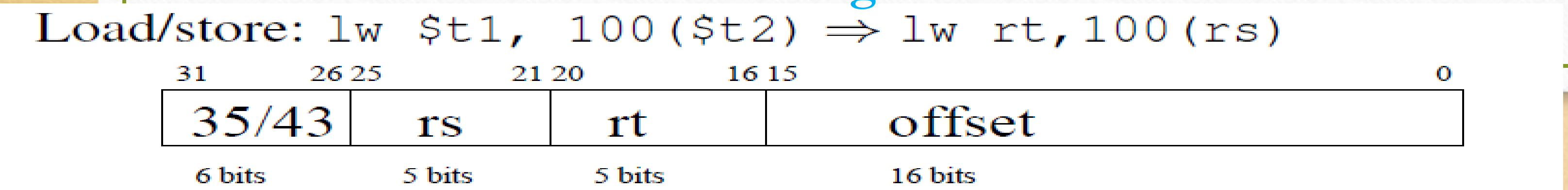
31 26	25 21	20 16	15 0
35/43	rs	rt	offset
6 bits	5 bits	5 bits	16 bits

lw $t1, 100($t2)



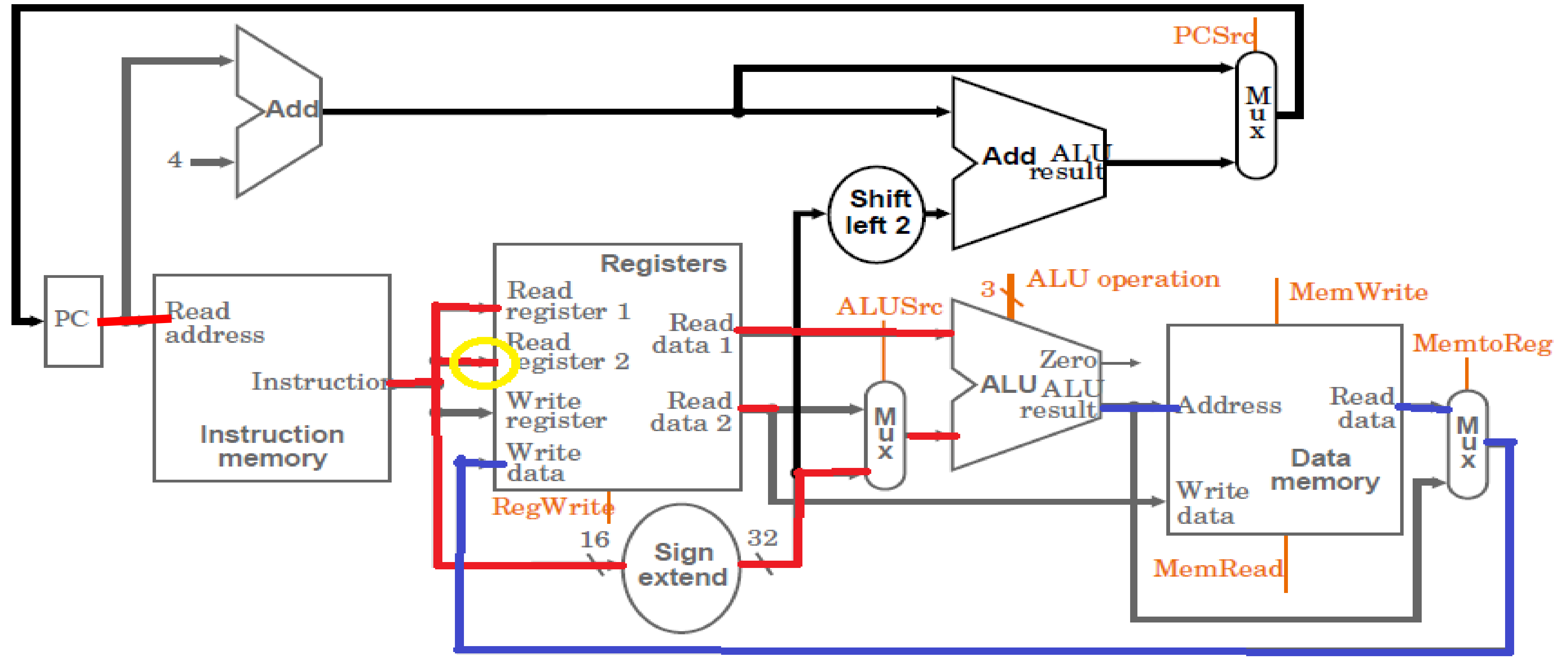
Once we access data memory → we now have 32 bits of data

* this data needs to be written to destination register (rt)
* therefore send data back to register file


