Topic V28

Single-Cycle Datapath

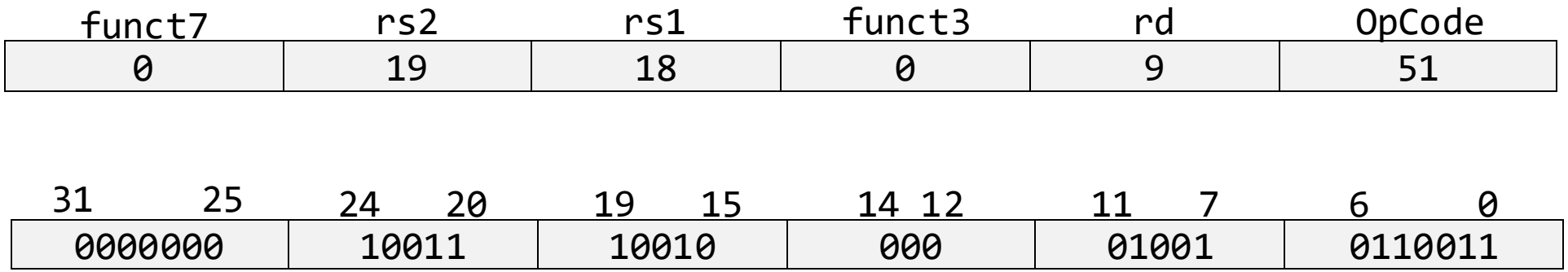
Reading: (Section 4.4)

R-Type Instructions

```
add x9, x18, x19
```

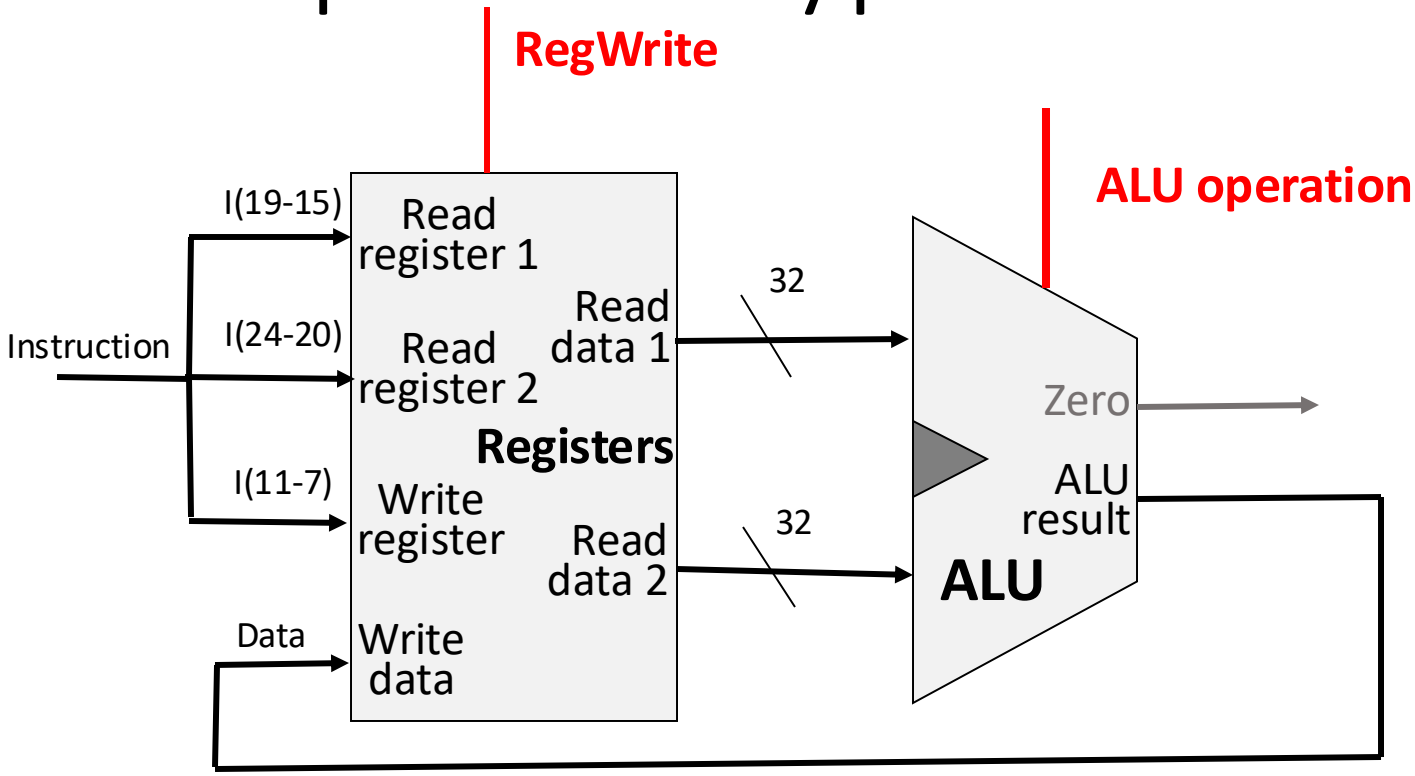
This instruction has 3 operands. The data that it operates on is stored in registers. It adds the content of registers x18 and x19 and stores the result in register x9. Its meaning can be summarized as follows:

```
x9 ← x18 + x19
```



R-Type Instruction Format

Datapath: R-Type Instruction




```
add x9, x18, x19
```

funct7	rs2	rs1	funct3	rd	OpCode
0	19	18	0	9	51

31 25	24 20	19 15	14 12	11 7	6 0
0000000	10011	10010	000	01001	0110011

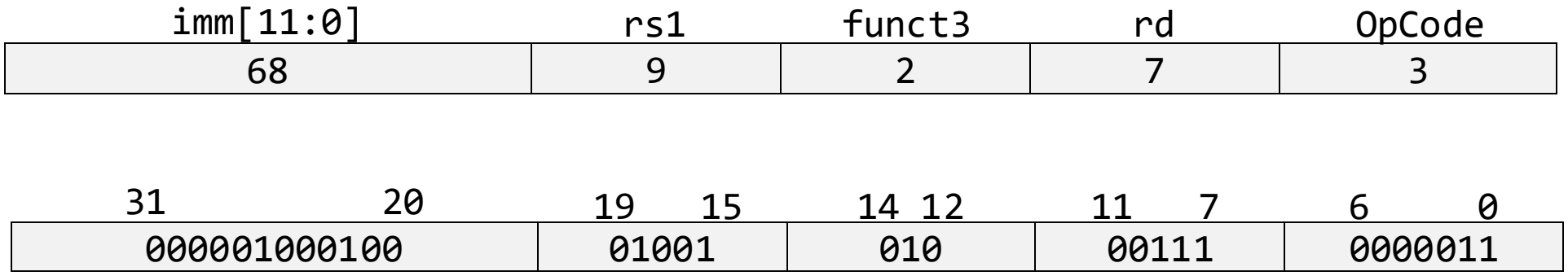
R-Type Instruction Format

Memory Instructions (load)

```
lw x7, 68(x9)
```

Read the memory location addressed by the value in register x9 plus 68 and store the value in register x7
Its meaning can be summarized as follows:

```
x7 ← Memory[x9 + 68]
```



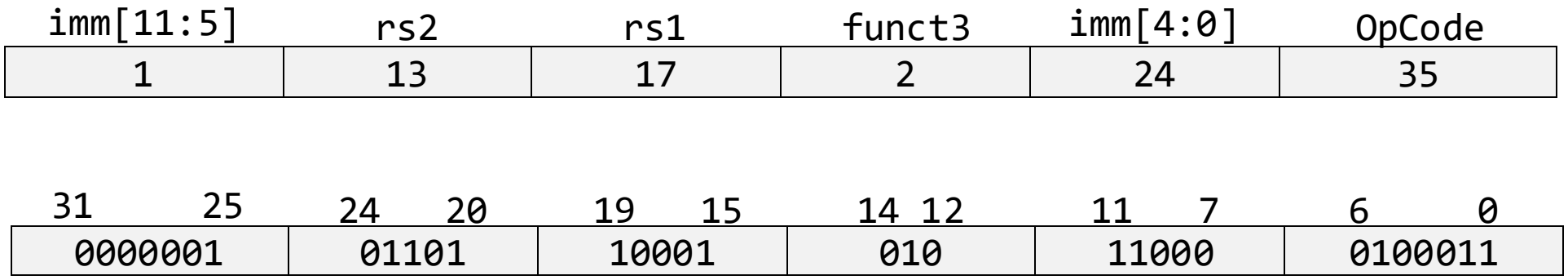
I-Type Instruction Format

Memory Instructions (store)

