The MIPS Processor Datapath

Module Outline

- MIPS datapath implementation
  - Register File, Instruction memory, Data memory
- Instruction interpretation and execution.
- Combinational control
- Assignment: Datapath design and Control Unit design using SystemC.

Logic Design Fundamentals

- Information encoded in binary
  - Low voltage = 0, High voltage = 1
  - One wire per bit
  - Multi-bit data encoded on multi-wire buses

Logic Design Fundamentals

- Information encoded in binary
  - Low voltage = 0, High voltage = 1
  - One wire per bit
  - Multi-bit data encoded on multi-wire buses
- Combinational element
  -
  -
- State (sequential) elements
  -

Logic Design Fundamentals

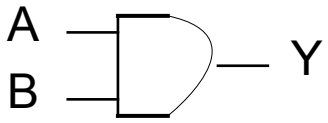
- Information encoded in binary
  - Low voltage = 0, High voltage = 1
  - One wire per bit
  - Multi-bit data encoded on multi-wire buses
- Combinational element
  - Operate on data
  - Output is a function of input
- State (sequential) elements
  -

Logic Design Fundamentals

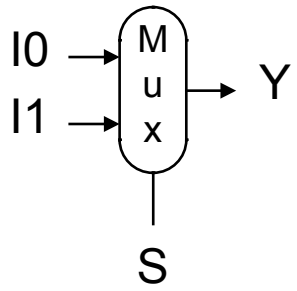
- Information encoded in binary
  - Low voltage = 0, High voltage = 1
  - One wire per bit
  - Multi-bit data encoded on multi-wire buses
- Combinational element
  - Operate on data
  - Output is a function of input
- State (sequential) elements
  - Store information

Combinational Elements

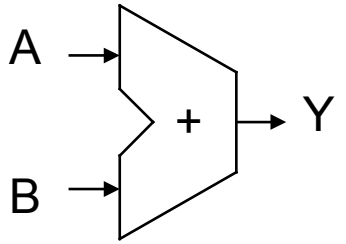
- AND-gate
  - Y = A & B



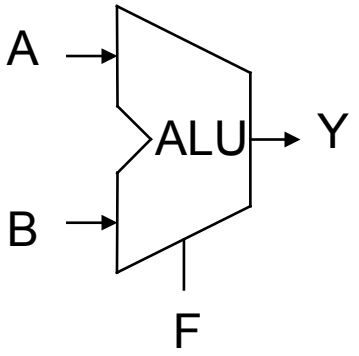
- Multiplexer
  - Y = S ? I1 : I0



- Adder
  - Y = A + B



- Arithmetic/Logic Unit
  - Y = F(A, B)