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

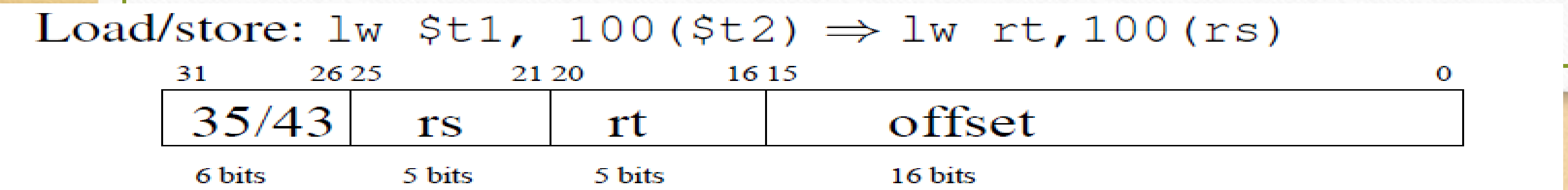
31 26	25 21	20 16	15 0
35/43	rs	rt	offset
6 bits	5 bits	5 bits	16 bits

lw $t1, 100($t2)



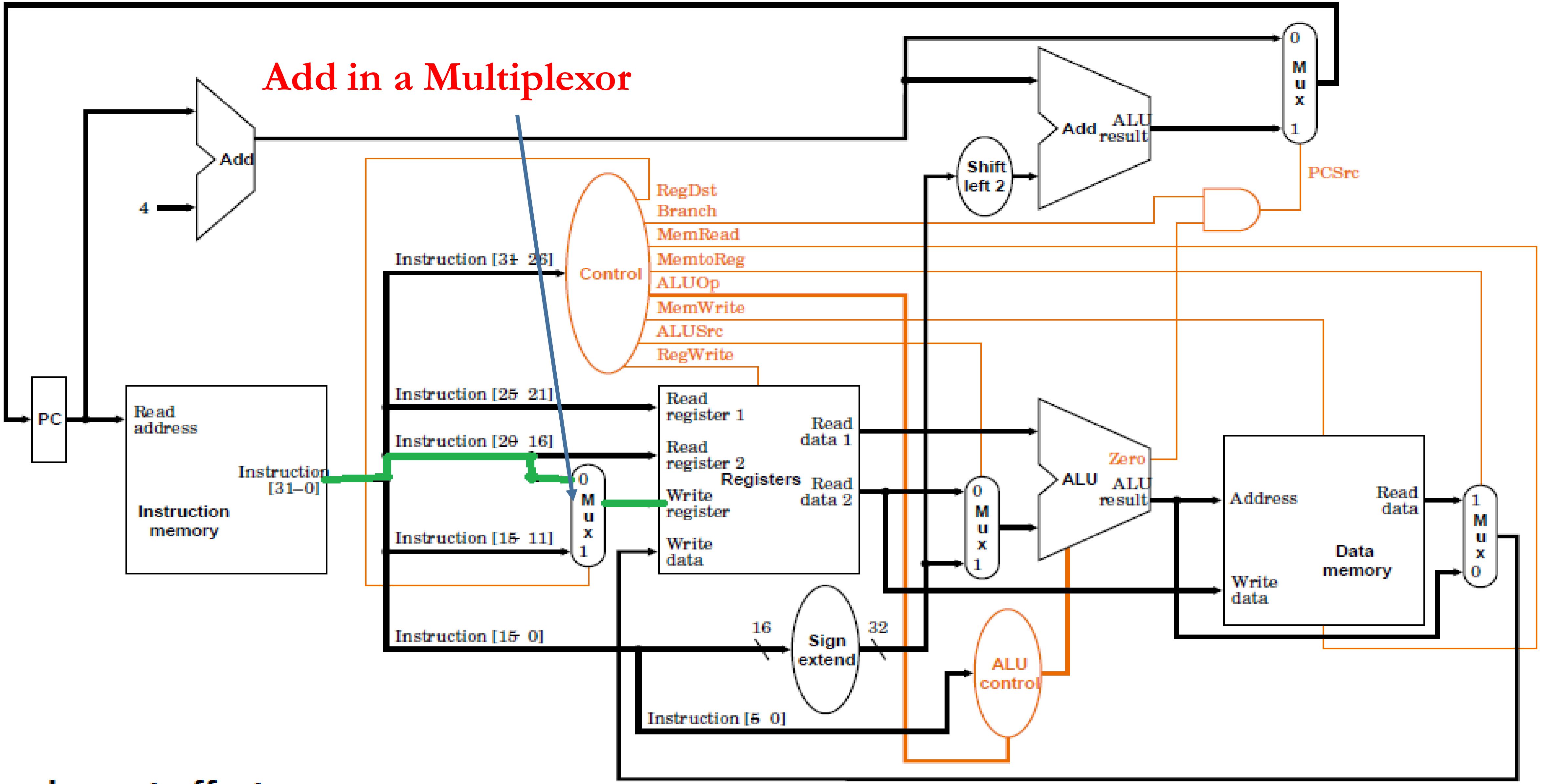
We need rt to be the Write Register

Need more hardware here : to connect rt as also an option for a write register



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



lw rs rt offset

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