`sw x13, 56(x17)`

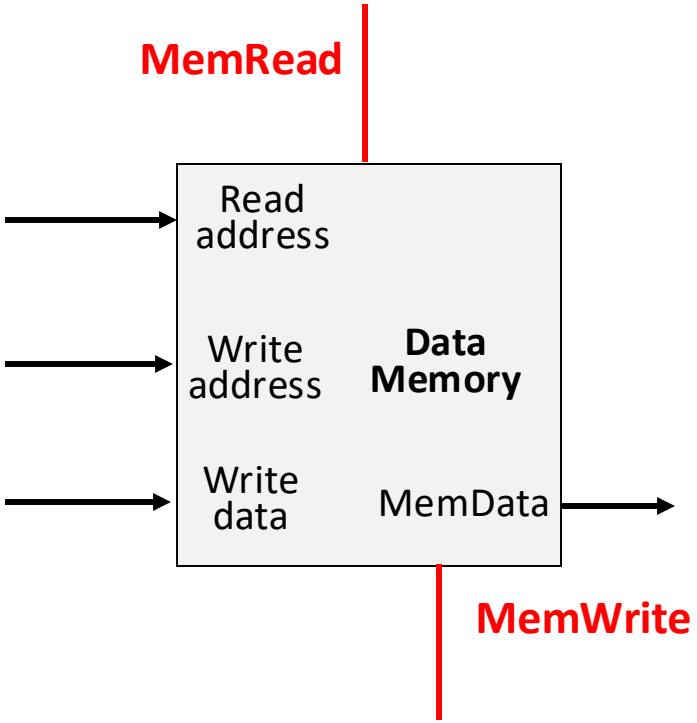
Write the value currently in register x13 in the memory location addressed by the value in register x17 plus 56
Its meaning can be summarized as follows:

`Memory[x17 + 56] ← x13`

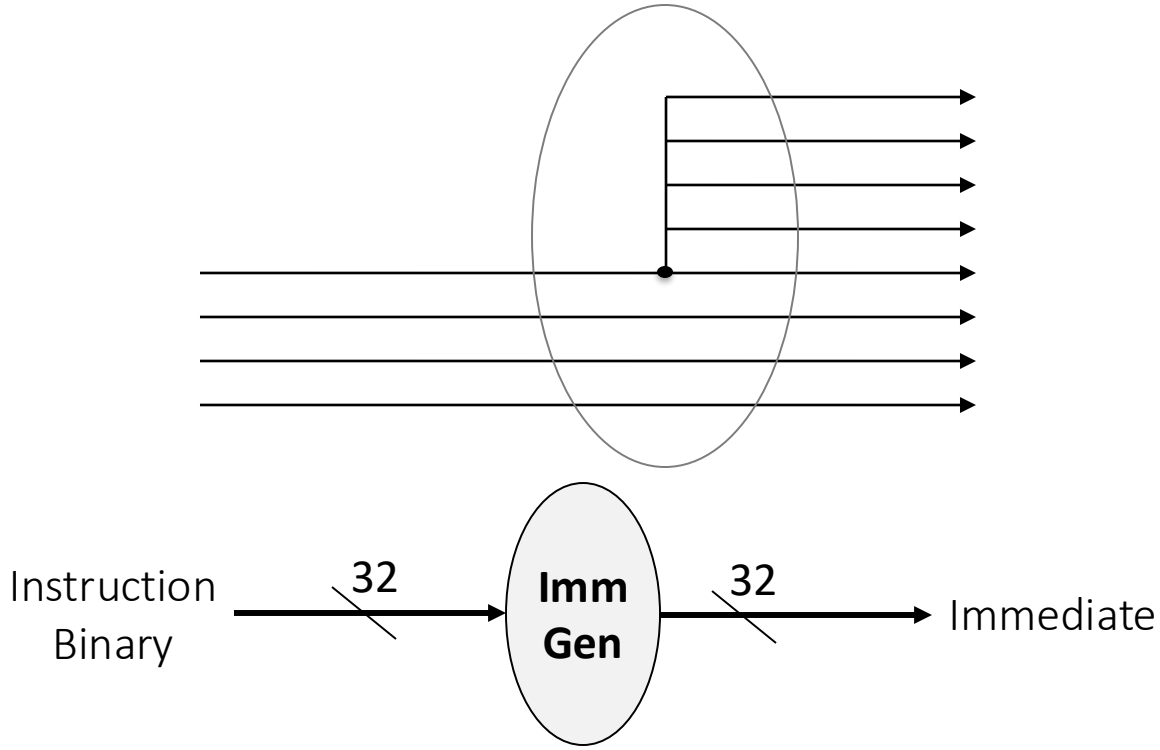


S-Type Instruction Format

DataPath: Load/Store Instructions



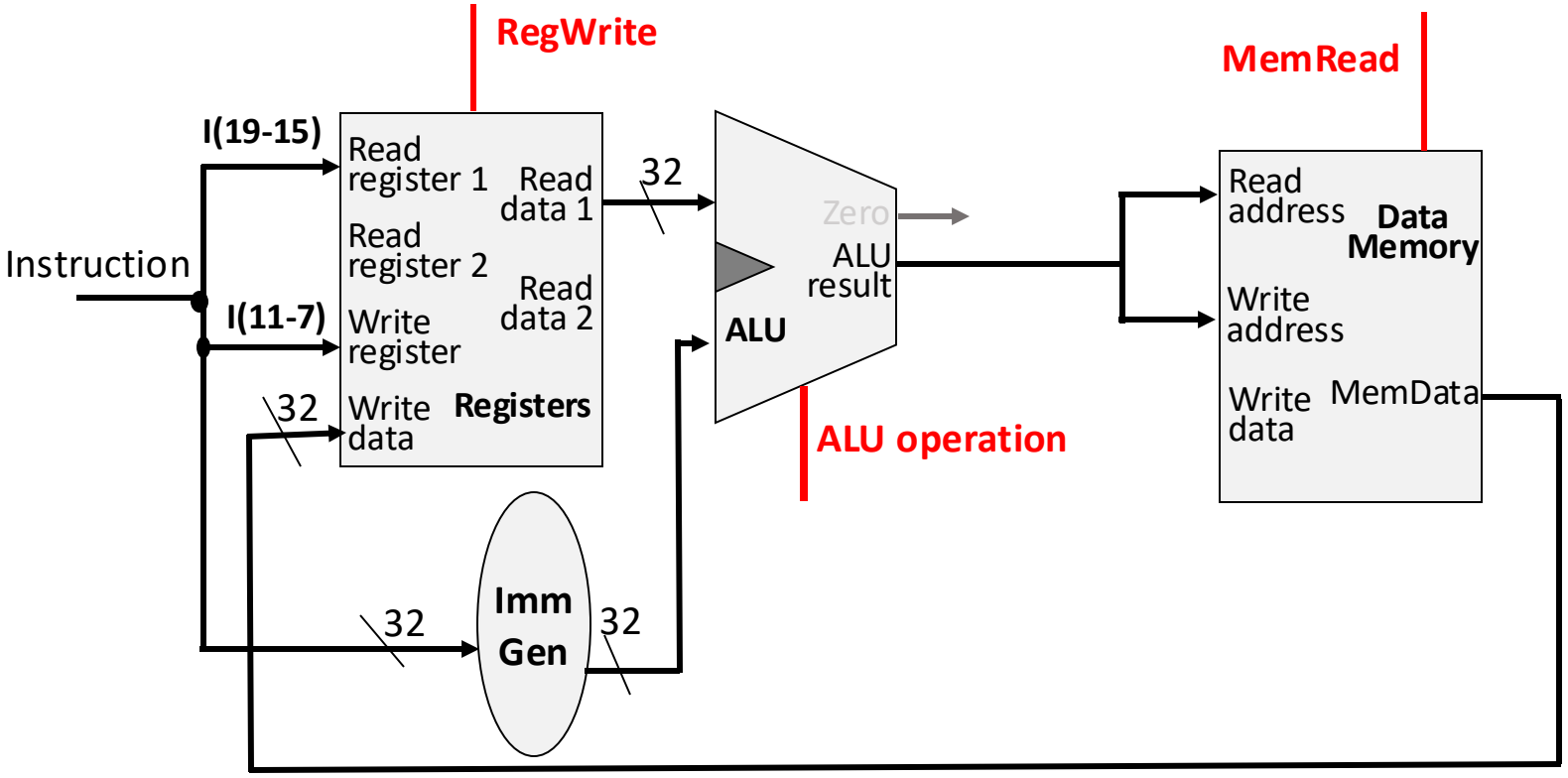
Data Memory Unit



Immediate Generation Unit

The immediate generation unit (ImmGen) has a 32-bit instruction as input that selects a 12-bit field for load, store, branch that is sign-extended into a 32-bit result appearing on the output.

