MIPS Datapath

CMSC 301

Prof Szajda

Goal

- Build an architecture to support the following instructions
  - Arithmetic: add, sub, addi, slt
  - Memory references: lw, sw
  - Branches: j, beq

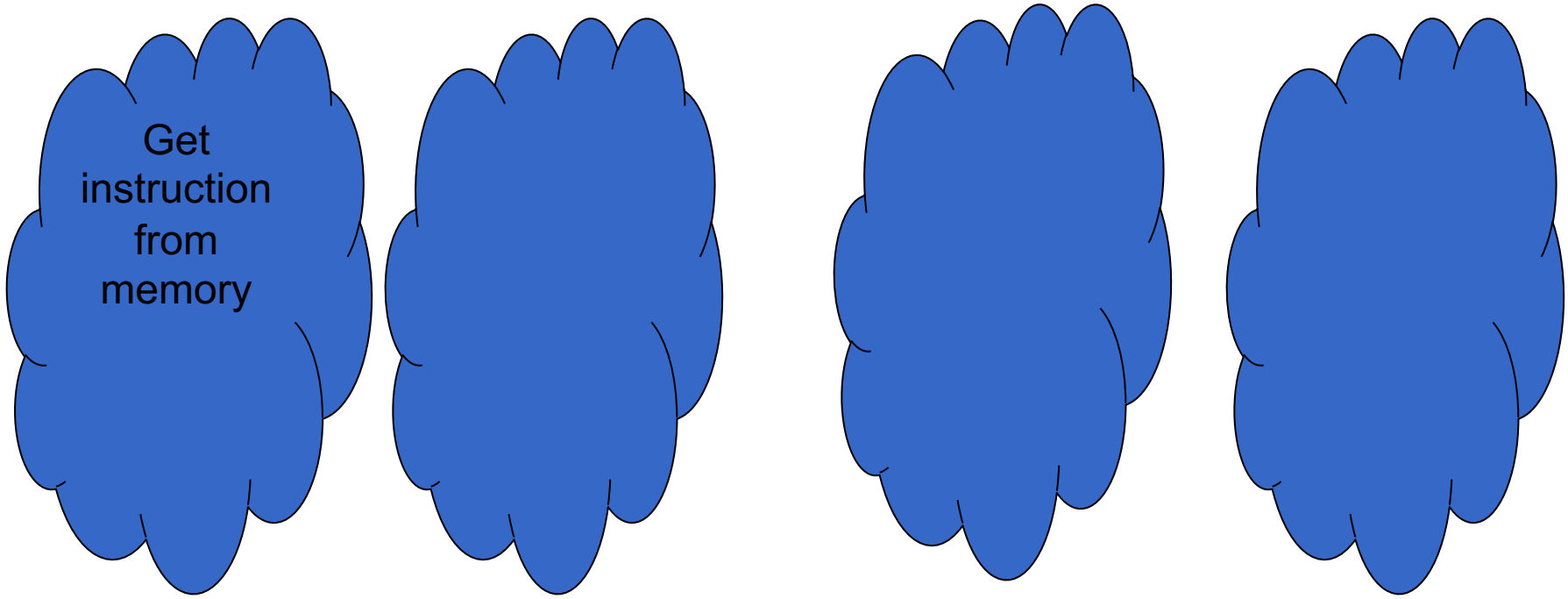
Process

1) Design basic framework that is needed by all instructions
2) Build a computer for each operation individually
3) Add MUXs to choose between different operations
4) Add control signals to control the MUXs

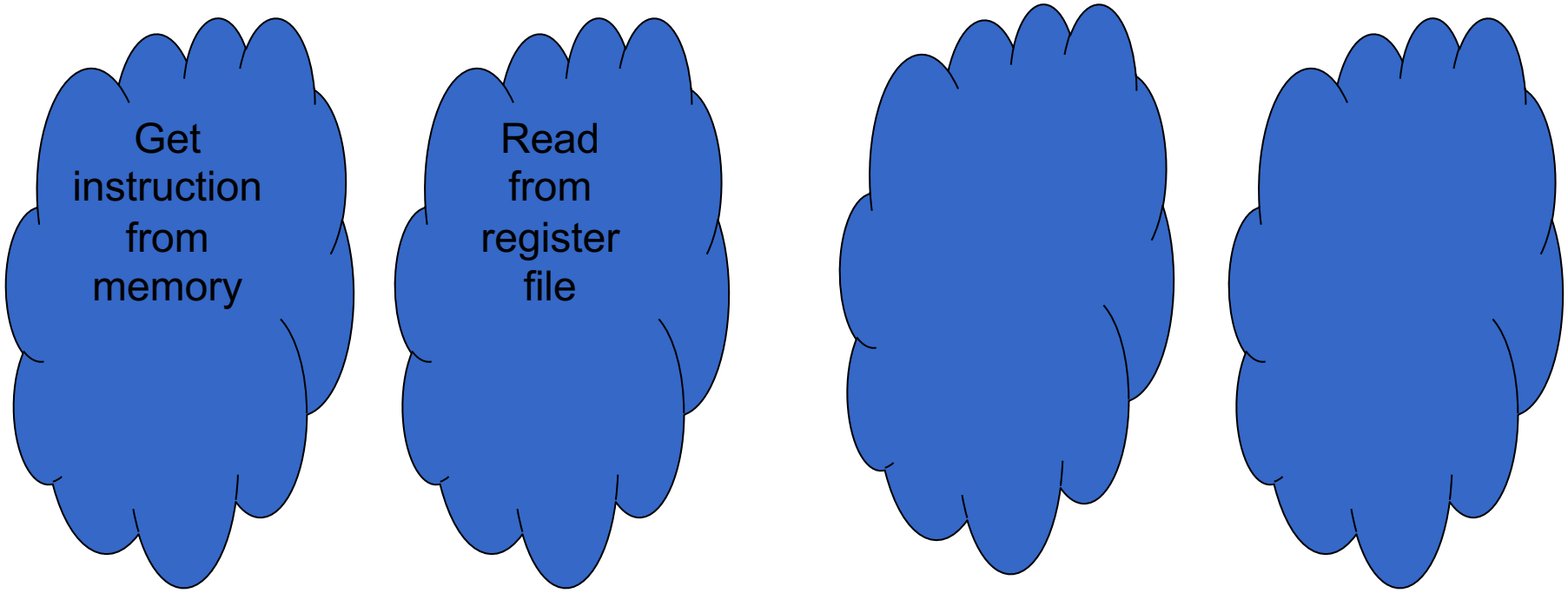
MIPS Steps

- Get an instruction from memory using the Program Counter (PC)
- Read one or two registers each instruction
  - One register: addi, lw
  - Two registers: add, sub, slt, sw, beq
- All instructions use ALU after reading regs
- Some instructions also access Memory
- Write result to Register file

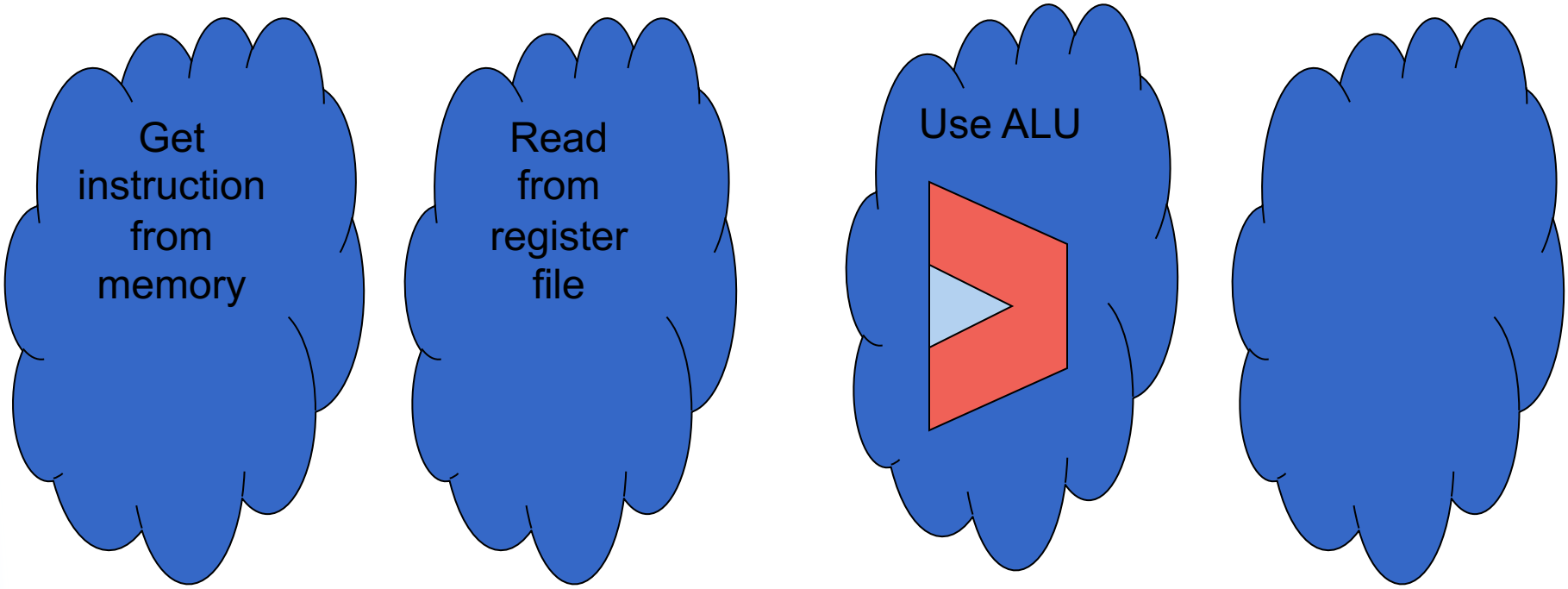
Framework



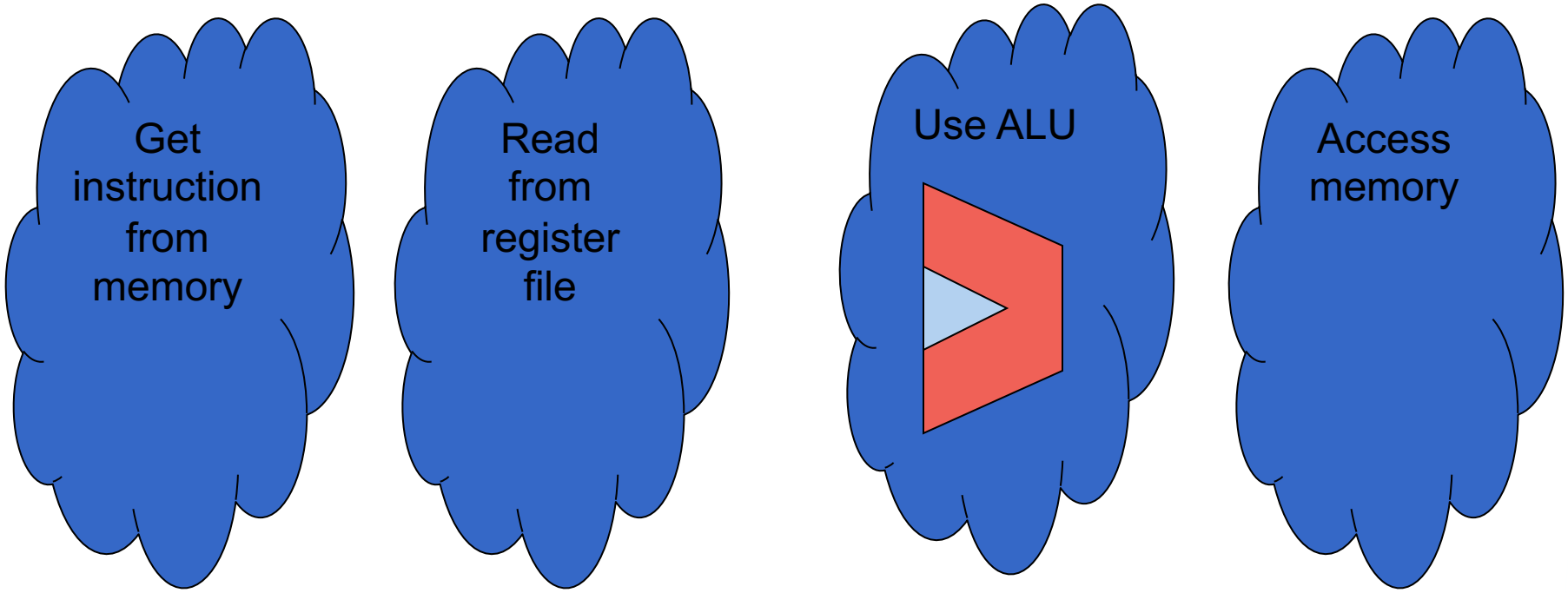
Framework



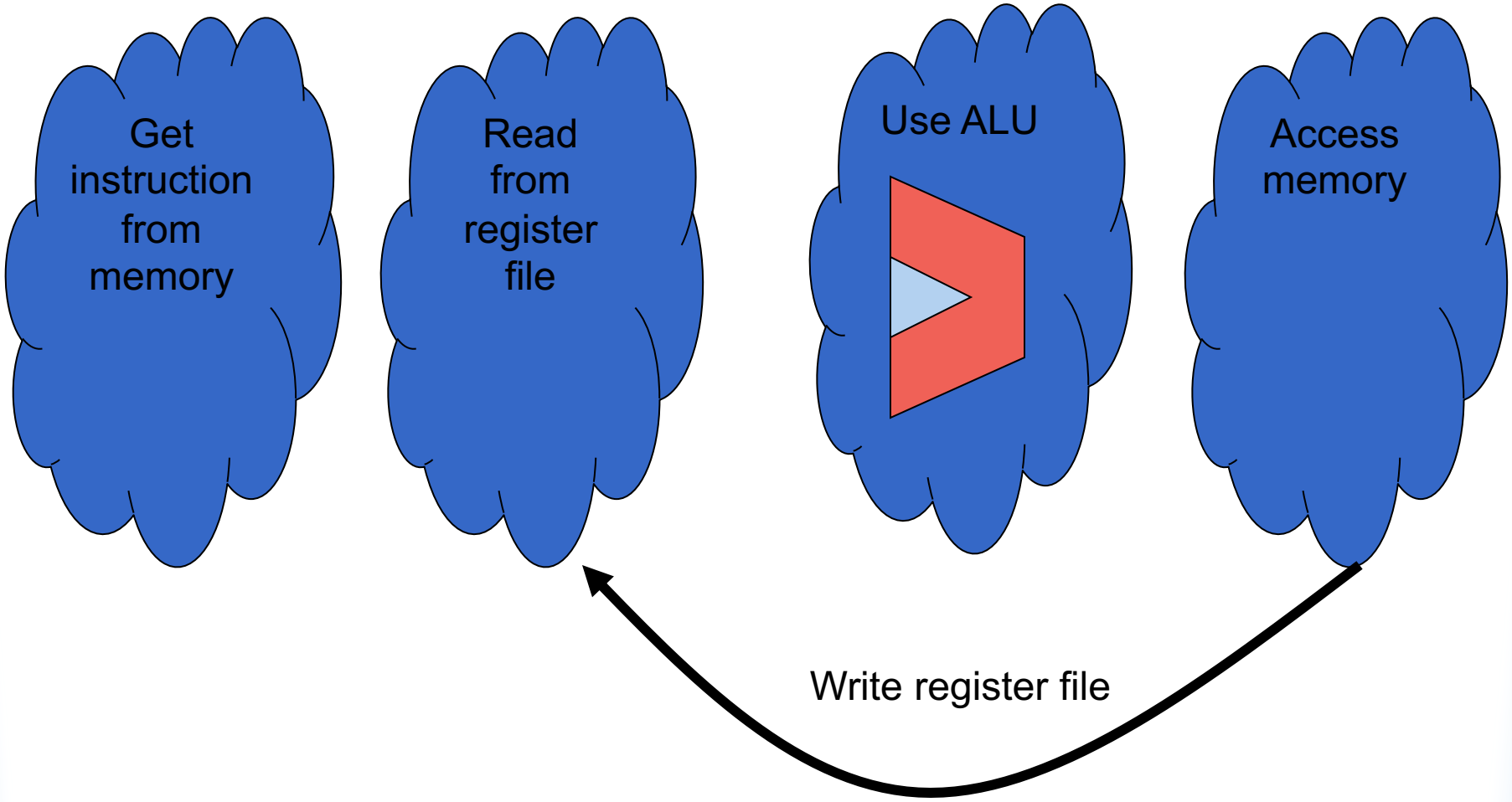
Framework


Get instruction from memory
Read from register file
Use ALU

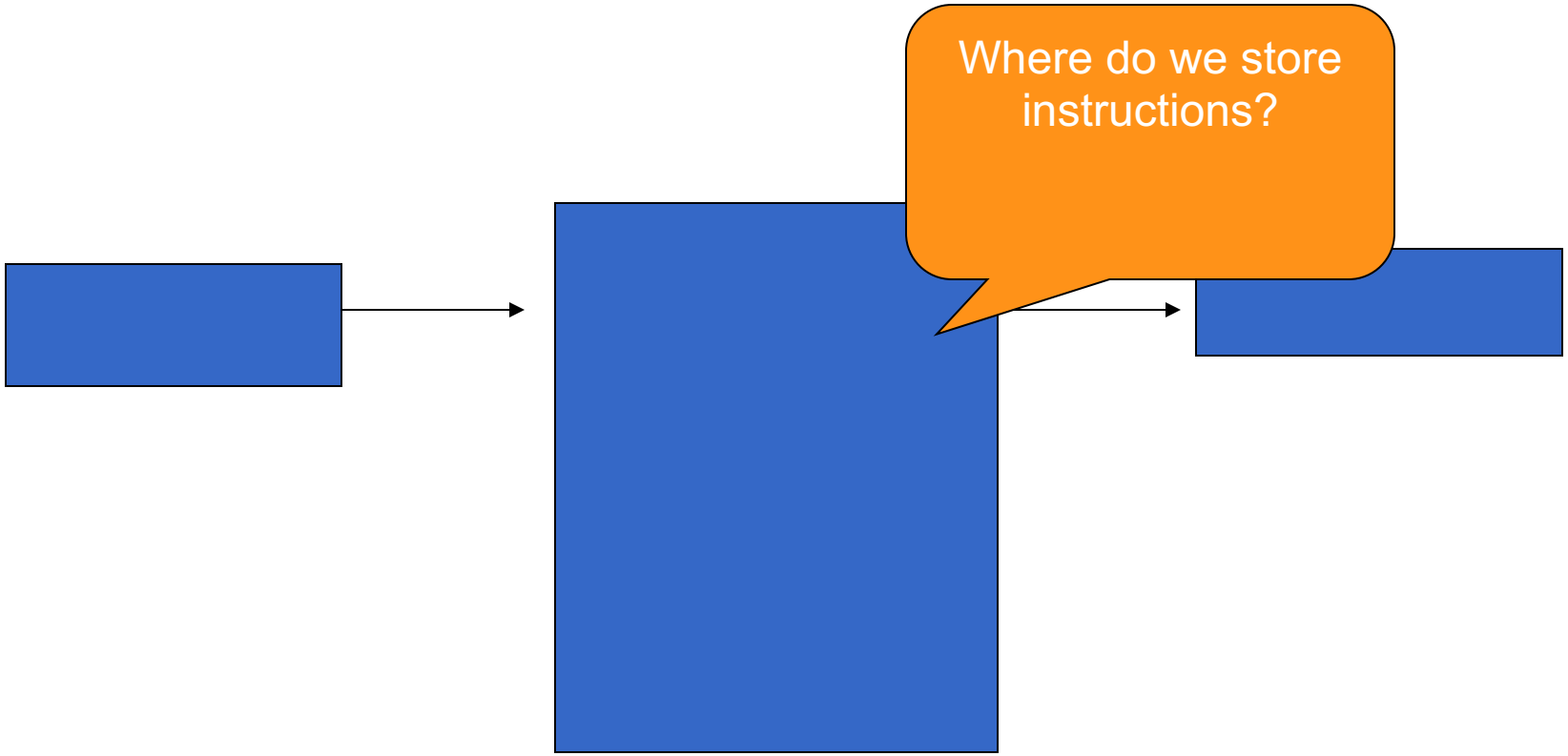
Framework


Get instruction from memory
Read from register file
Use ALU
Access memory

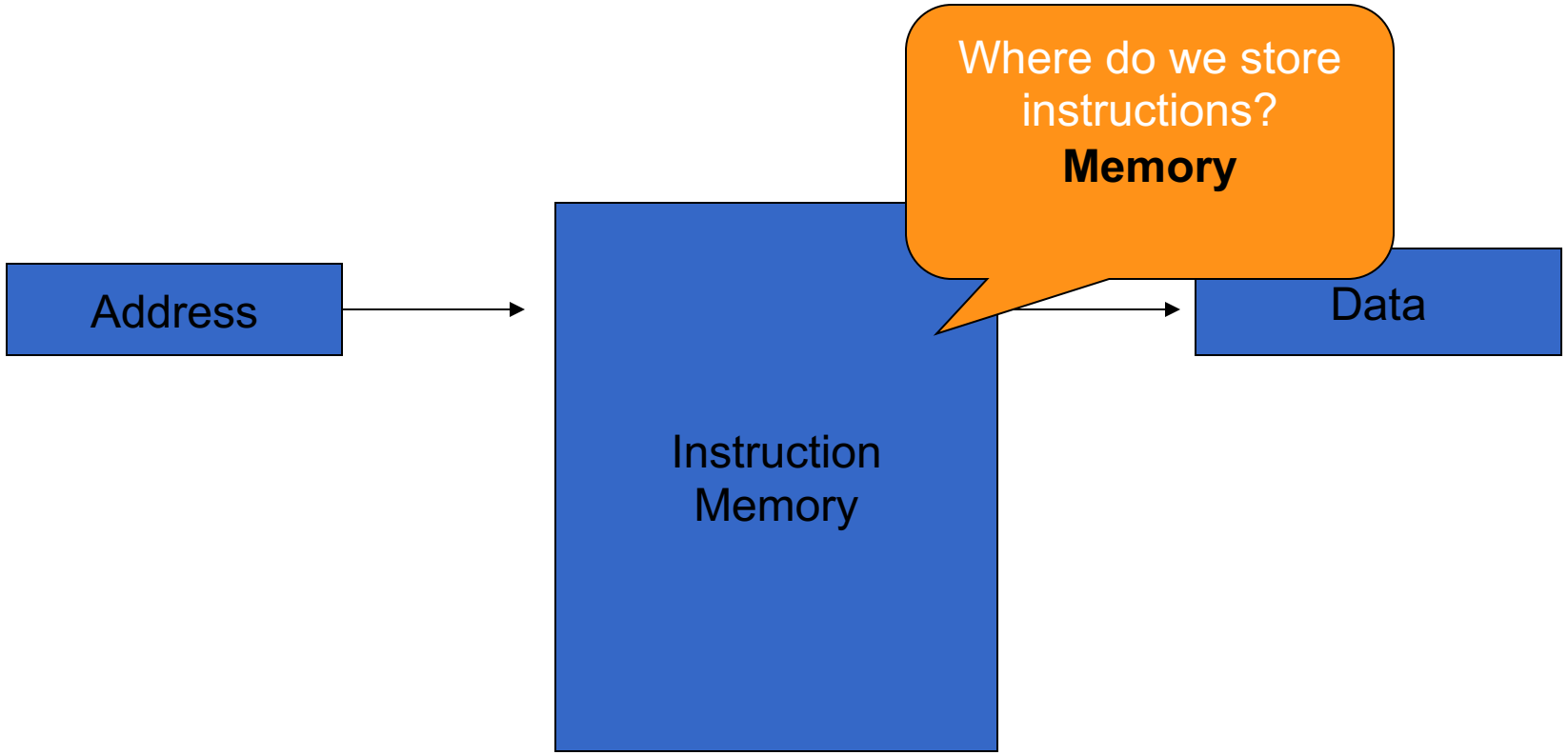
Framework


Get instruction from memory
Read from register file
Use ALU
Access memory
Write register file

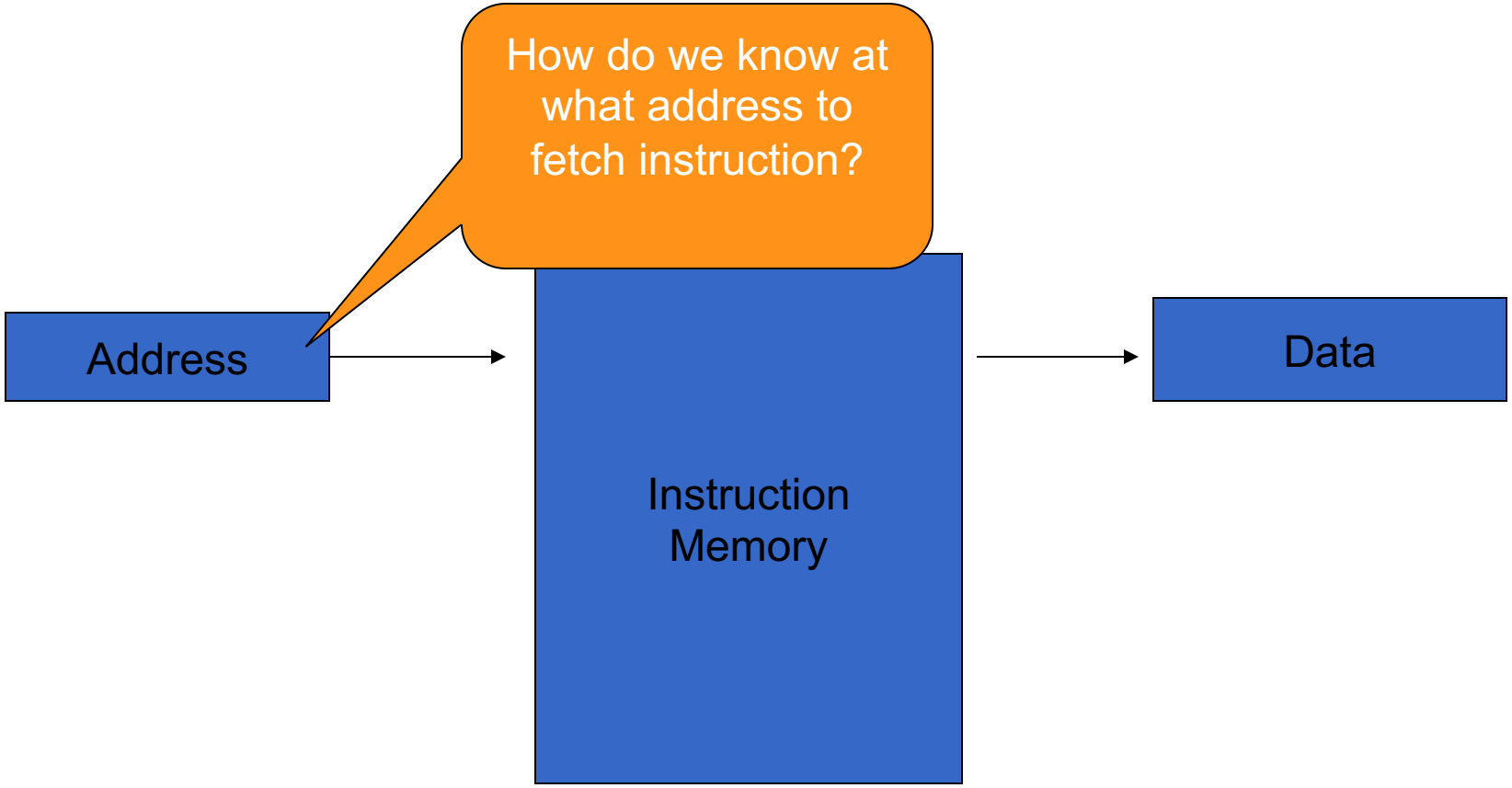
Get Instruction



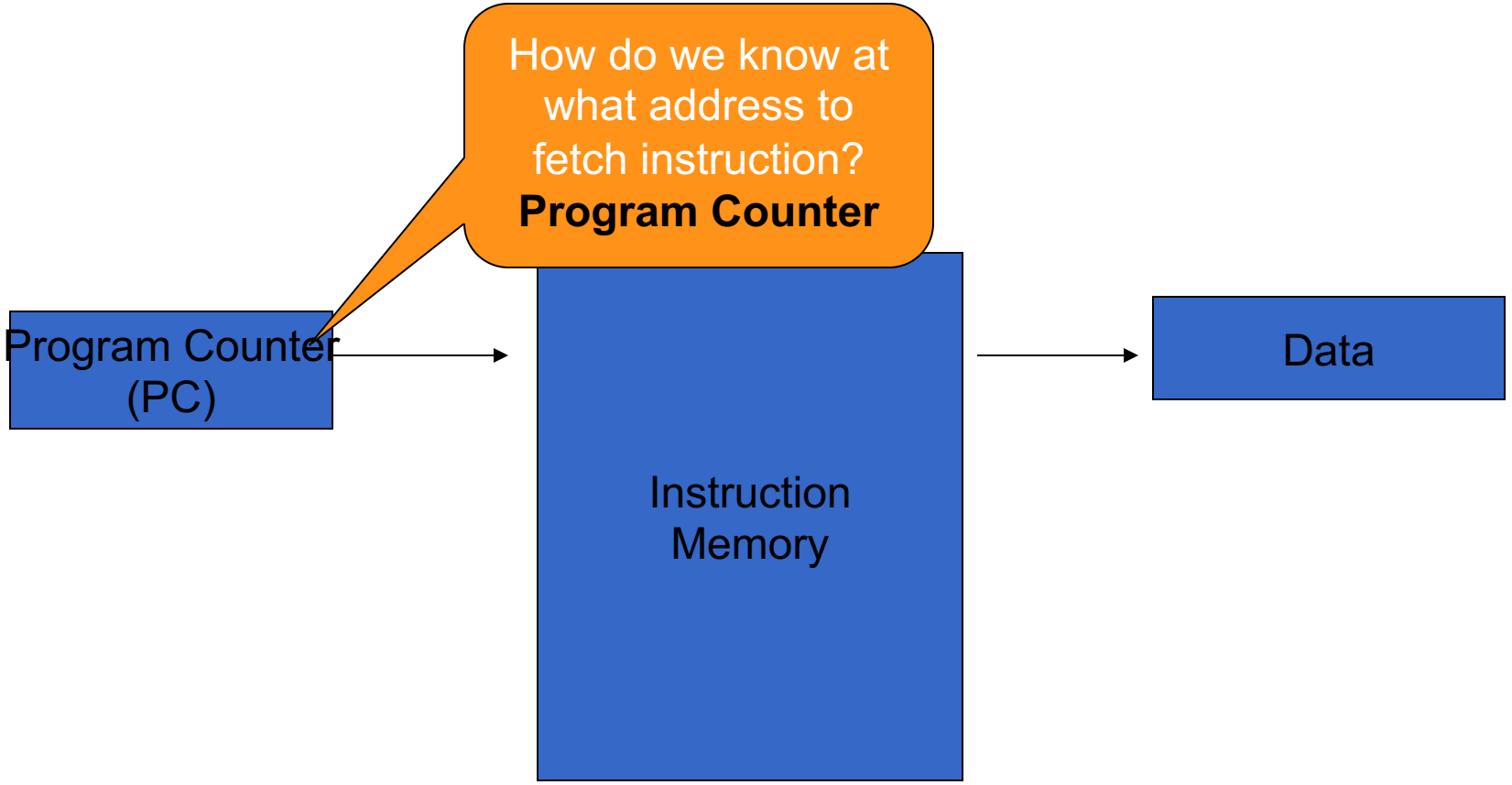
Get Instruction



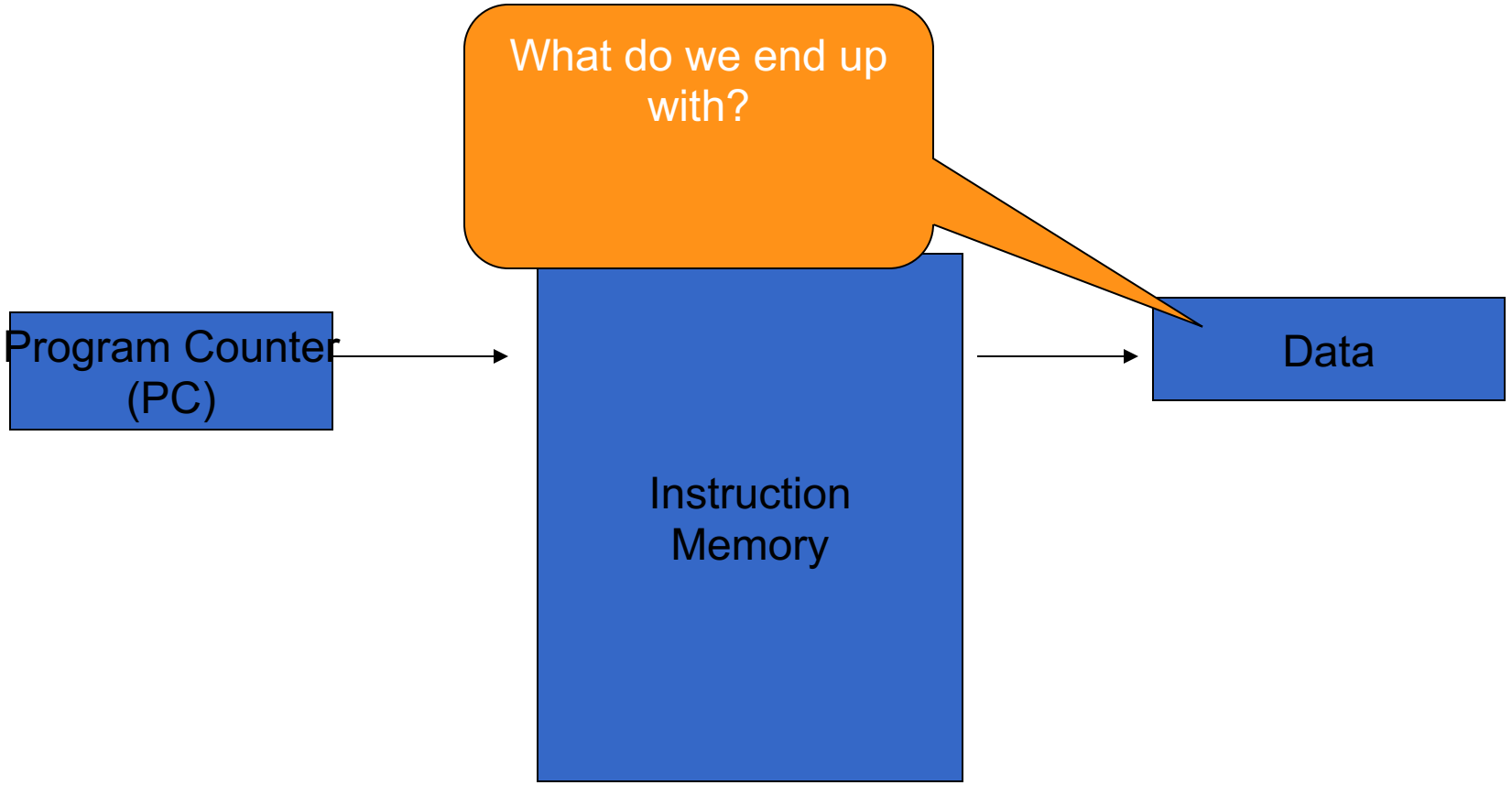
Get Instruction



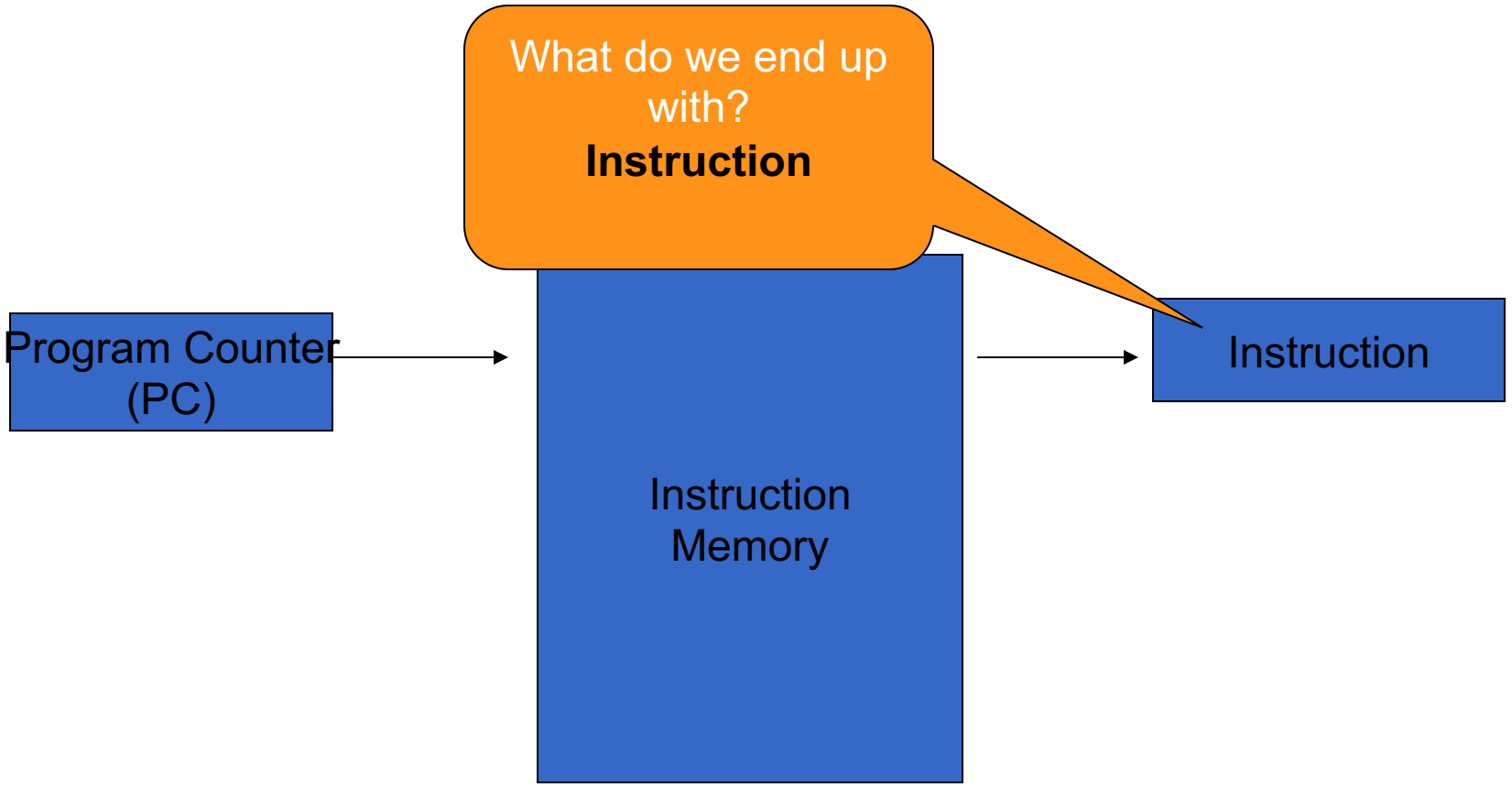
Get Instruction



Get Instruction

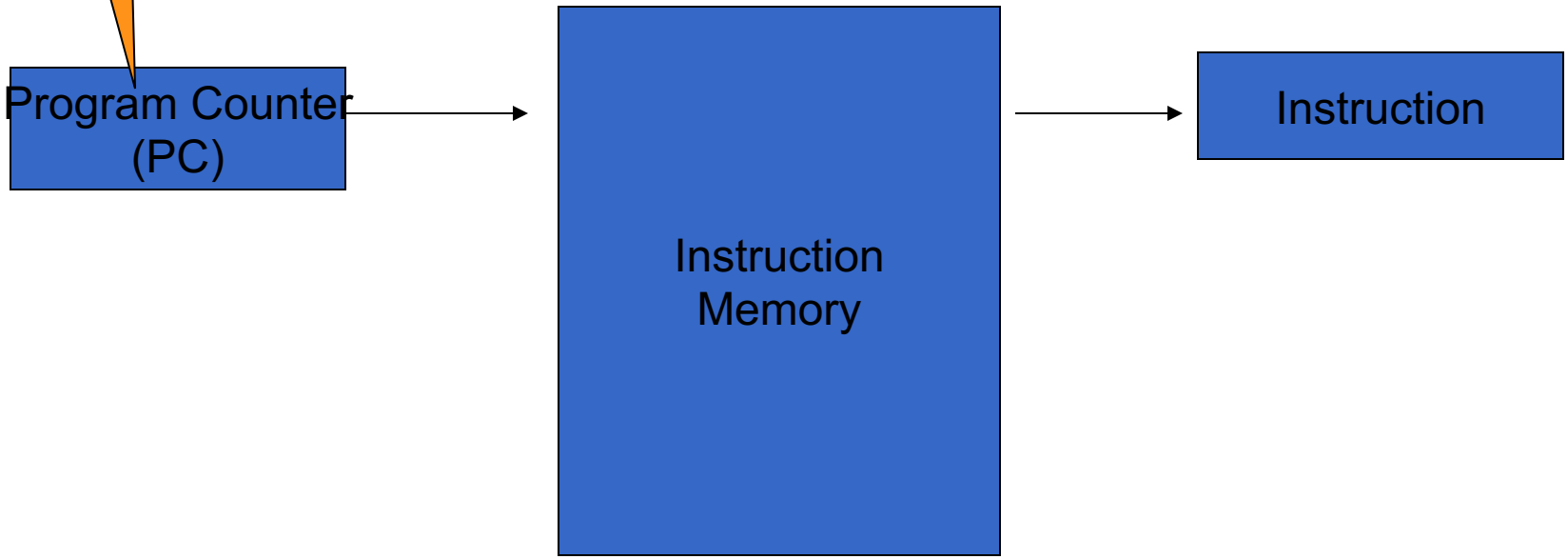


Get Instruction



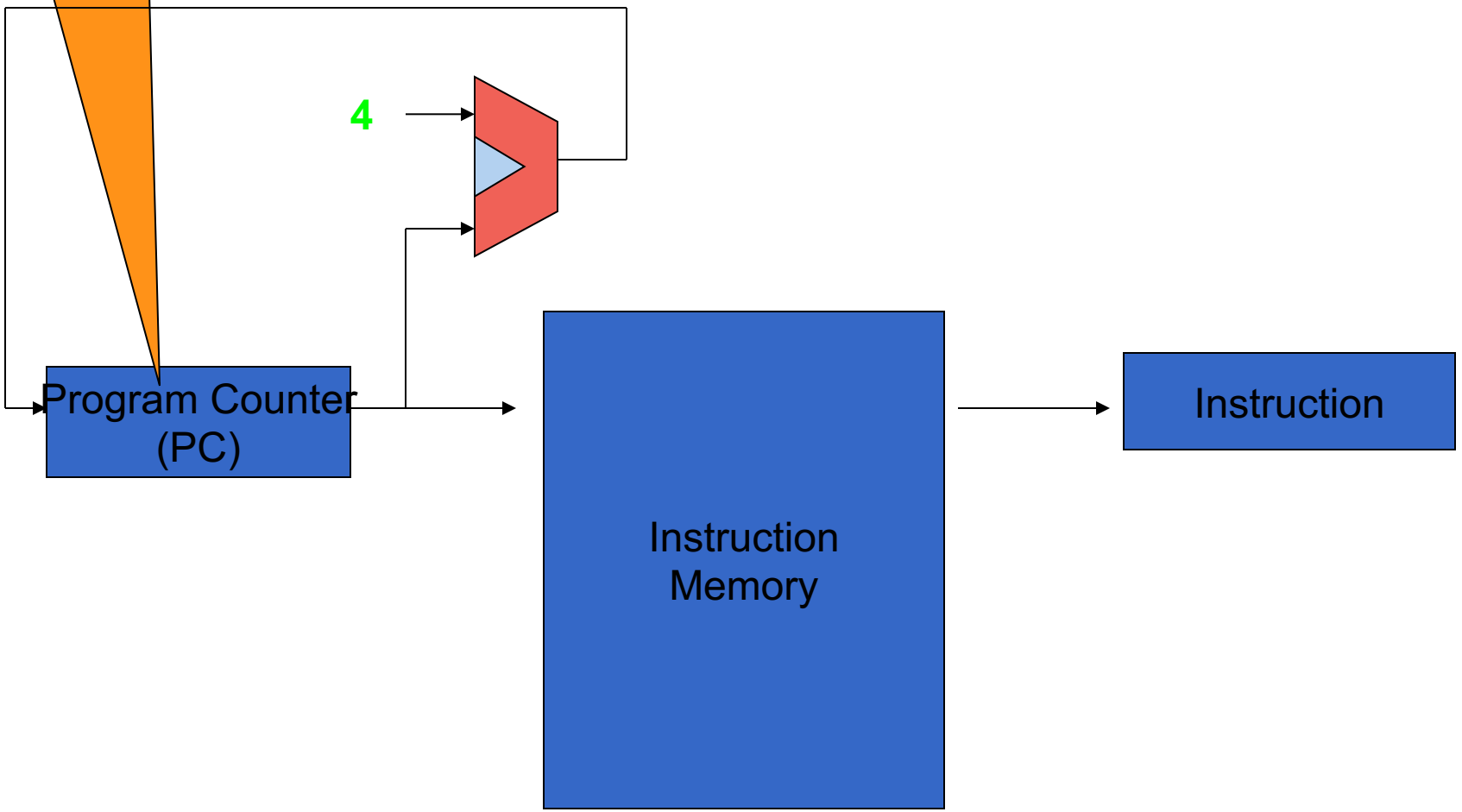
Get Instruction

What happens to the PC each instruction?