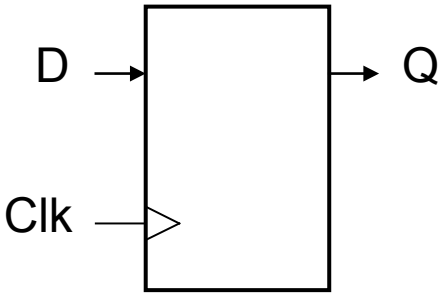


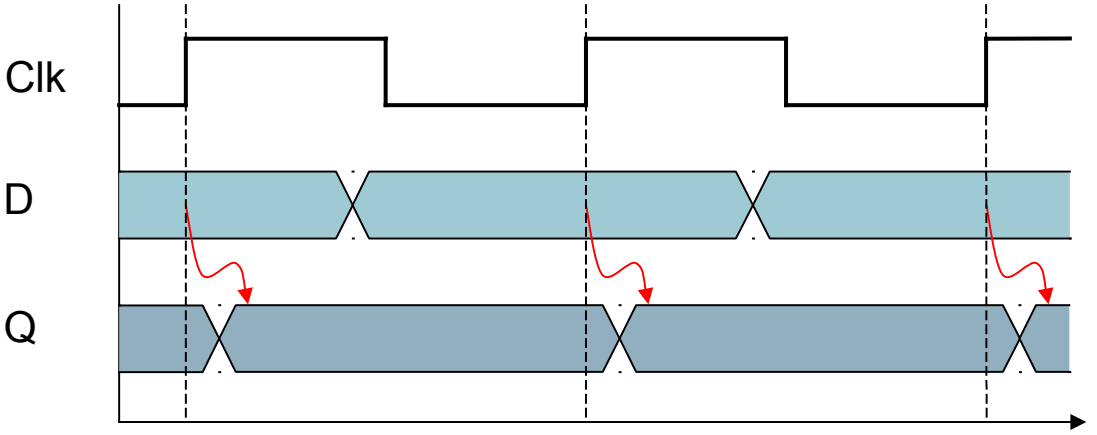
Sequential Elements

- Register: stores data in a circuit
  - Uses a clock signal to determine when to update the stored value
  - Edge-triggered: update when Clk changes from 0 to 1

Sequential Elements

- Register: stores data in a circuit
  - Uses a clock signal to determine when to update the stored value
  - Edge-triggered: update when Clk changes from 0 to 1



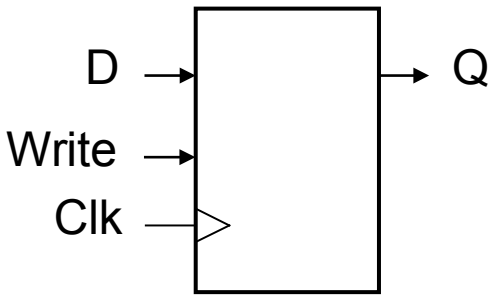


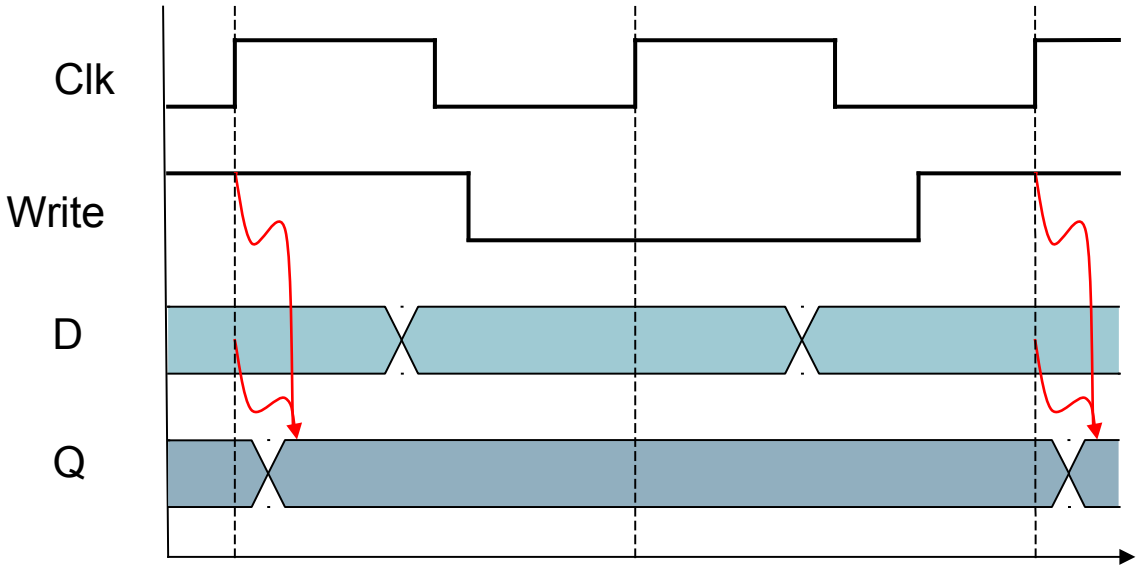
Sequential Elements

- Register with write control
  - Only updates on clock edge when write control input is 1
  - Used when stored value is required later