DataPath: Load/Store Instructions

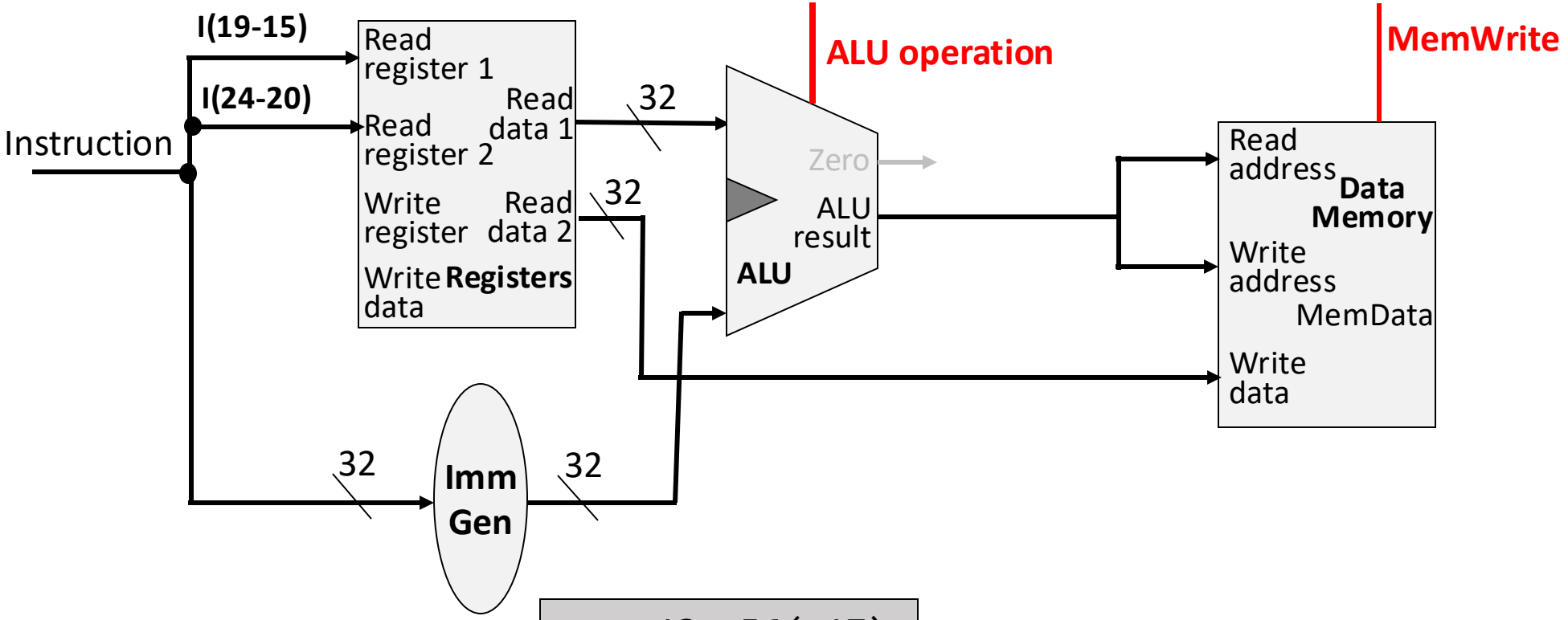


`lw x7, 68(x9)`

imm[11:0]	rs1	funct3	rd	OpCode
68	9	2	7	3

31 20	19 15	14 12	11 7	6 0
000001000100	01001	010	00111	0000011

DataPath: Load/Store Instructions

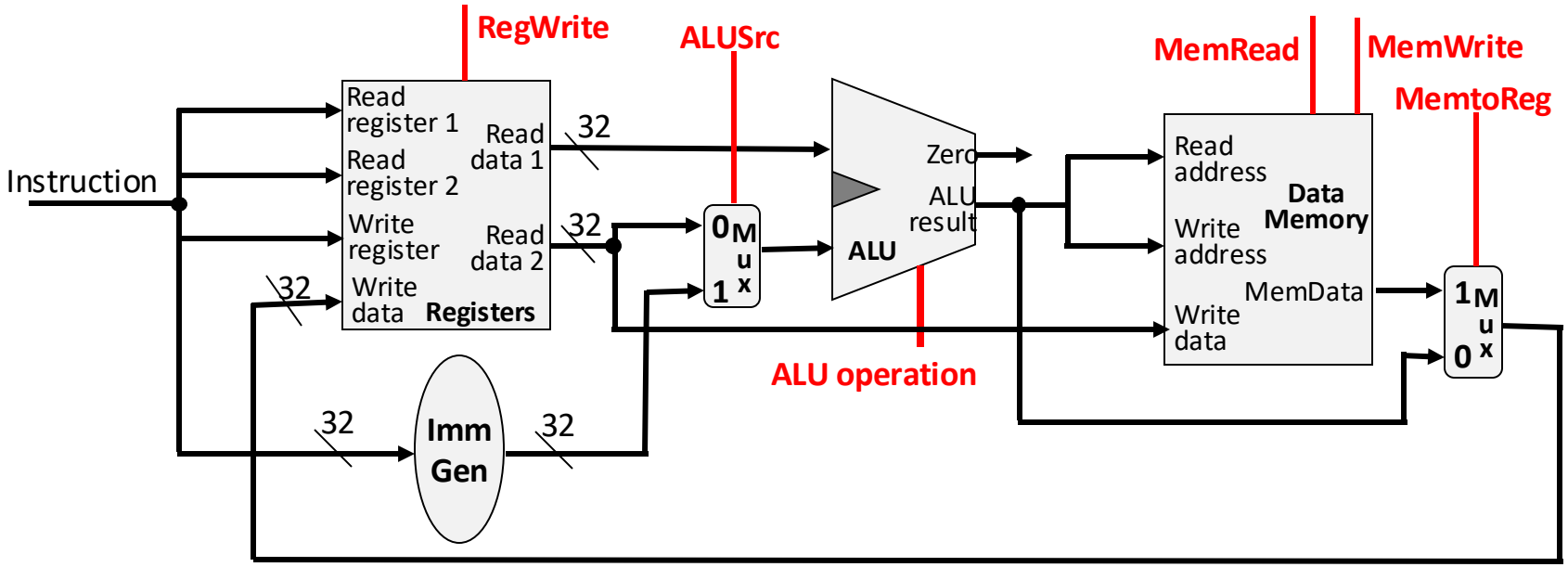



```
sw x13, 56(x17)
```

imm[11:5]	rs2	rs1	funct3	imm[4:0]	OpCode
1	13	17	2	24	35

31 25	24 20	19 15	14 12	11 7	6 0
0000001	01101	10001	010	11000	0100011

Combining Memory and Register Instr. Datapath's



`sw x13, 56(x17)`

imm[11:5]	rs2	rs1	funct3	imm[4:0]	OpCode
1	13	17	2	24	35

`add x9, x18, x19`

funct7	rs2	rs1	funct3	rd	OpCode
0	19	18	0	9	51

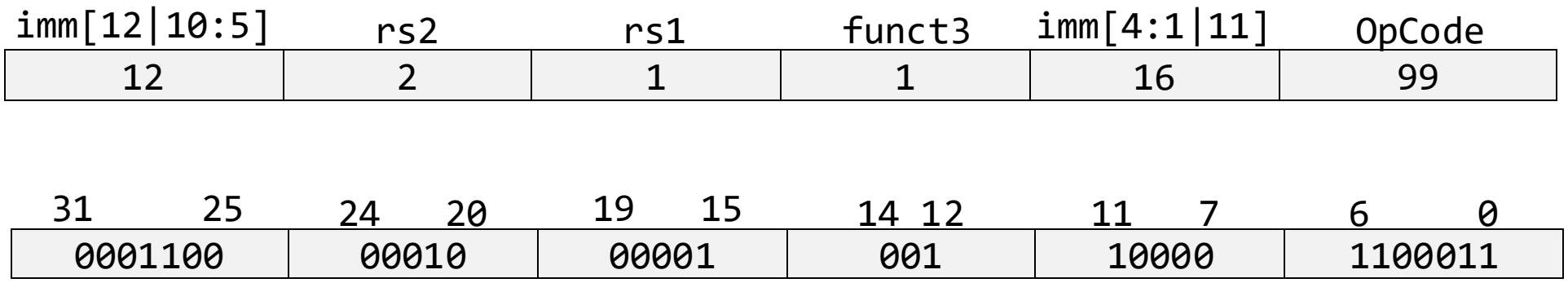
Branch Instructions

Branch Instructions

```
bne x1, x2, 400
```

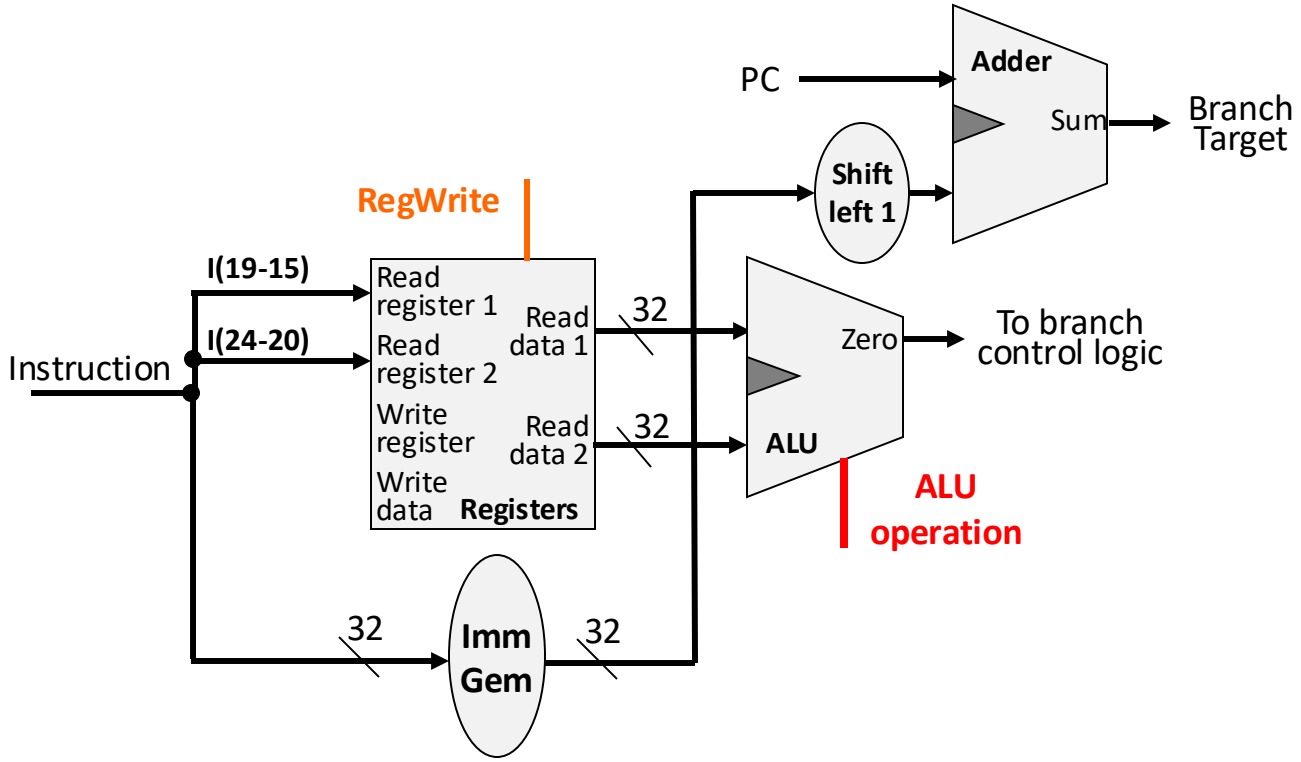
If the value in register x1 is not equal the value in register x2,