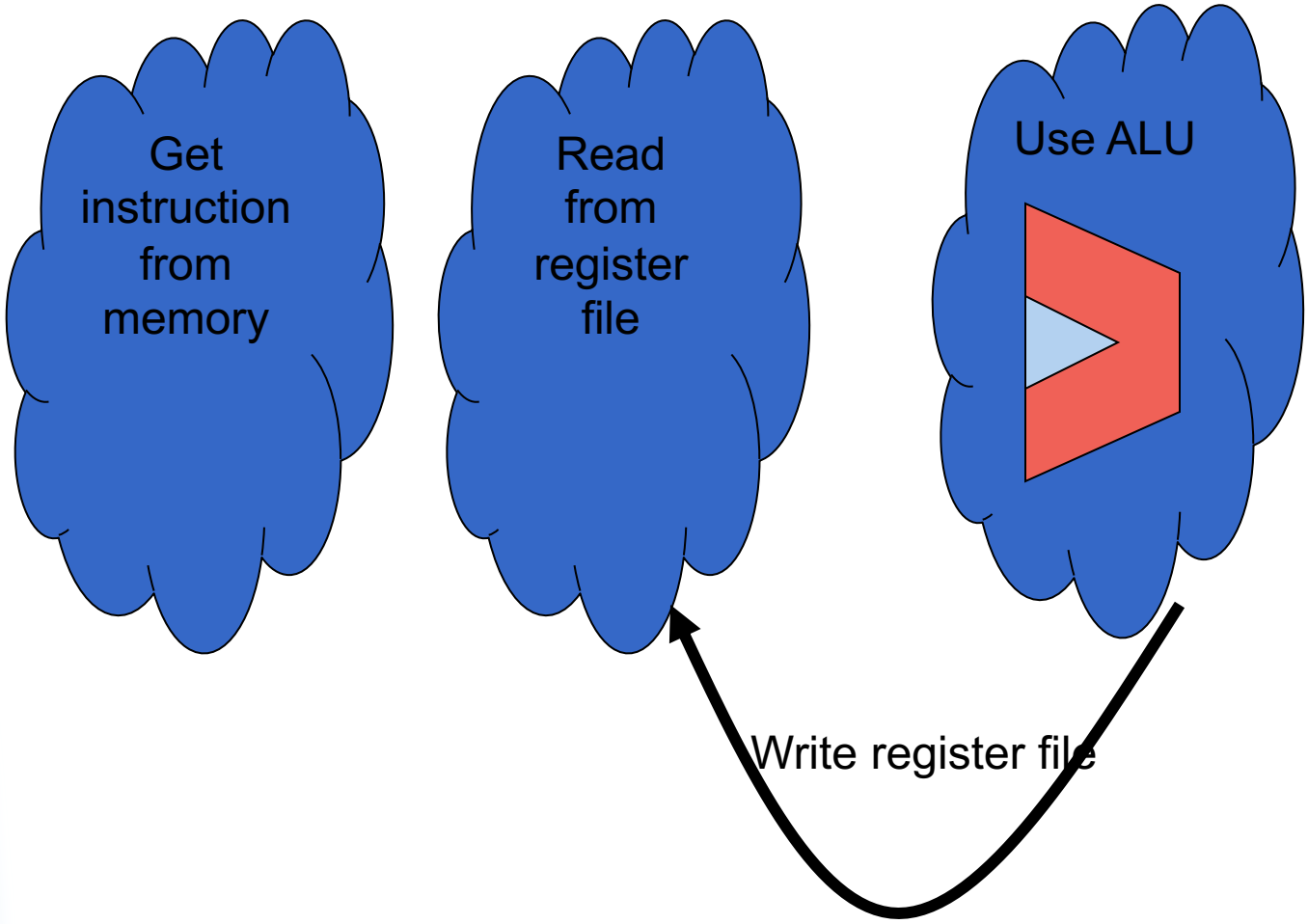


Get Instruction

What happens to the PC each instruction?
Increment by 4B

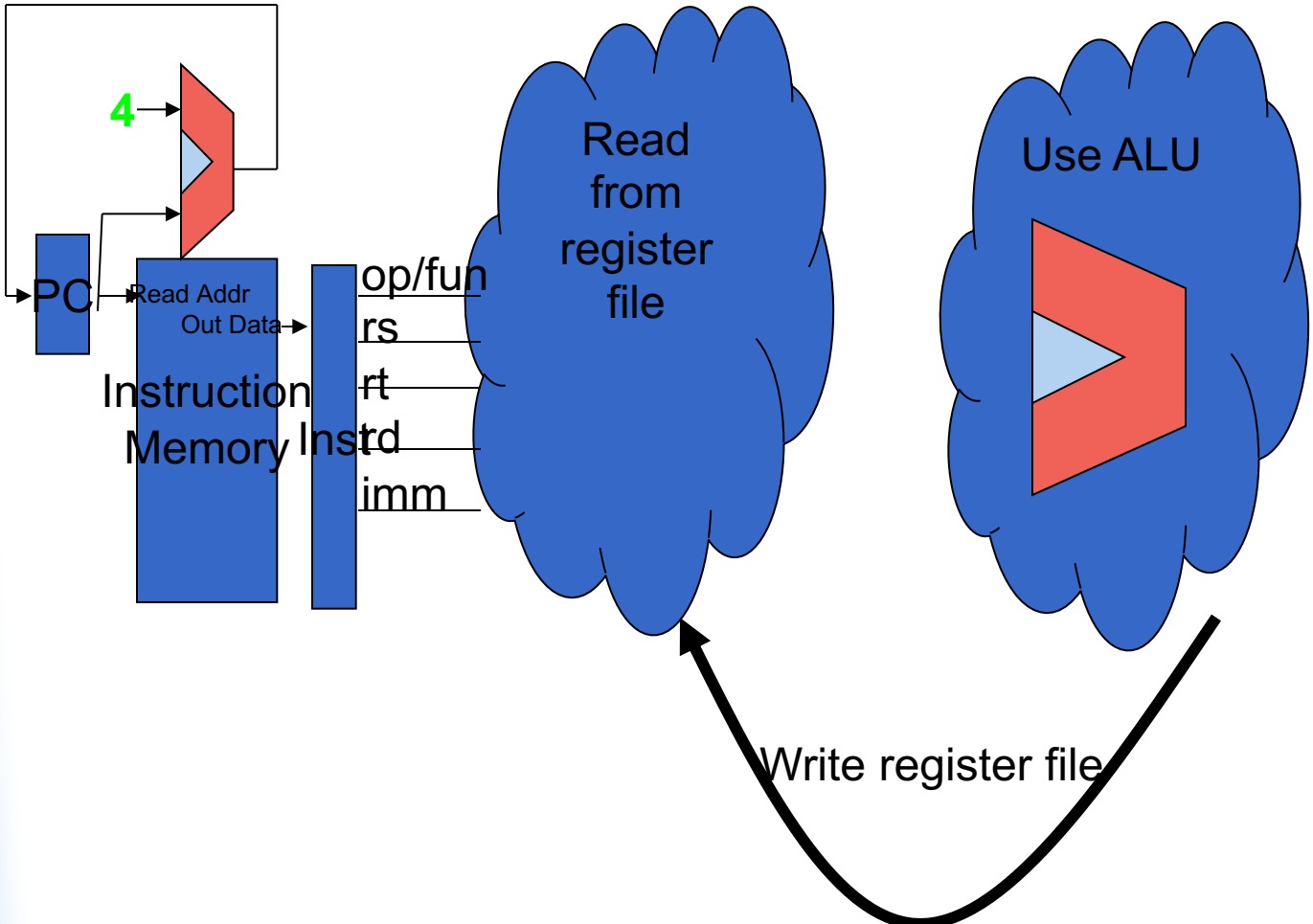


“Add” Instruction



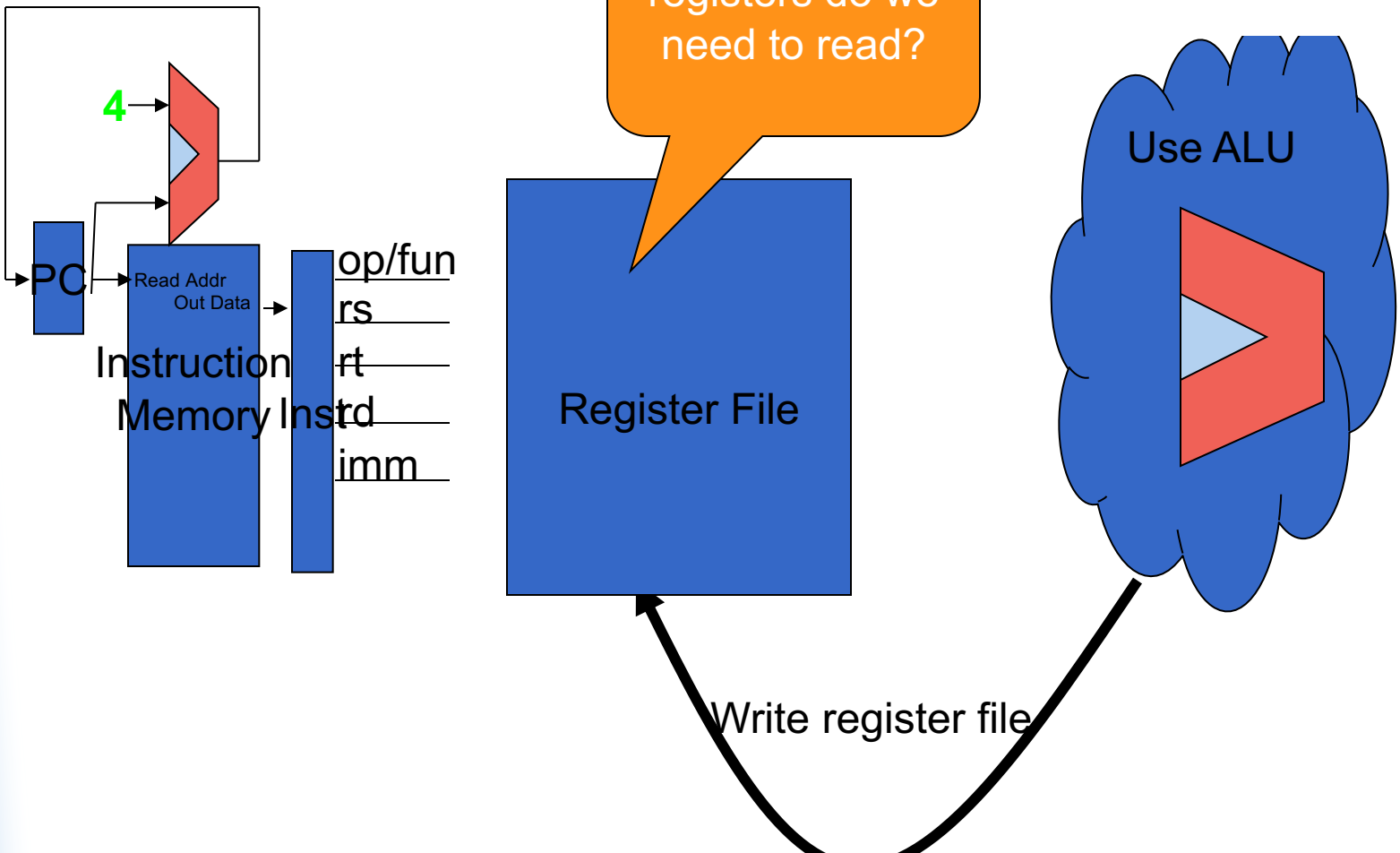
“Add” Instruction

Operation	rs	rt	rd	shamt	funct	# meaning
add	3	5	2	0	32	# $2 <- $3 + $5



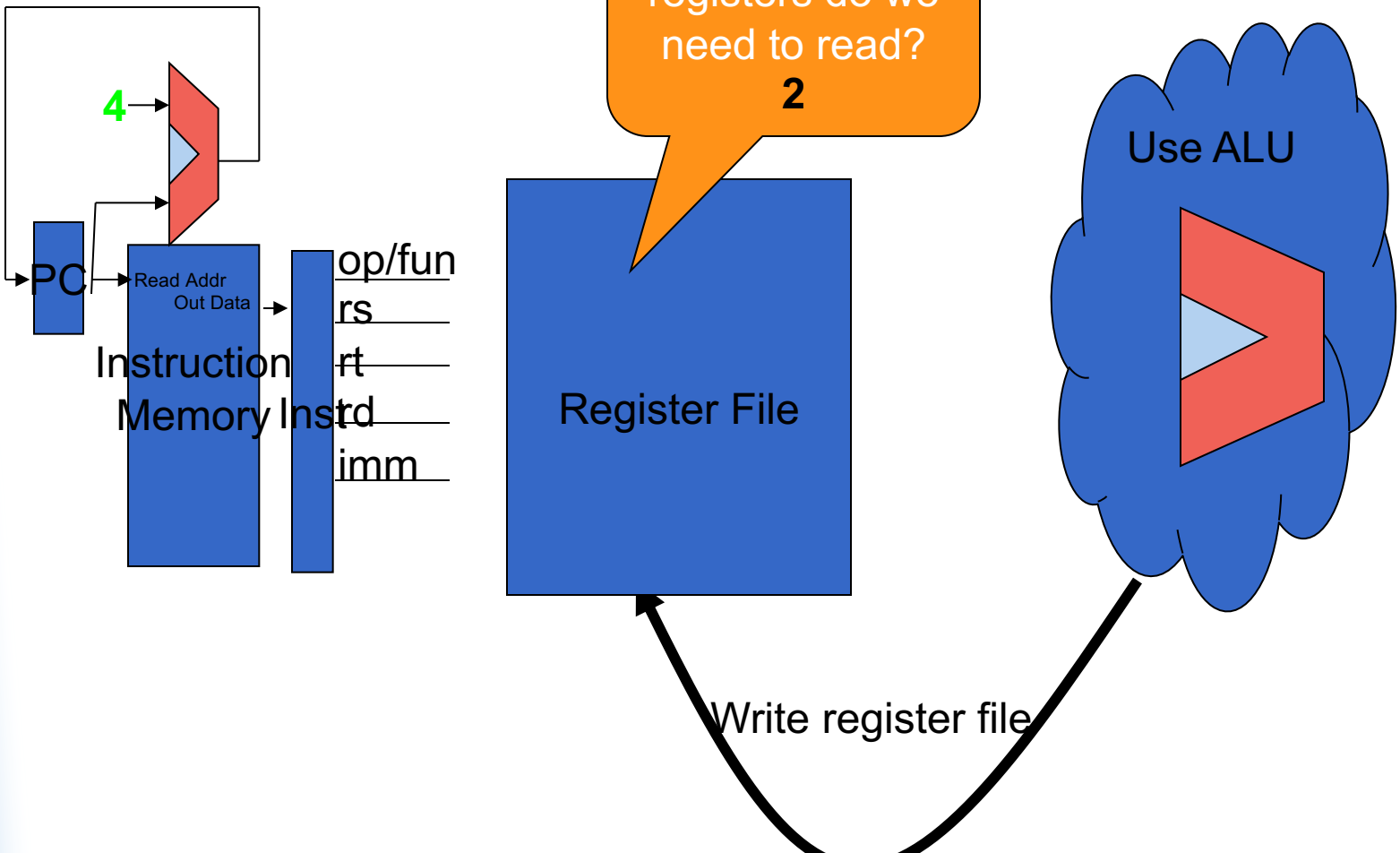
"Add" Instruction

Operation	rs	rt	rd	shamt	funct	# meaning
add	3	5	2			# $2 <- $3 + $5



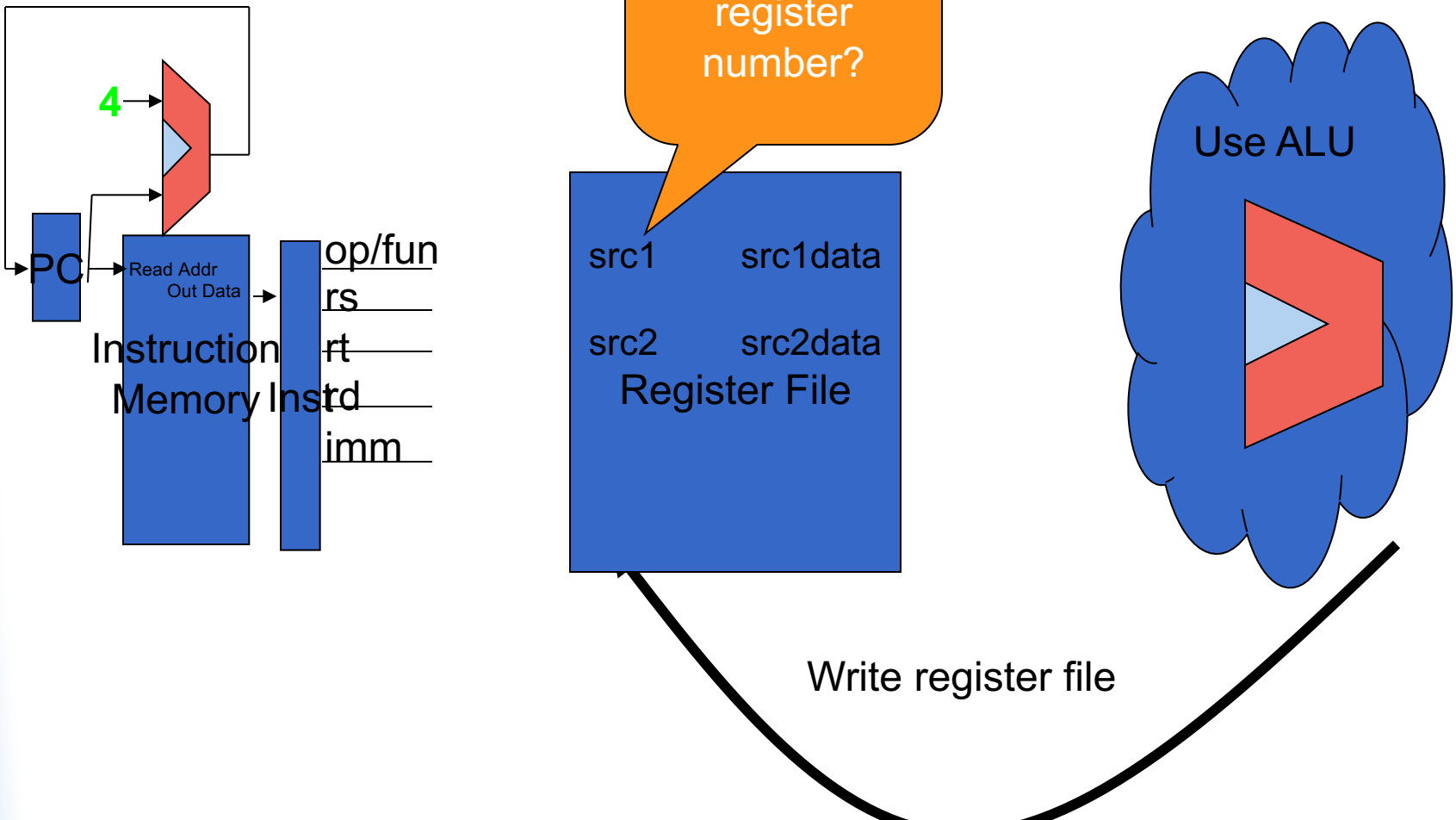
"Add" Instruction

Operation	rs	rt	rd	shamt	funct	# meaning
add	3	5	2			# $2 <- $3 + $5



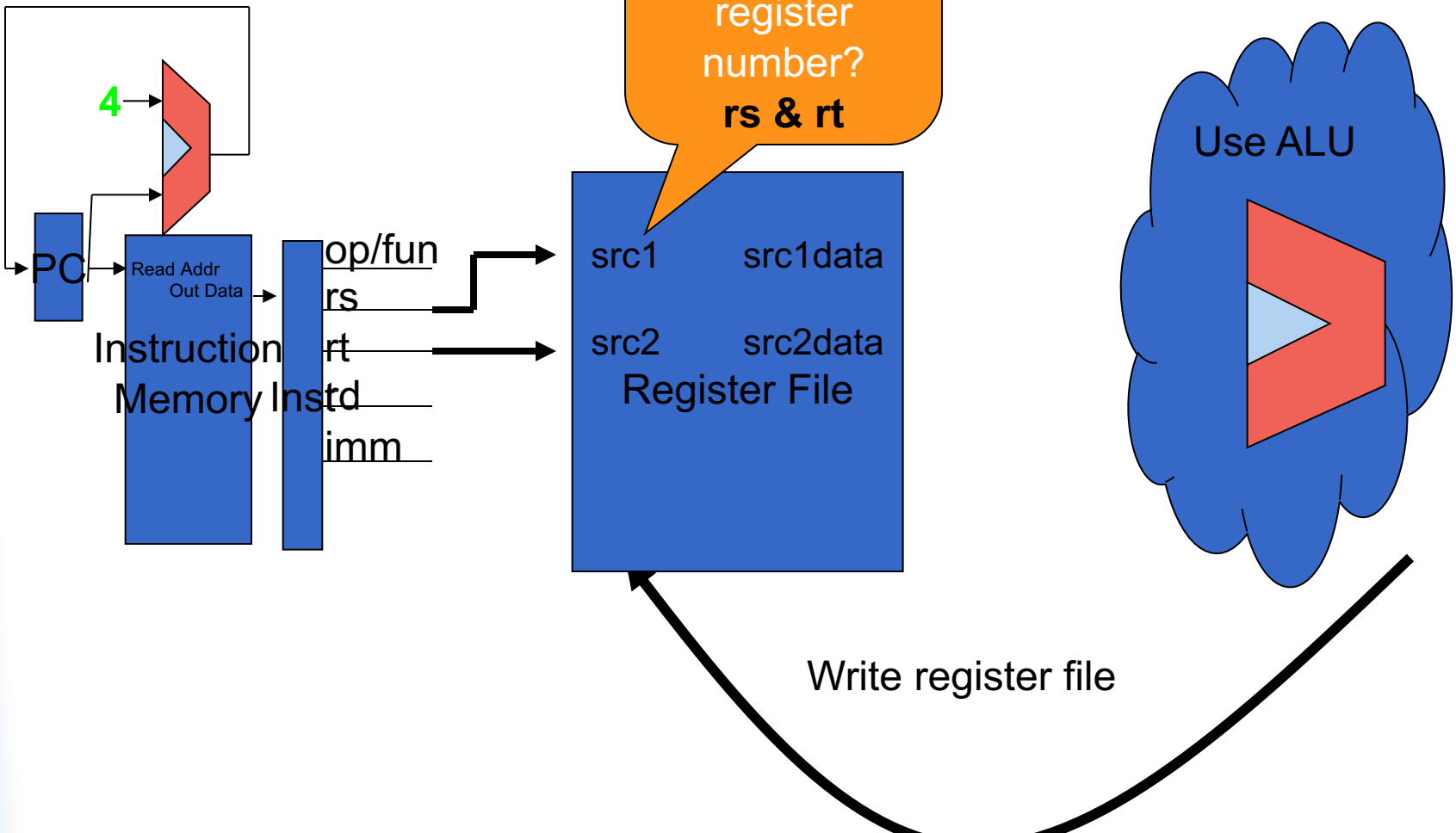
"Add" Instruction

Operation	rs	rt	rd	...	# meaning
add	3	5	2		# $2 <- $3 + $5



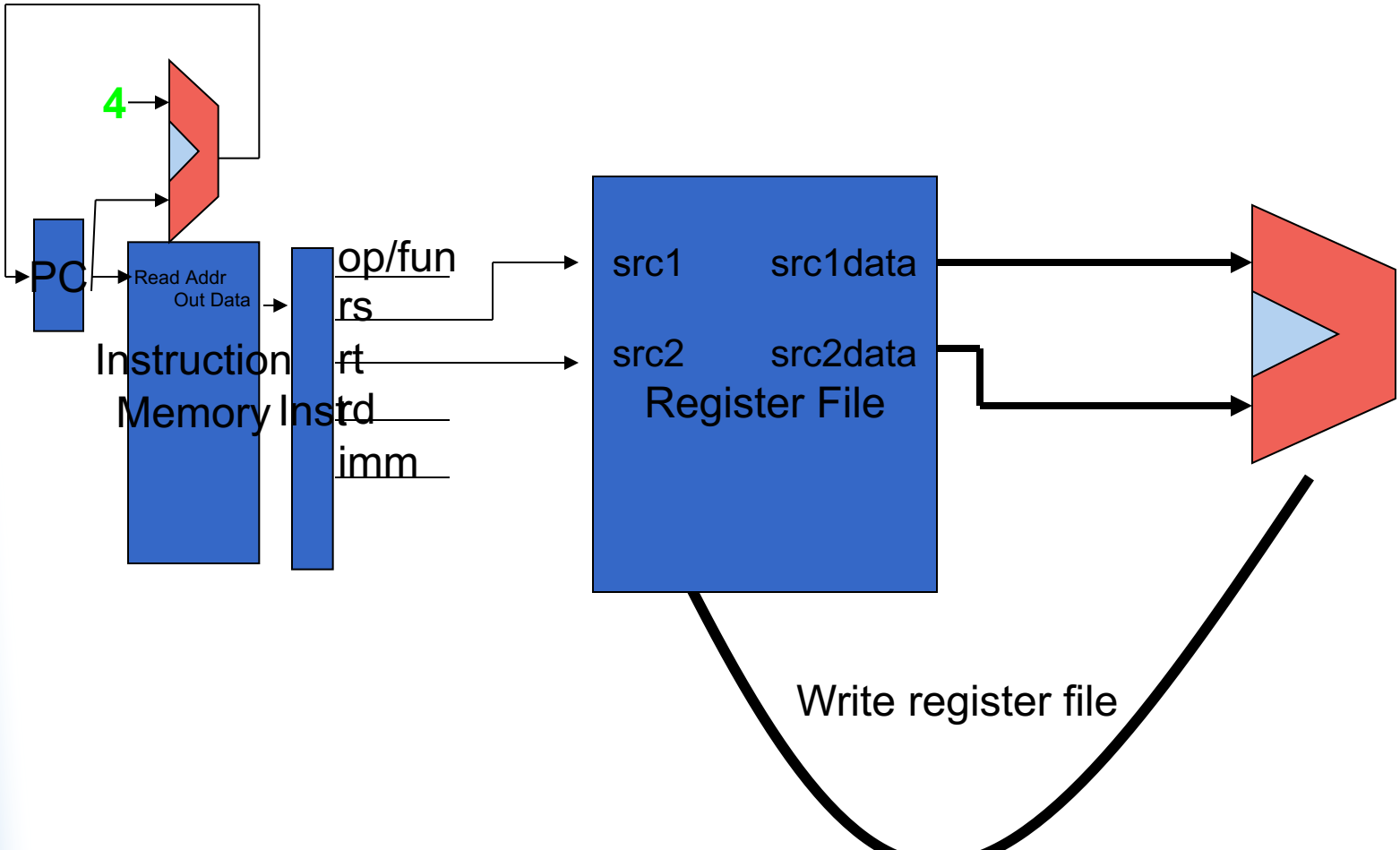
“Add” Instruction

Operation	rs	rt	rd		# meaning
add	3	5	2		# $2 <- $3 + $5



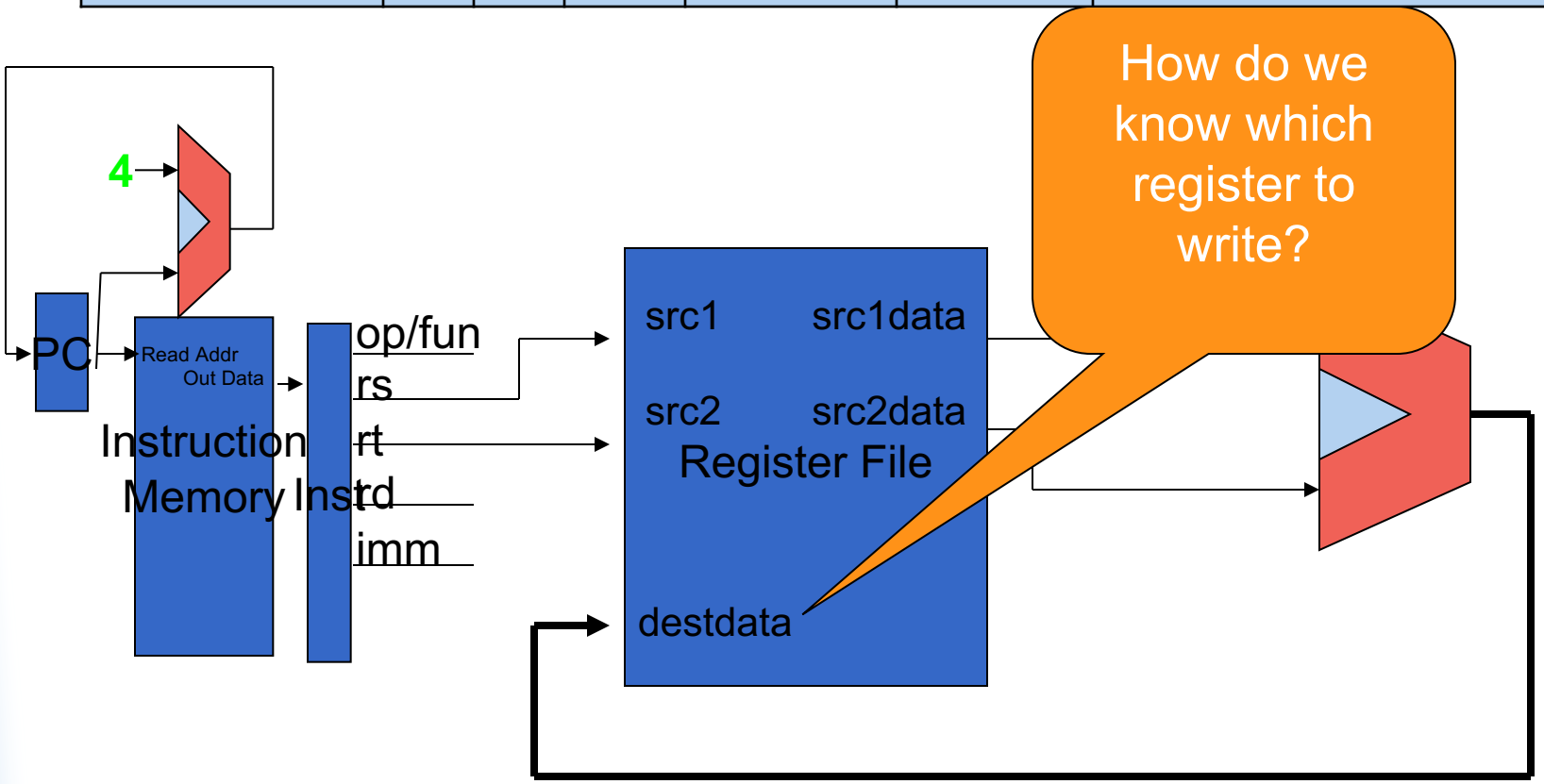
“Add” Instruction

Operation	rs	rt	rd	shamt	funct	# meaning
add	3	5	2	0	32	# $2 <- $3 + $5



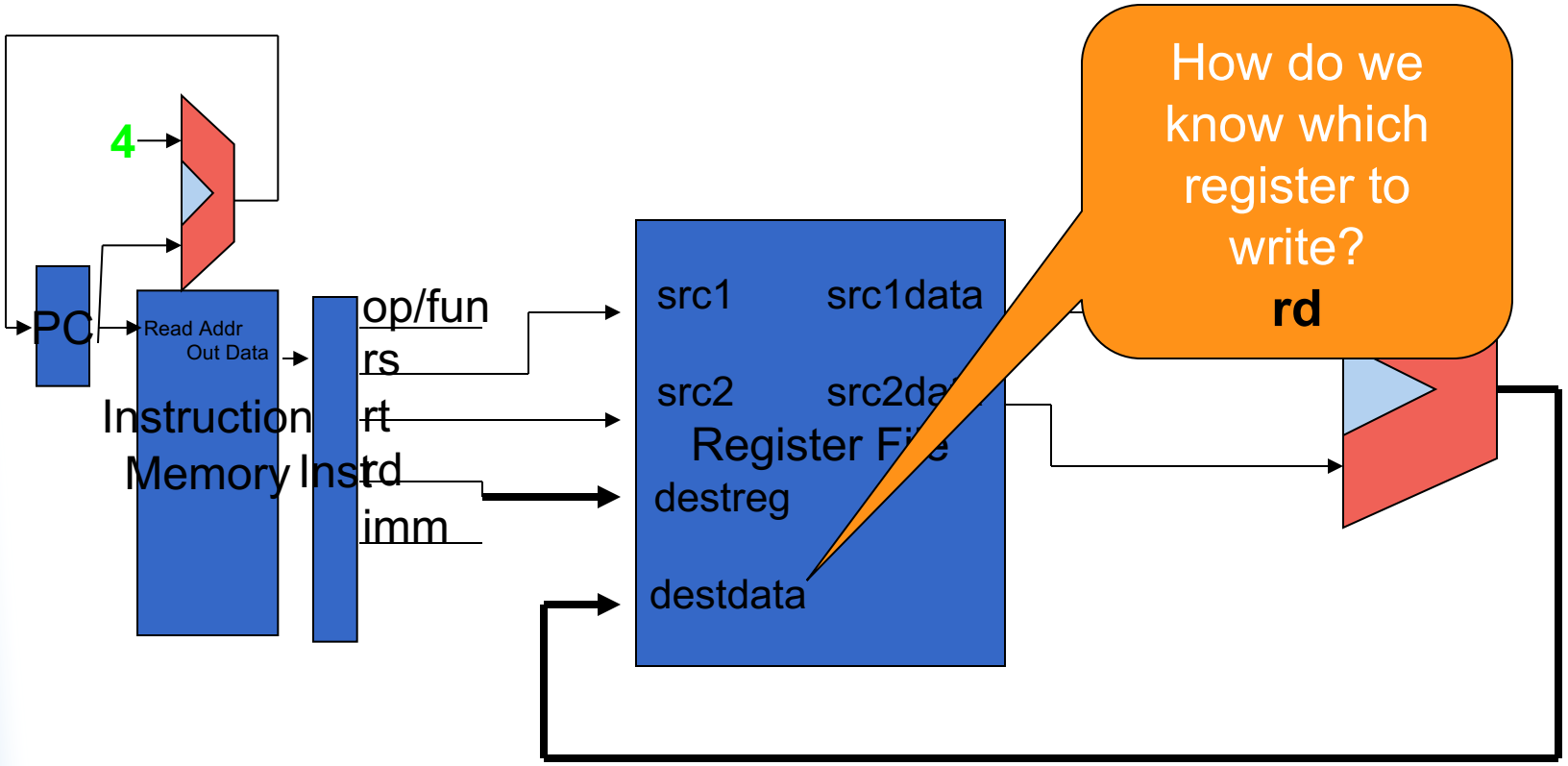
“Add” Instruction

Operation	rs	rt	rd	shamt	funct	# meaning
add	3	5	2	0	32	# $2 <- $3 + $5



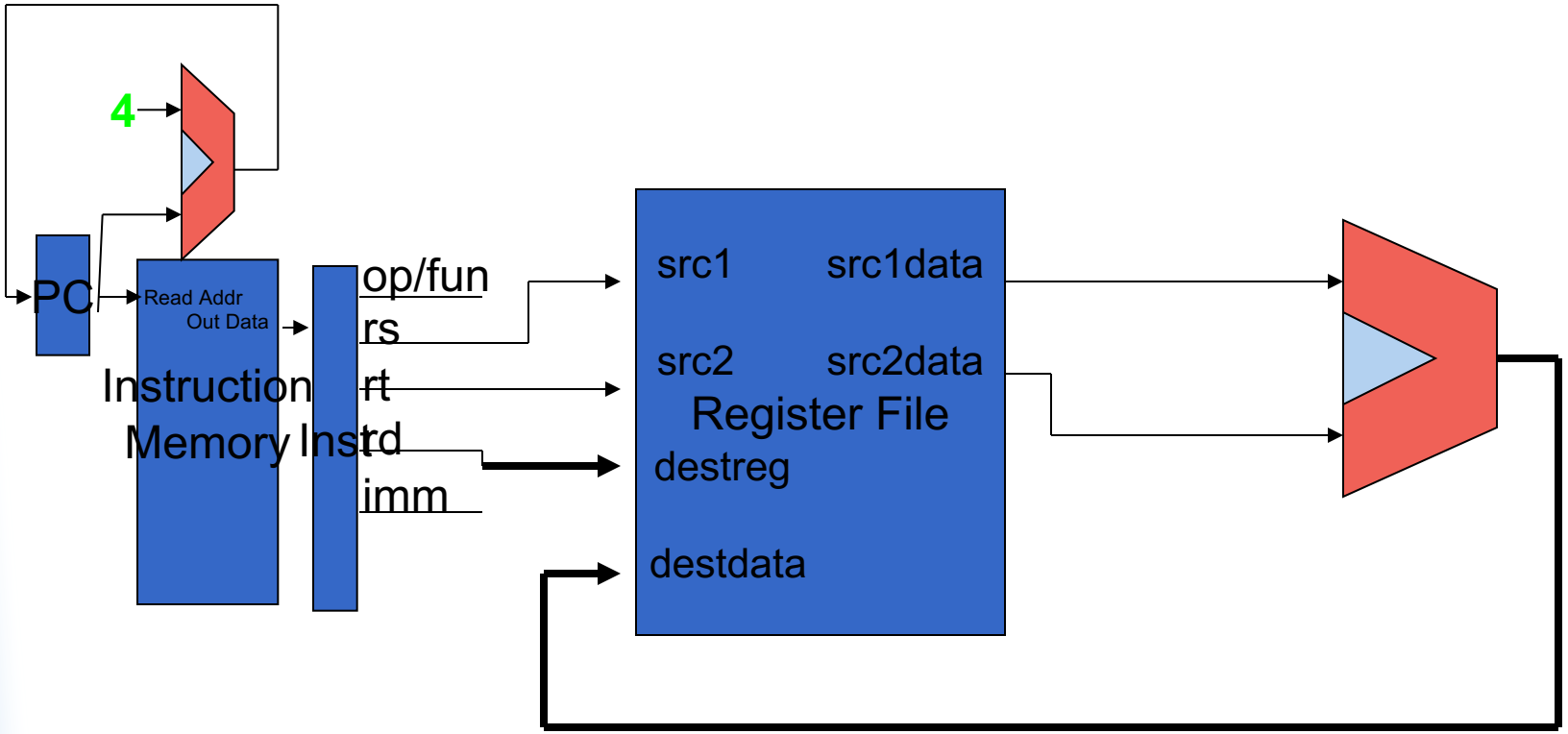
“Add” Instruction

Operation	rs	rt	rd	shamt	funct	# meaning
add	3	5	2	0	32	# $2 <- $3 + $5



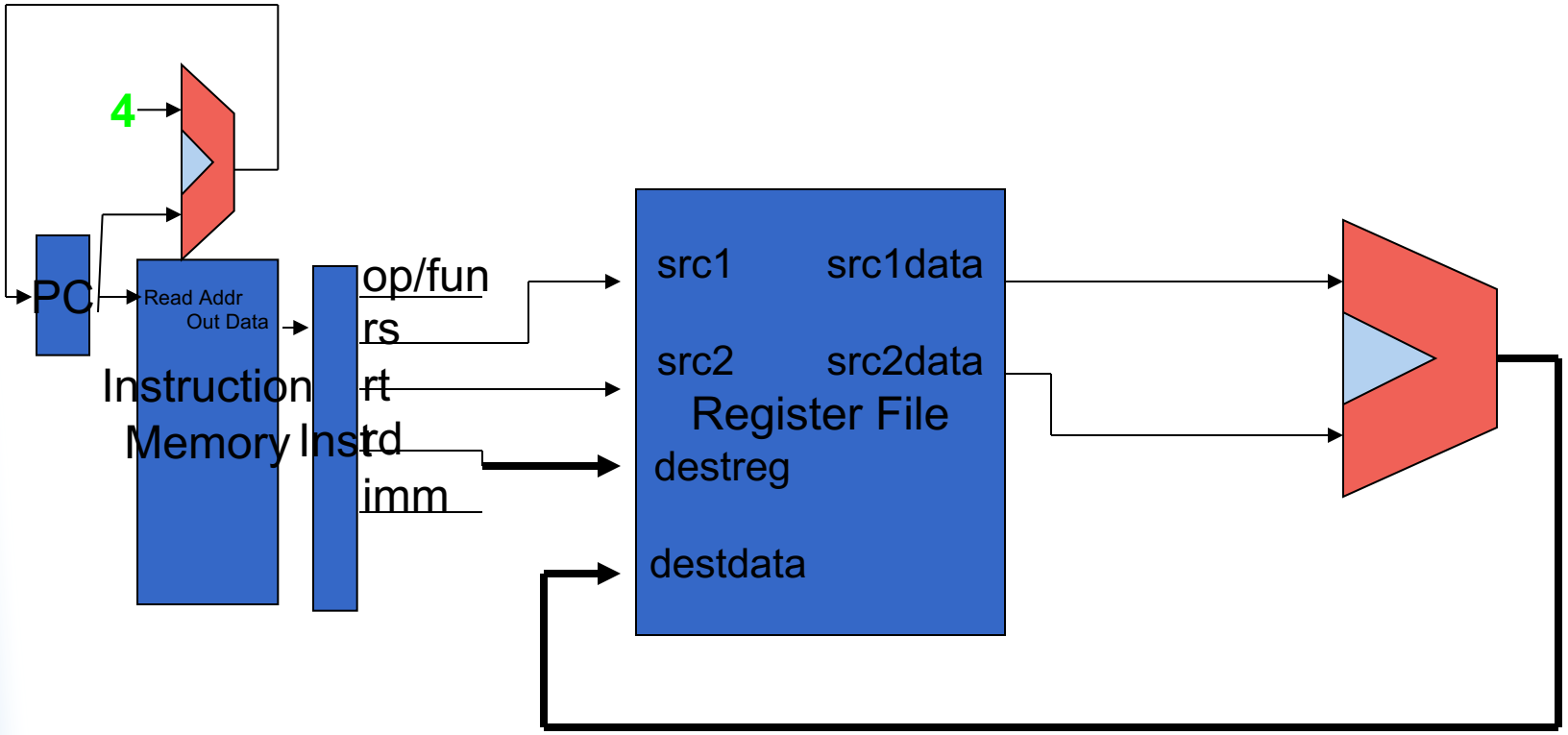
“Add” Instruction

Operation	rs	rt	rd	shamt	funct	# meaning
add	3	5	2	0	32	# $2 <- $3 + $5



What happens if instruction reads and writes same register?

Operation	rs	rt	rd	shamt	funct	# meaning
add	3	5	3	0	32	# $3 <- $3 + $5



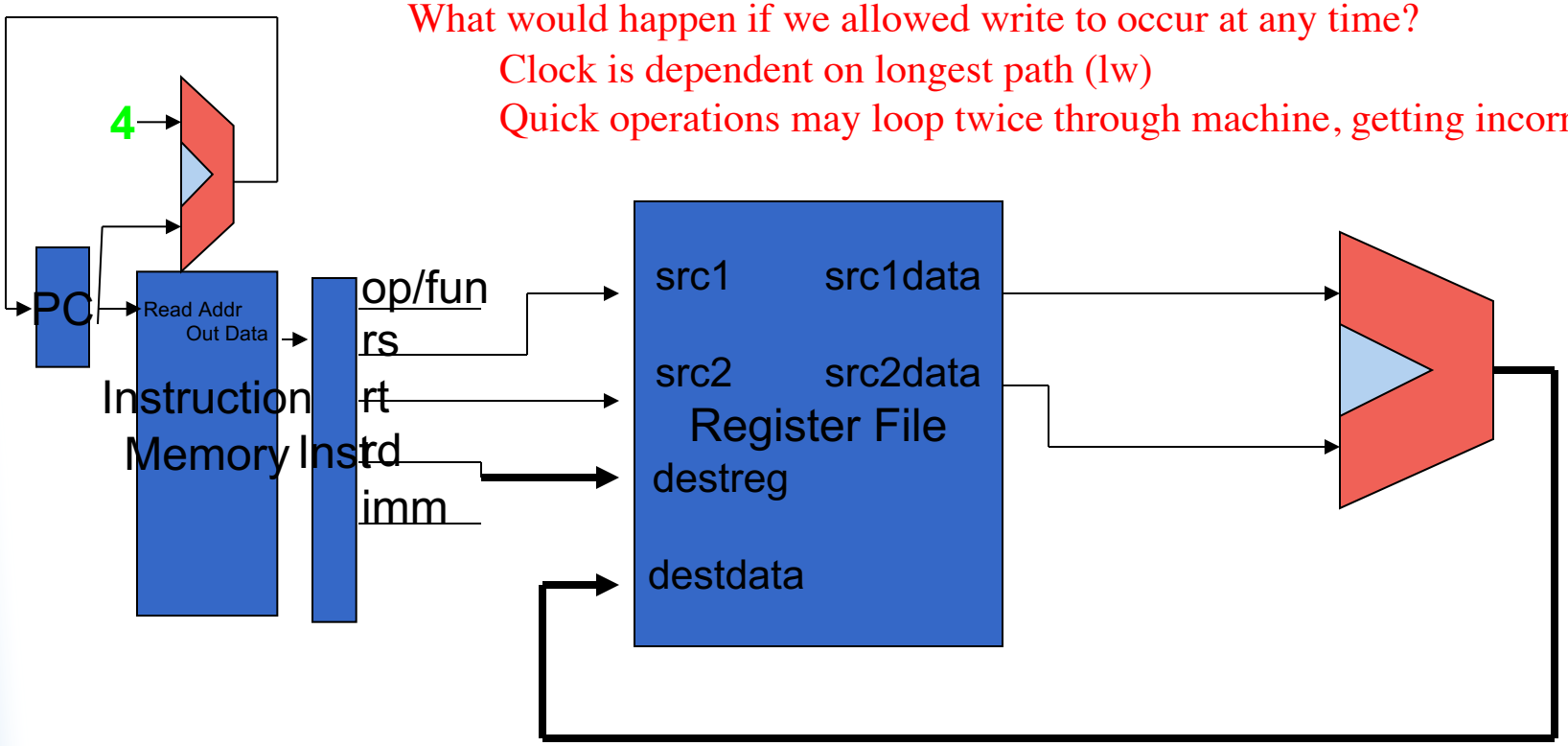
What happens if instruction reads and writes same register?

Operation	rs	rt	rd	shamt	funct	# meaning