Sequential Elements

- Register with write control
  - Only updates on clock edge when write control input is 1
  - Used when stored value is required later





Clocking Methodology

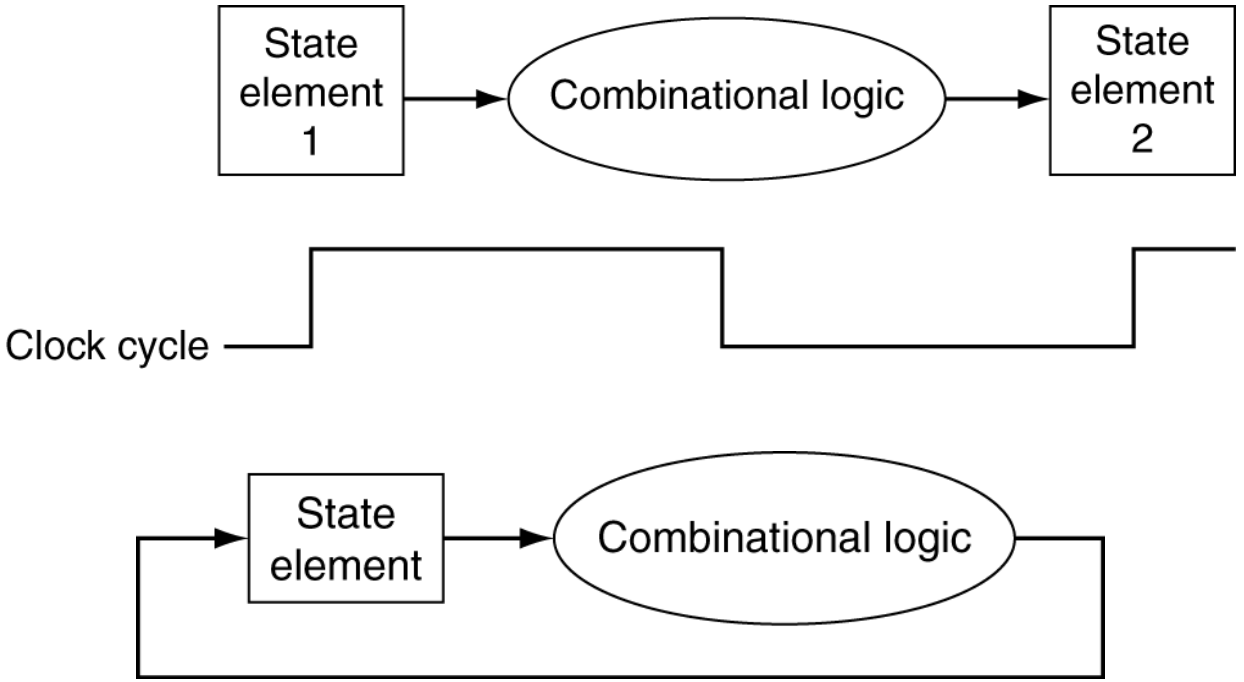
- Combinational logic transforms data during clock cycles
  -

Clocking Methodology

- Combinational logic transforms data during clock cycles
  - Between clock edges
  - Input from state elements, output to state element
  -