then add 100 to the PC before fetching the next instruction.
Its meaning can be summarized as follows:

```
if(x1 ≠ x2)  PC ← PC + (400)
{400 = {imm, 1'b0}}
```



SB-Type Instruction Format

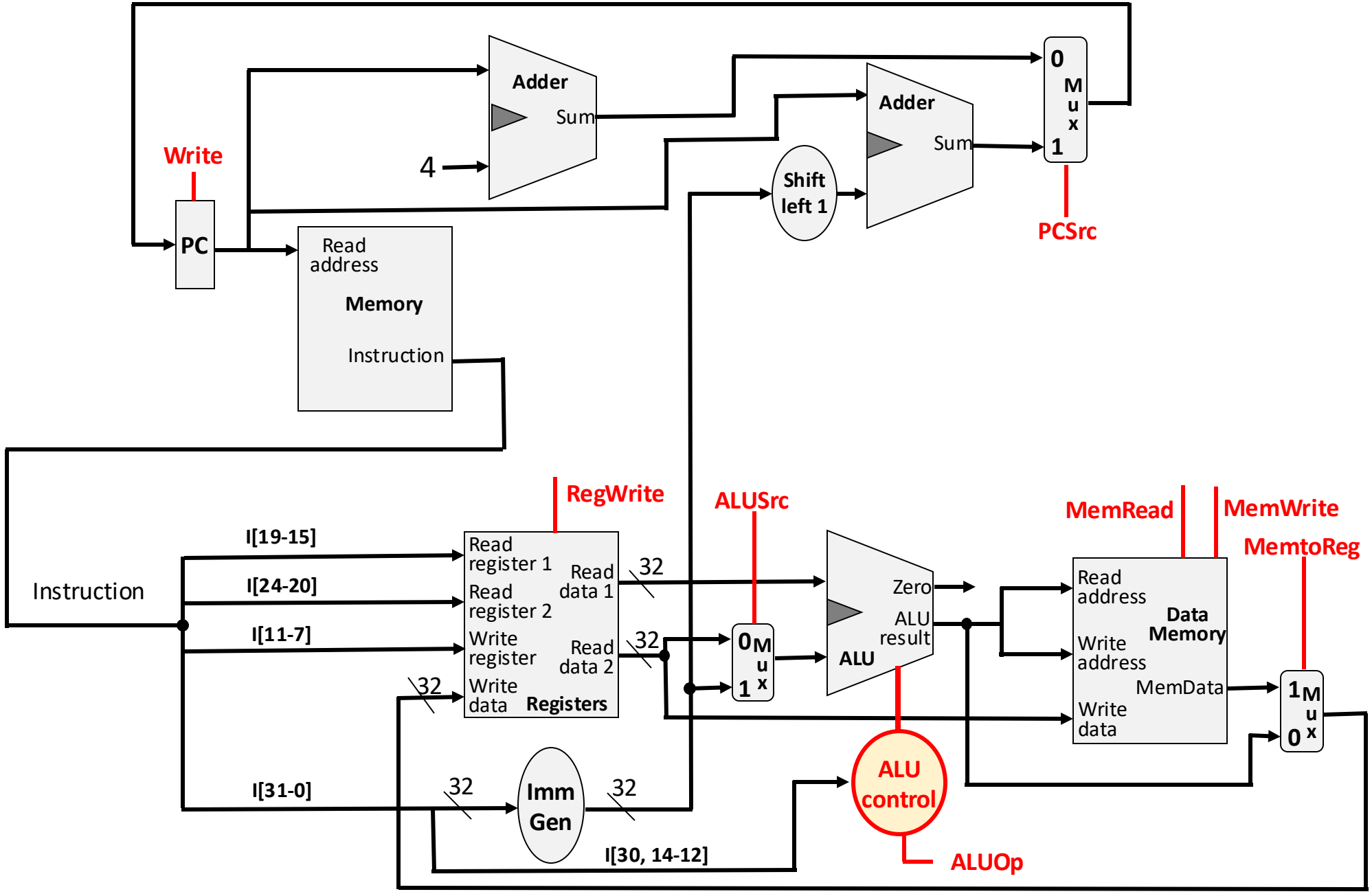
DataPath: Branch Instructions



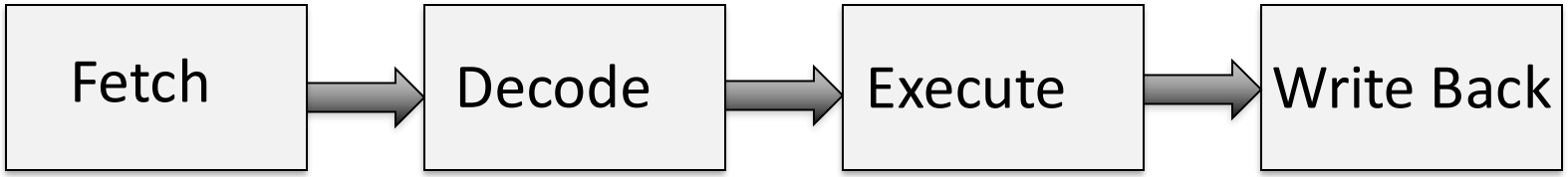

```
bne x1, x2, 400
```

imm[12|10:5]	rs2	rs1	funct3	imm[4:1|11]	OpCode
12	2	1	1	16	99

31 25	24 20	19 15	14 12	11 7	6 0
0001100	00010	00001	001	10000	1100011


Write
PC
Adder
4
Sum
Adder
Sum
Shift left 1
0
M
u
x
1
PCSrc
Read address
Memory
Instruction
Instruction
I[19-15]
I[24-20]
I[11-7]
I[31-0]
RegWrite
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
32
32
32
ALUSrc
0
M
u
x
1
Zero
ALU result
ALU
MemRead
MemWrite
MemtoReg
Read address
Write address
Write data
Data Memory
MemData
1
M
u
x
0
Imm Gen
32
32
I[30, 14-12]
ALU control
ALUOp