add	3	5	3	0	32	# $3 <- $3 + $5

What would happen if we allowed write to occur at any time?

Clock is dependent on longest path (lw)

Quick operations may loop twice through machine, getting incorrect result.

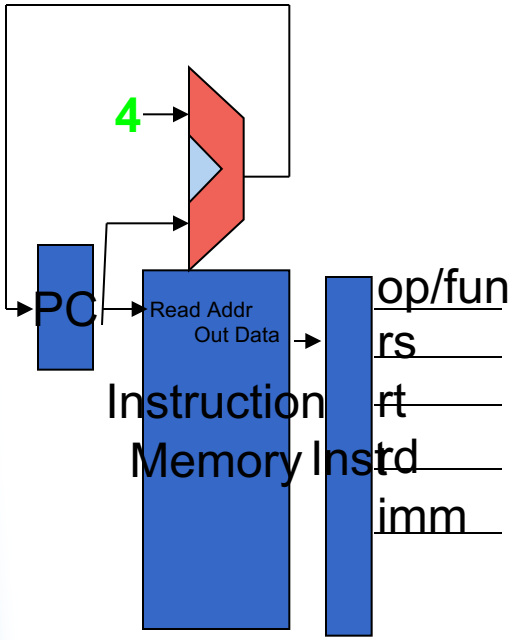


Reading/Write Registers

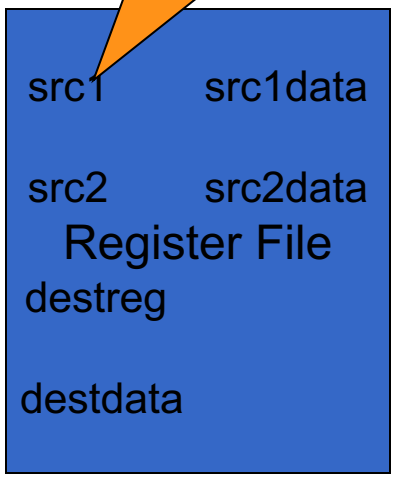
- When does register get written?
  - At the end of the clock cycle
  - Edge-triggered circuits

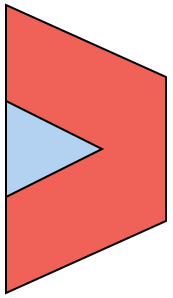
"Addi" Instruction

Operation	rs	rt	imm	# meaning
addi $5,$3,6	3	5	6	# $5 <- $3 + 6



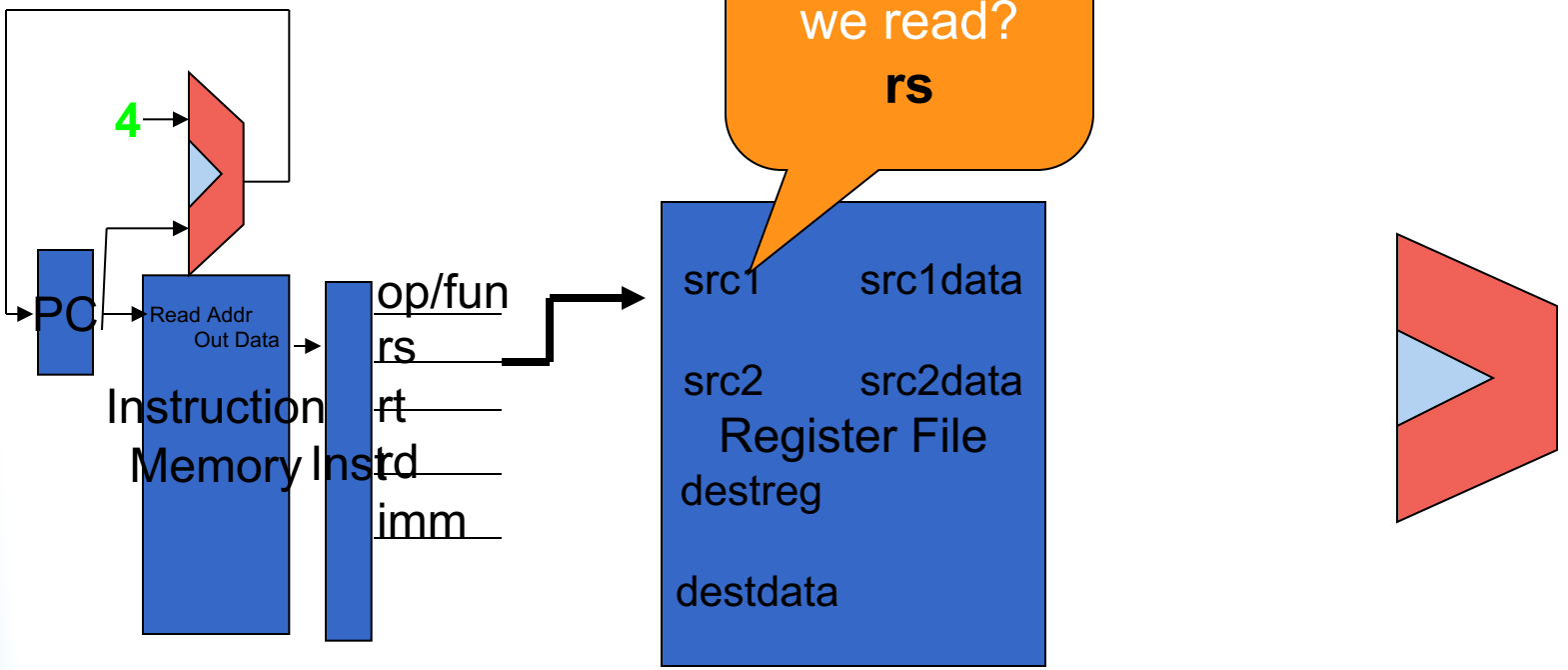
What registers do we read?





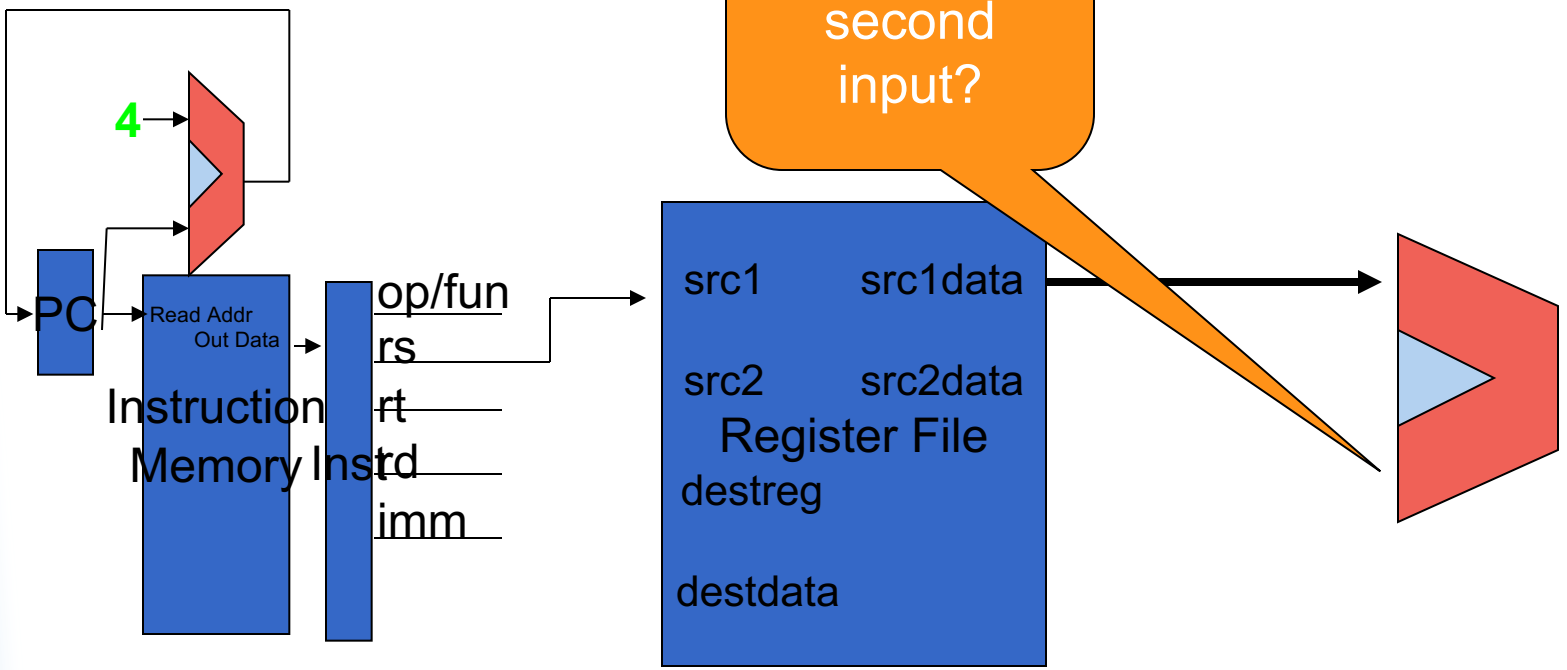
“Addi” Instruction

Operation	rs	rt	imm	# meaning
addi $5,$3,6	3	5	6	# $5 <- $3 + 6



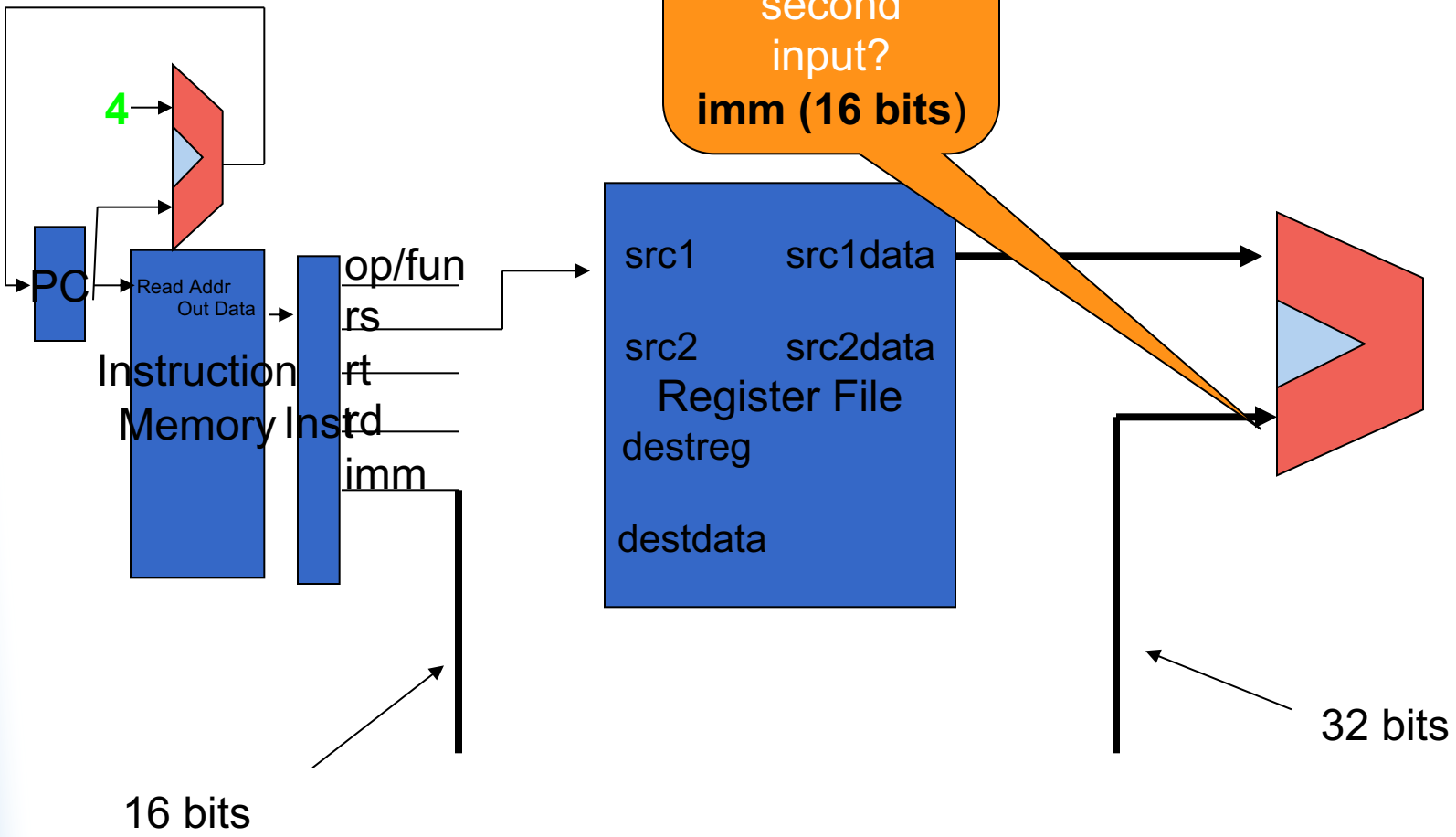
"Addi" Instruction

Operation	rs	rt	imm	# meaning
addi $5,$3,6	3	5	6	# $5 <- $3 + 6



"Addi" Instruction

Operation	rs	rt	imm	# meaning
addi $5,$3,6	3	5	6	# $5 <- $3 + 6

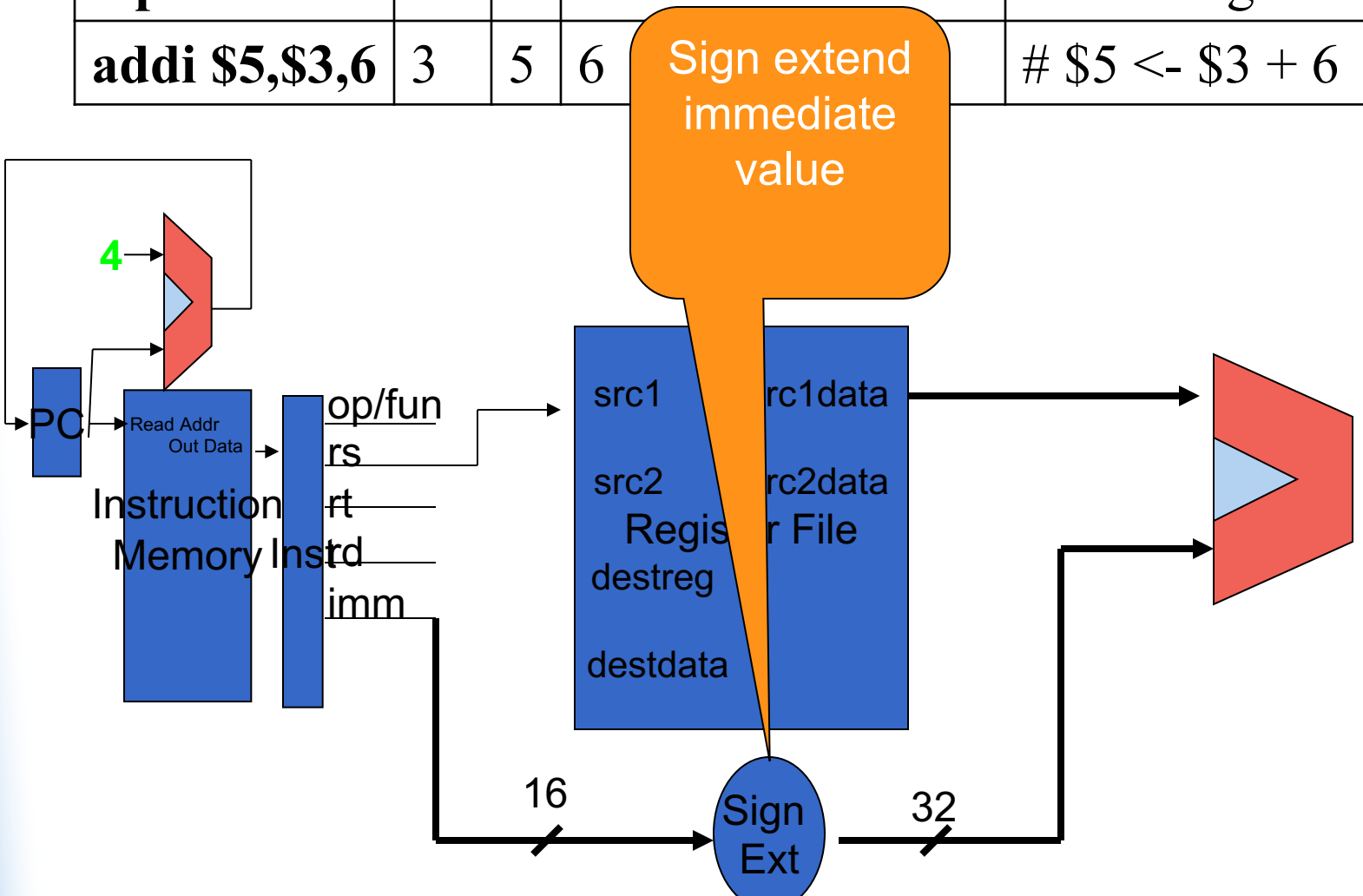


Sign Extension

- How do we go from 16-bit number to 32-bit number?
- How about 4-bit to 8-bit.
  - 0111 = 7 = 00000111
  - 1110 = -2 = 11111110
- Take the top bit and copy it to all the other bits

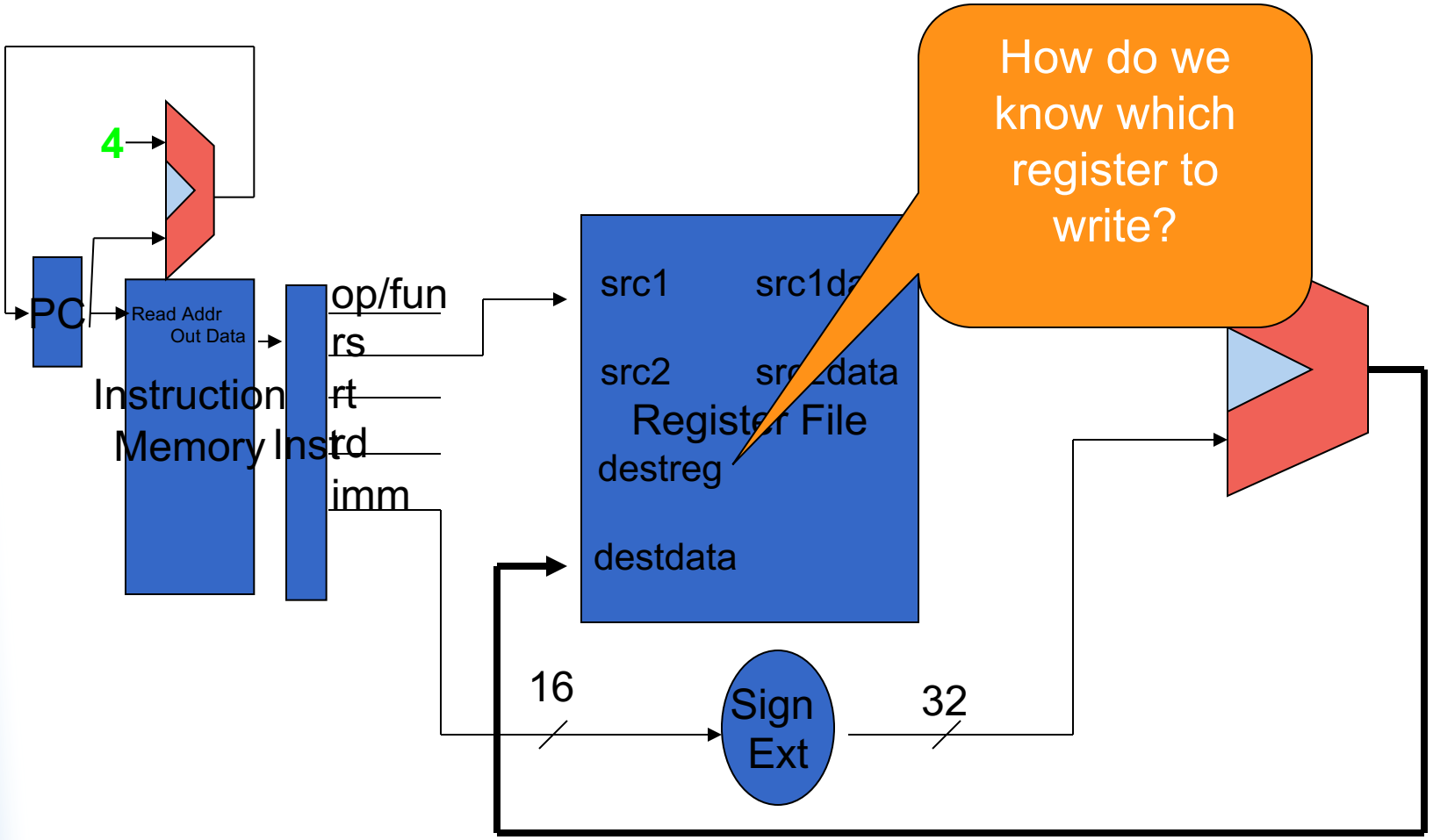
“Addi” Instruction

Operation	rs	rt	imm	# meaning
addi $5,$3,6	3	5	6	# $5 <- $3 + 6



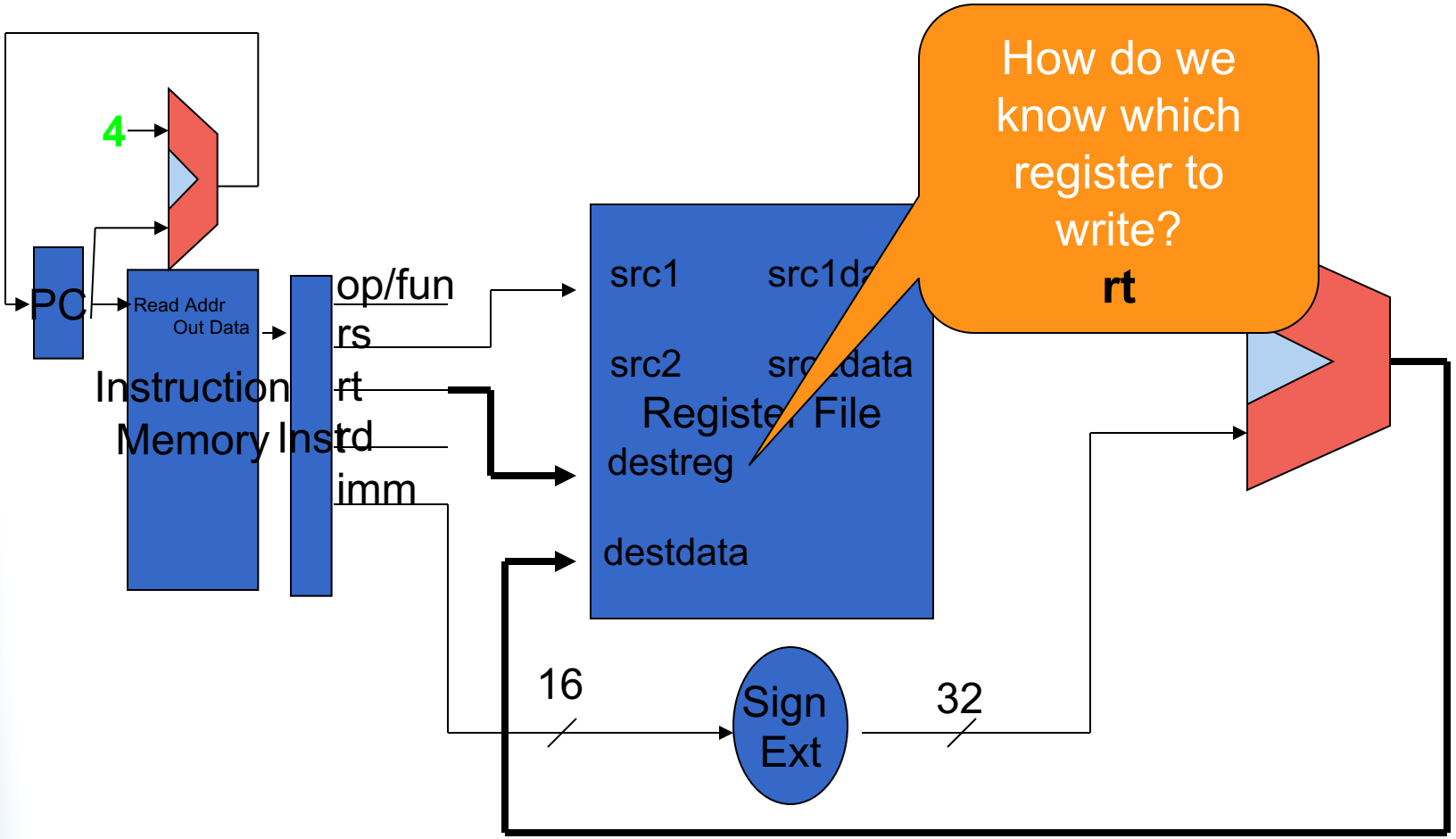
"Addi" Instruction

Operation	rs	rt	imm	# meaning
addi $5,$3,6	3	5	6	# $5 <- $3 + 6

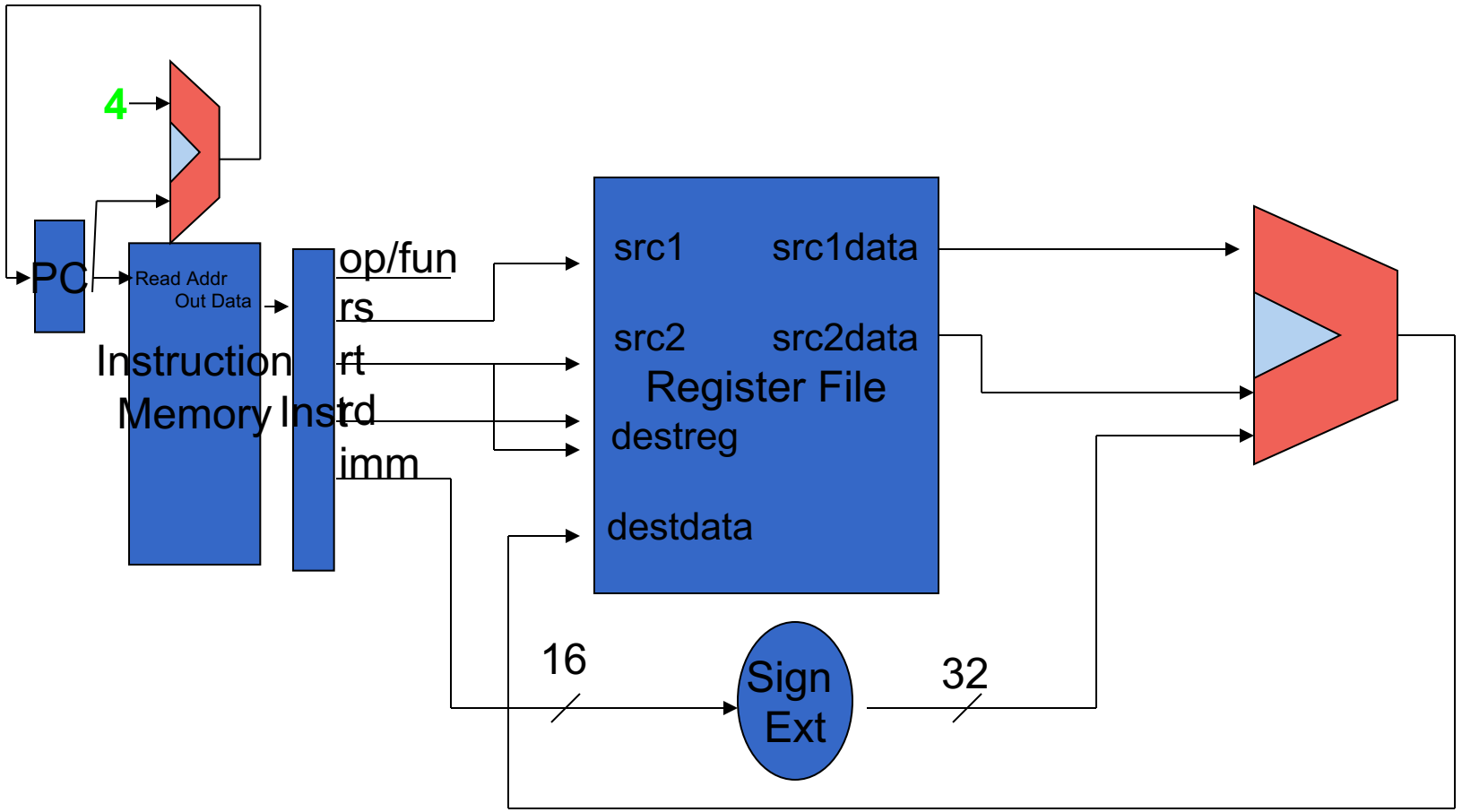


“Addi” Instruction

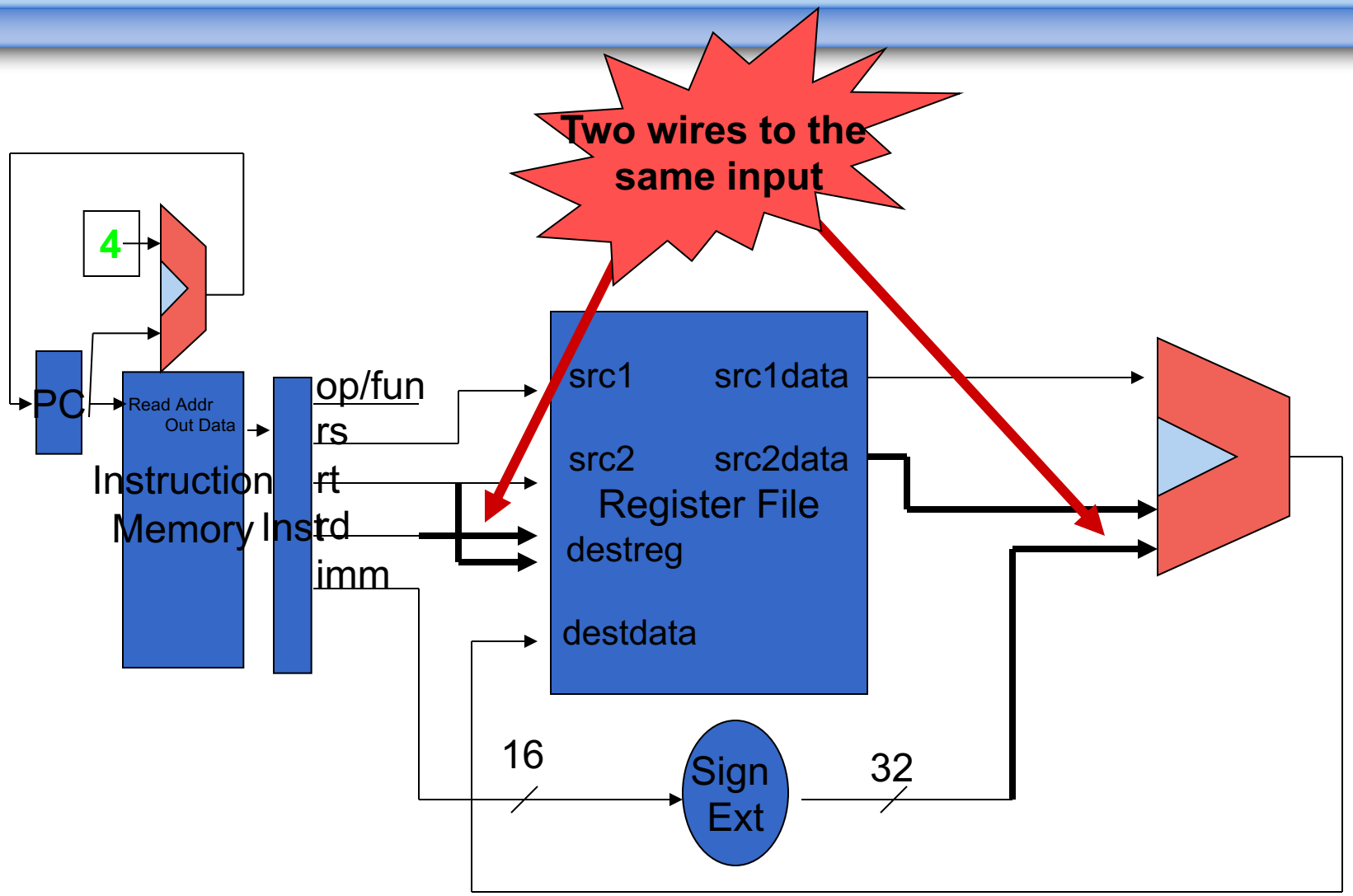
Operation	rs	rt	imm	# meaning
addi $5,$3,6	3	5	6	# $5 <- $3 + 6