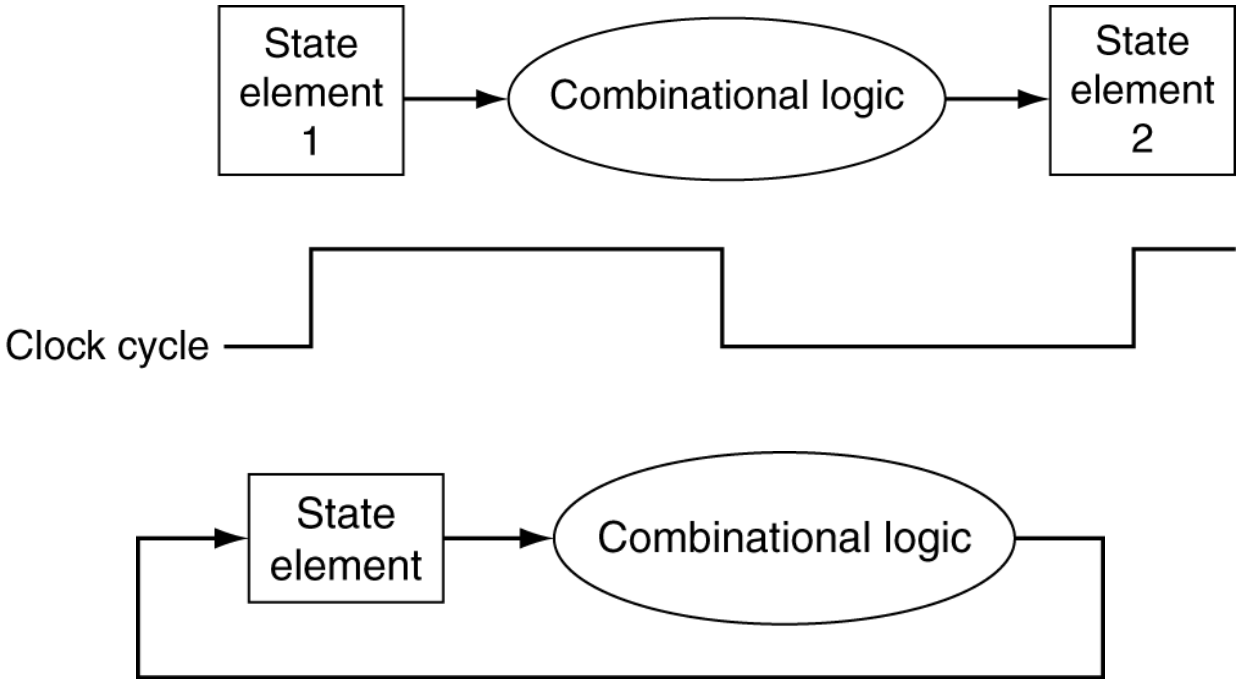
Clocking Methodology

- Combinational logic transforms data during clock cycles
  - Between clock edges
  - Input from state elements, output to state element
  -



Clocking Methodology

- Combinational logic transforms data during clock cycles
  - Between clock edges
  - Input from state elements, output to state element
  - Longest delay determines clock period



Building a Datapath

- Datapath
  -

Building a Datapath

- Datapath
  - Elements that process data and addresses in the CPU
    -
-

Building a Datapath

- Datapath
  - Elements that process data and addresses in the CPU
    - Registers, ALUs, muxes, memories, …
-

Building a Datapath

- Datapath
  - Elements that process data and addresses in the CPU
    - Registers, ALUs, muxes, memories, …
- We will build a MIPS datapath incrementally

A Basic MIPS Implementation

- Memory-reference instructions – Load Word (lw) and Store Word (sw)
- ALU instructions – add, sub, AND, OR and slt
- Branch on equal (beq)

Instruction Execution – Steps

Instruction Execution – Steps

- Instruction Fetch

Instruction Execution – Steps

- Instruction Fetch
- Instruction Decode/Register Fetch

Instruction Execution – Steps

- Instruction Fetch
- Instruction Decode/Register Fetch
- Execute
  - ALU
  - Effective Address Calculation

Instruction Execution – Steps

- Instruction Fetch
- Instruction Decode/Register Fetch
- Execute
  - ALU
  - Effective Address Calculation
- Memory Access

Instruction Execution – Steps

- Instruction Fetch
- Instruction Decode/Register Fetch
- Execute
  - ALU
  - Effective Address Calculation
- Memory Access
- Write back (Update RF)