Four Steps of an R-type Instruction

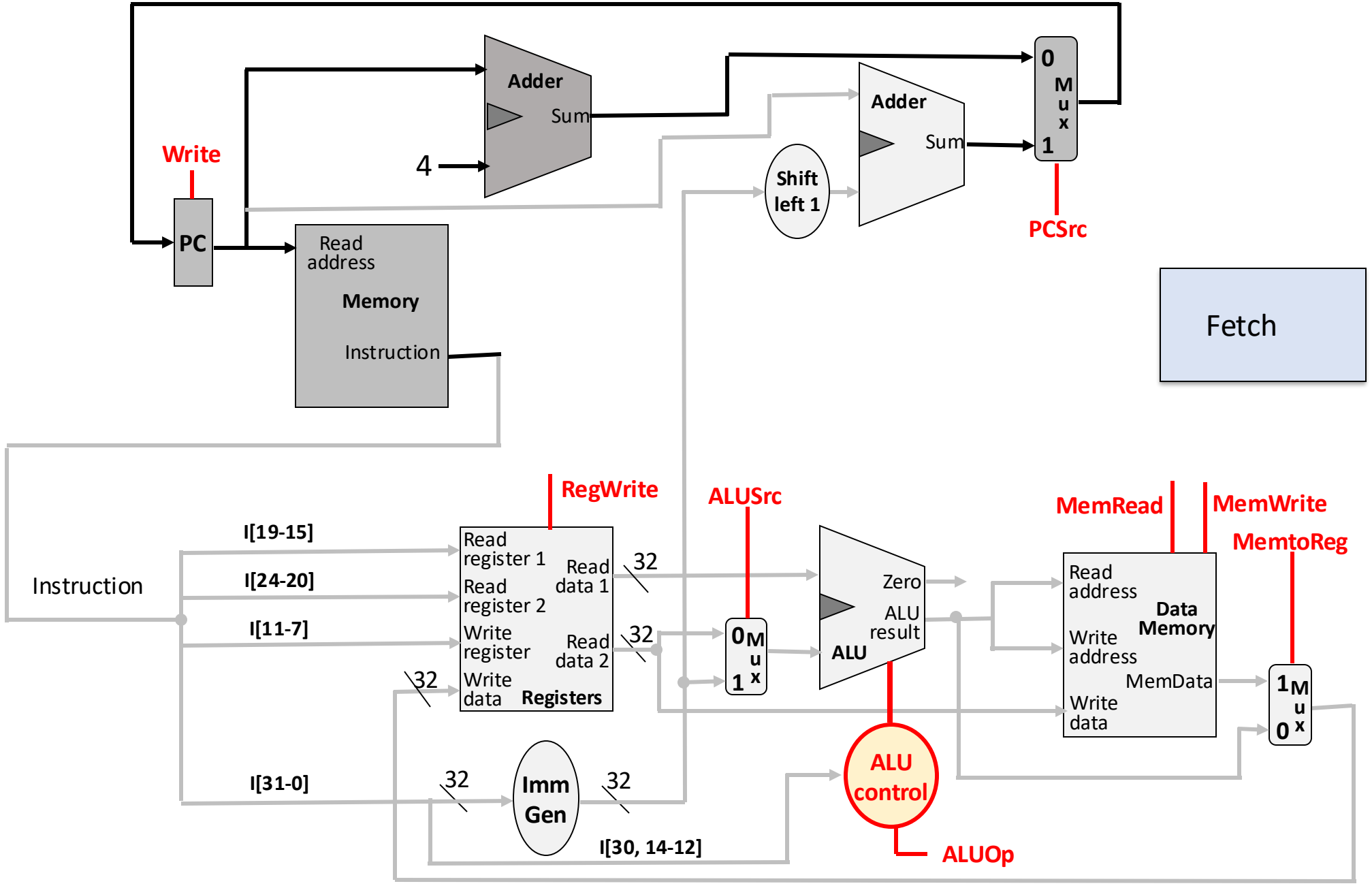


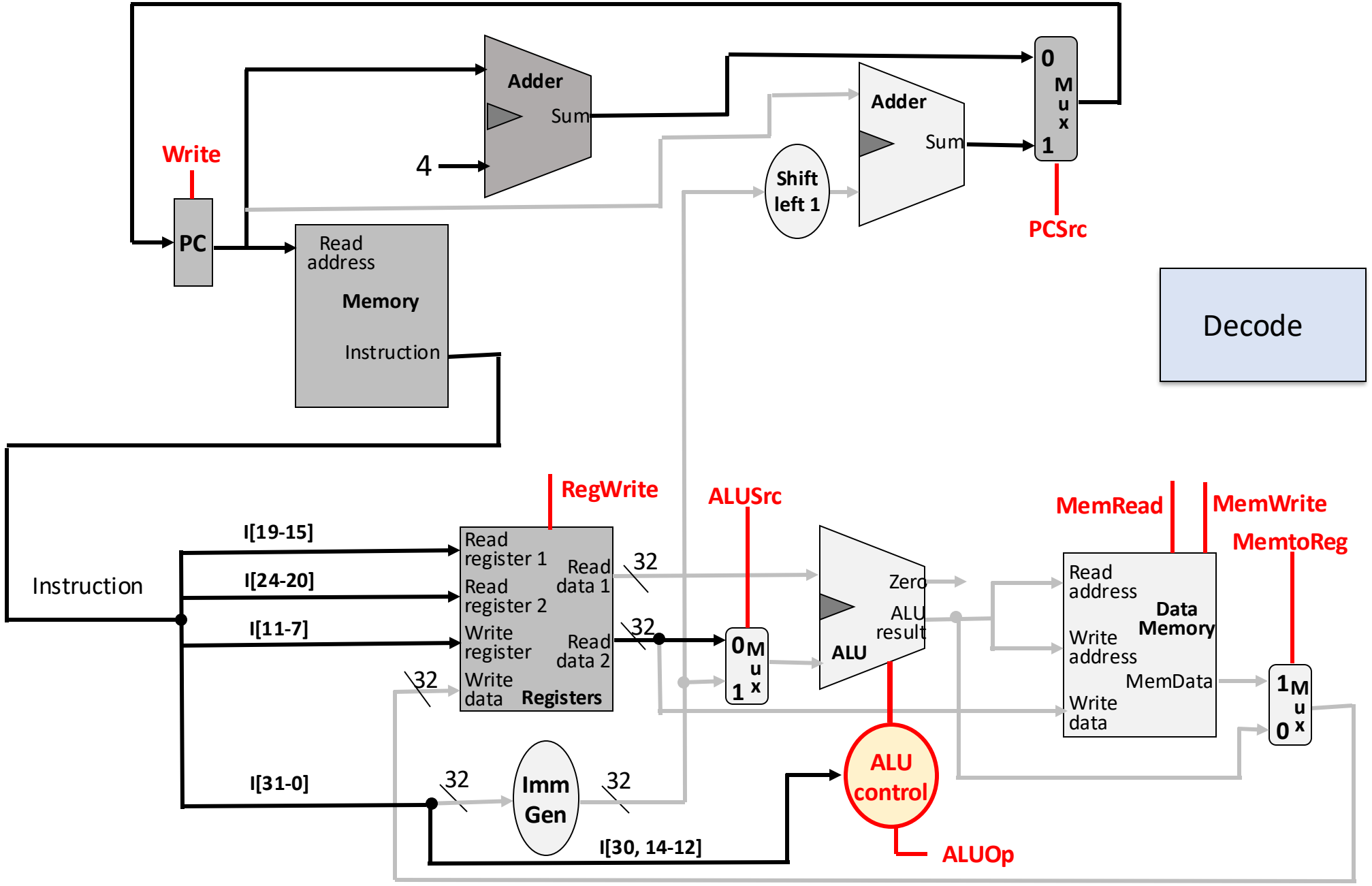
1. Fetch Instruction from instruction memory and increment PC.