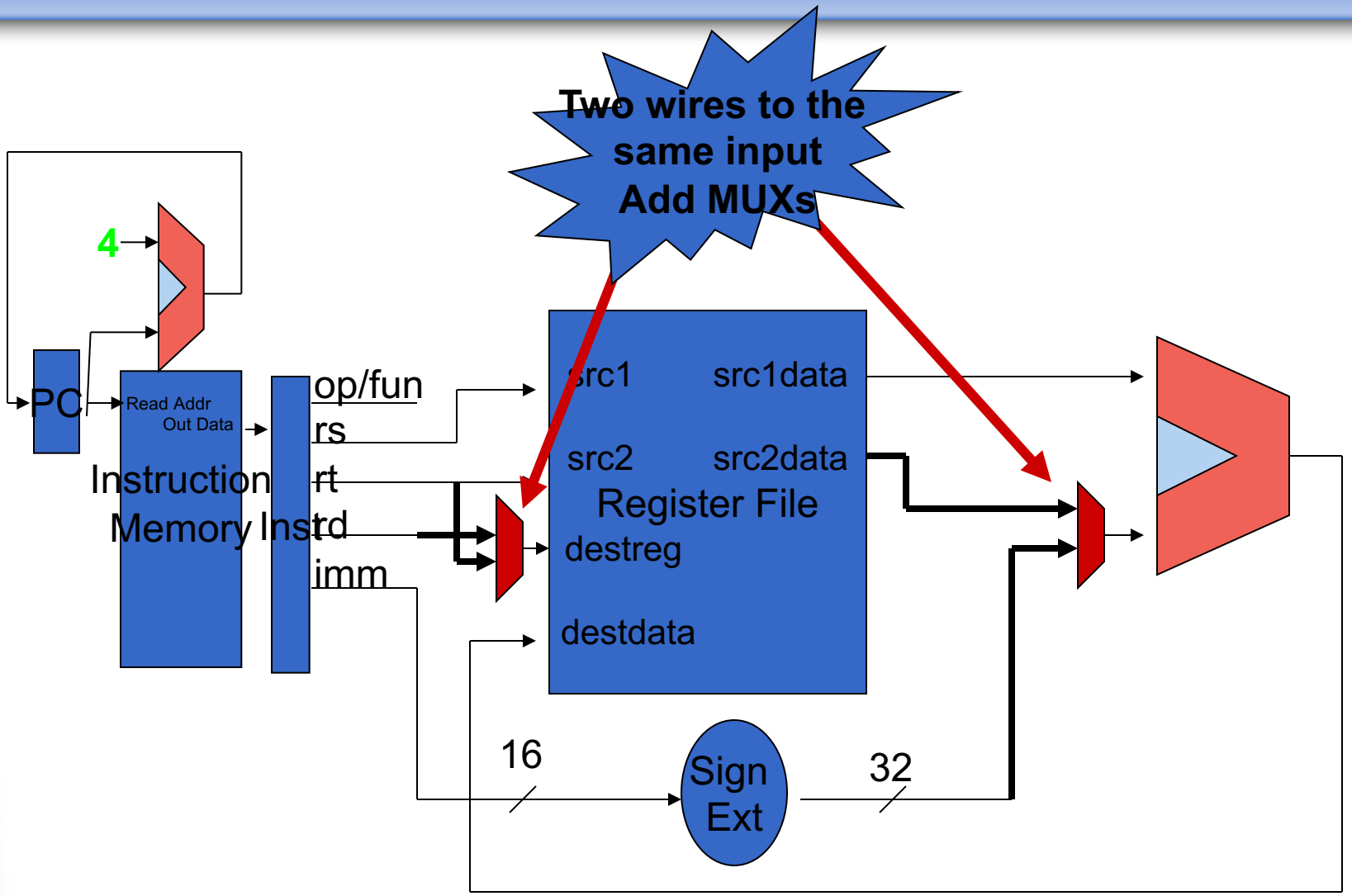
Putting them Together



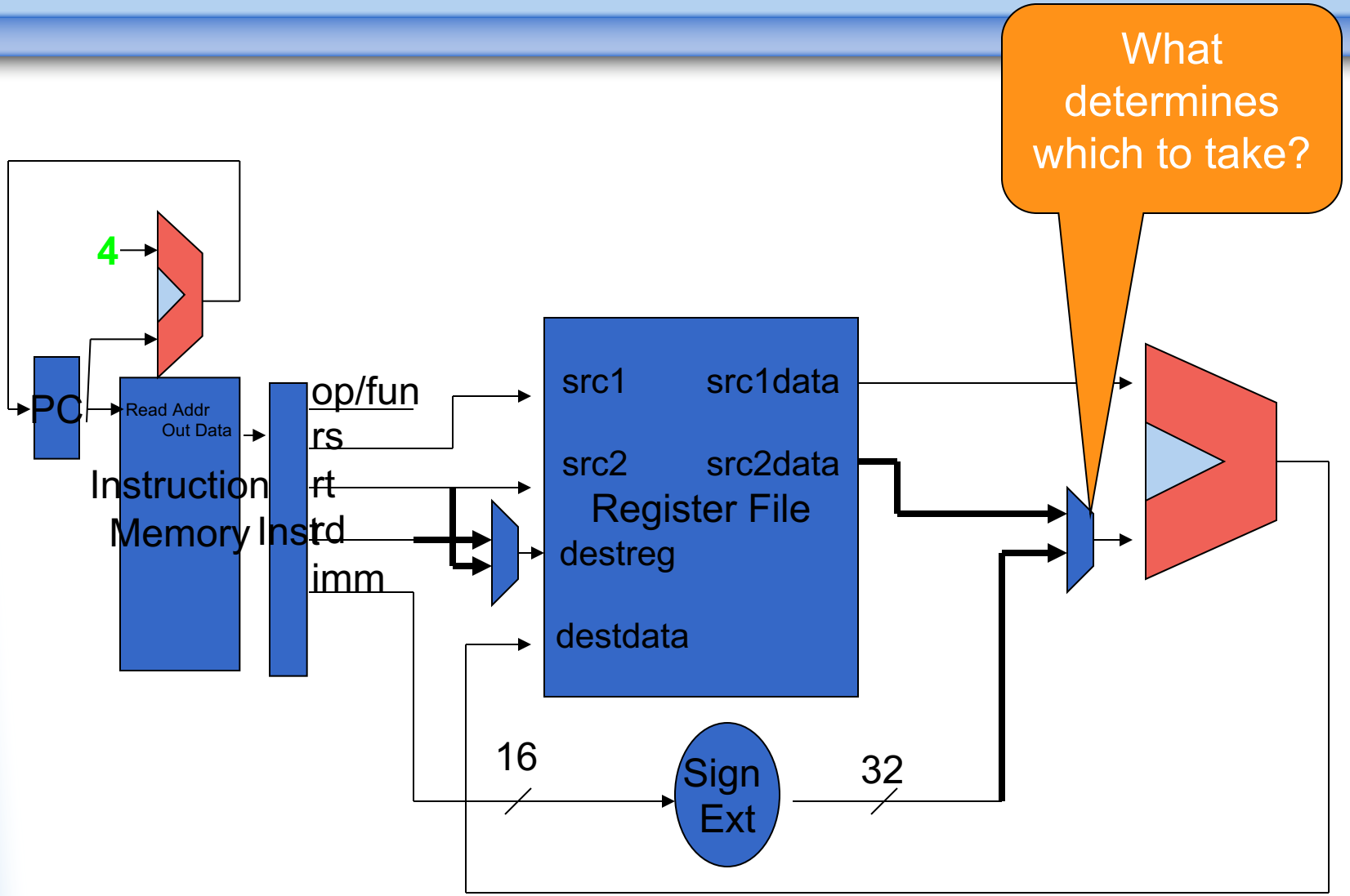
Putting them Together



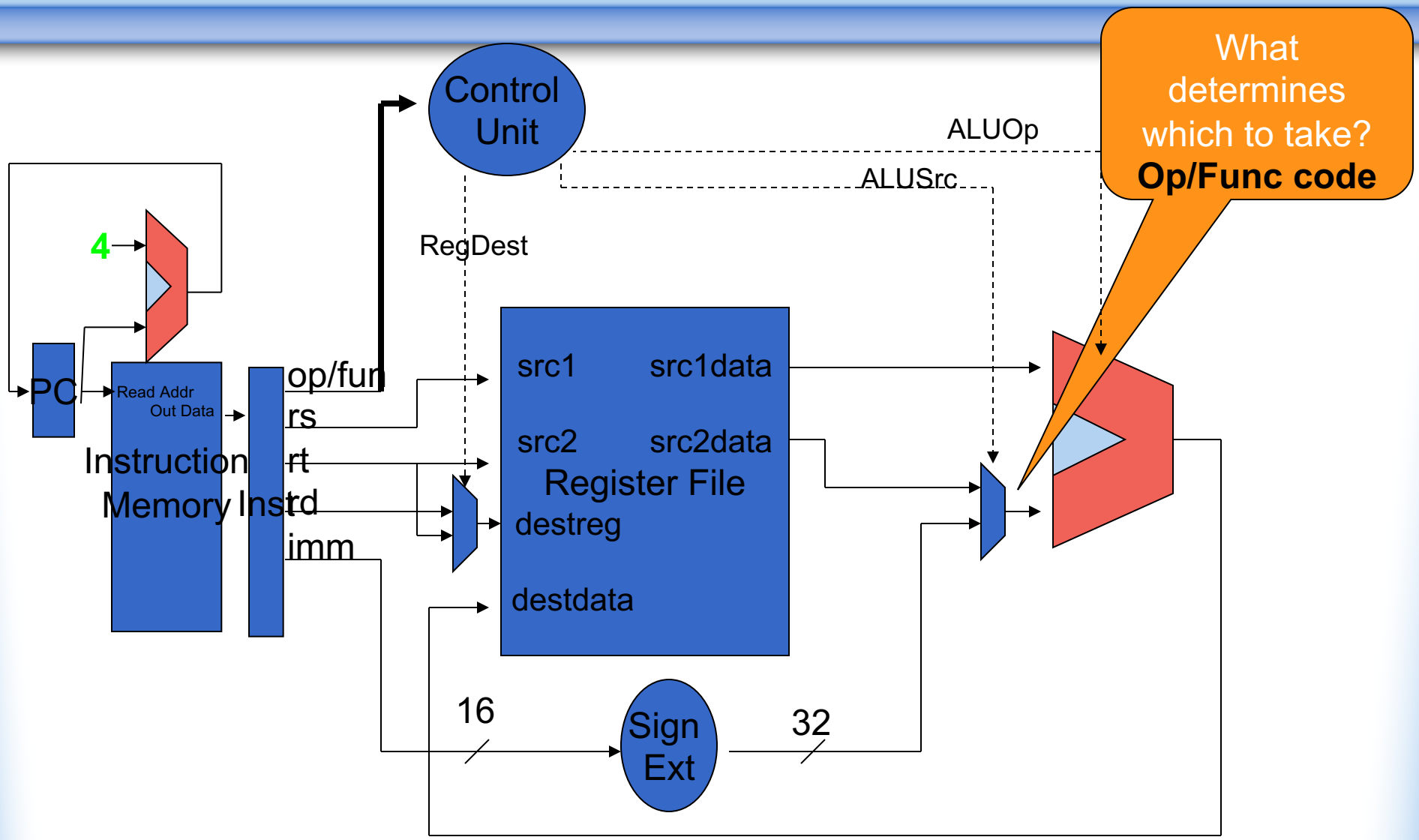
Putting them Together



Putting them Together

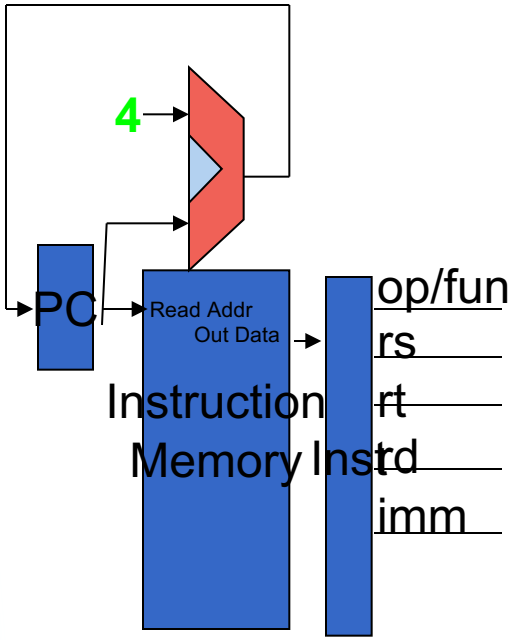


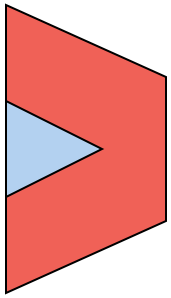
Putting them Together

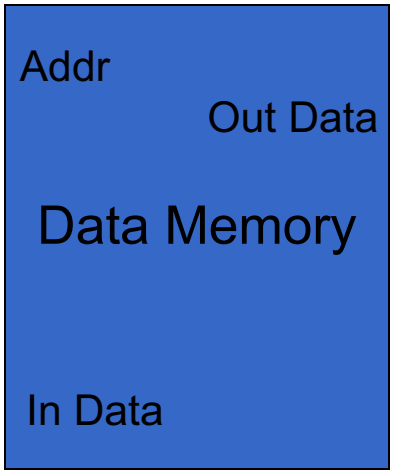


Load Operation

Operation	rs	rt	imm	# meaning
lw $5,8($3)	3	5	8	# $5 <- M[$3 + 8]

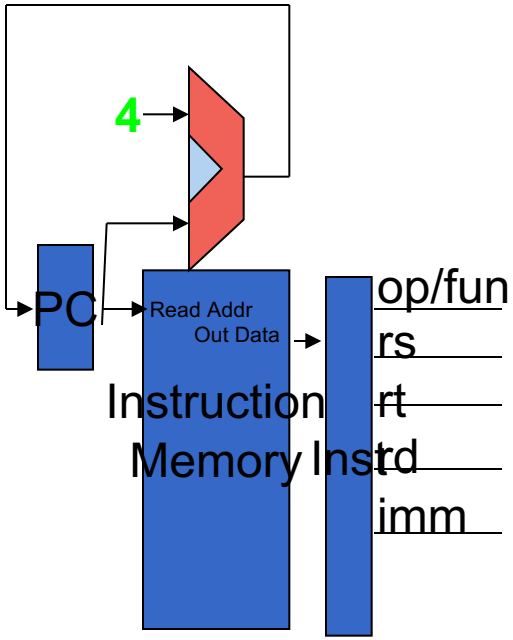




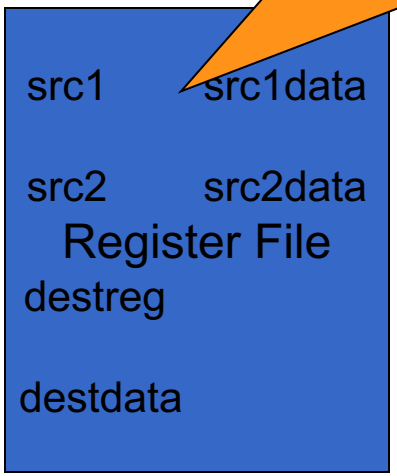


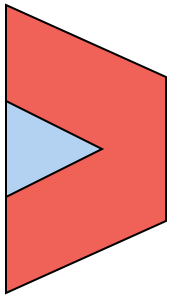
Load Operation

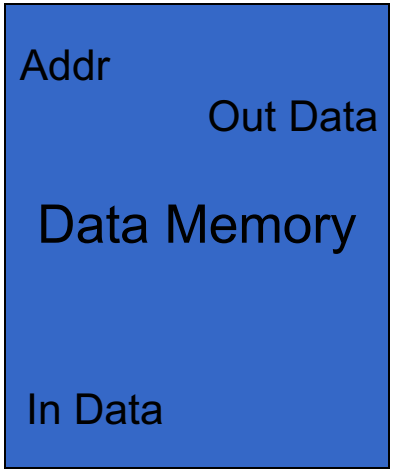
Operation	rs	rt	imm	# meaning
lw $5,8($3)	3	5	8	# $5 <- M[$3 + 8]



How many source regs?
What part of instruction?

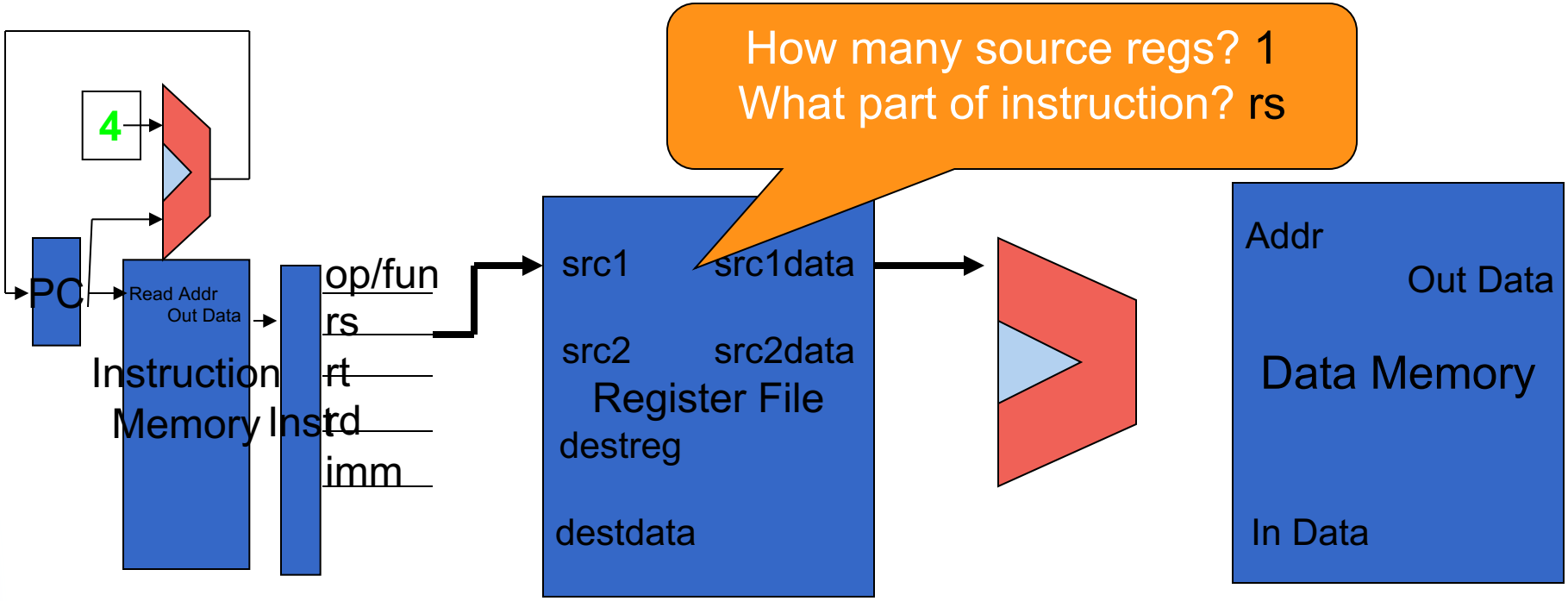






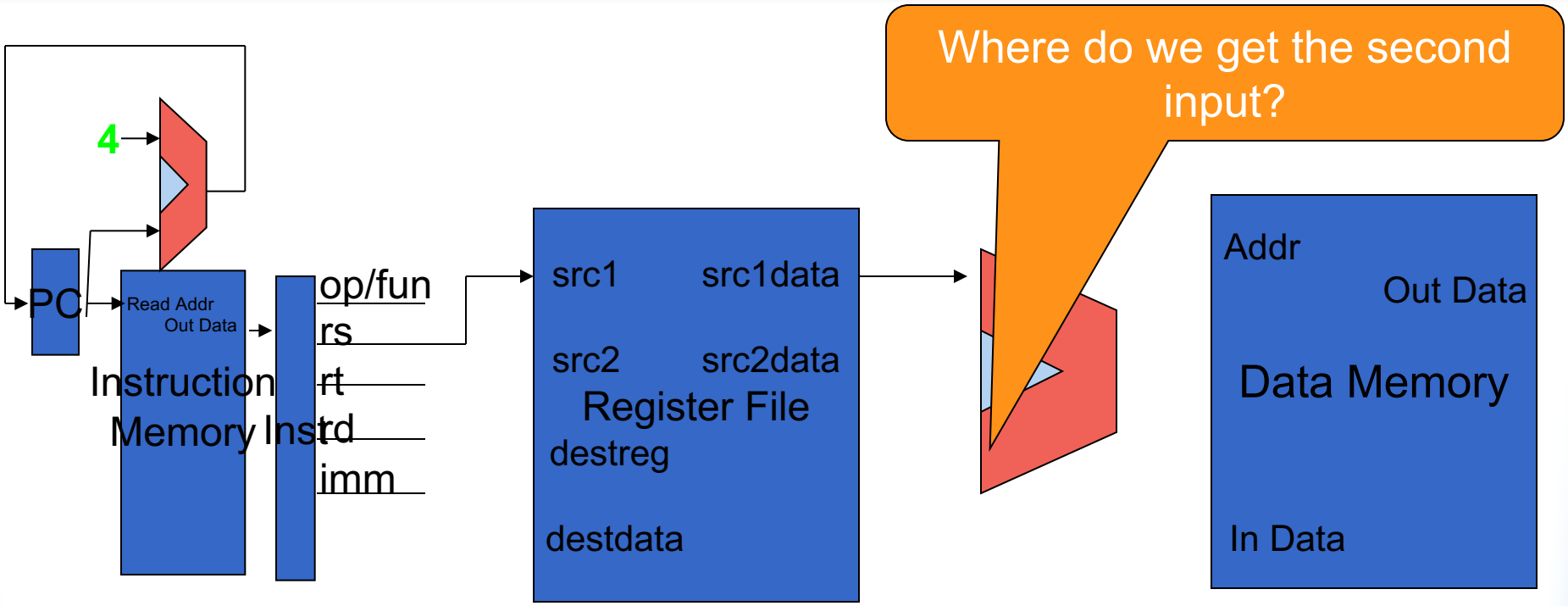
Load Operation

Operation	rs	rt	imm	# meaning
lw $5,8($3)	3	5	8	# $5 <- M[$3 + 8]



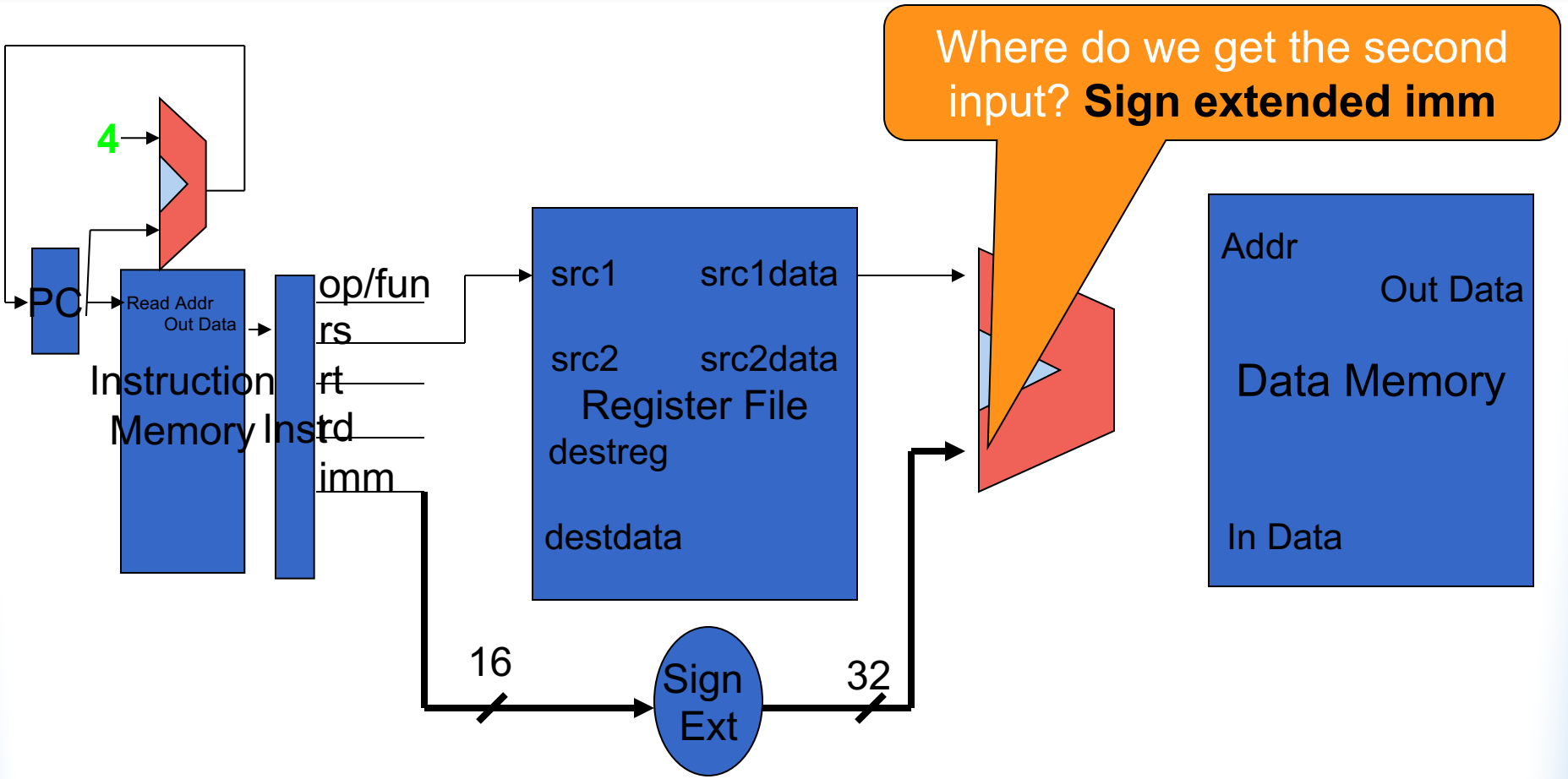
Load Operation

Operation	rs	rt	imm	# meaning
lw $5,8($3)	3	5	8	# $5 <- M[$3 + 8]



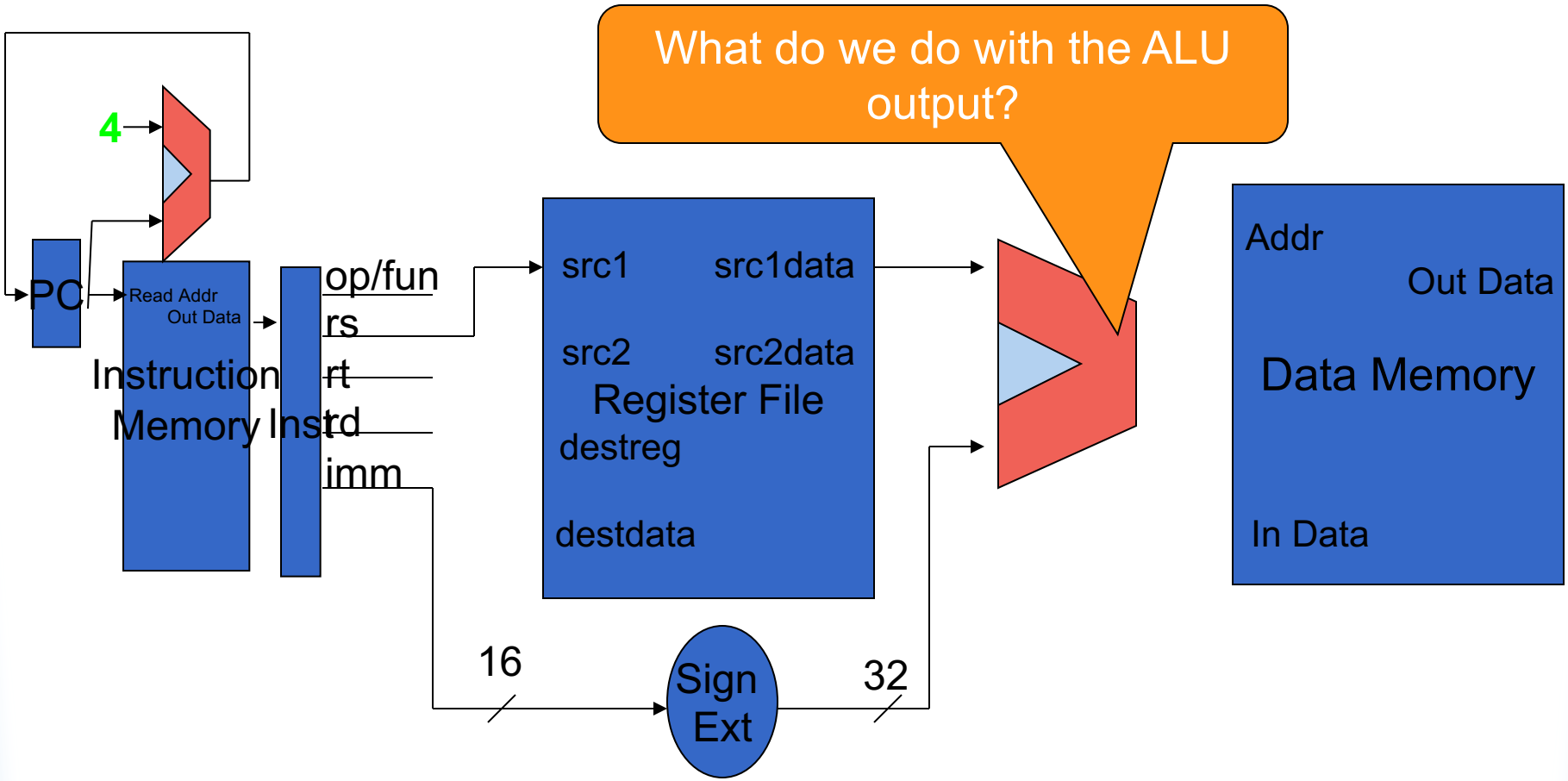
Load Operation

Operation	rs	rt	imm	# meaning
lw $5,8($3)	3	5	8	# $5 <- M[$3 + 8]



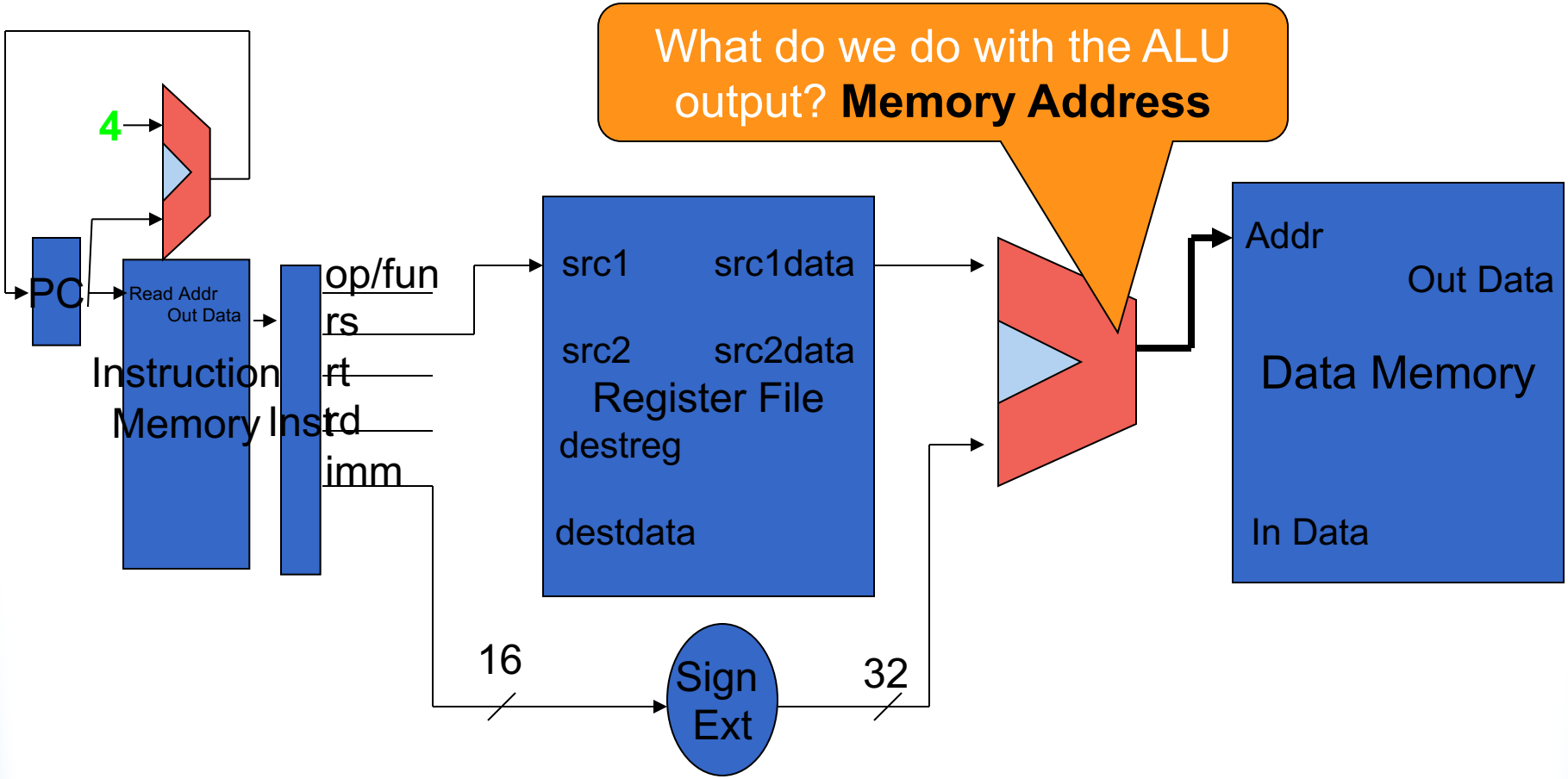
Load Operation

Operation	rs	rt	imm	# meaning
lw $5,8($3)	3	5	8	# $5 <- M[$3 + 8]



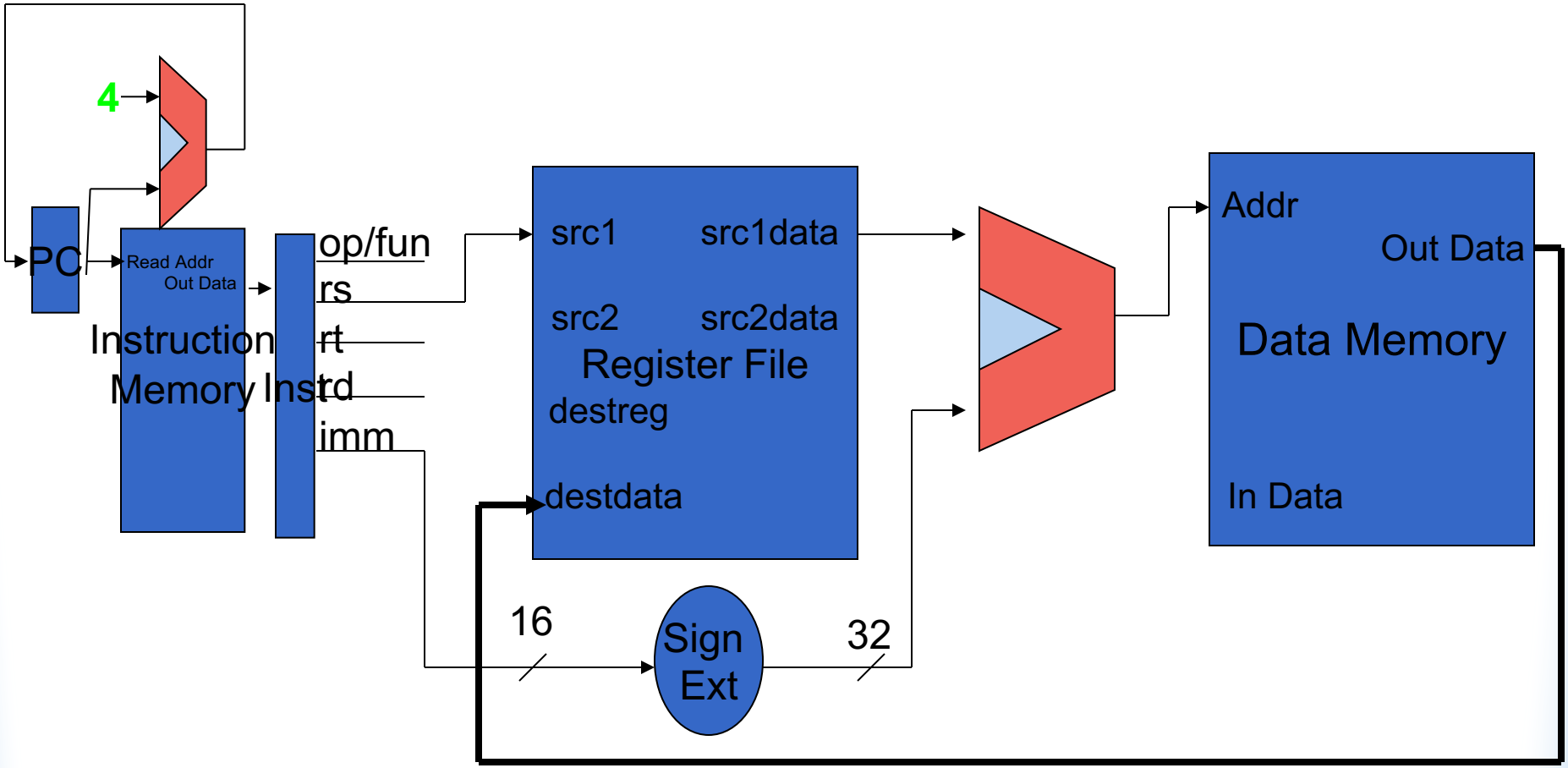
Load Operation

Operation	rs	rt	imm	# meaning
lw $5,8($3)	3	5	8	# $5 <- M[$3 + 8]