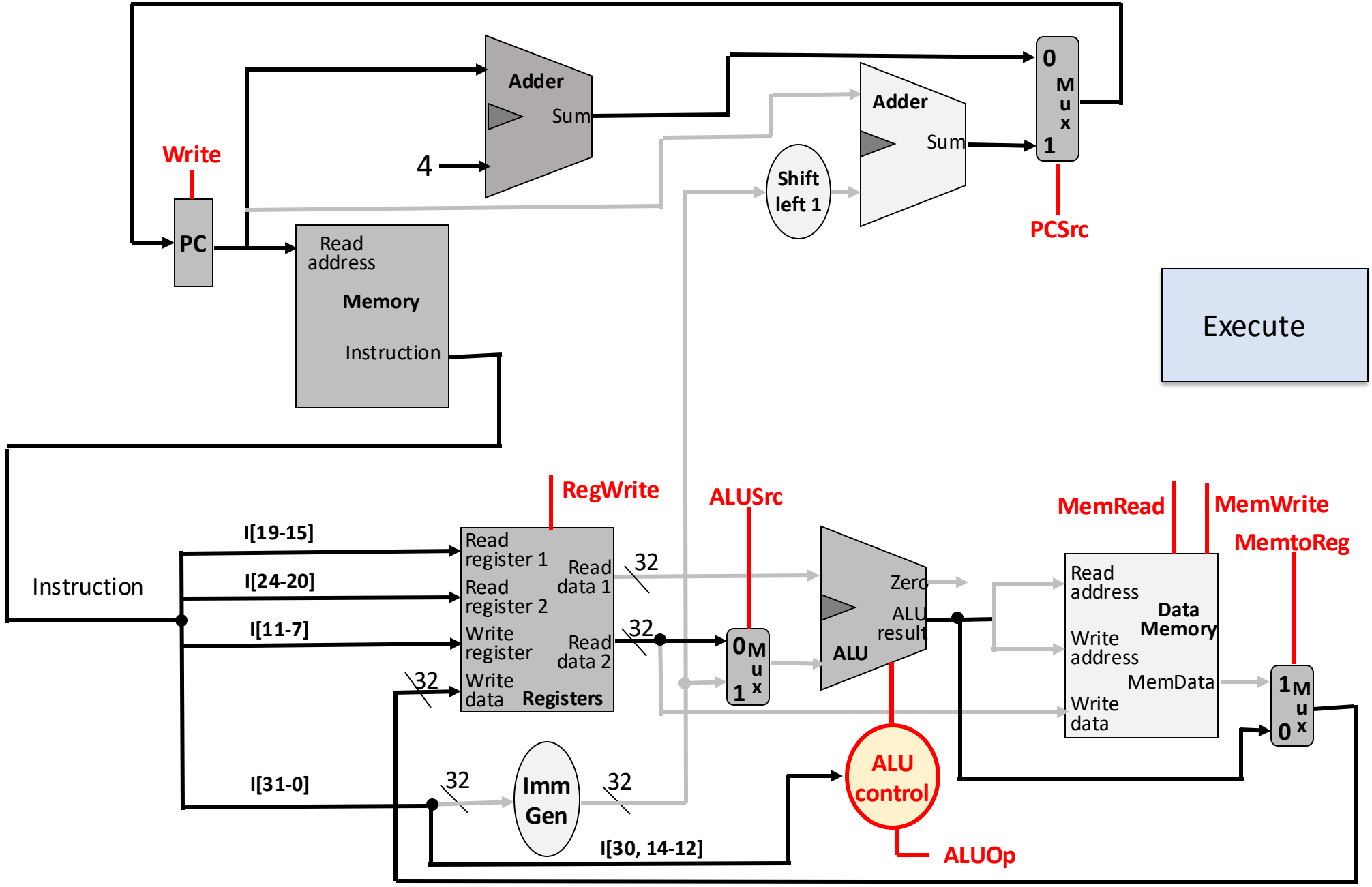
2. Read two registers (I19-I15) and (I24-I20) from the register file; also, the main control unit computes the setting of the control lines during this step.

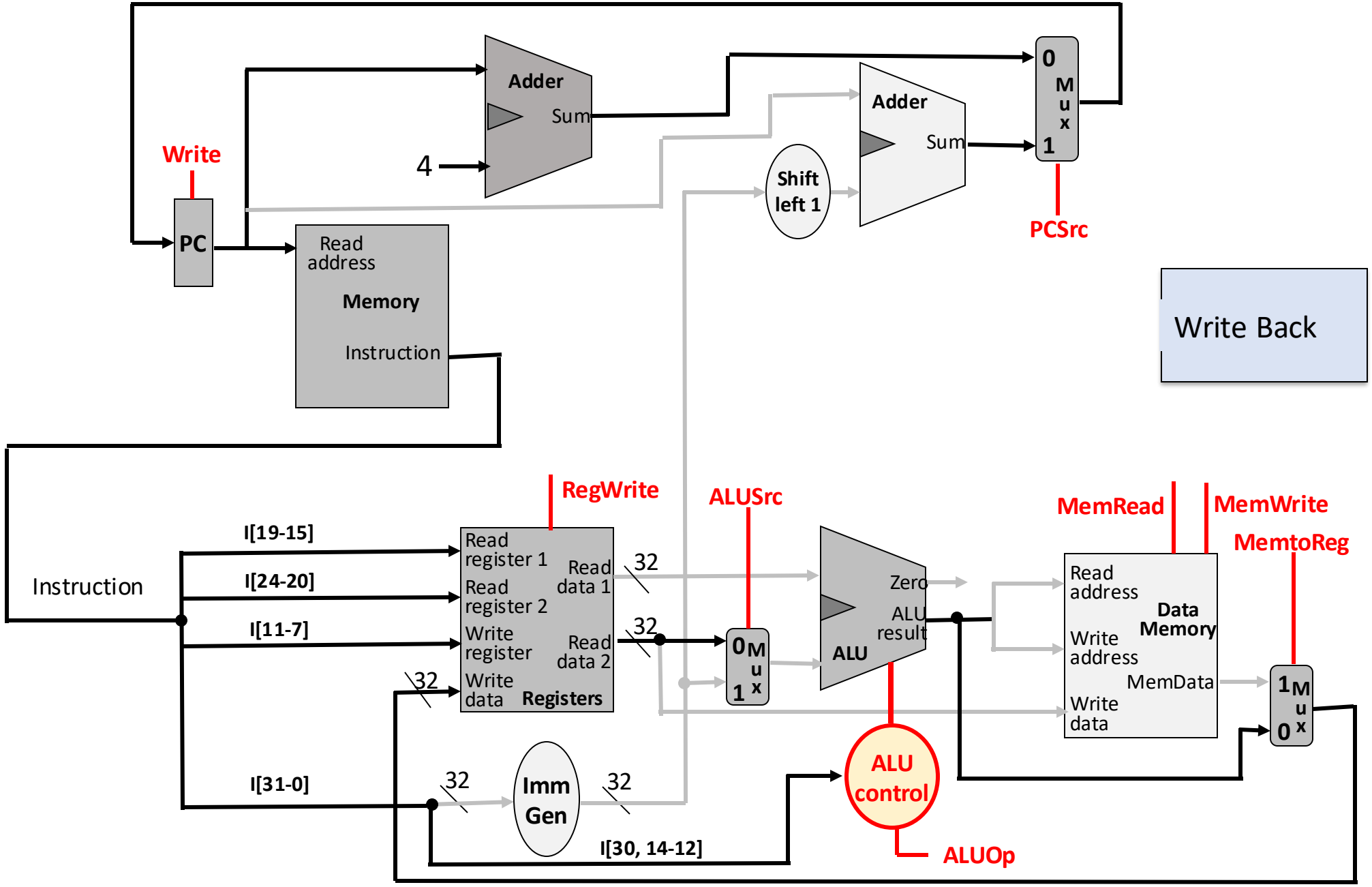
3. The ALU operates on the data read from the register file, using portions of the opcode to generate the ALU function.