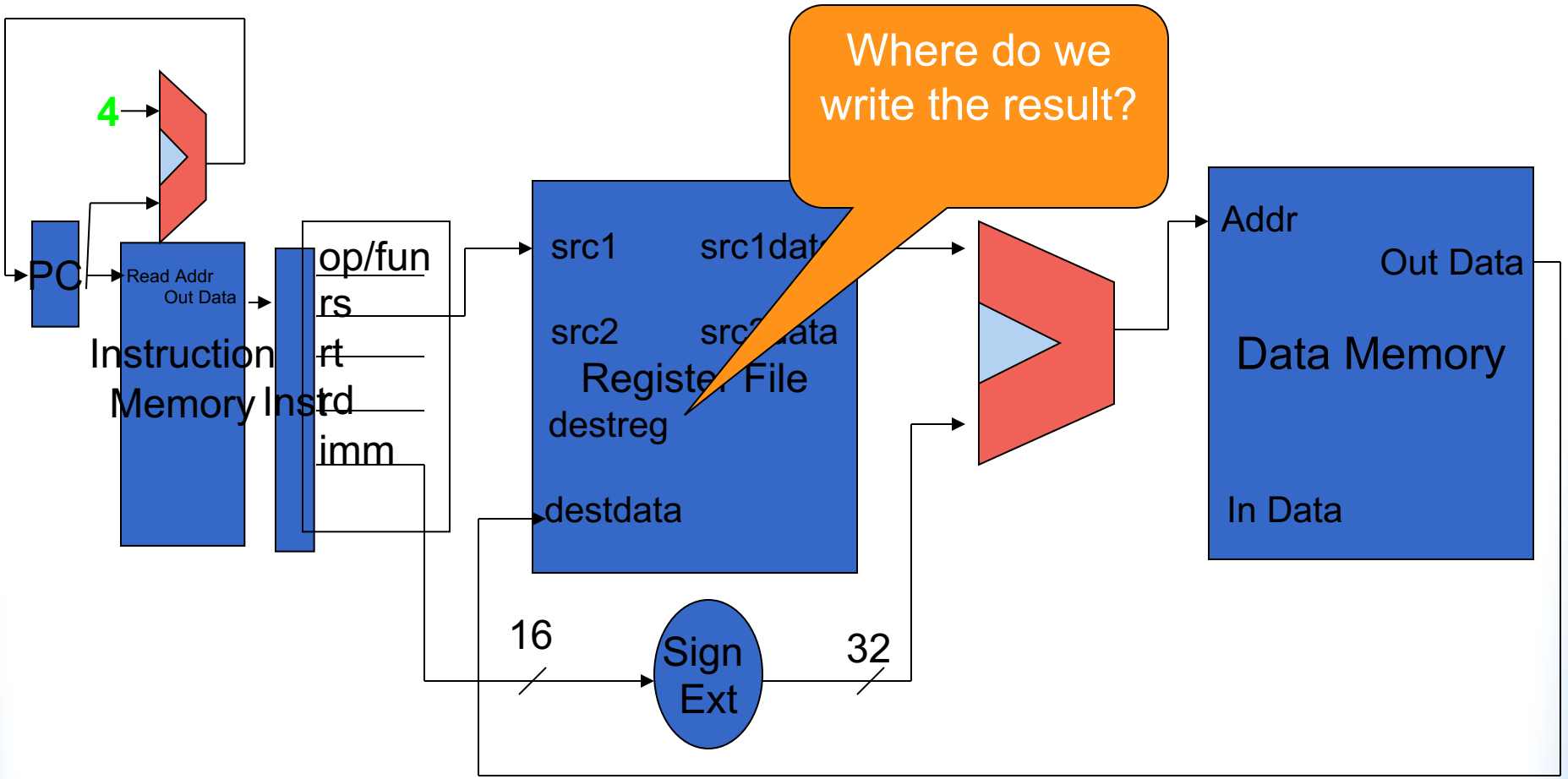
Load Operation

Operation	rs	rt	imm	# meaning
lw $5,8($3)	3	5	8	# $5 <- M[$3 + 8]



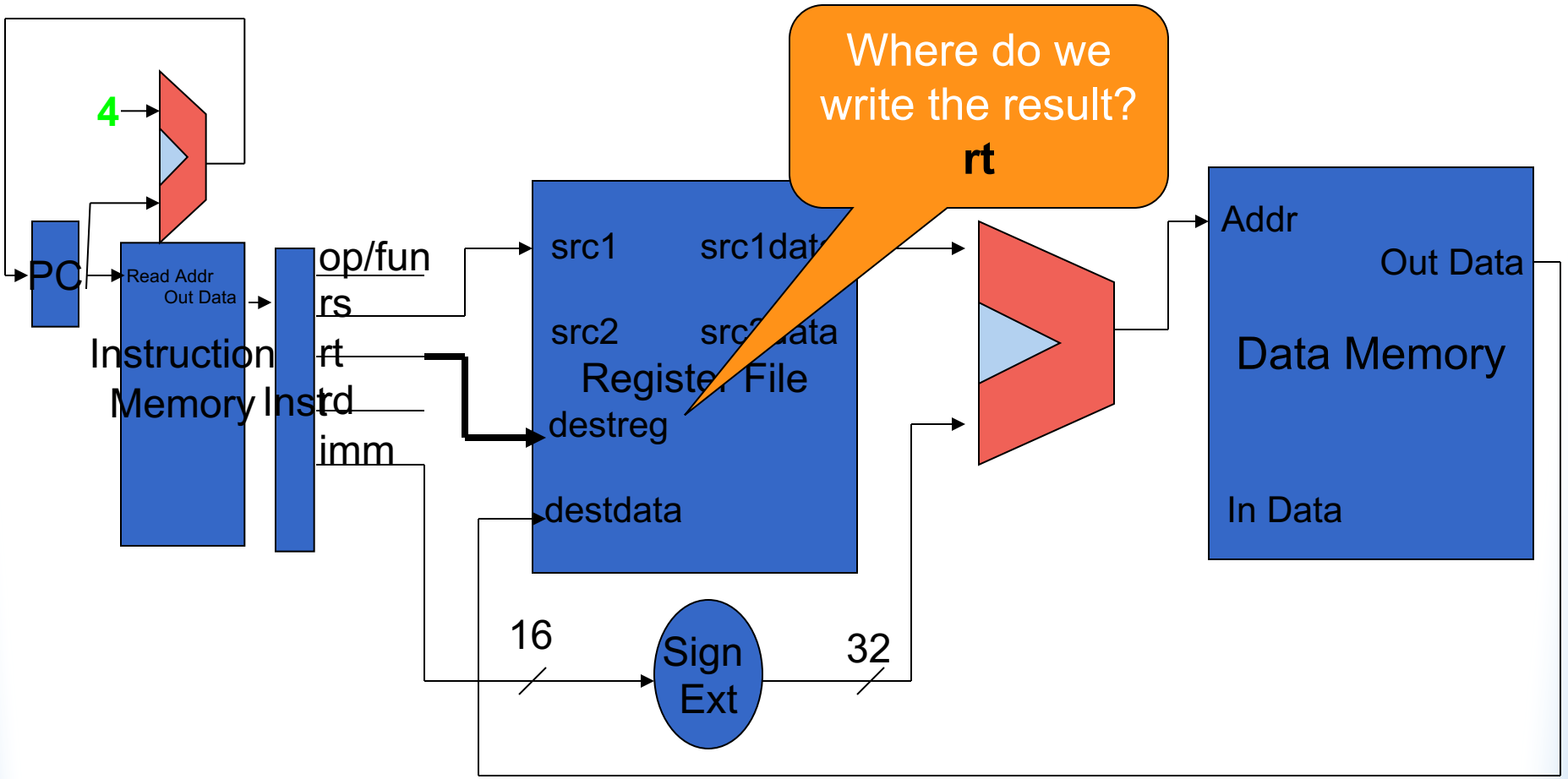
Load Operation

Operation	rs	rt	imm	# meaning
lw $5,8($3)	3	5	8	# $5 <- M[$3 + 8]



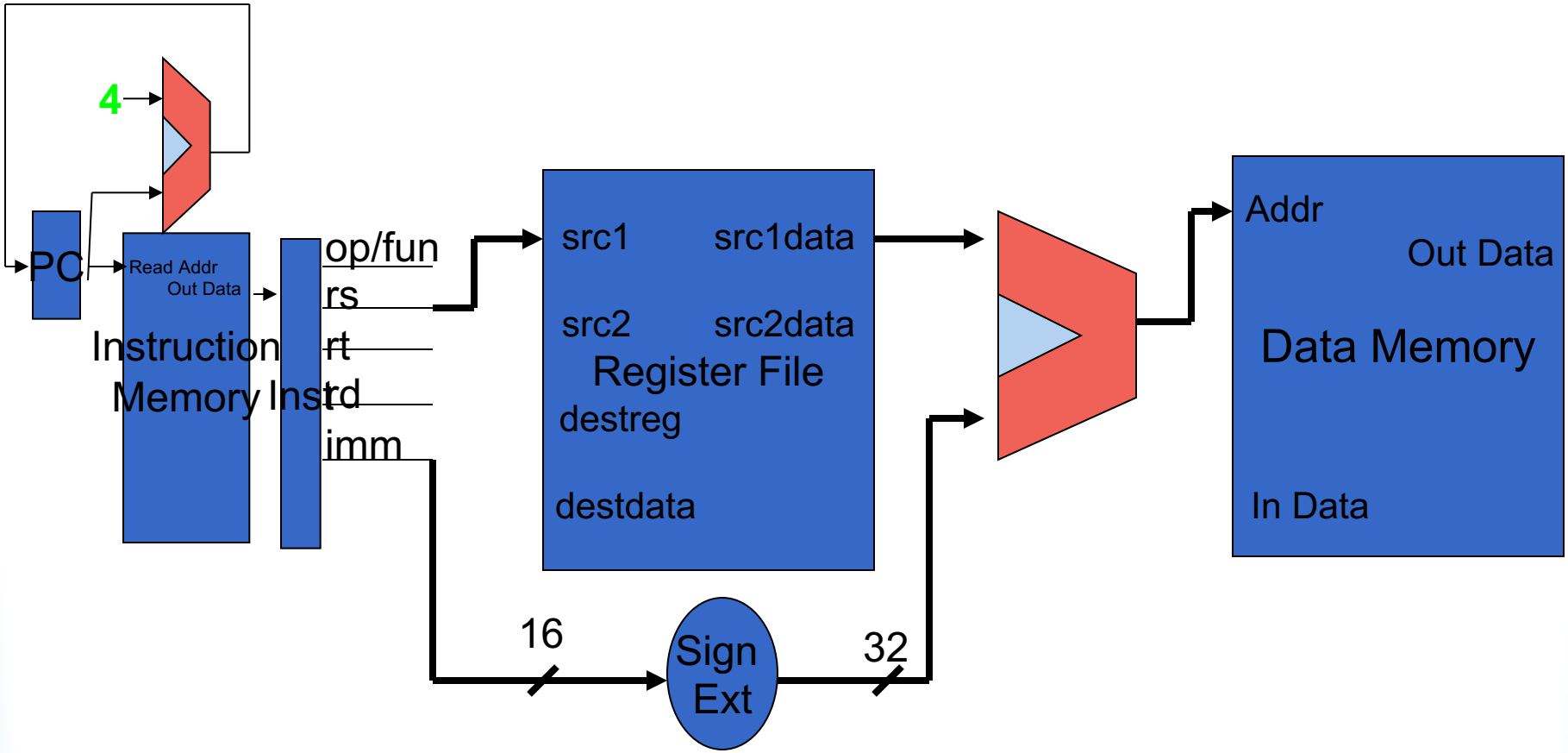
Load Operation

Operation	rs	rt	imm	# meaning
lw $5,8($3)	3	5	8	# $5 <- M[$3 + 8]



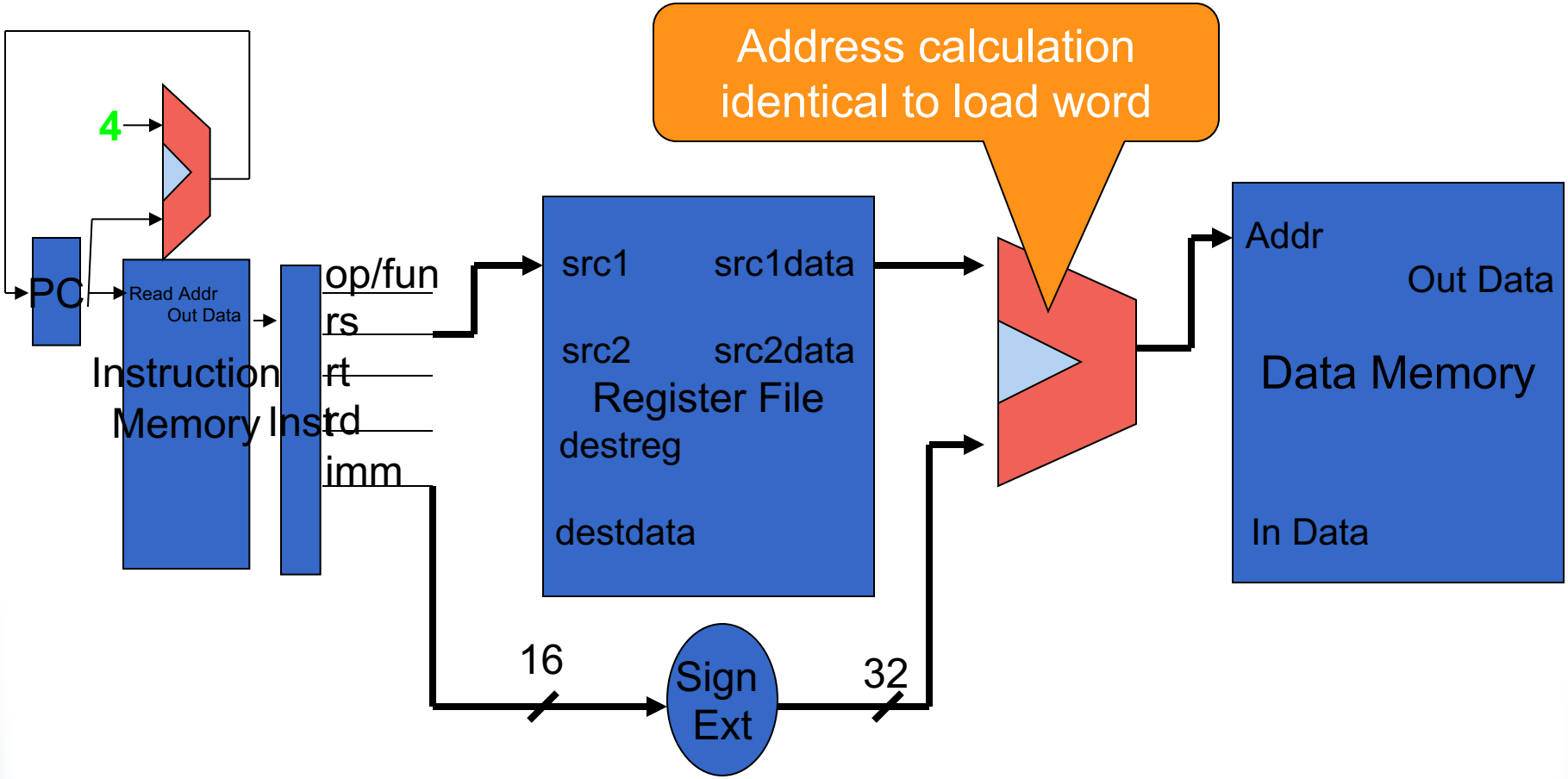
Store Operation

Operation	rs	rt	imm	# meaning
sw $5,8($3)	3	5	8	# M[$3 + 8] <- $5



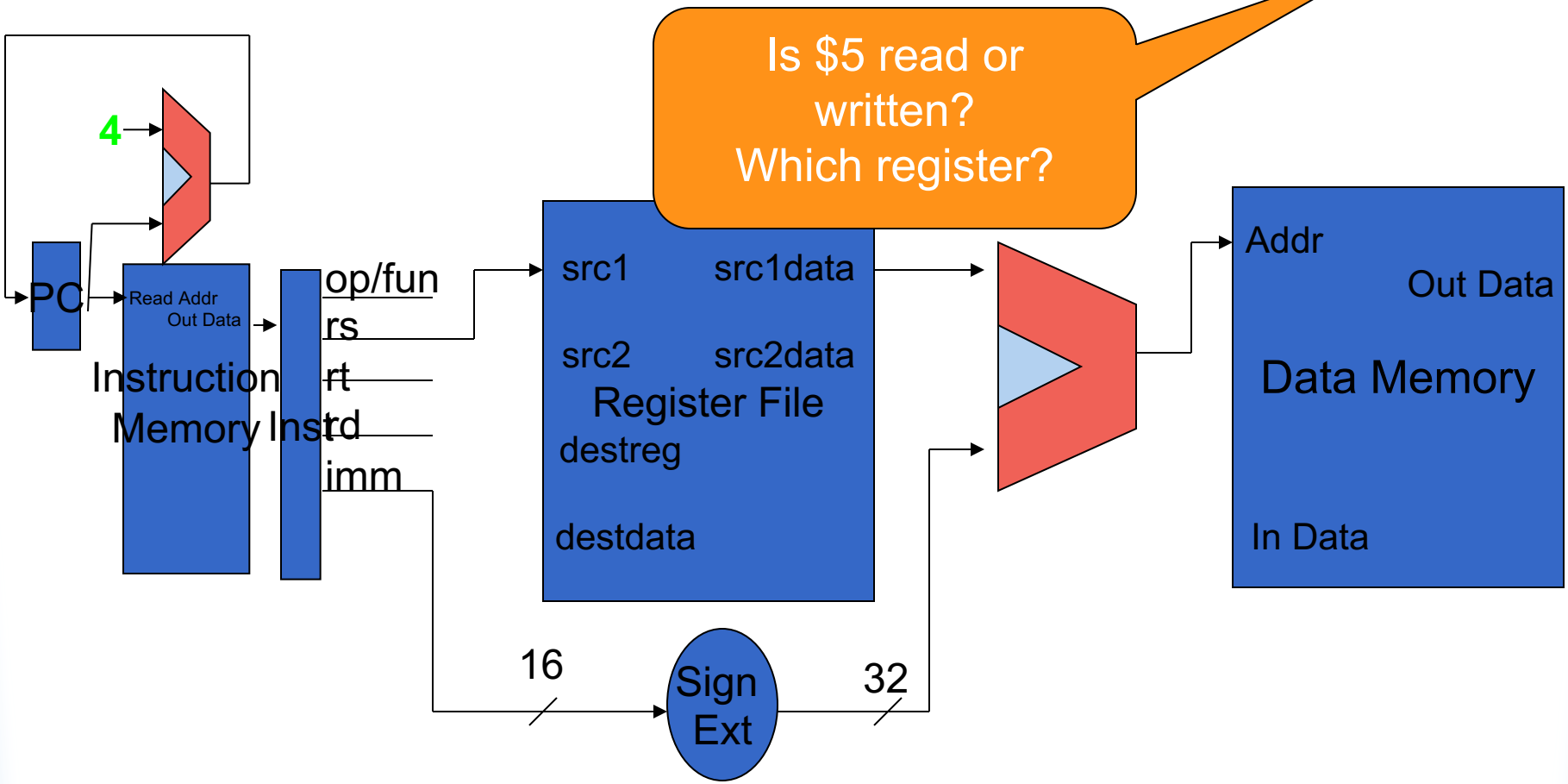
Store Operation

Operation	rs	rt	imm	# meaning
sw $5,8($3)	3	5	8	# M[$3 + 8] <- $5



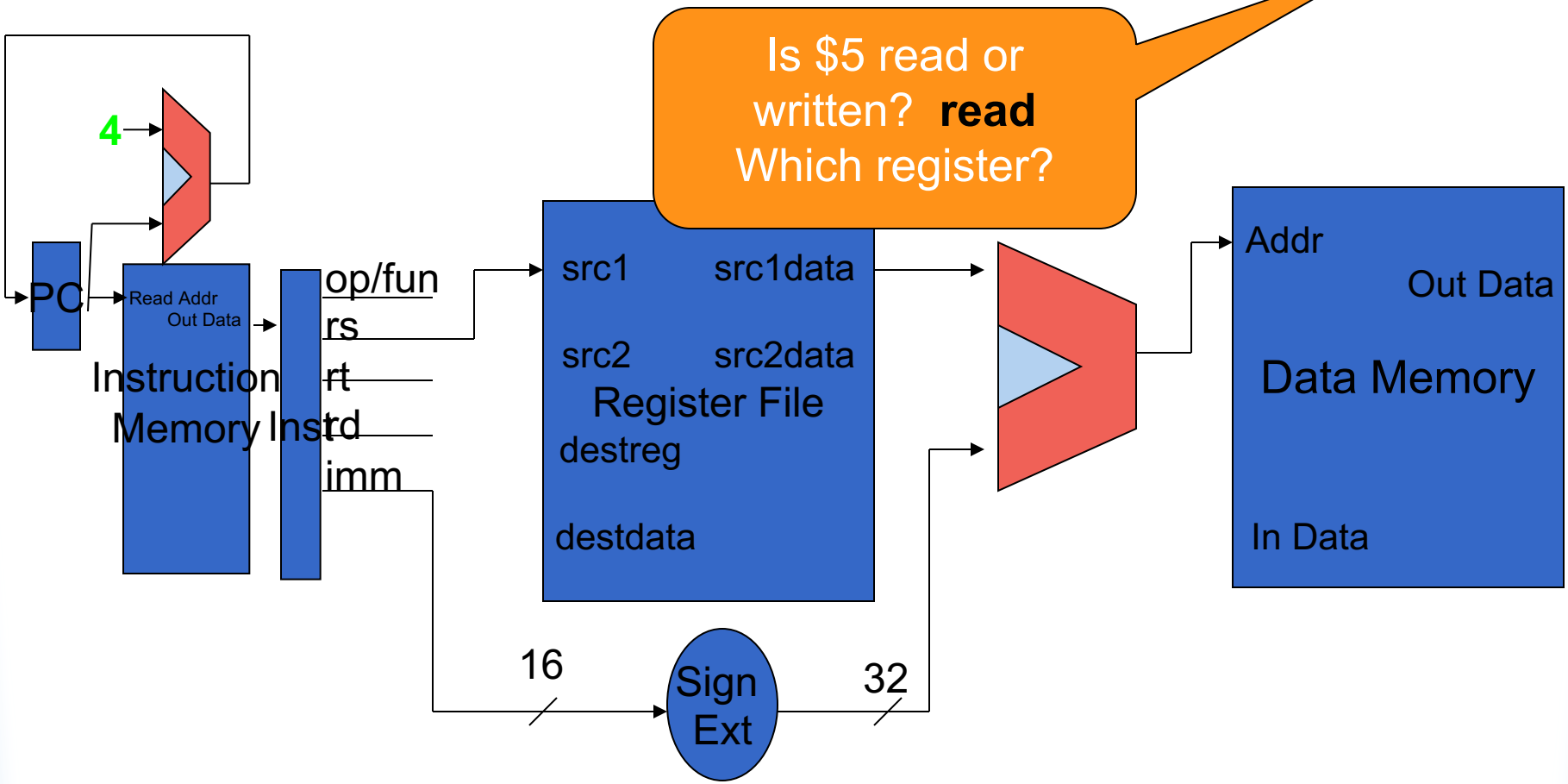
Store Operation

Operation	rs	rt	imm	# meaning
sw $5,8($3)	3	5	8	# M[$3 + 8] <- $5



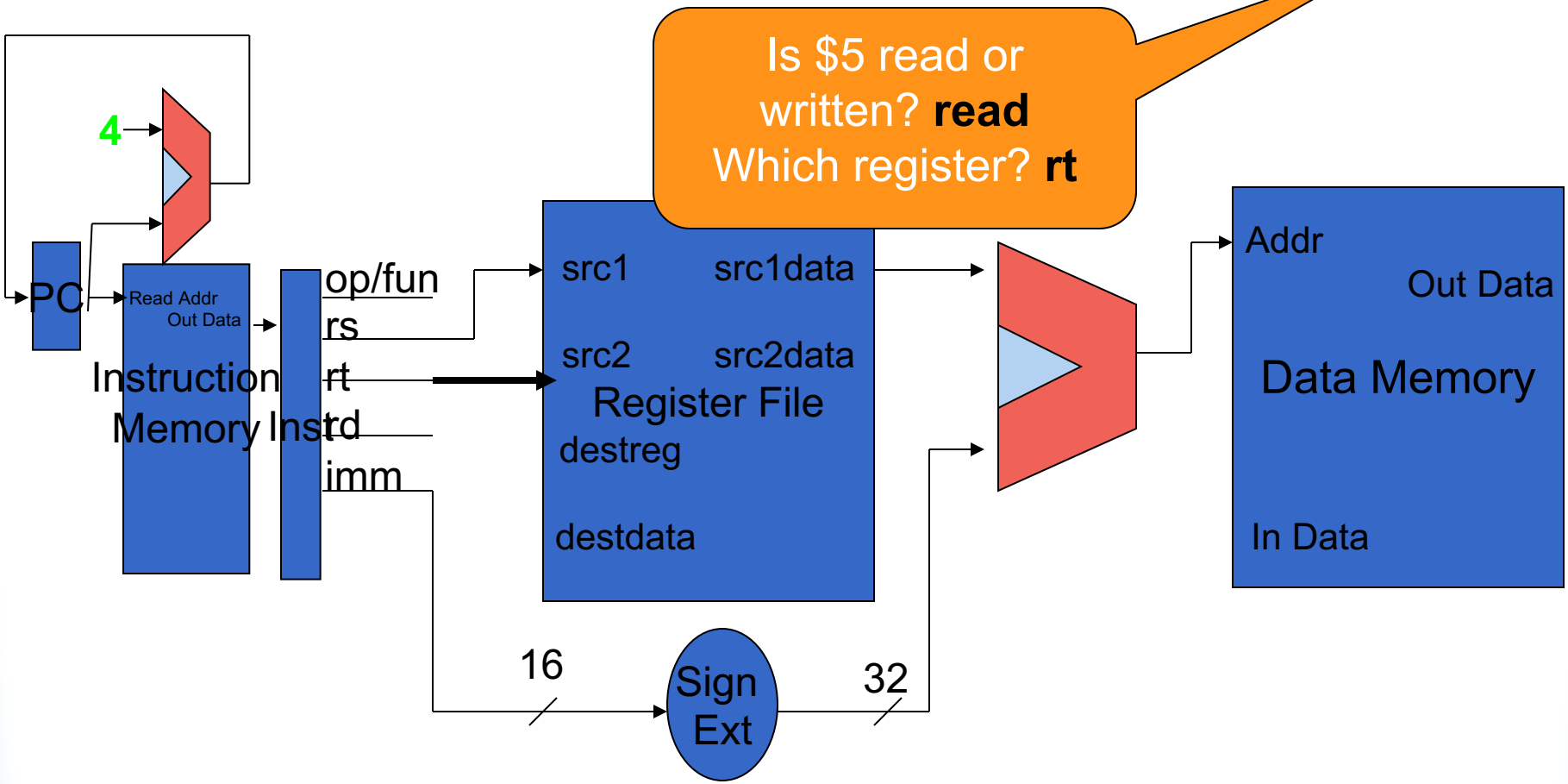
Store Operation

Operation	rs	rt	imm	# meaning
sw $5,8($3)	3	5	8	# M[$3 + 8] <- $5



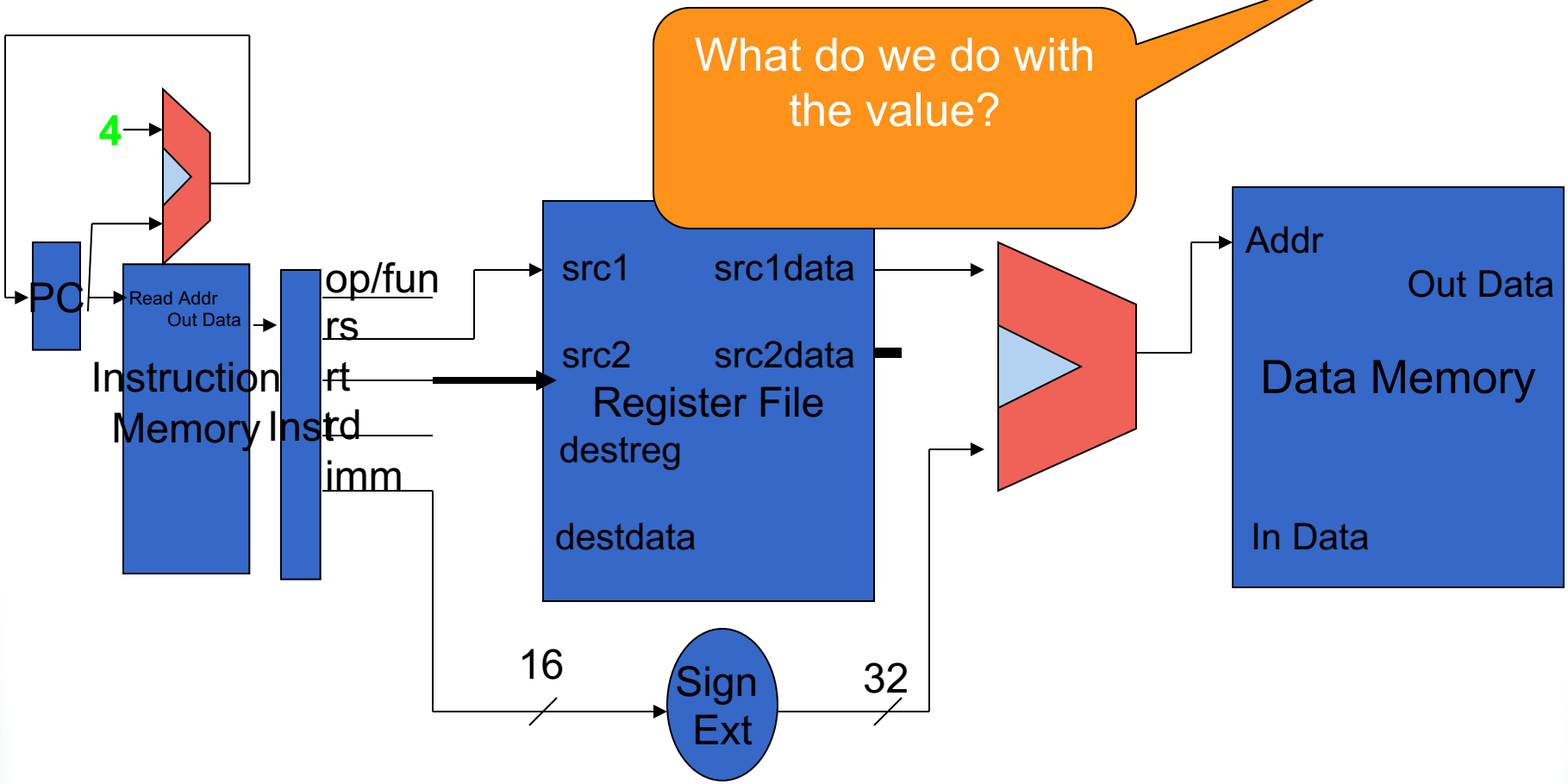
Store Operation

Operation	rs	rt	imm	# meaning
sw $5,8($3)	3	5	8	# M[$3 + 8] <- $5



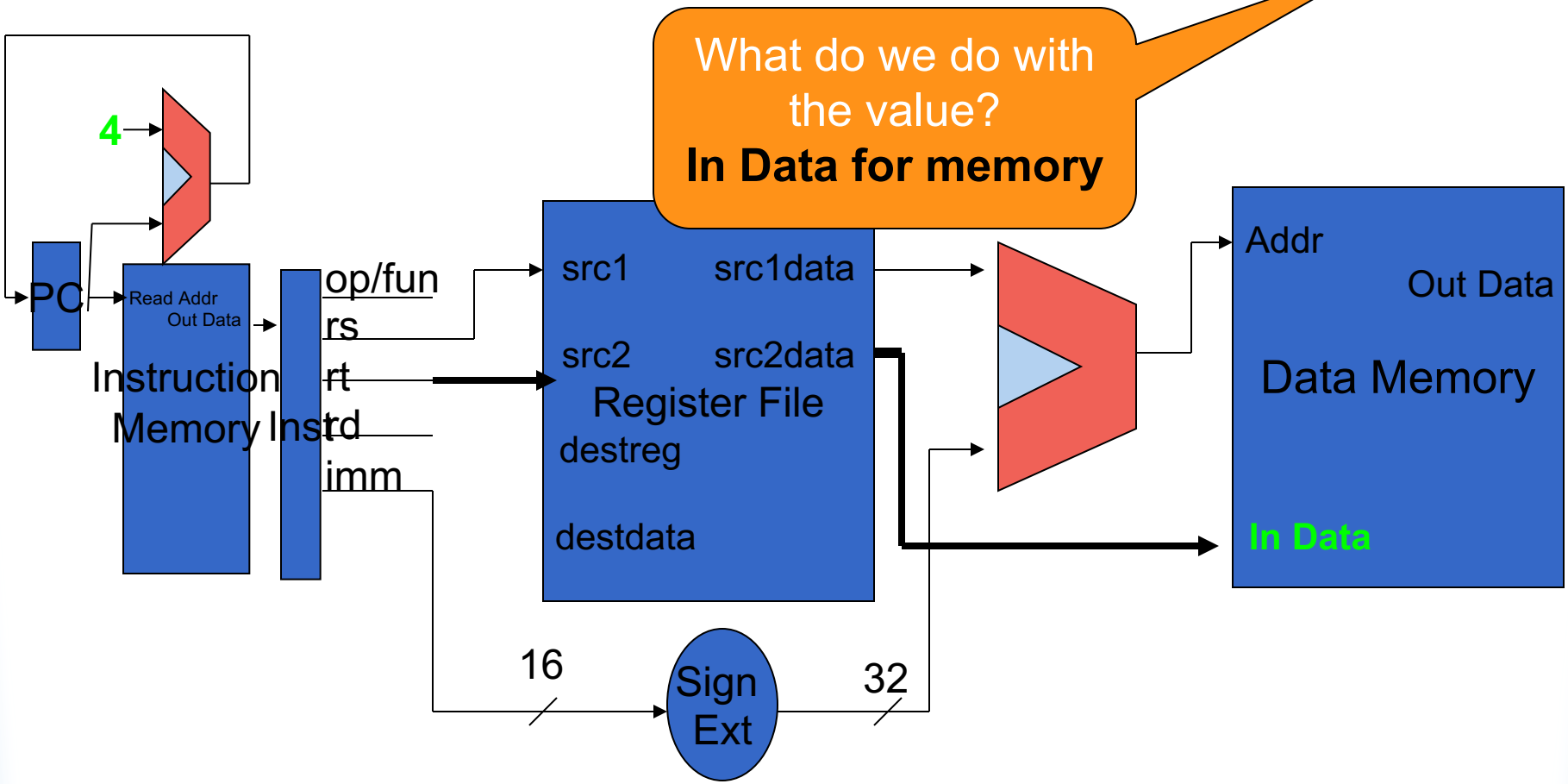
Store Operation

Operation	rs	rt	imm	# meaning
sw $5,8($3)	3	5	8	# M[$3 + 8] <- $5



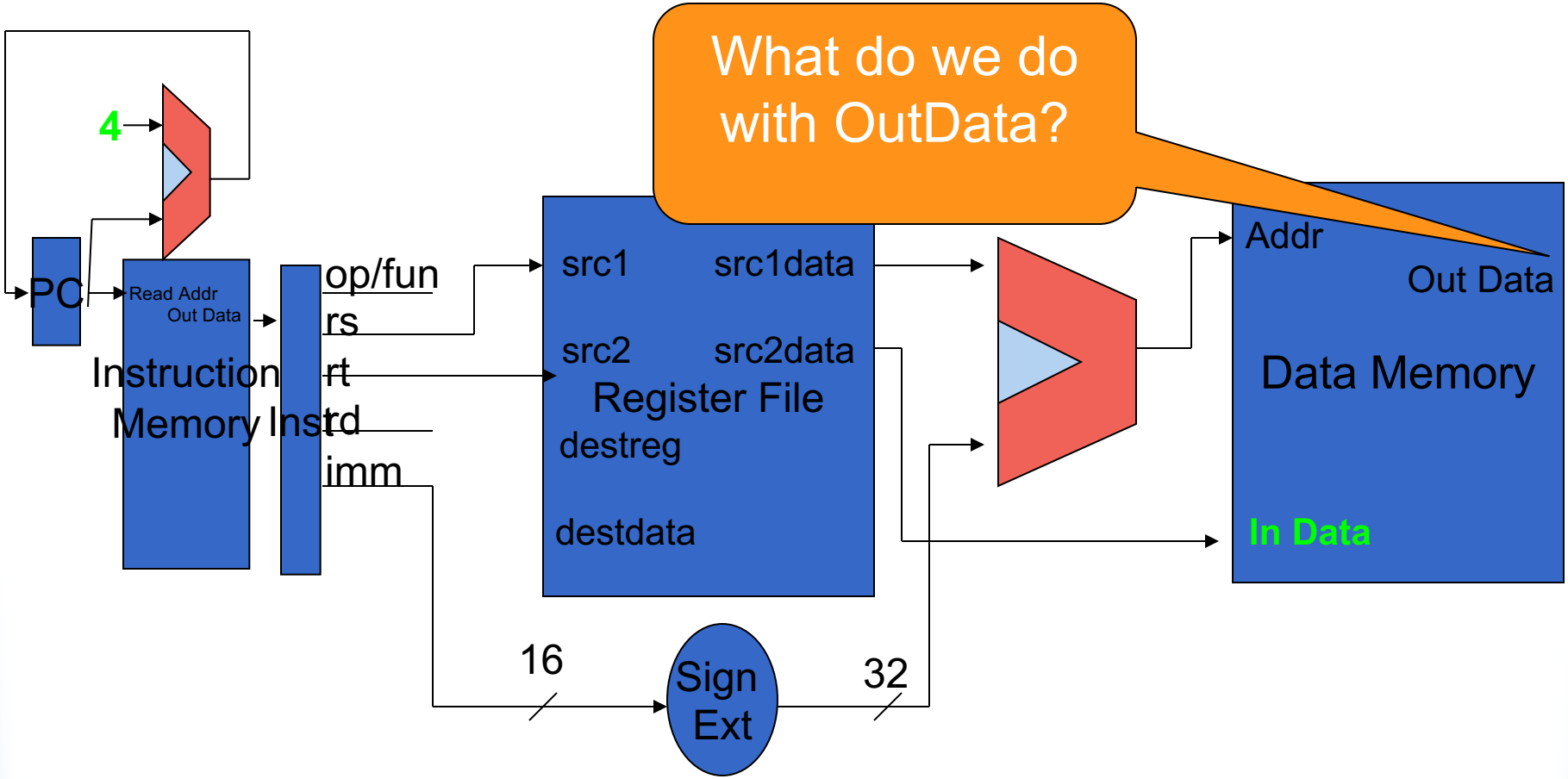
Store Operation

Operation	rs	rt	imm	# meaning
sw $5,8($3)	3	5	8	# M[$3 + 8] <- $5



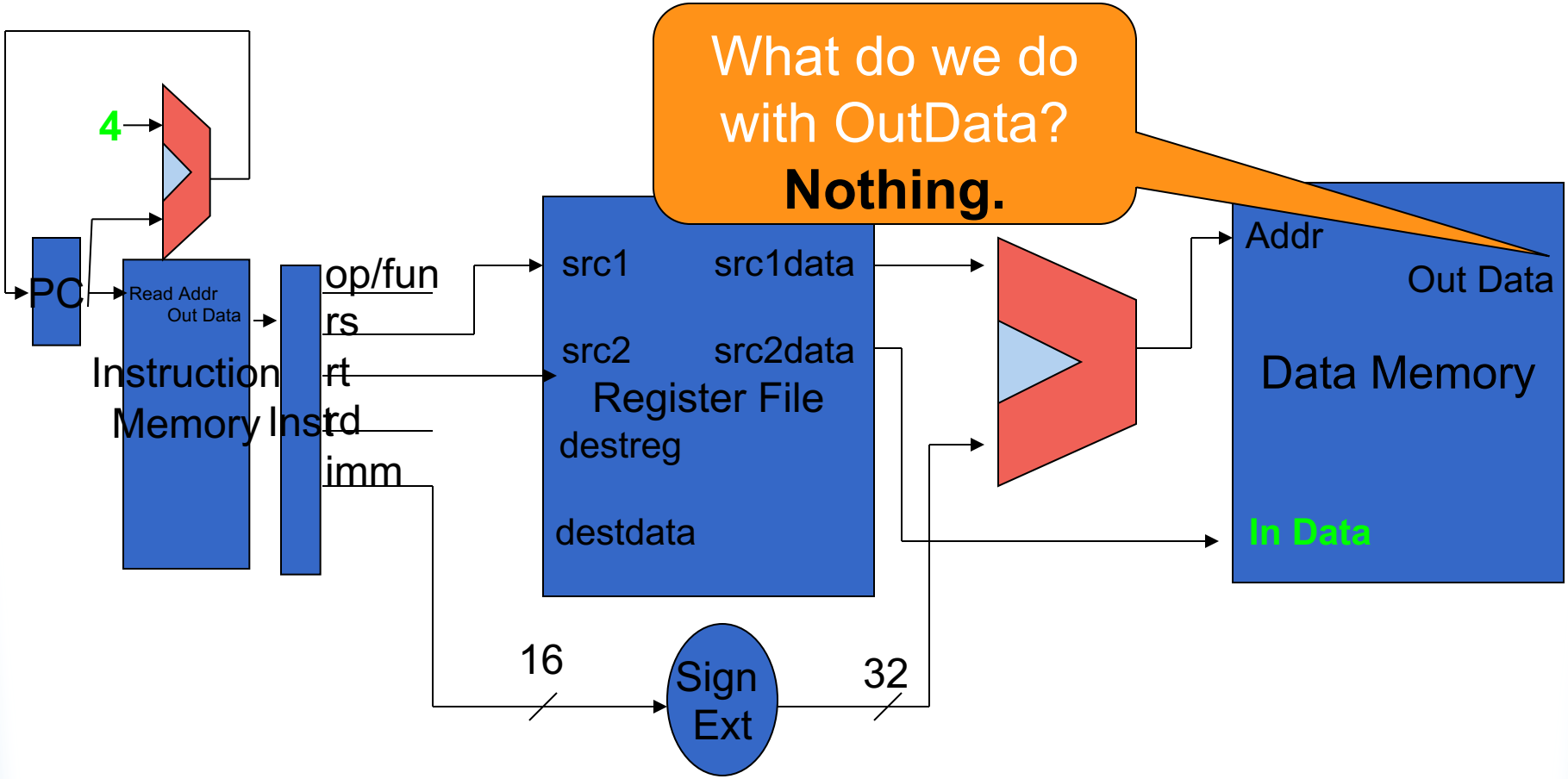
Store Operation

Operation	rs	rt	imm	# meaning
sw $5,8($3)	3	5	8	# M[$3 + 8] <- $5

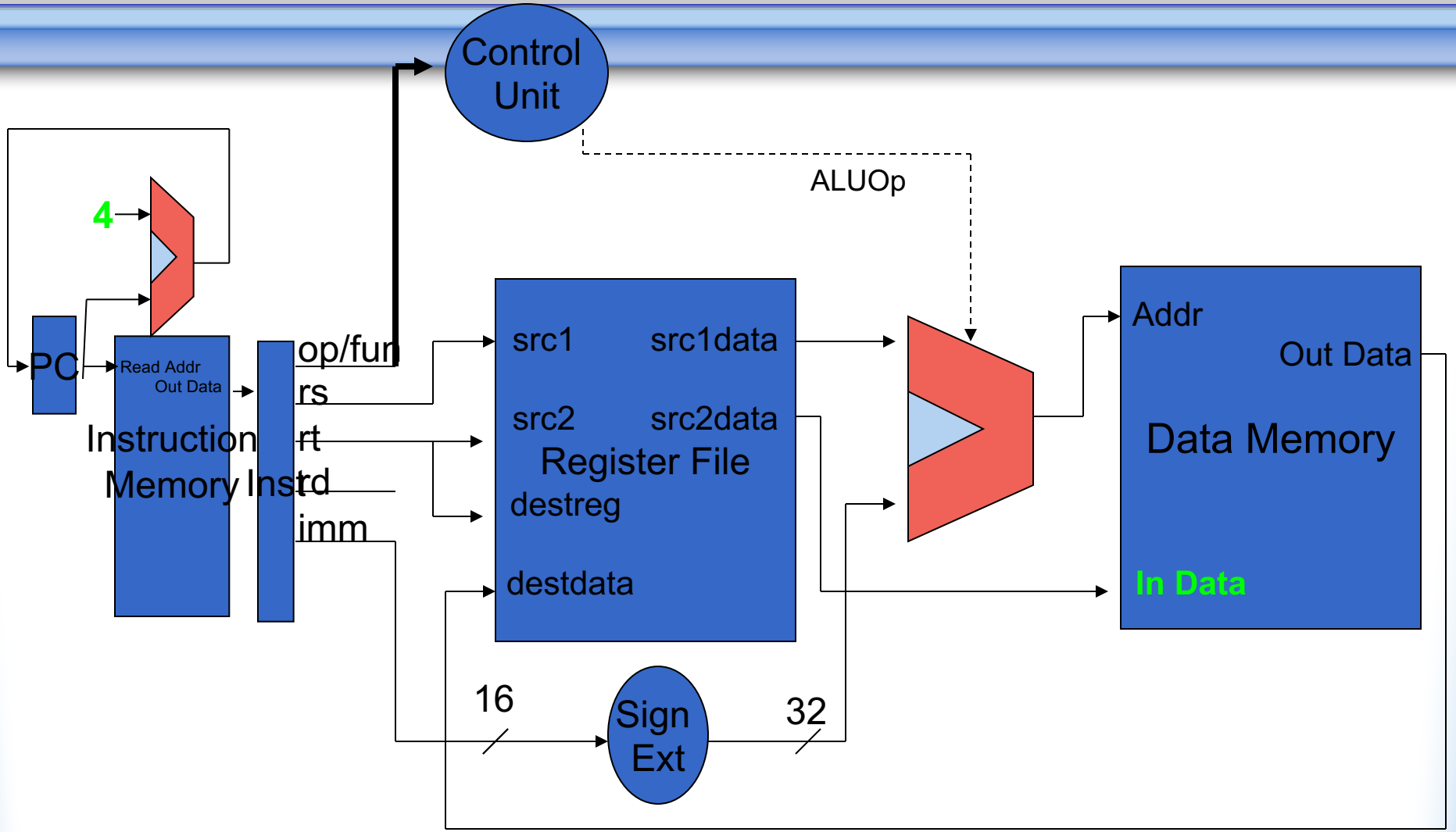


Store Operation

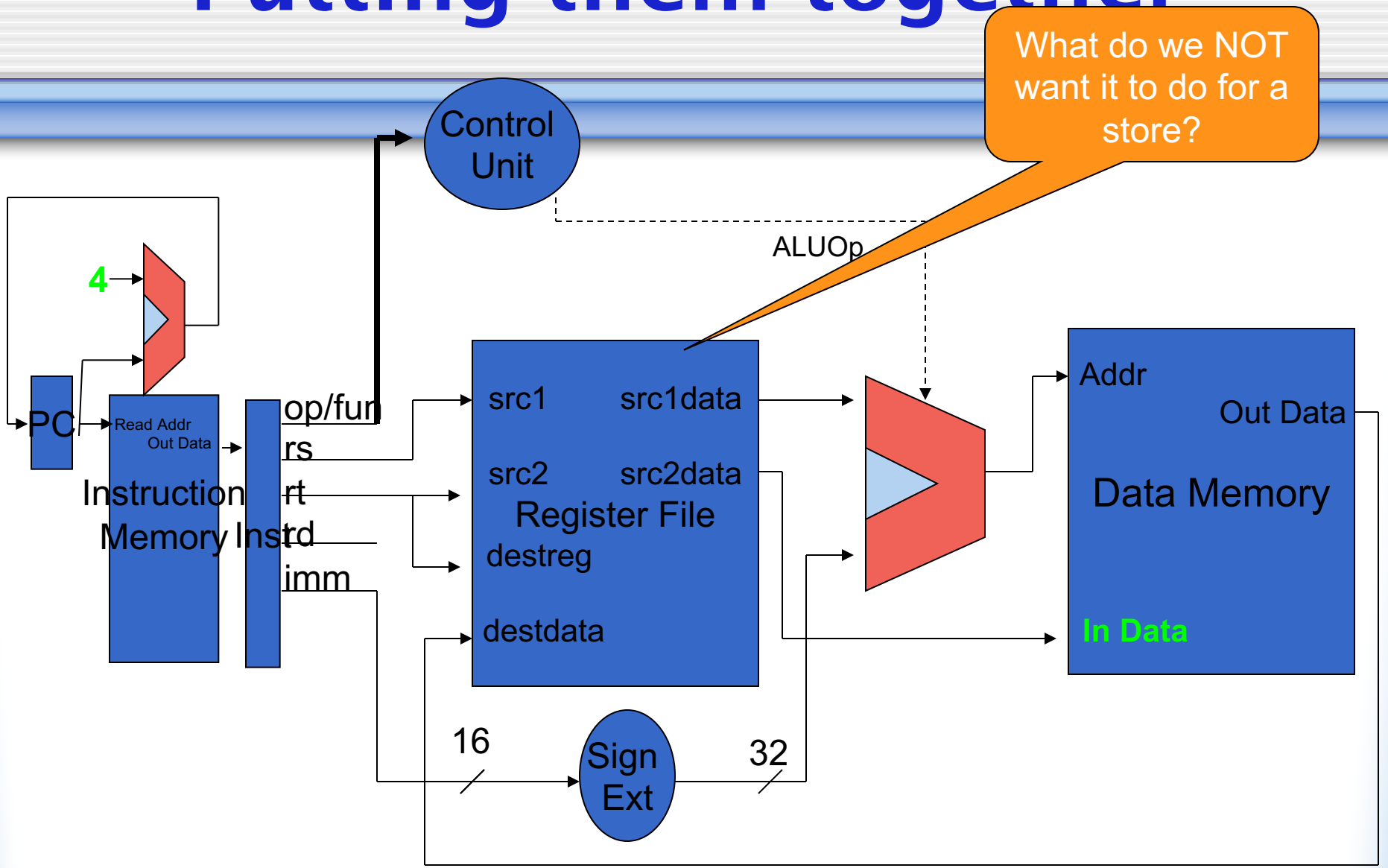
Operation	rs	rt	imm	# meaning
sw $5,8($3)	3	5	8	# M[$3 + 8] <- $5