4. Write the ALU result to the destination register (I11-I7).


Adder
Sum
4
Write
PC
Read address
Memory
Instruction
Shift left 1
Adder
Sum
0
M
u
x
1
PCSrc
Fetch
RegWrite
ALUSrc
MemRead
MemWrite
MemtoReg
Instruction
I[19-15]
I[24-20]
I[11-7]
I[31-0]
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
32
32
32
32
0
M
u
x
1
Zero
ALU
ALU result
Read address
Write address
Write data
Data Memory
MemData
1
M
u
x
0
Imm Gen
I[30, 14-12]
ALU control
ALUOp


Adder
Sum
4
Write
PC
Read address
Memory
Instruction
Shift left 1
Adder
Sum
0
M
u
x
1
PCSrc
Decode
RegWrite
ALUSrc
MemRead
MemWrite
MemtoReg
I[19-15]
I[24-20]
I[11-7]
I[31-0]
Instruction
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
32
32
32
32
0
M
u
x
1
ALU
Zero
ALU result
Read address
Write address
Write data
Data Memory
MemData
1
M
u
x
0
Imm Gen
ALU control
ALUOp
I[30, 14-12]


Write
PC
Adder
4
Sum
Adder
Sum
Shift left 1
0
M
u
x
1
PCSrc
Read address
Memory
Instruction
Execute
RegWrite
ALUSrc
MemRead
MemWrite
MemtoReg
Instruction
I[19-15]
I[24-20]
I[11-7]
I[31-0]
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
32
32
32
32
0
M
u
x
1
ALU
Zero
ALU result
Read address
Write address
Write data
Data Memory
MemData
1
M
u
x
0
Imm Gen
ALU control
ALUOp
I[30, 14-12]


Write
PC
Adder
4
Sum
Adder
Sum
Shift left 1
0
M
u
x
1
PCSrc
Read address
Memory
Instruction
Write Back
RegWrite
ALUSrc
MemRead
MemWrite
MemtoReg
Instruction
I[19-15]
I[24-20]
I[11-7]
I[31-0]
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
32
32
32
0
M
u
x
1
ALU
Zero
ALU result
Read address
Write address
Write data
Data Memory
MemData
1
M
u
x
0
Imm Gen
32
32
ALU control
ALUOp
I[30, 14-12]

Five Steps for a load Instruction

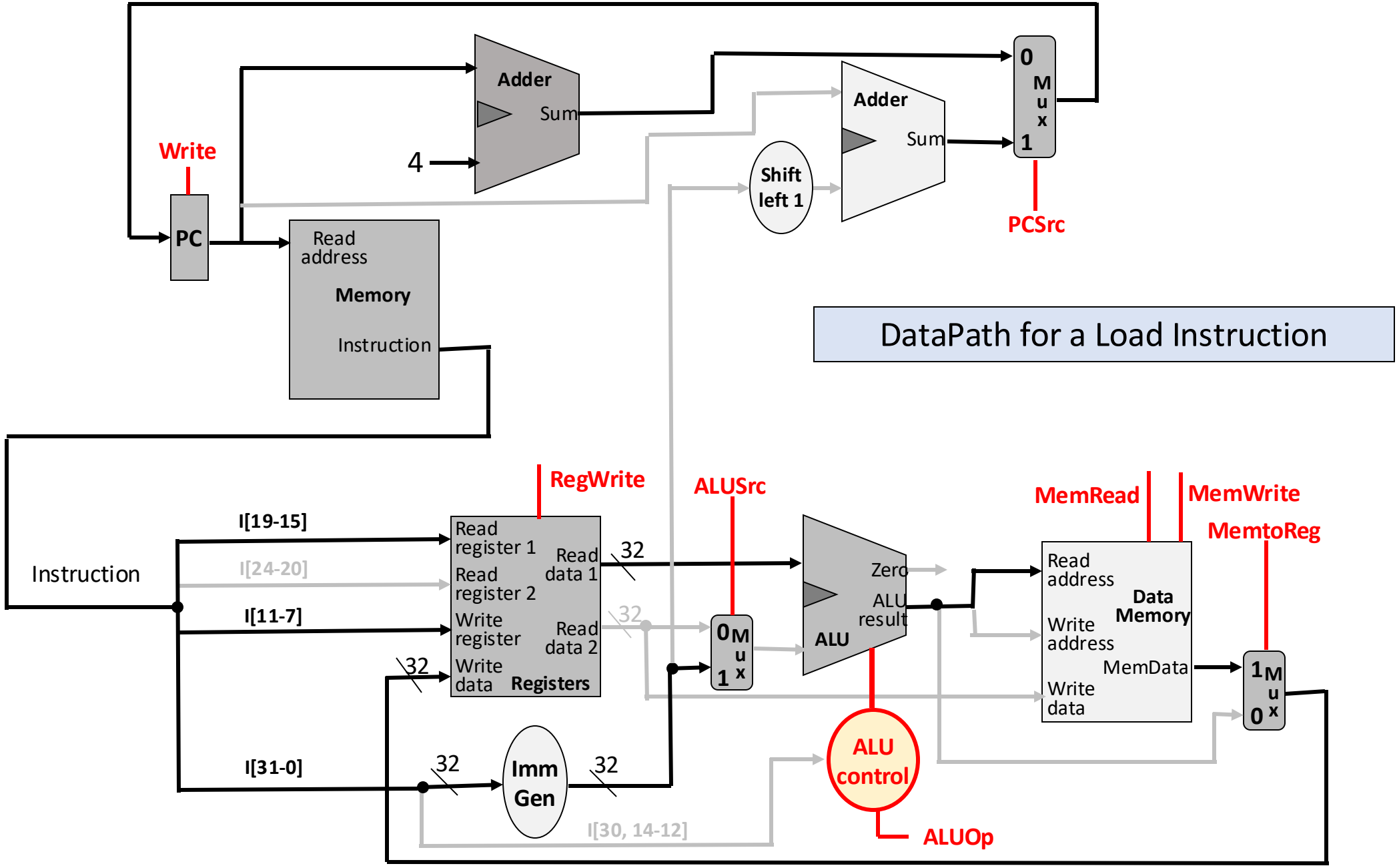
1. Fetch Instruction from instruction memory and increment PC.

2. Read one register (I19-I15) from the register file.

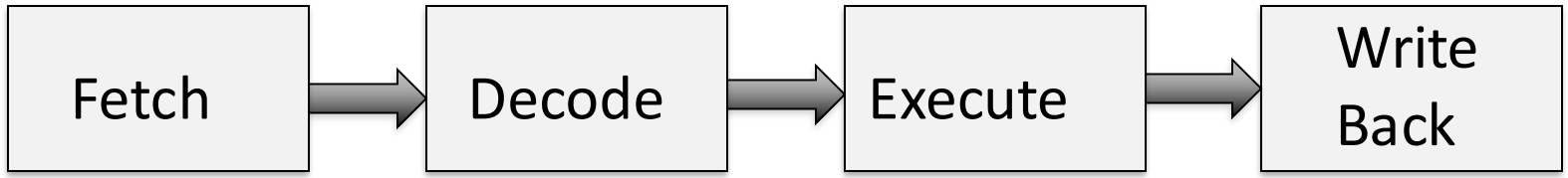
3. The ALU computes the sum of the value read from the register file and the sign extended 12 bits of the instruction (offset).

4. Use the result of the ALU as the address for the data memory.

5. Write the data from the memory unit to the destination register (I11-I7).


DataPath for a Load Instruction
Adder
Sum
4
Adder
Sum
Shift left 1
0
M
u
x
1
PCSrc
Write
PC
Read address
Memory
Instruction
Instruction
I[19-15]
I[24-20]
I[11-7]
I[31-0]
RegWrite
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
32
32
32
ALUSrc
0
M
u
x
1
Zero
ALU
ALU result
MemRead
MemWrite
MemtoReg
Read address
Write address
Write data
Data Memory
MemData
1
M
u
x
0
Imm Gen
32
32
I[30, 14-12]
ALU control
ALUOp