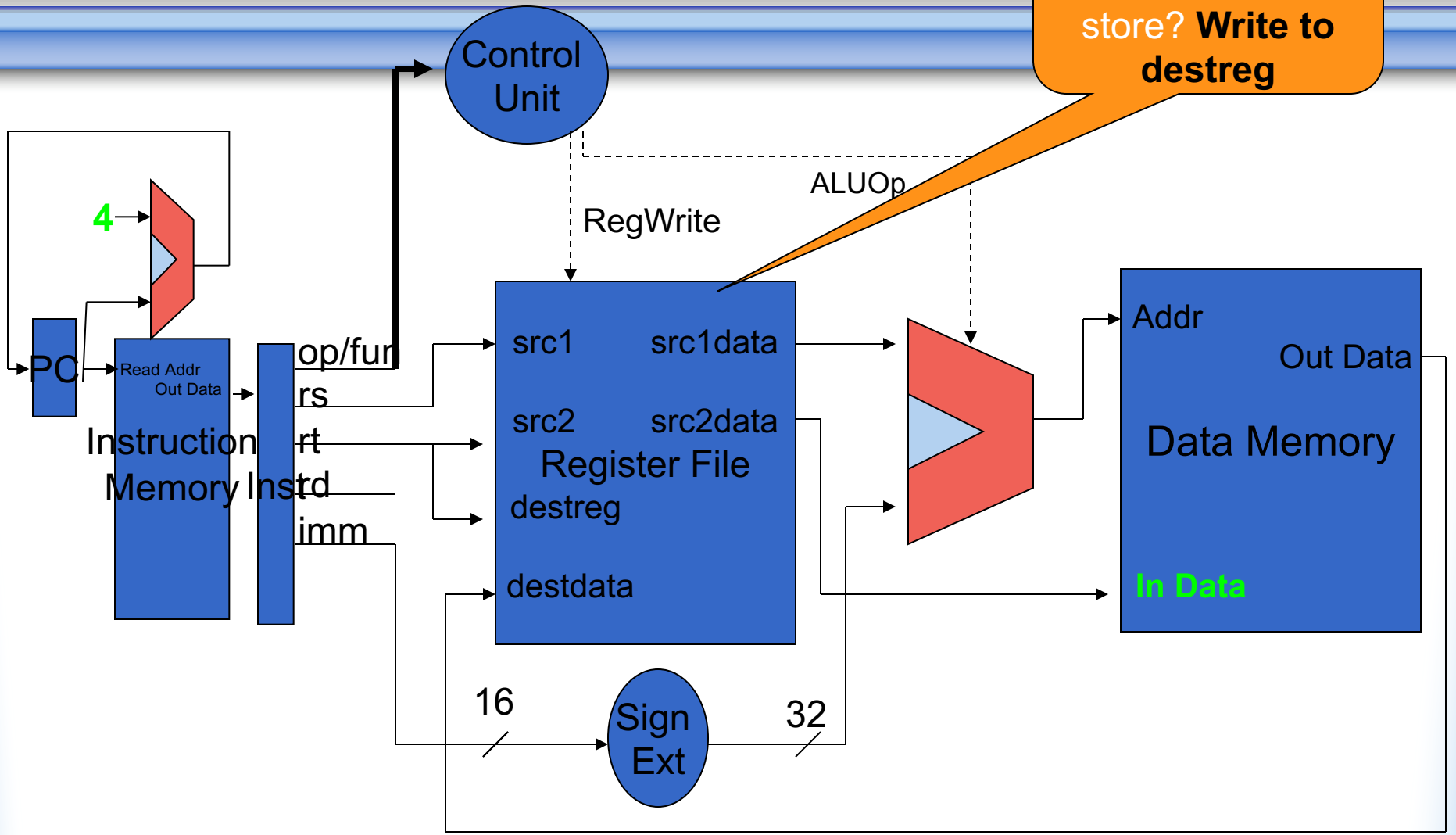
Putting them together



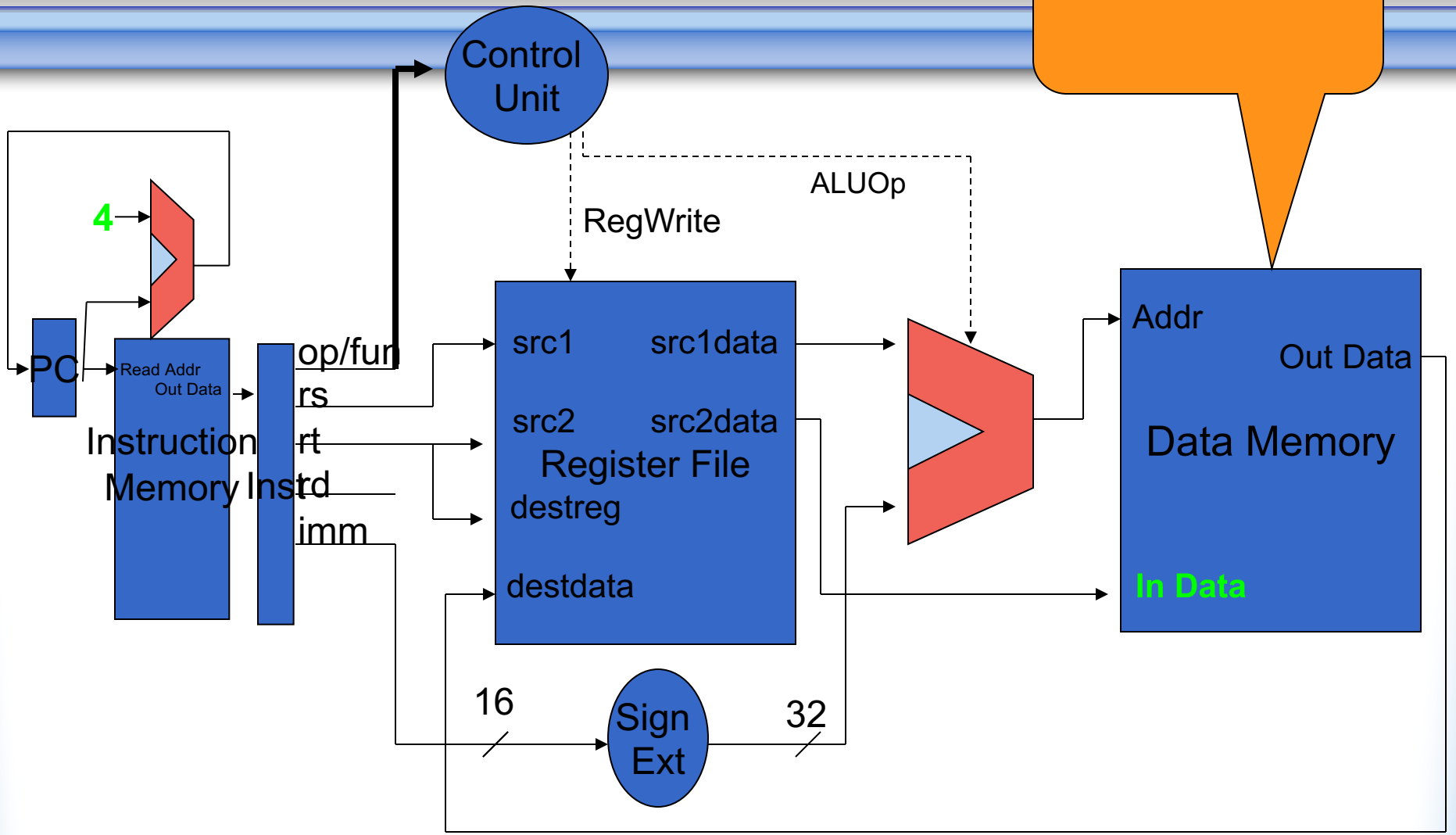
Putting them together



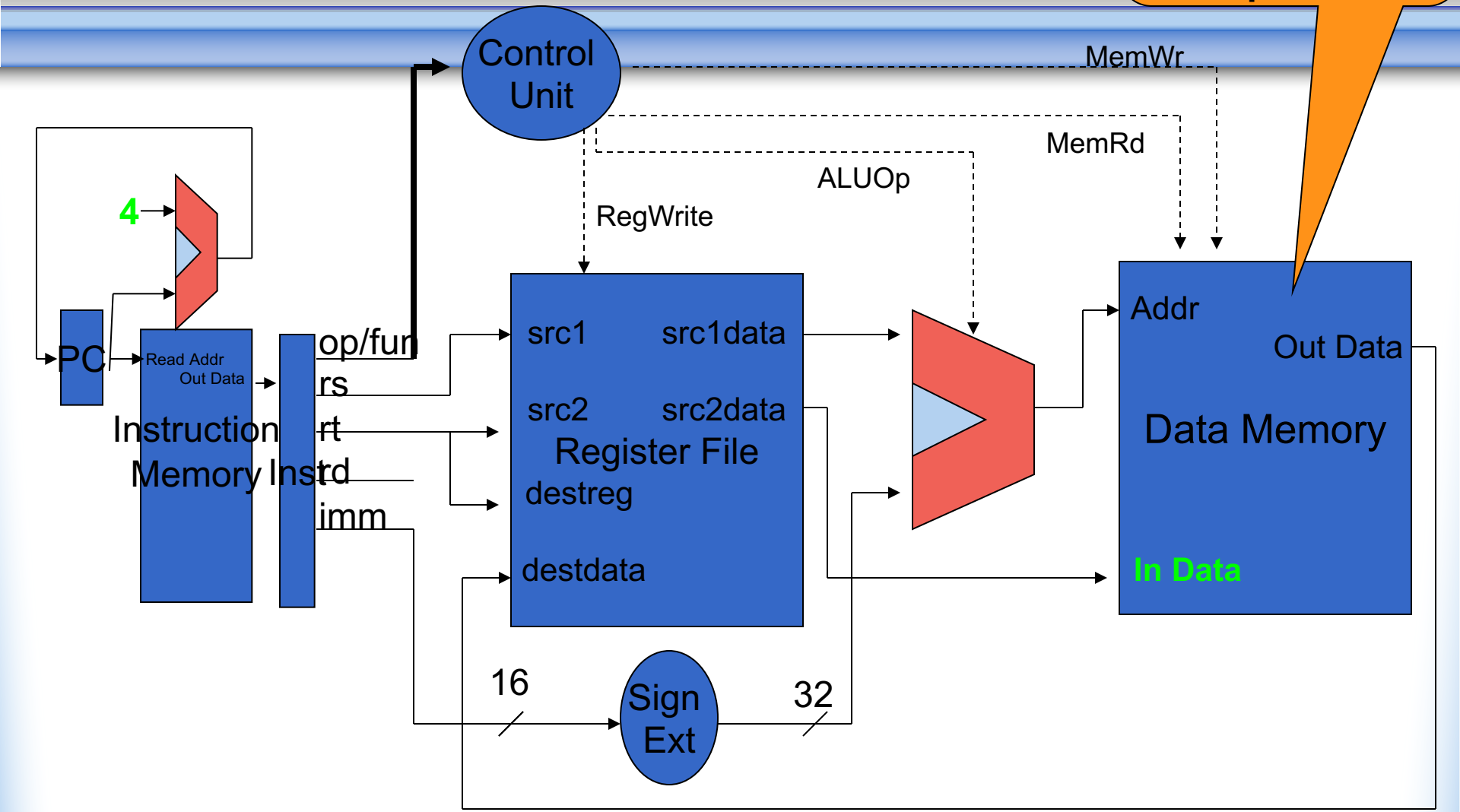
Putting them together



Putting them together

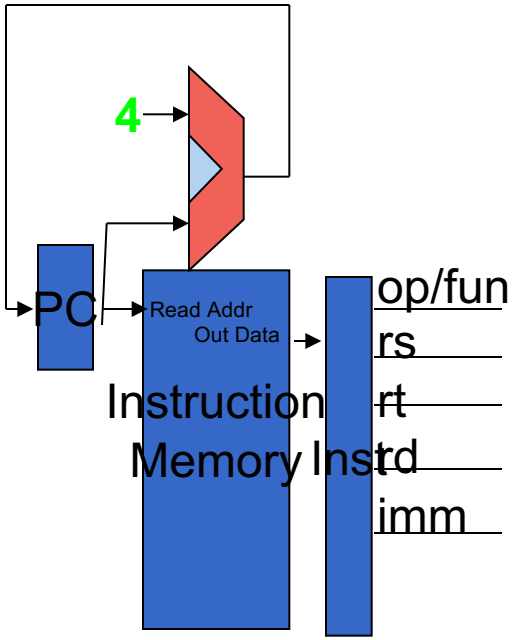


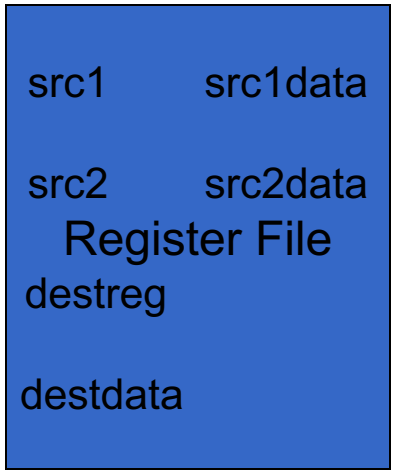
Putting them togeth

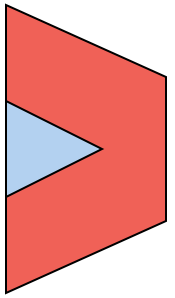


“beq” Instruction

Operation	rs	rt	imm	# meaning
beq $3,$5,lp	3	5	6	# if ($3 == $5) goto lp

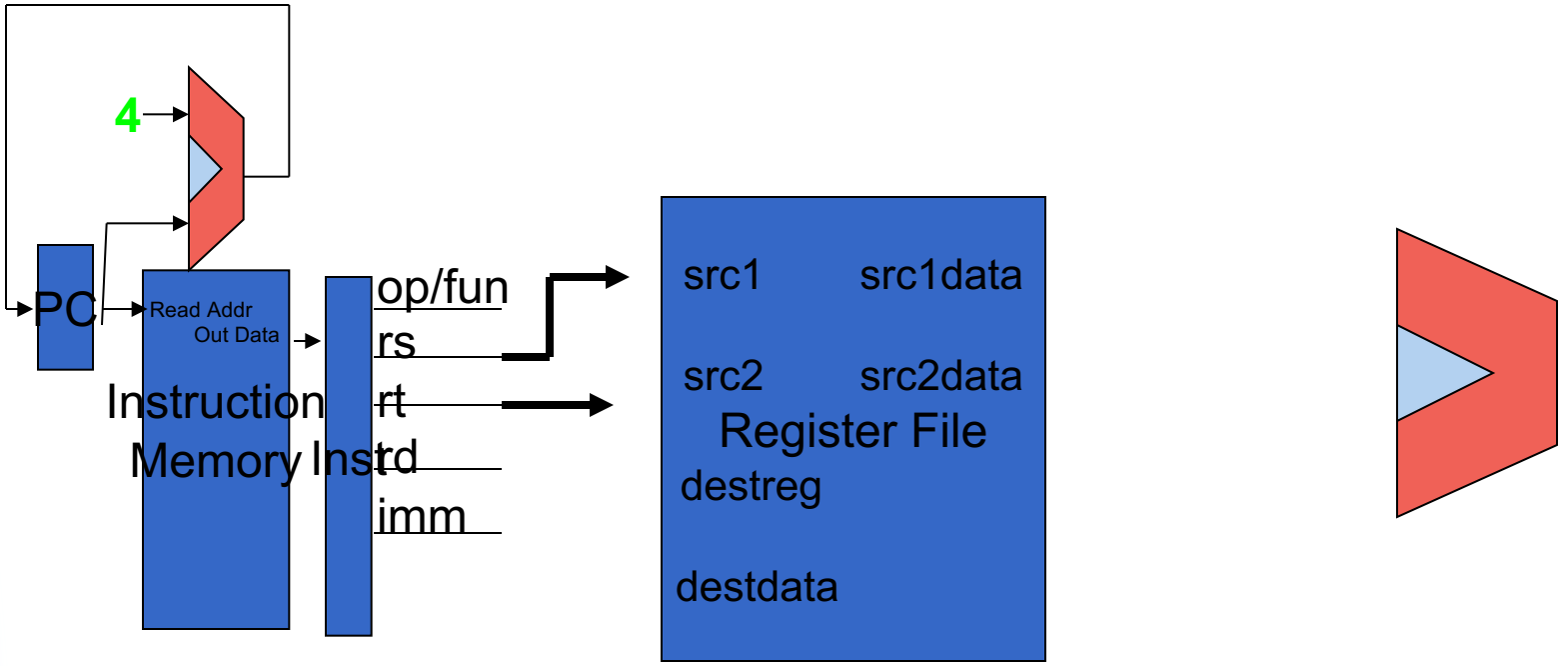






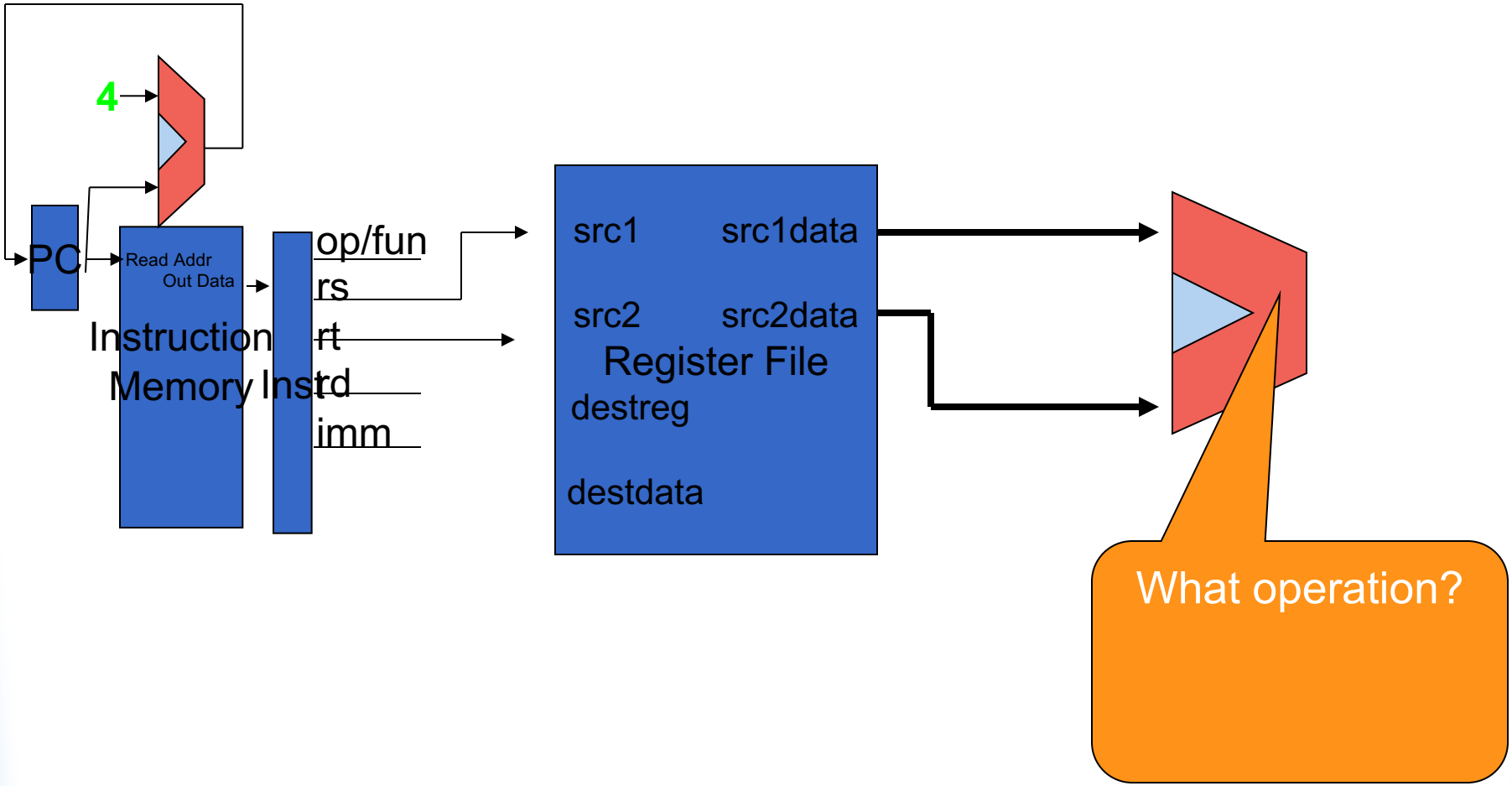
"beq" Instruction

Operation	rs	rt	imm	# meaning
beq $3,$5,lp	3	5	6	# if ($3 == $5) goto lp



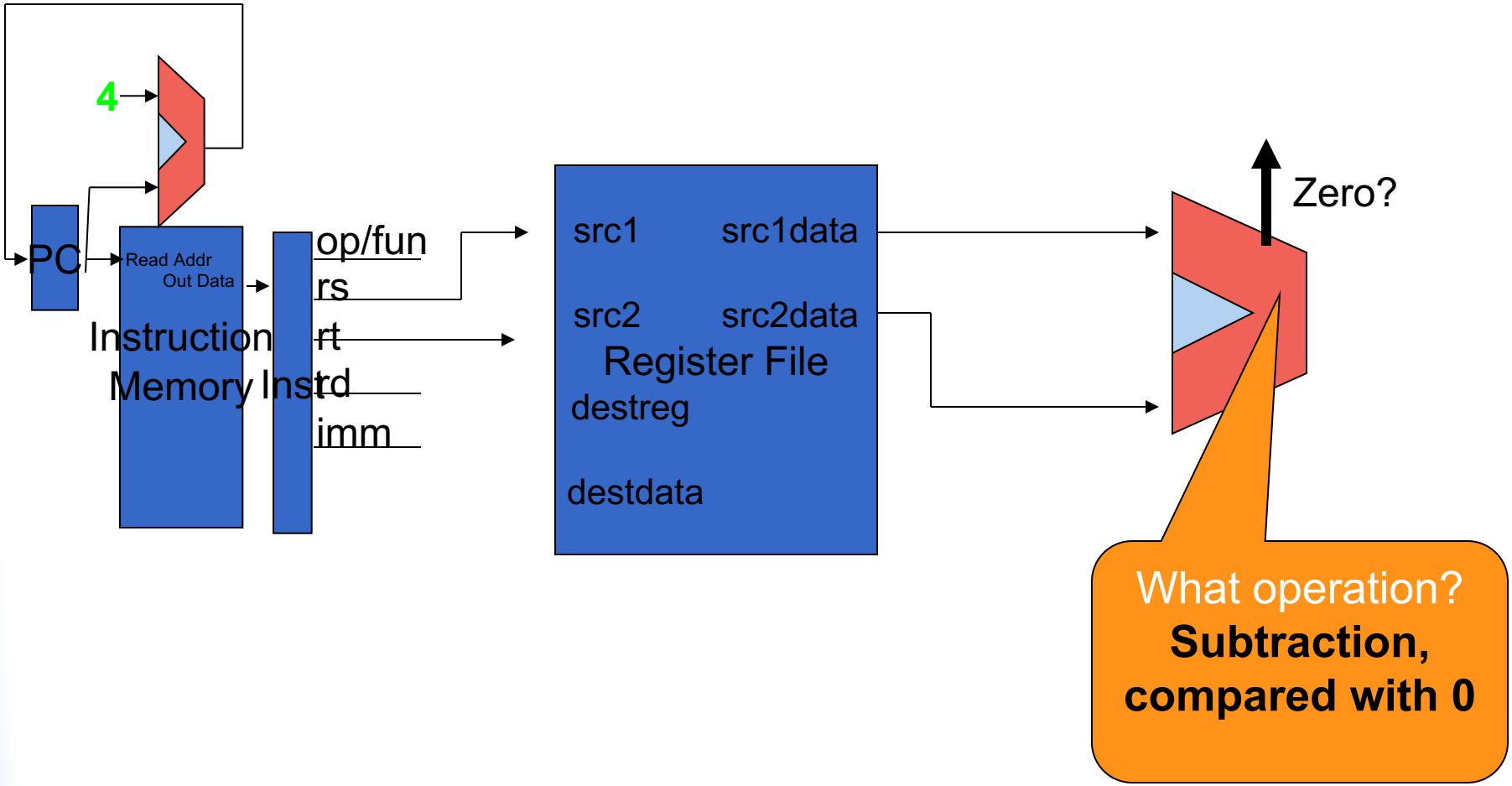
"beq" Instruction

Operation	rs	rt	imm	# meaning
beq $3,$5,lp	3	5	6	# if ($3 == $5) goto lp



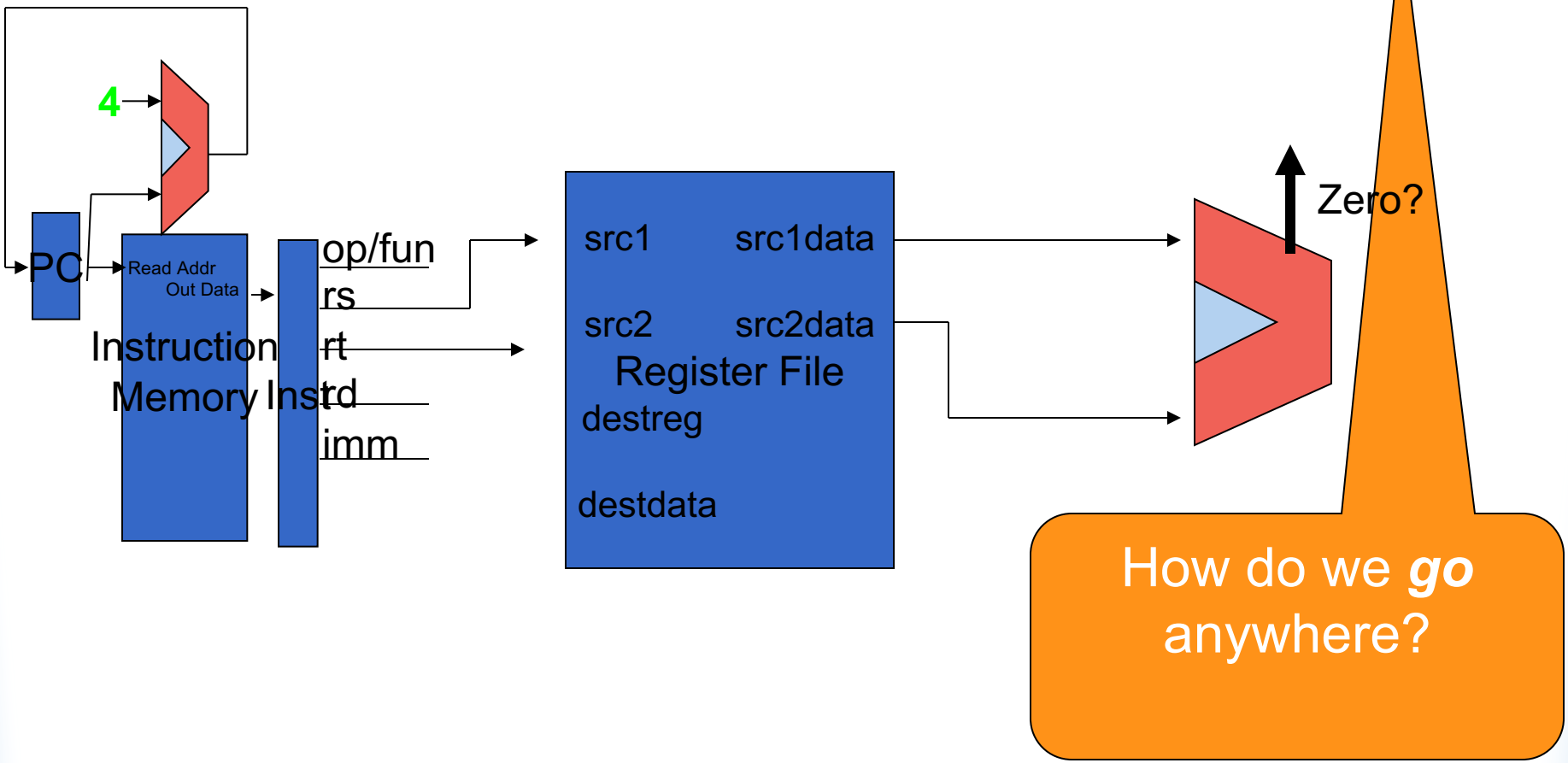
“beq” Instruction

Operation	rs	rt	imm	# meaning
beq $3,$5,lp	3	5	6	# **if ($3 == $5)** goto lp



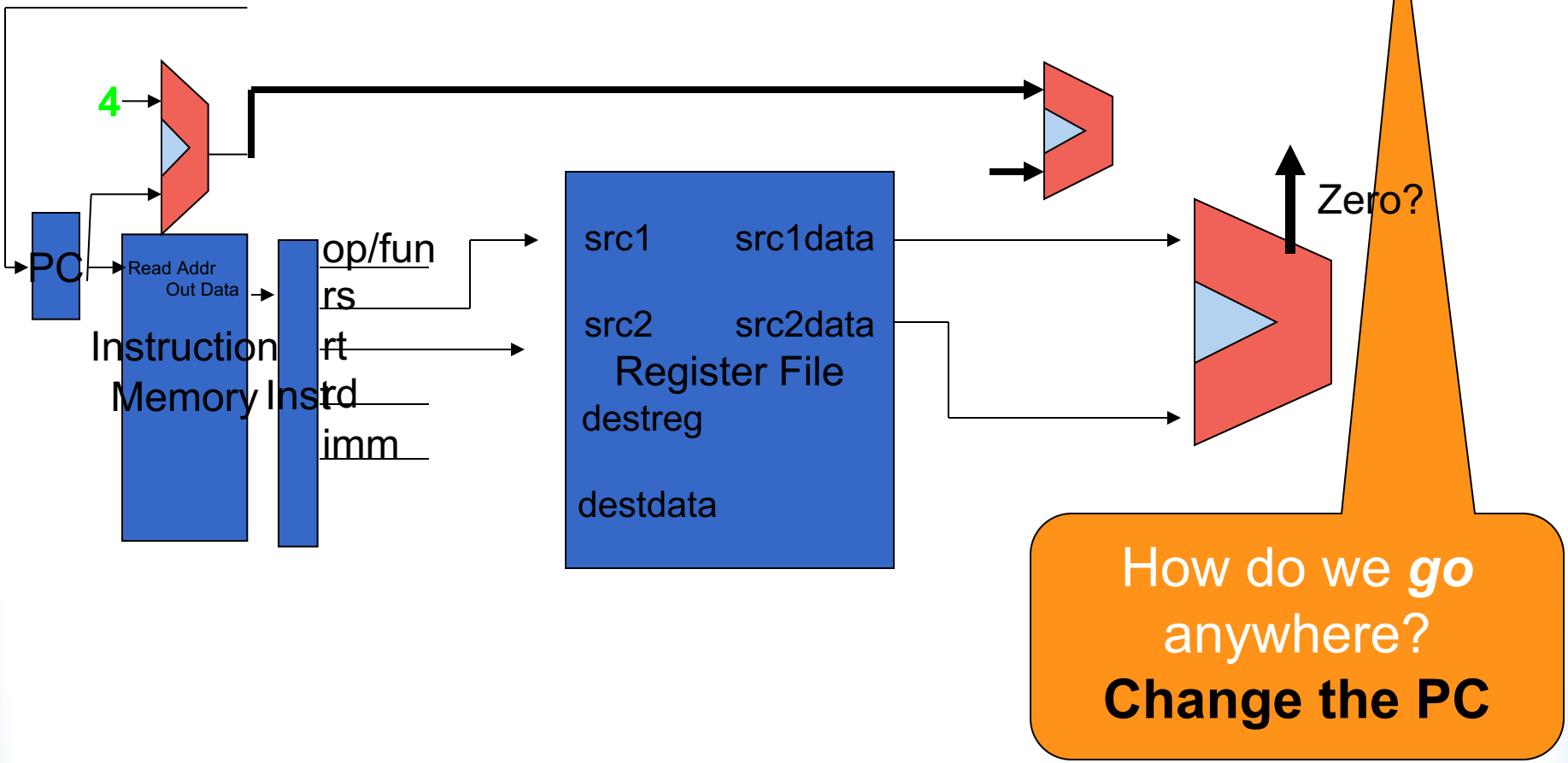
"beq" Instruction

Operation	rs	rt	imm	# meaning
beq $3,$5,lp	3	5	6	# if ($3 == $5) goto lp



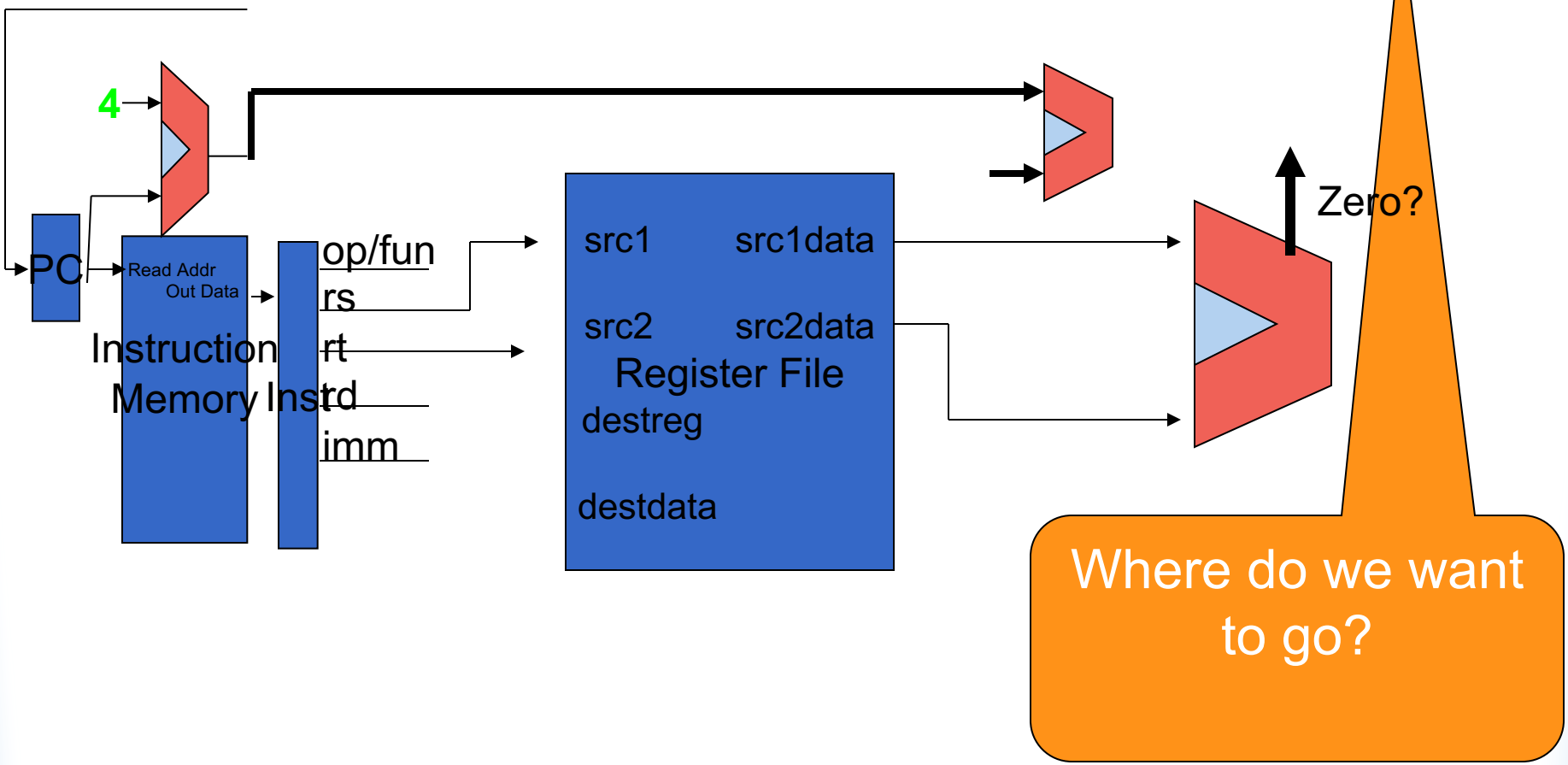
"beq" Instruction

Operation	rs	rt	imm	# meaning
beq $3,$5,lp	3	5	6	# if ($3 == $5) goto lp



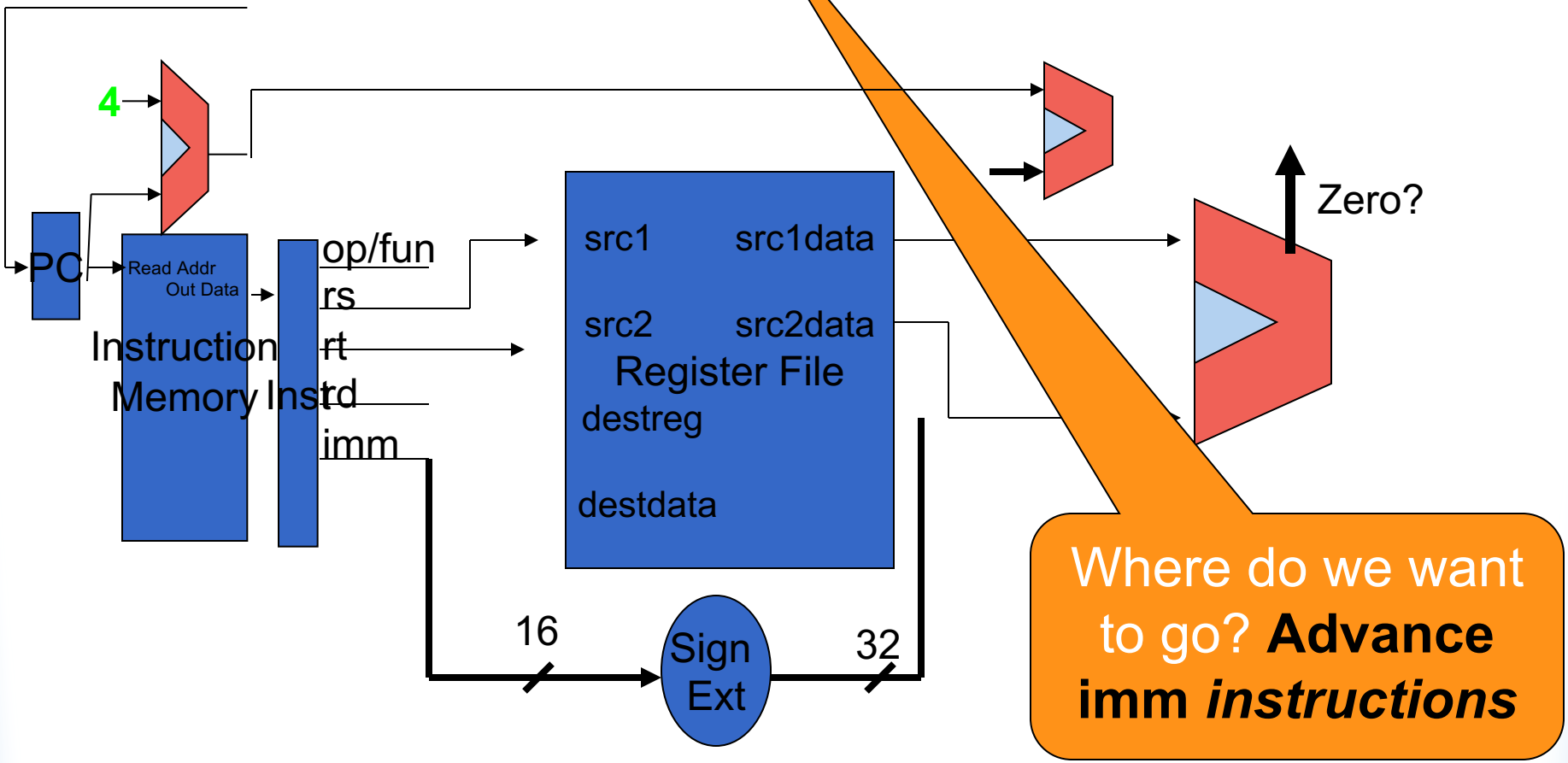
"beq" Instruction

Operation	rs	rt	imm	# meaning
beq $3,$5,lp	3	5	6	# if ($3 == $5) goto lp



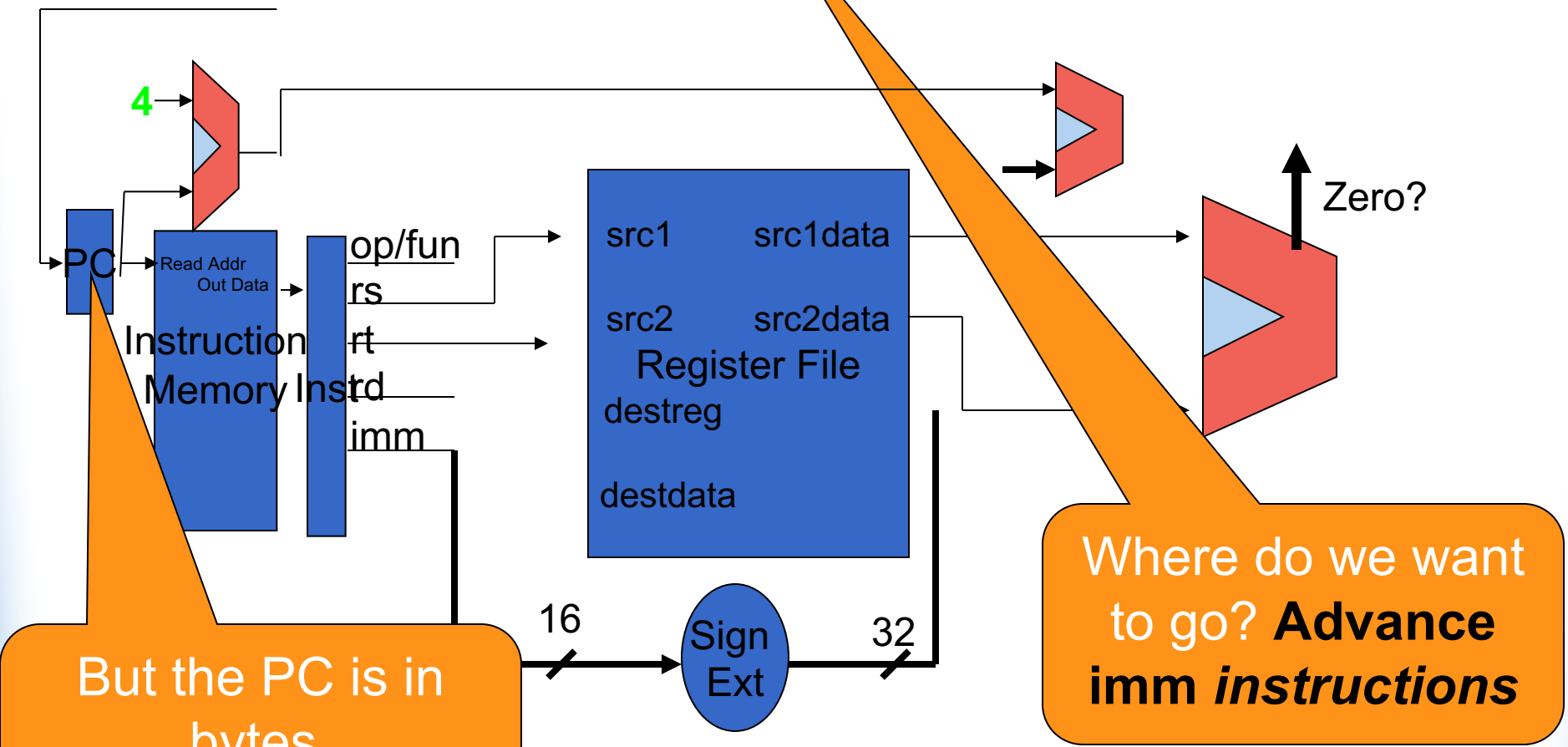
"beq" Instruction

Operation	rs	rt	imm	# meaning
beq $3,$5,lp	3	5	6	# if ($3 == $5) **goto lp**



"beq" Instruction

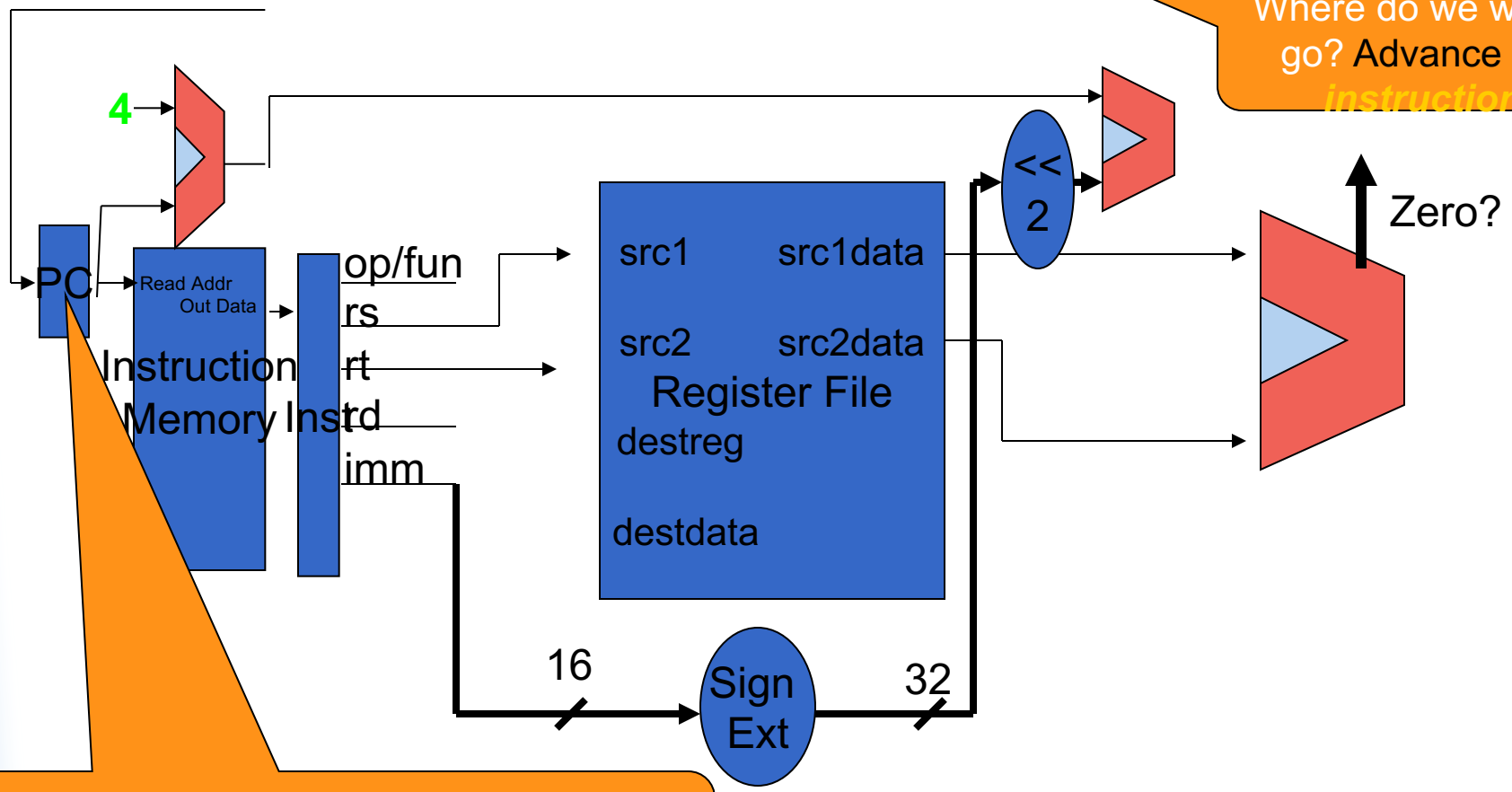
Operation	rs	rt	imm	# meaning
beq $3,$5,lp	3	5	6	# if ($3 == $5) goto lp



"beq" Instruction

Operation	rs	rt	imm	# meaning
beq $3,$5,lp	3	5	6	# if ($3 == $5) goto lp

Where do we want to go? Advance imm *instructions*

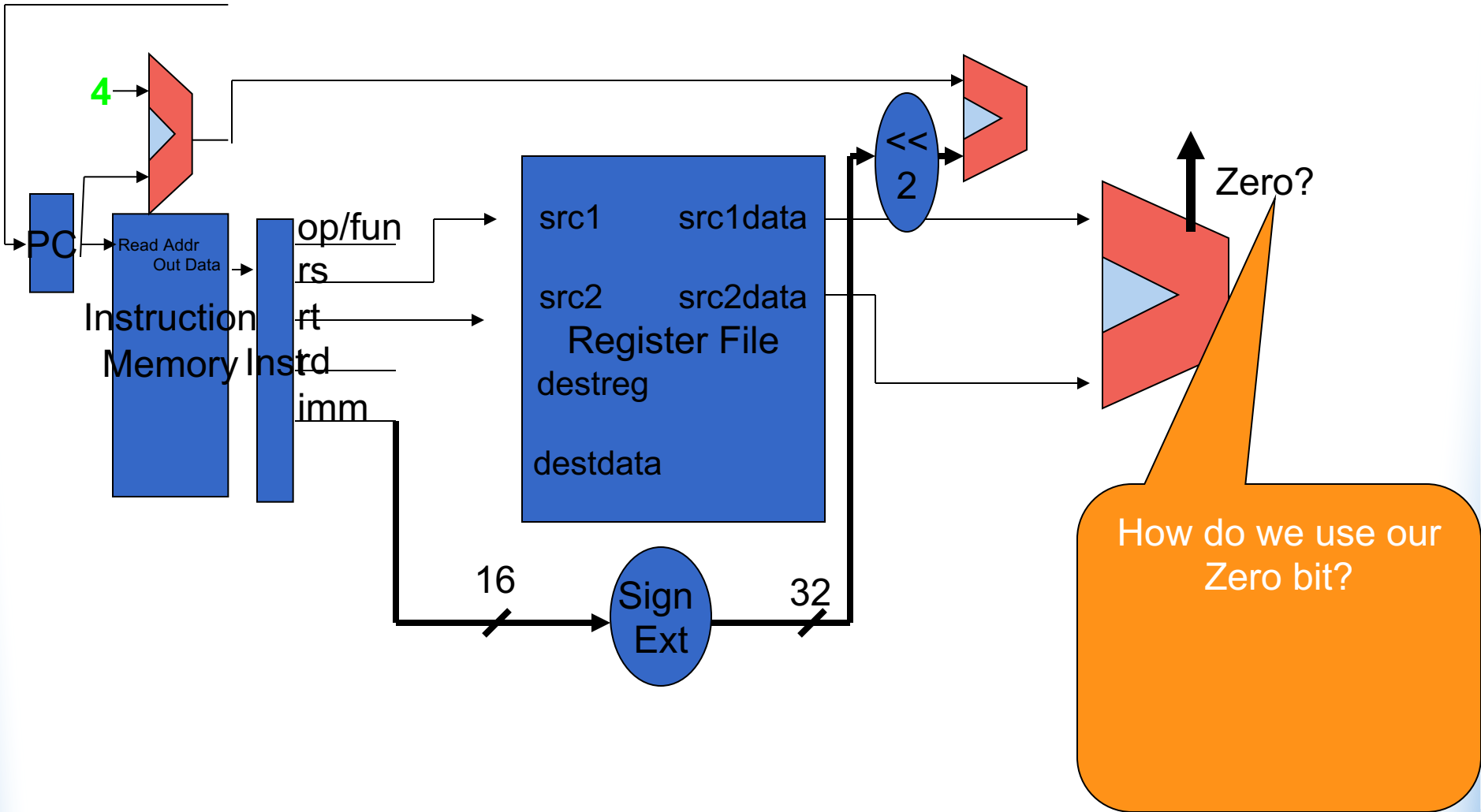


But the PC is in bytes.

PC = (PC + 4)+ Imm<<2

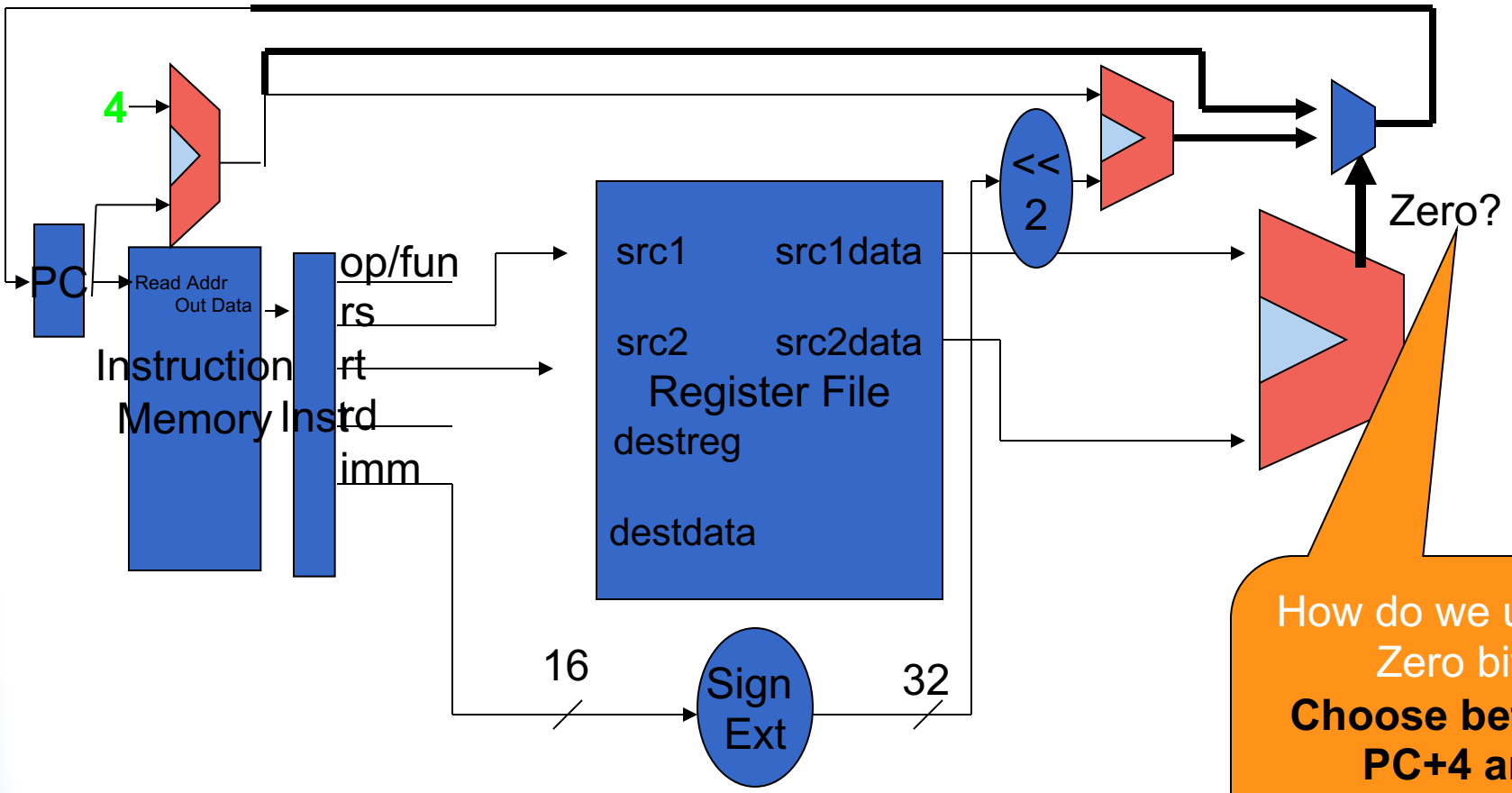
"beq" Instruction

Operation	rs	rt	imm	# meaning
beq $3,$5,lp	3	5	6	# if ($3 == $5) goto lp



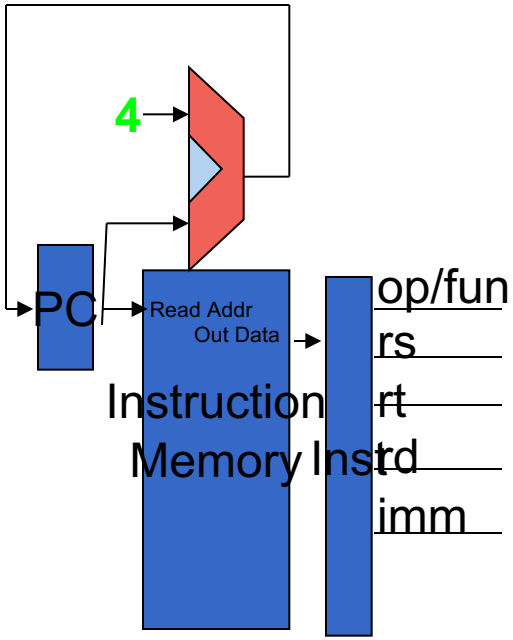
"beq" Instruction

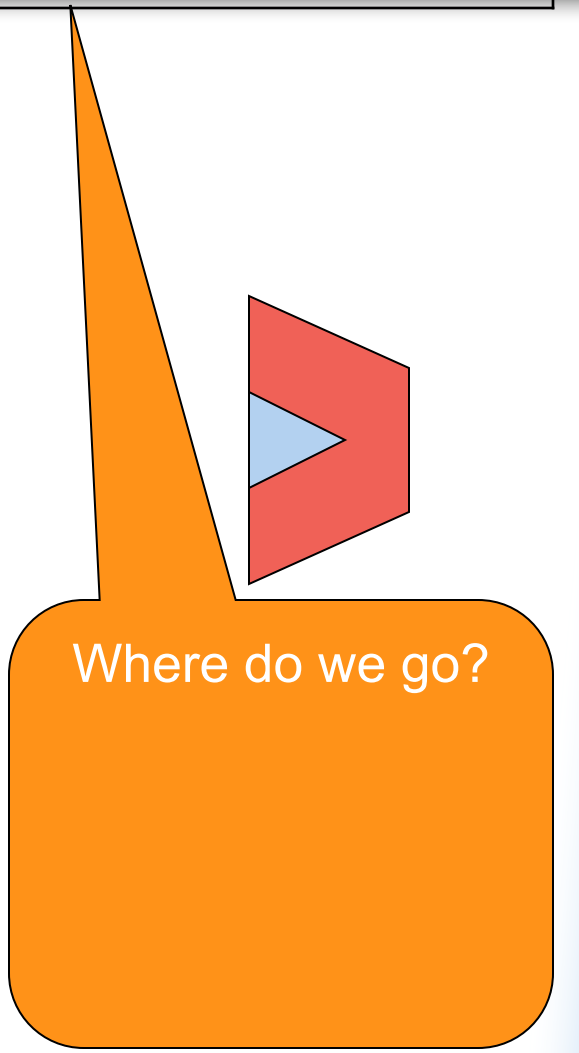
Operation	rs	rt	imm	# meaning
beq $3,$5,lp	3	5	6	# if ($3 == $5) goto lp



“j” Instruction

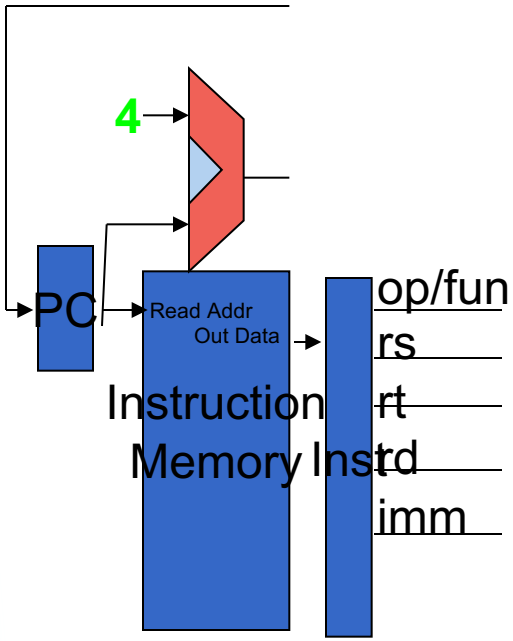
Operation	Target address	# meaning
j loop	0x0174837	# goto loop

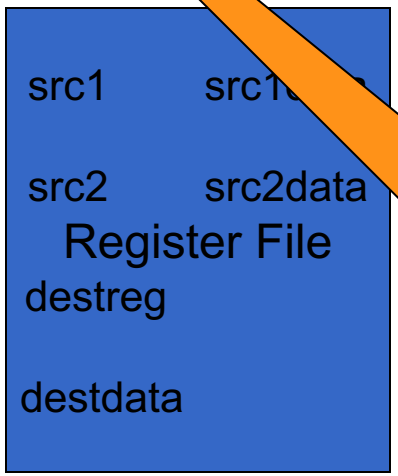


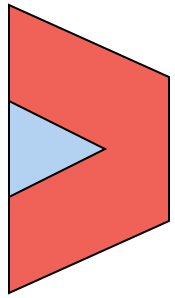


“j” Instruction

Operation	Target address	# meaning
j loop	0x0174837	# goto loop





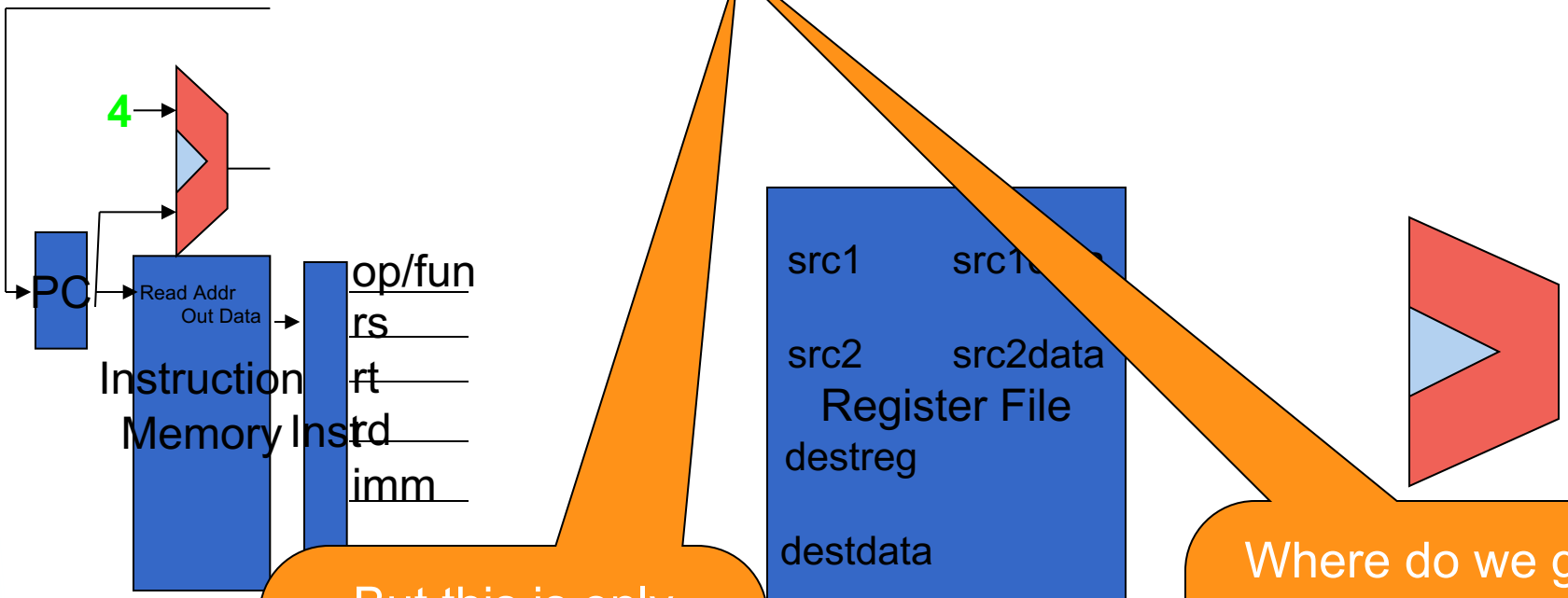


Where do we go?

To this absolute address

"j" Instruction

Operation	Target address	# meaning
j loop	0x0174837	# goto loop

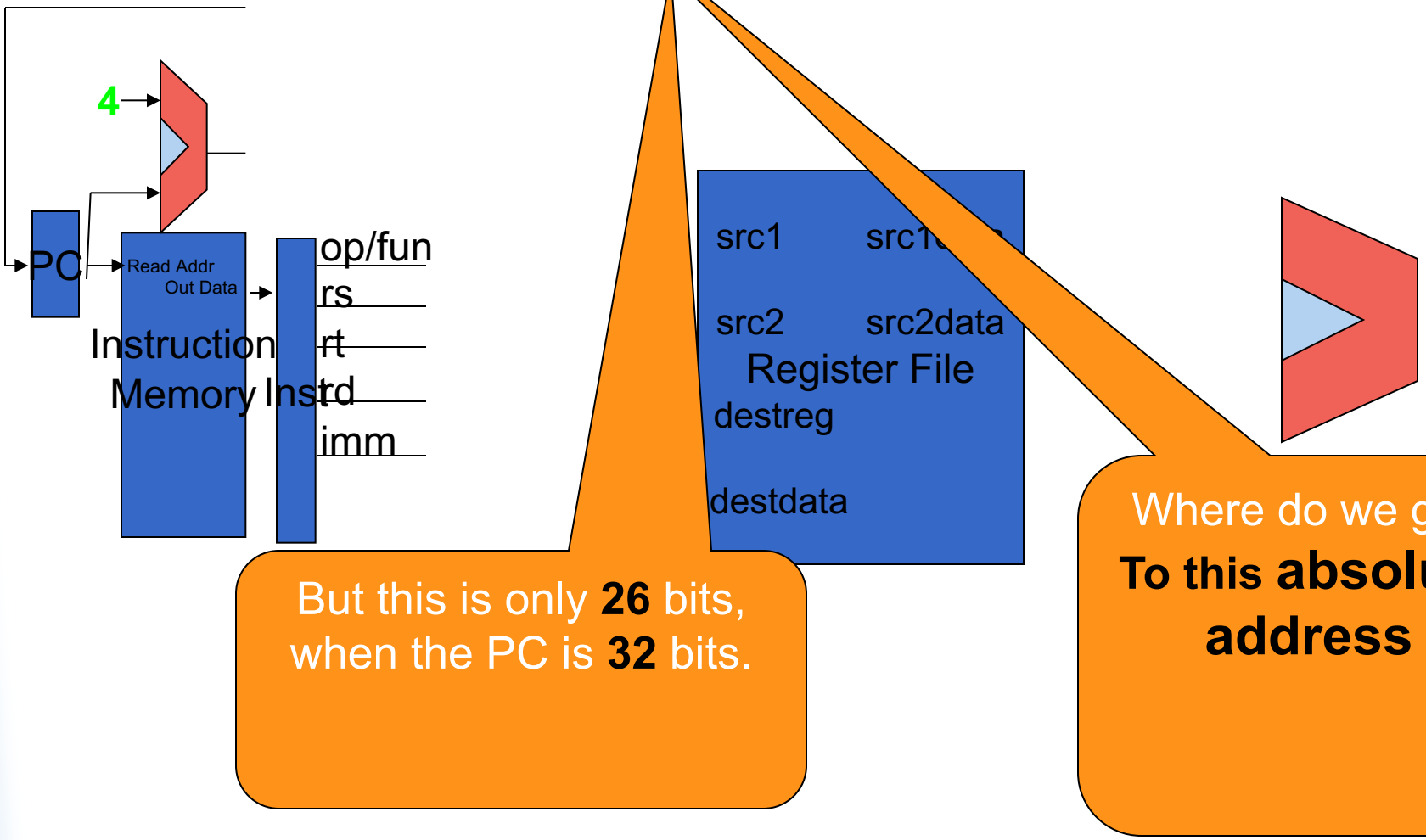


But this is only _____ bits, when the PC is _____ bits.