Four Steps for a branch-on-equal Instruction

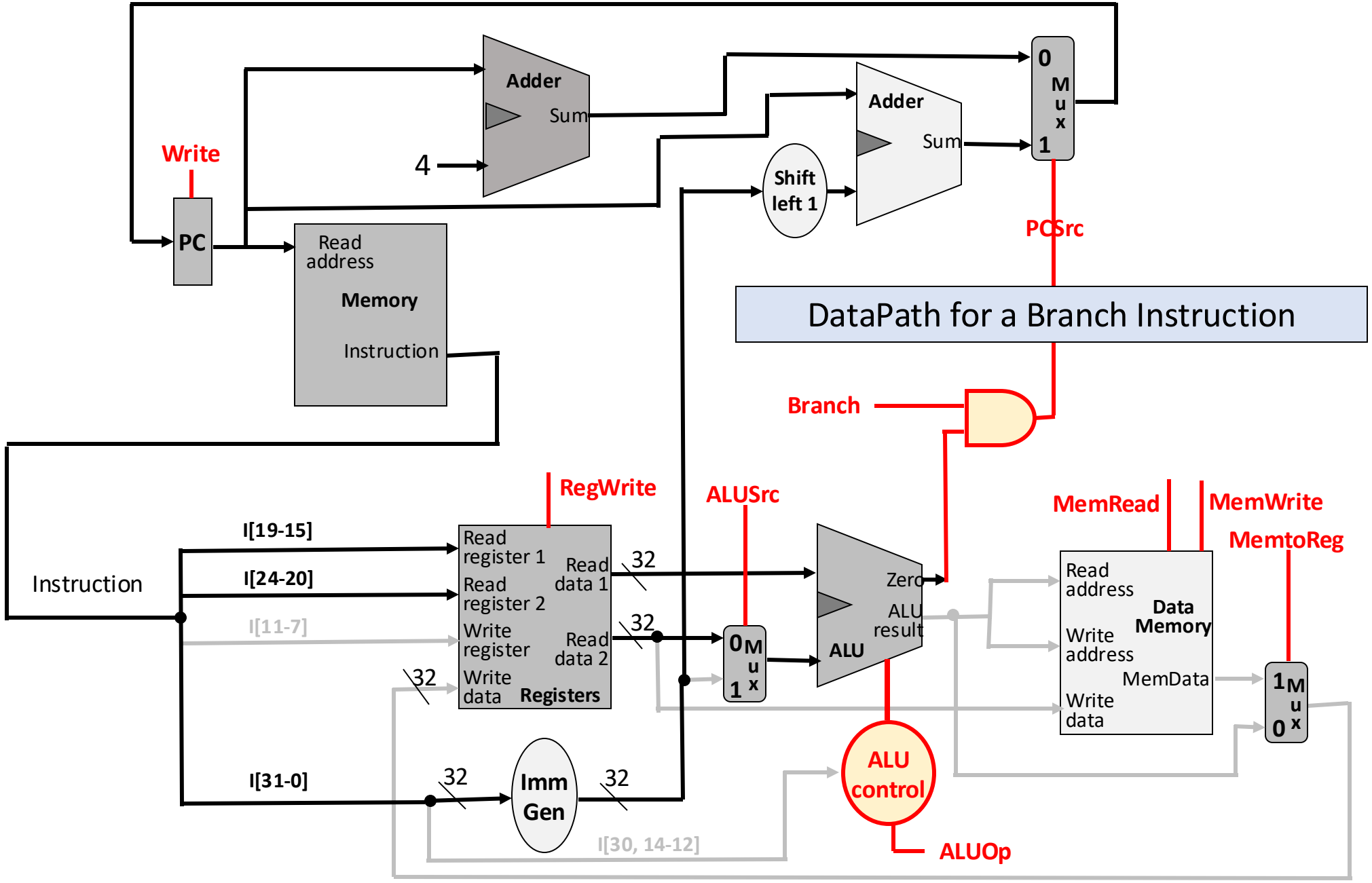


1. Fetch Instruction from instruction memory and increment PC.

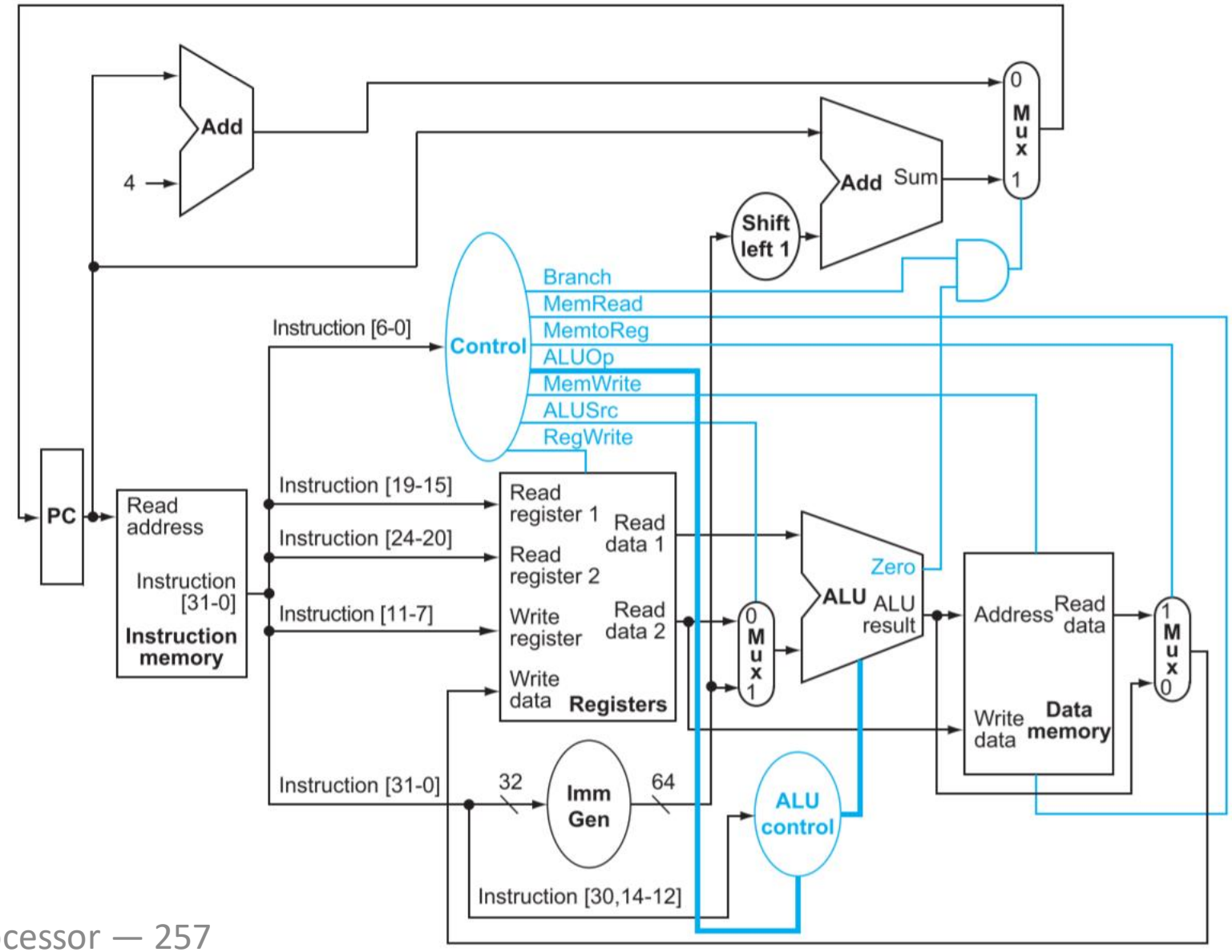
2. Read two registers (I19-I15) and (I24-I20) from the register file.

3. The ALU subtracts one data value from the other data value, both read from the register file. The value of PC is added to the sign-extended, 12 bits of the instruction (offset) left shifted by one; the result is the branch target address.

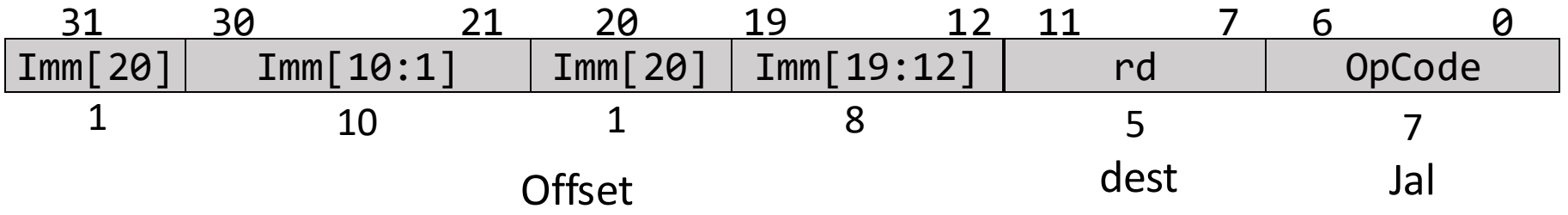
4. The Zero status information from the ALU is used to decide which adder result to store in the PC


DataPath for a Branch Instruction
Adder
Sum
4
Adder
Sum
Shift left 1
0
M
u
x
1
Write
PC
Read address
Memory
Instruction
PCSrc
Branch
RegWrite
ALUSrc
MemRead
MemWrite
MemtoReg
Instruction
I[19-15]
I[24-20]
I[11-7]
I[31-0]
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
32
32
32
32
0
M
u
x
1
Zero
ALU
ALU result
Read address
Write address
Write data
Data Memory
MemData
1
M
u
x
0
Imm Gen
ALU control
ALUOp
I[30, 14-12]

Datapath With Control



Implementing Jump and Link



1) JAL saves PC+4 in Reg[rd] (the return address)

2) Set PC = PC + offset (PC-relative jump)

3) Target somewhere within $\pm 2^{19}$ locations, 2 bytes apart

$\pm 2^{18}$ 32-bit instructions

4) Immediate encoding optimized similarly to branch instruction to reduce hardware cost

DataPath for a JAL Instruction