Where do we go?
To this absolute address

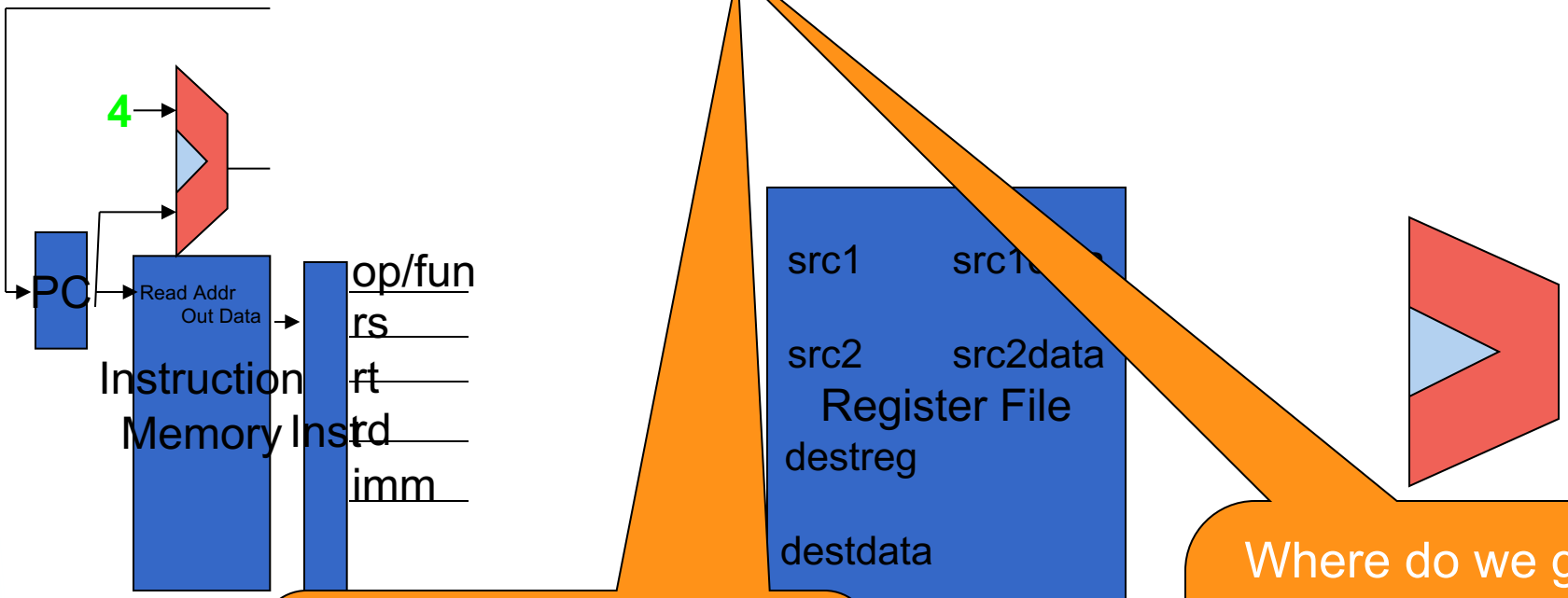
"j" Instruction

Operation	Target address	# meaning
j loop	0x0174837	# goto loop



"j" Instruction

Operation	Target address	# meaning
j loop	0x0174837	# goto loop

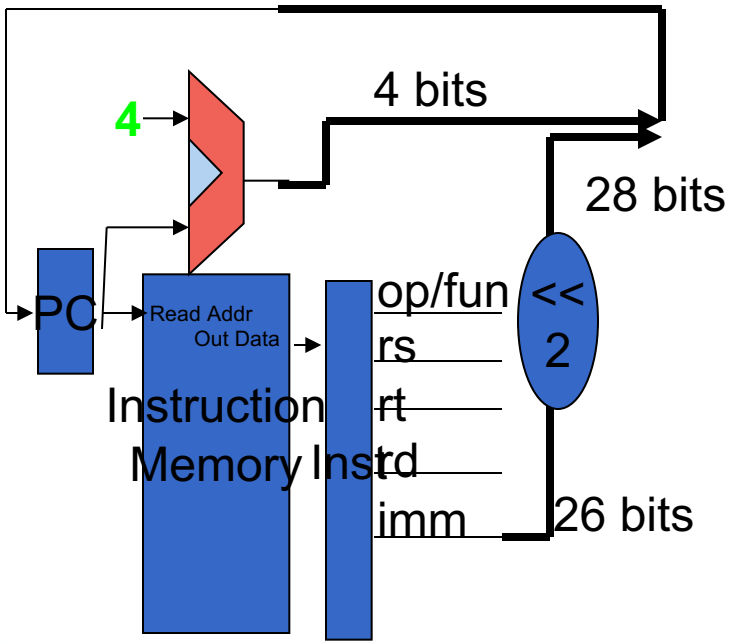


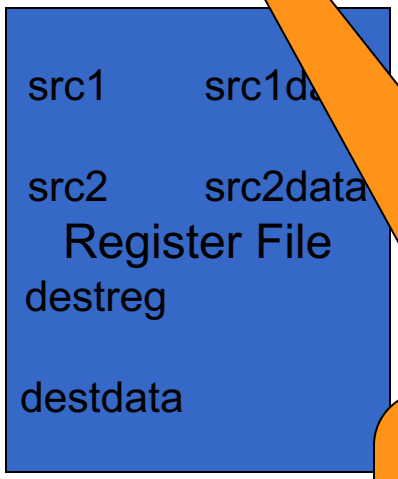
But this is only **26** bits, when the PC is **32** bits. **Shift left, Concatenate PC's upper bits**

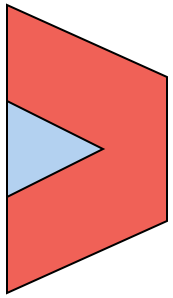
Where do we go? **To this absolute address**

“j” Instruction

Operation	Target address	# meaning
j loop	0x0174837	# goto loop

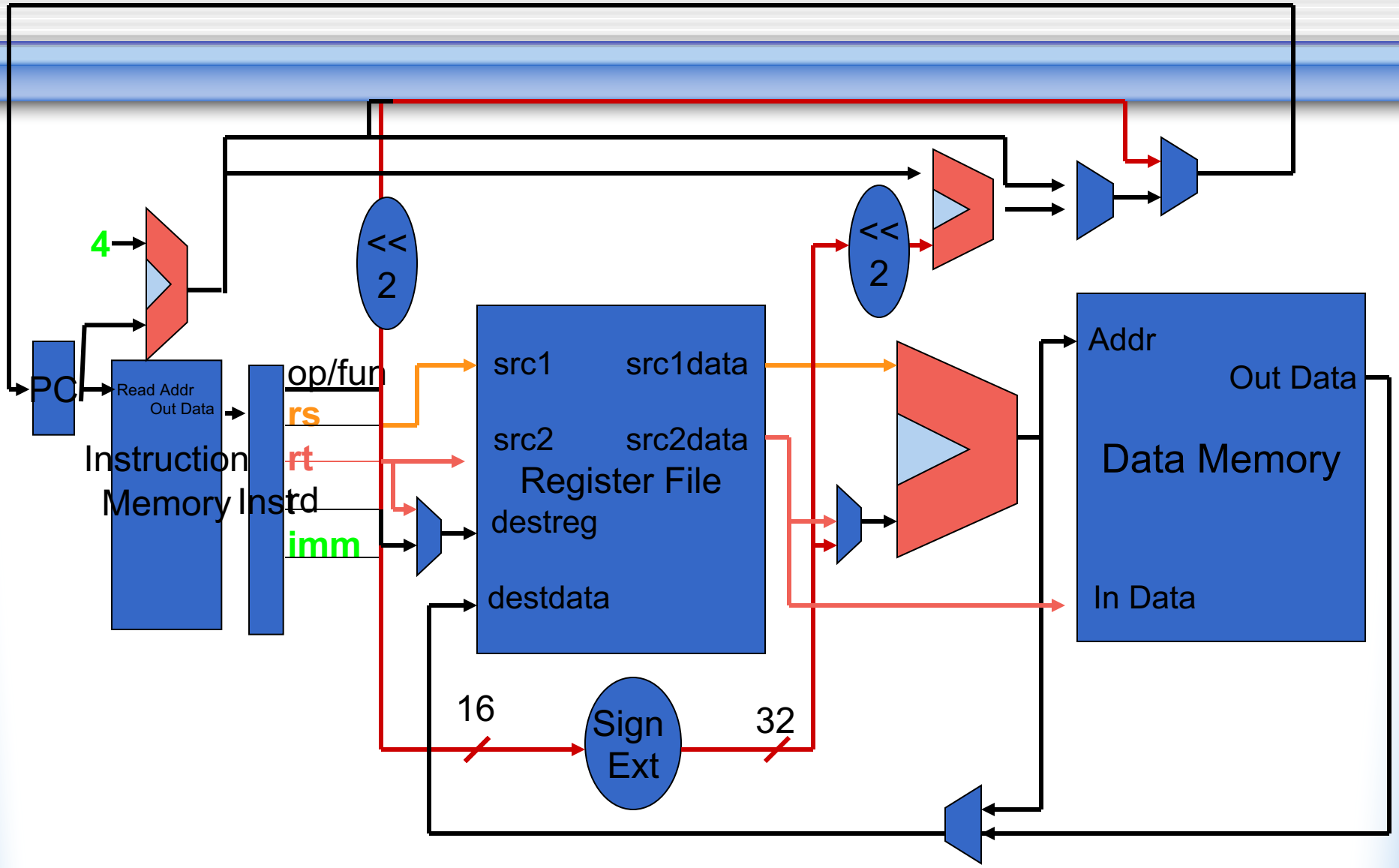






But this is only **26** bits, when the PC is **32** bits. **Shift left, Concatenate current PC’s upper bits**

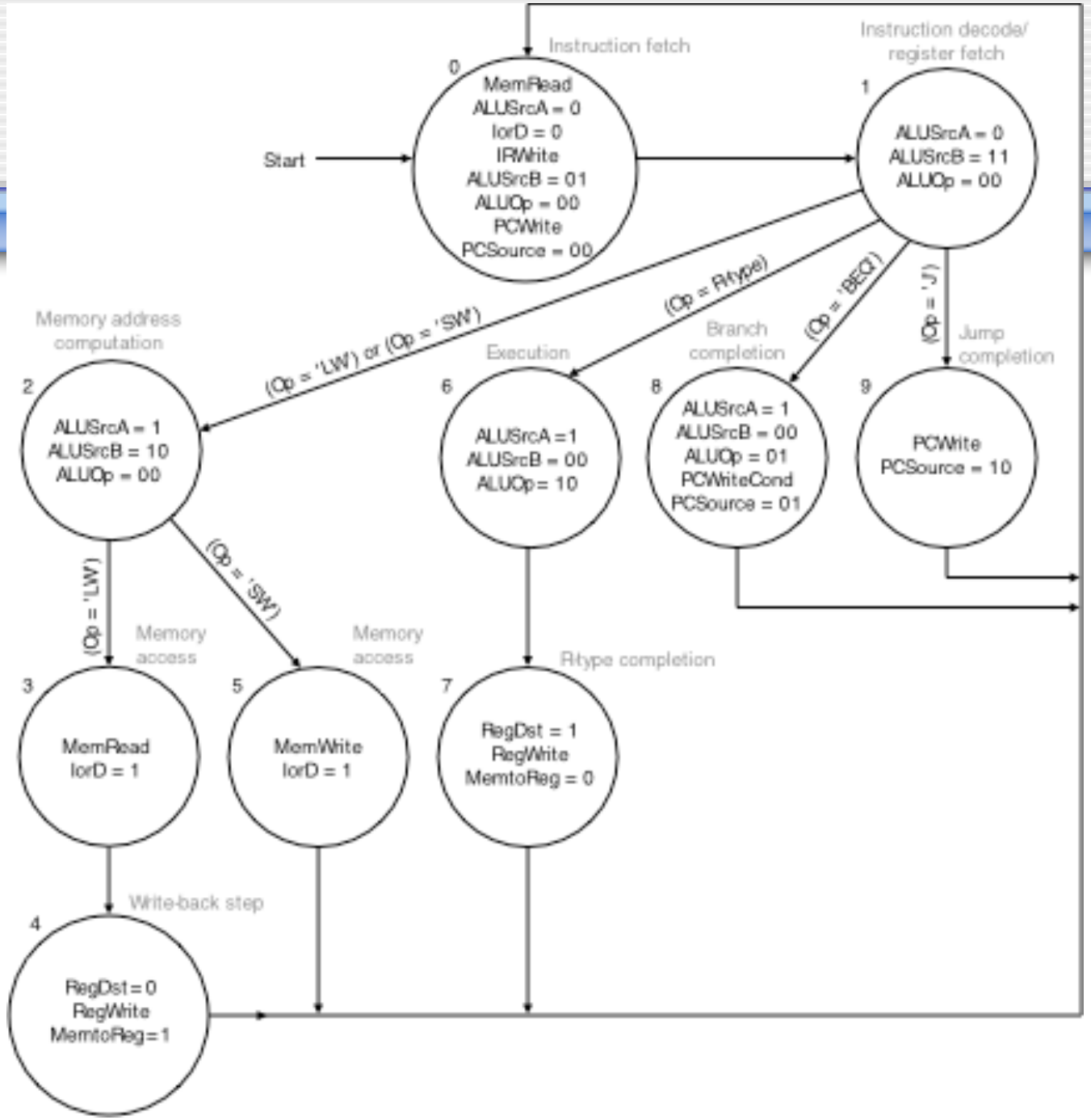
The Whole Shebang



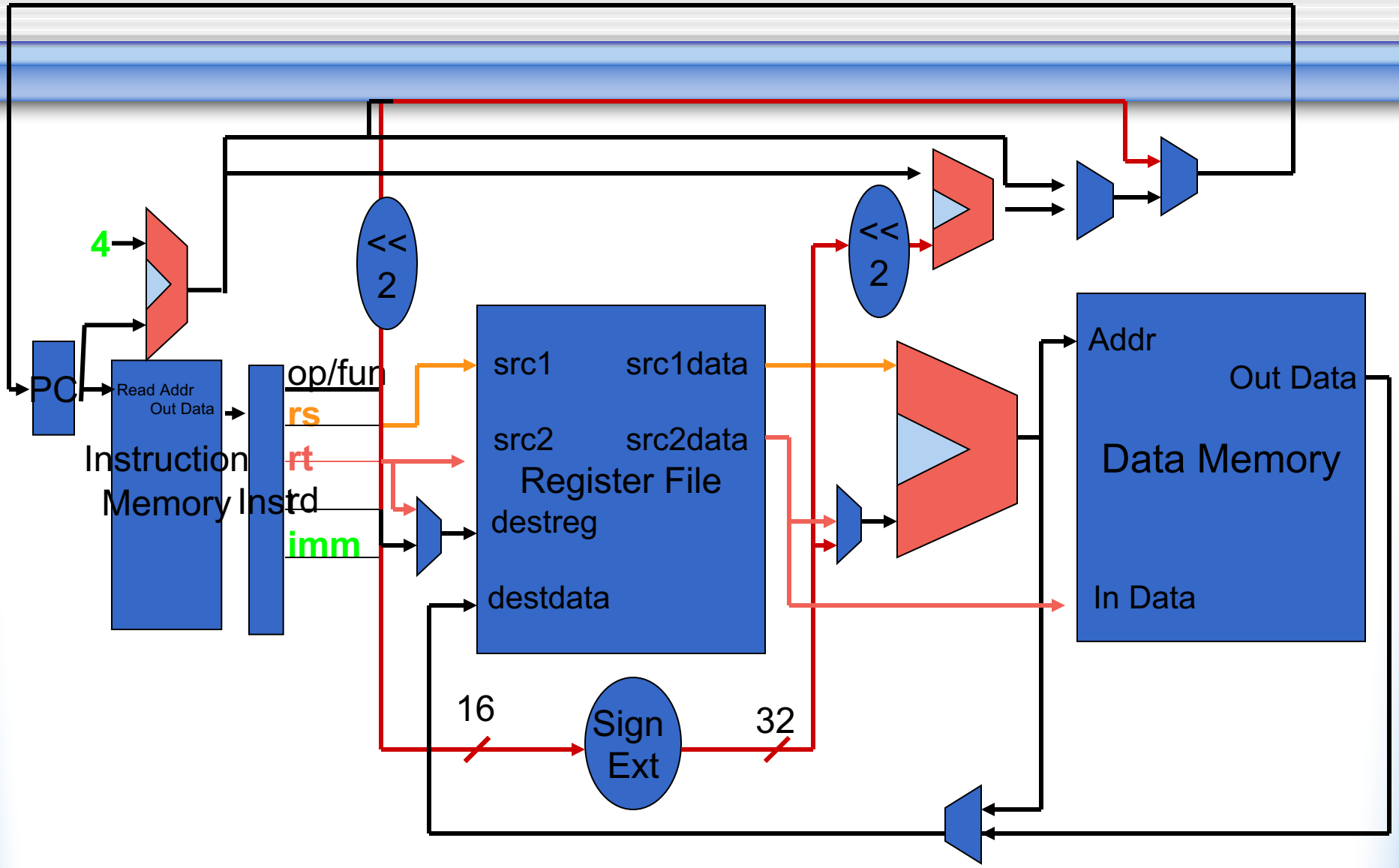
Control Unit

- Set of control line values cause appropriate actions to be taken at each step
- Finite state machine determines what needs to be done at each step
  - Fetch
  - Decode
  - Execute
  - Memory
  - Writeback

ACTIONS DEPEND ON OPCODE


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

Single Cycle Latency

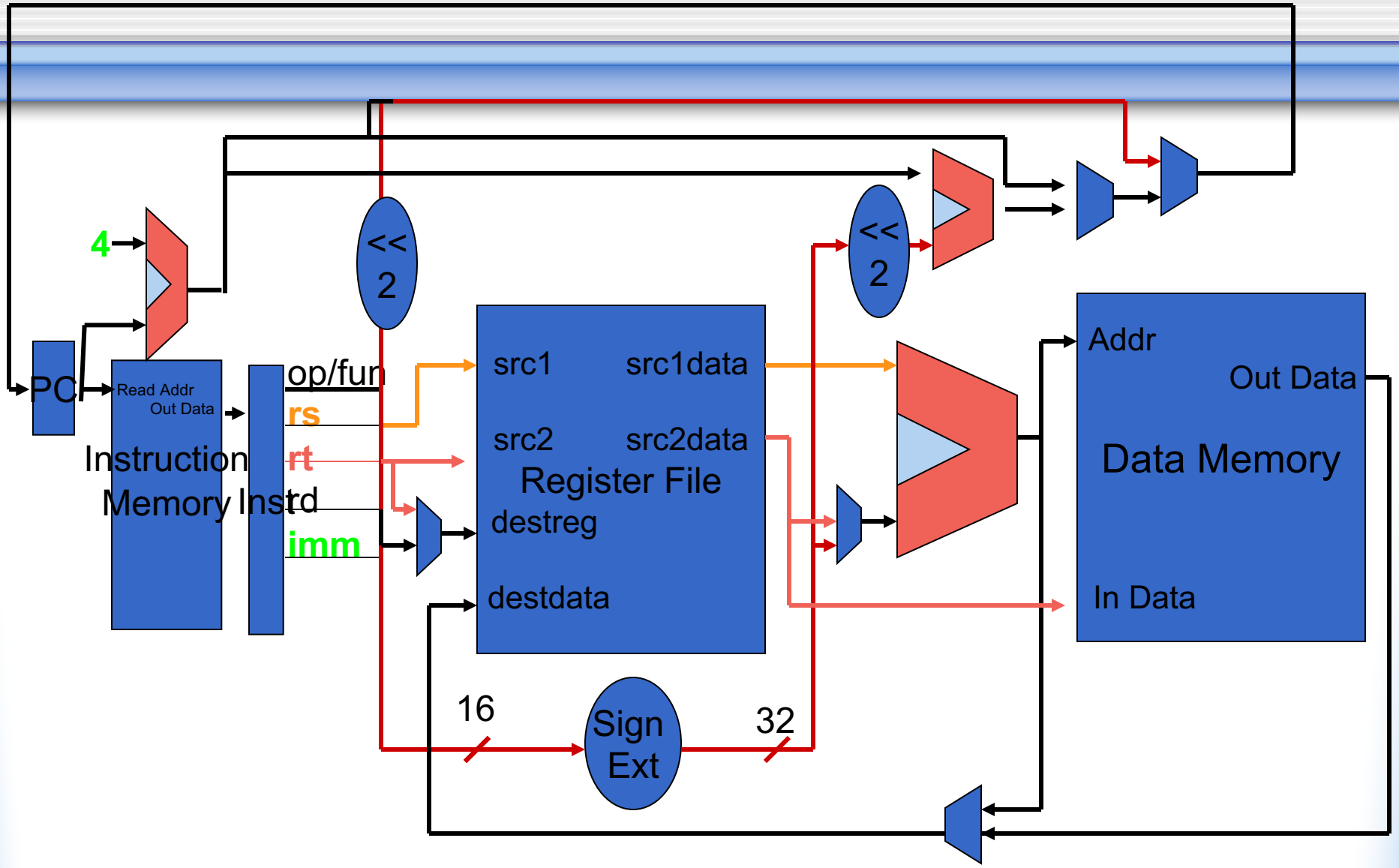


Time Diagram

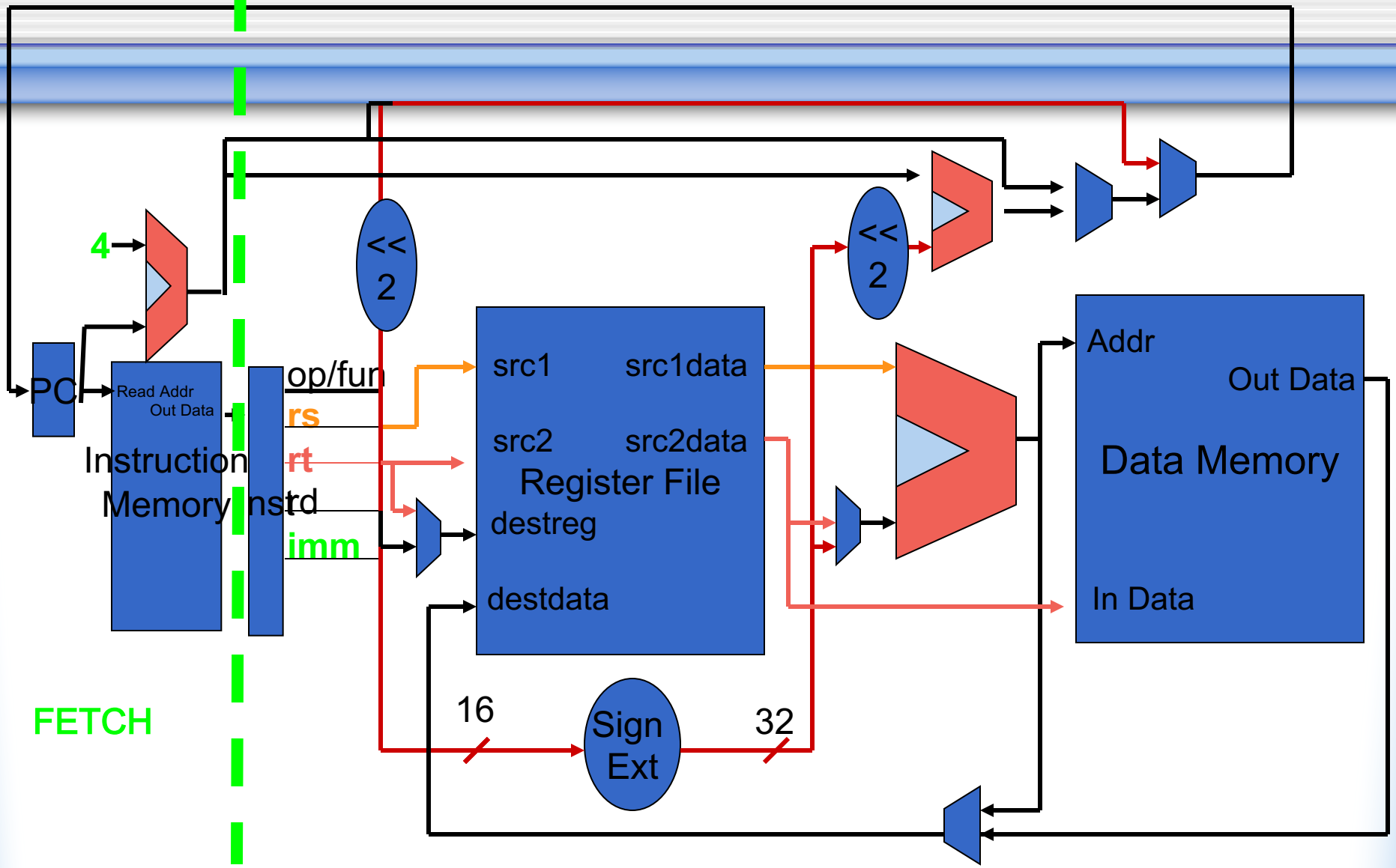
Cycle Time

- Not all instructions must go through all steps
  - add doesn't need to go to memory
- Single long clock cycle makes add take as long as load
- Can we change this?
  - Break single instruction execution into small execution steps

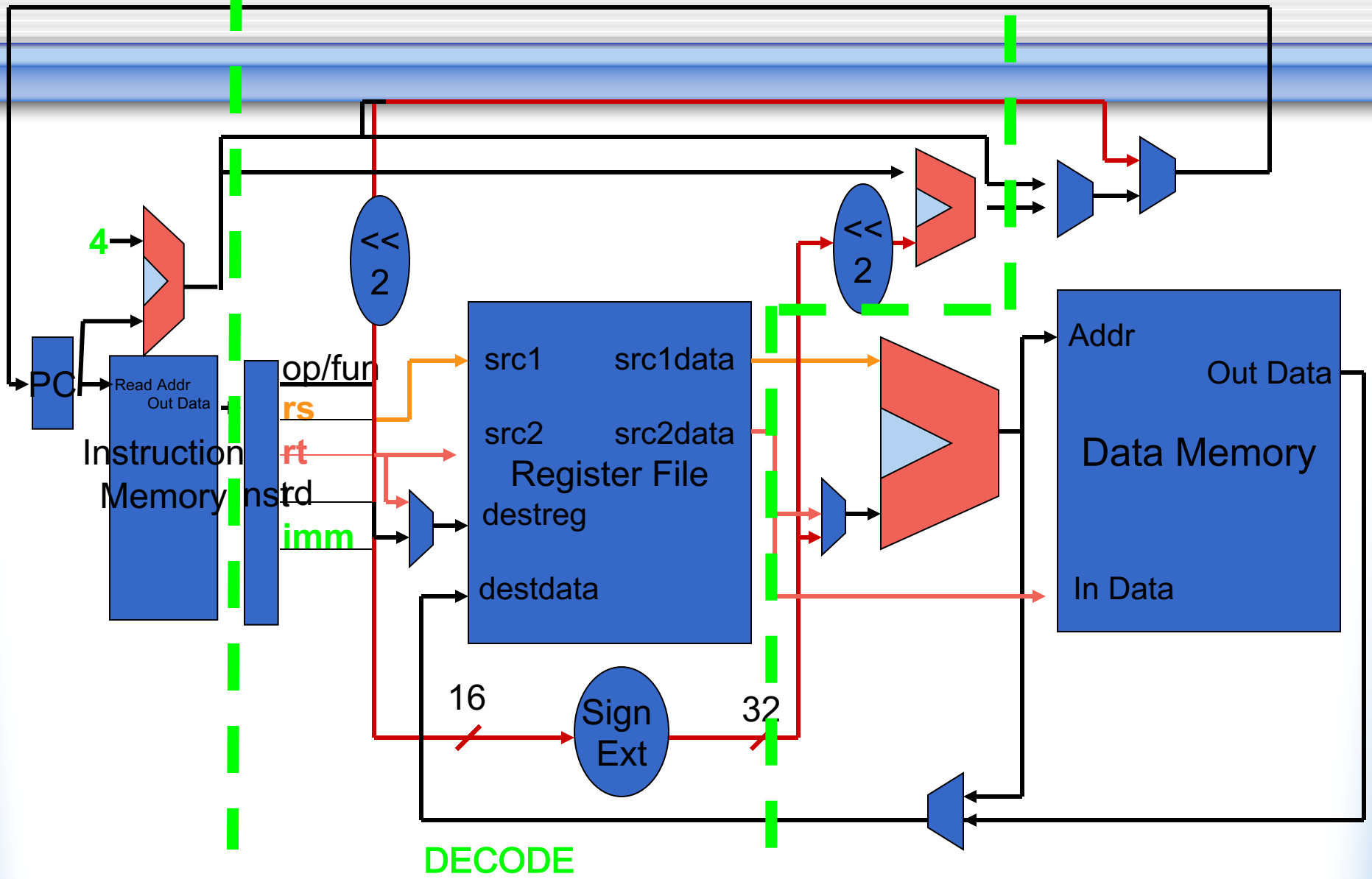
Five Cycle Implementation



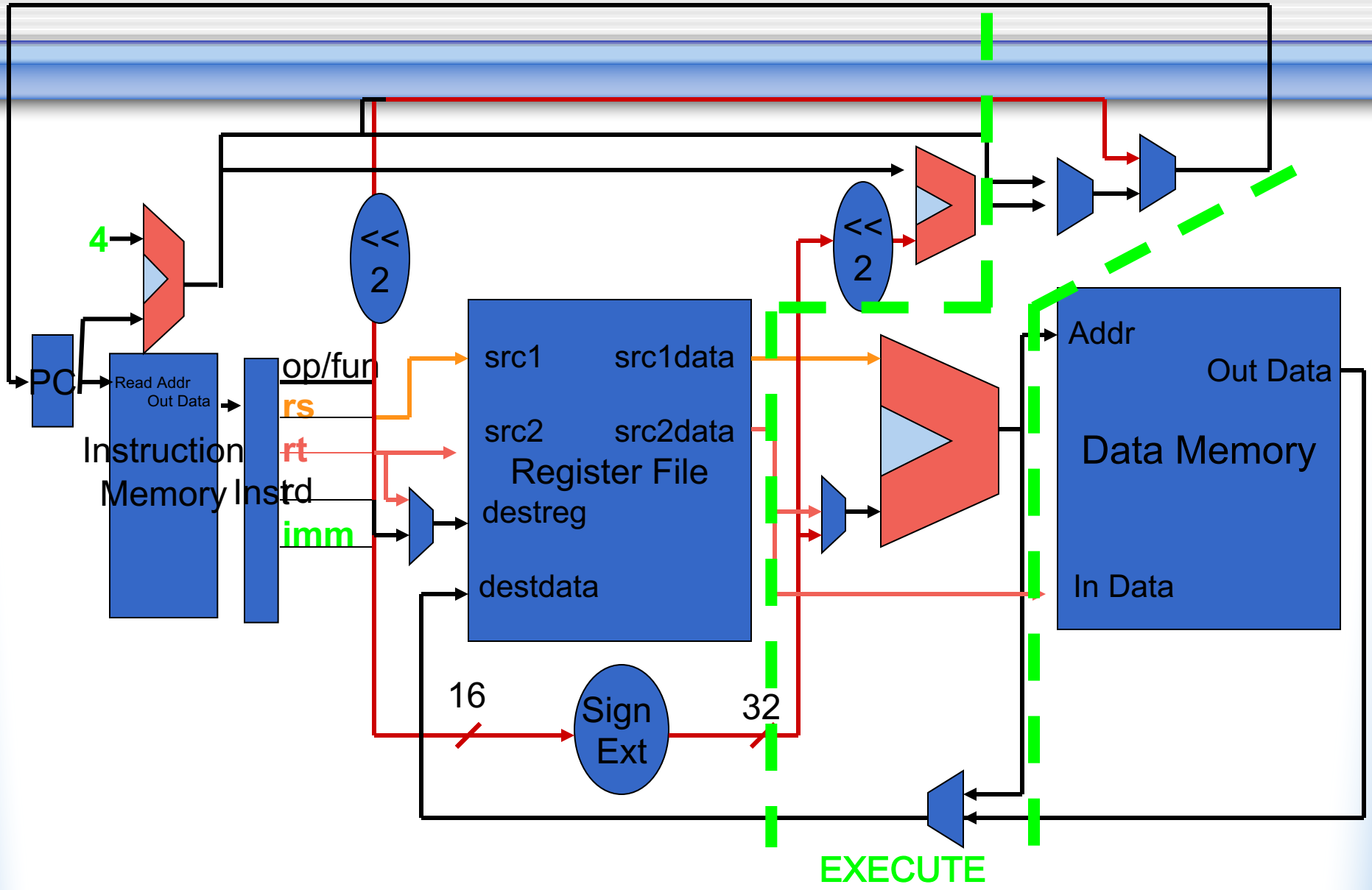
Five Cycle Implementation



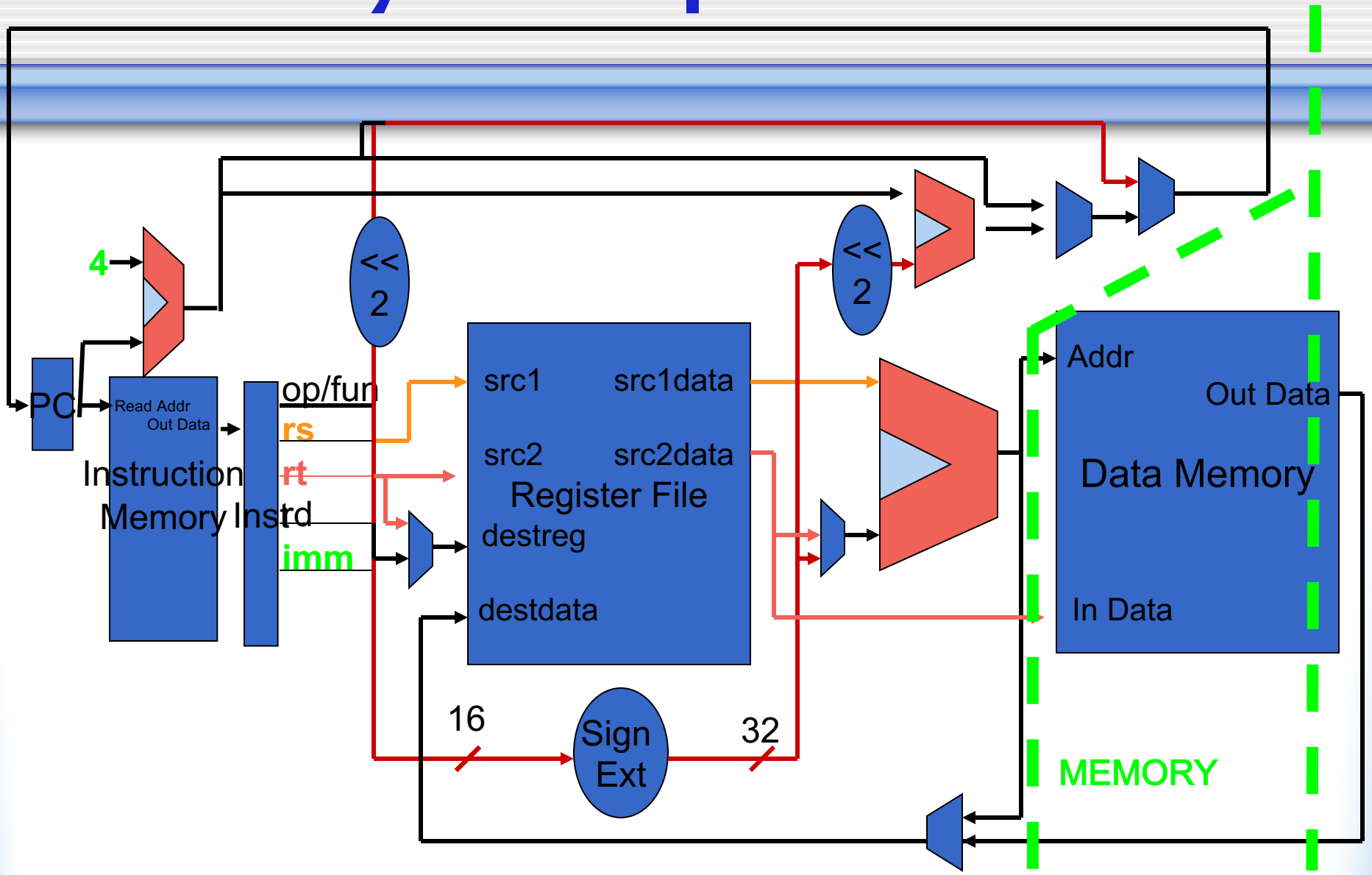
Five Cycle Implementation



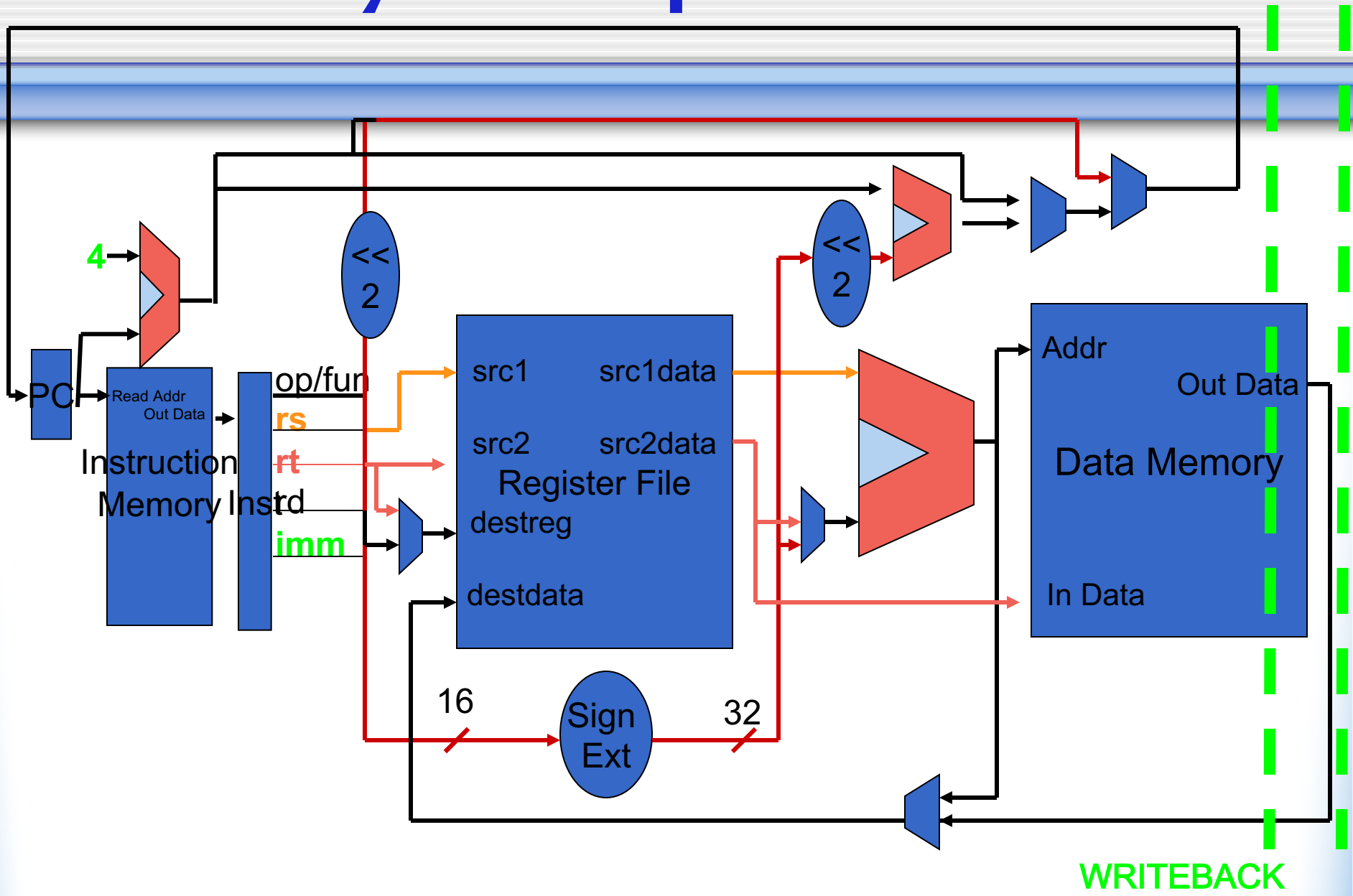
Five Cycle Implementation



Five Cycle Implementation



Five Cycle Implementation



How Many Cycles For:

- add
- sw
- lw
- blt
- j