Instruction Fetch – Actions

Instruction Fetch – Actions

- Read Program Counter

Instruction Fetch – Actions

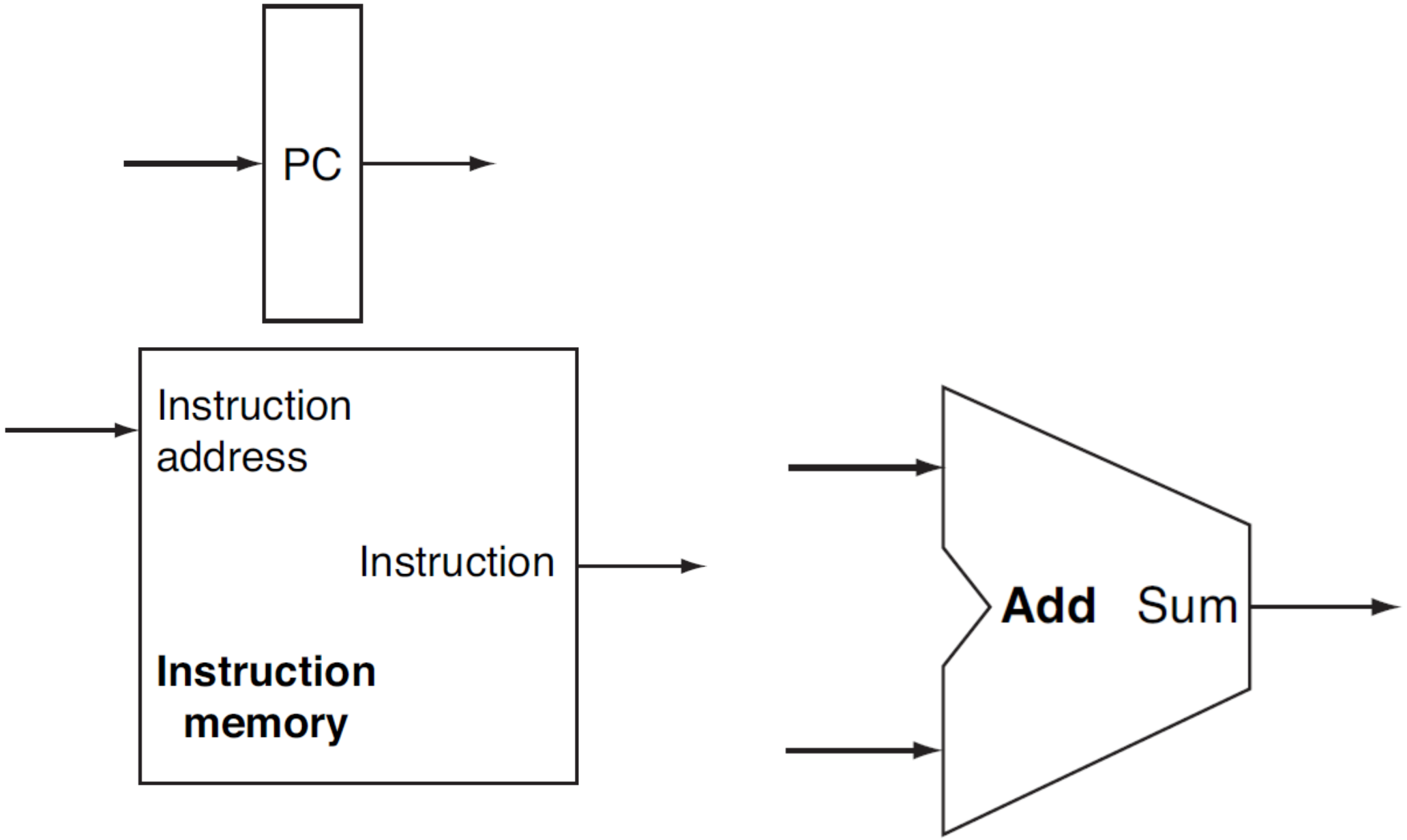
- Read Program Counter
- Fetch instruction from Instruction memory pointed to by the PC

Instruction Fetch – Actions

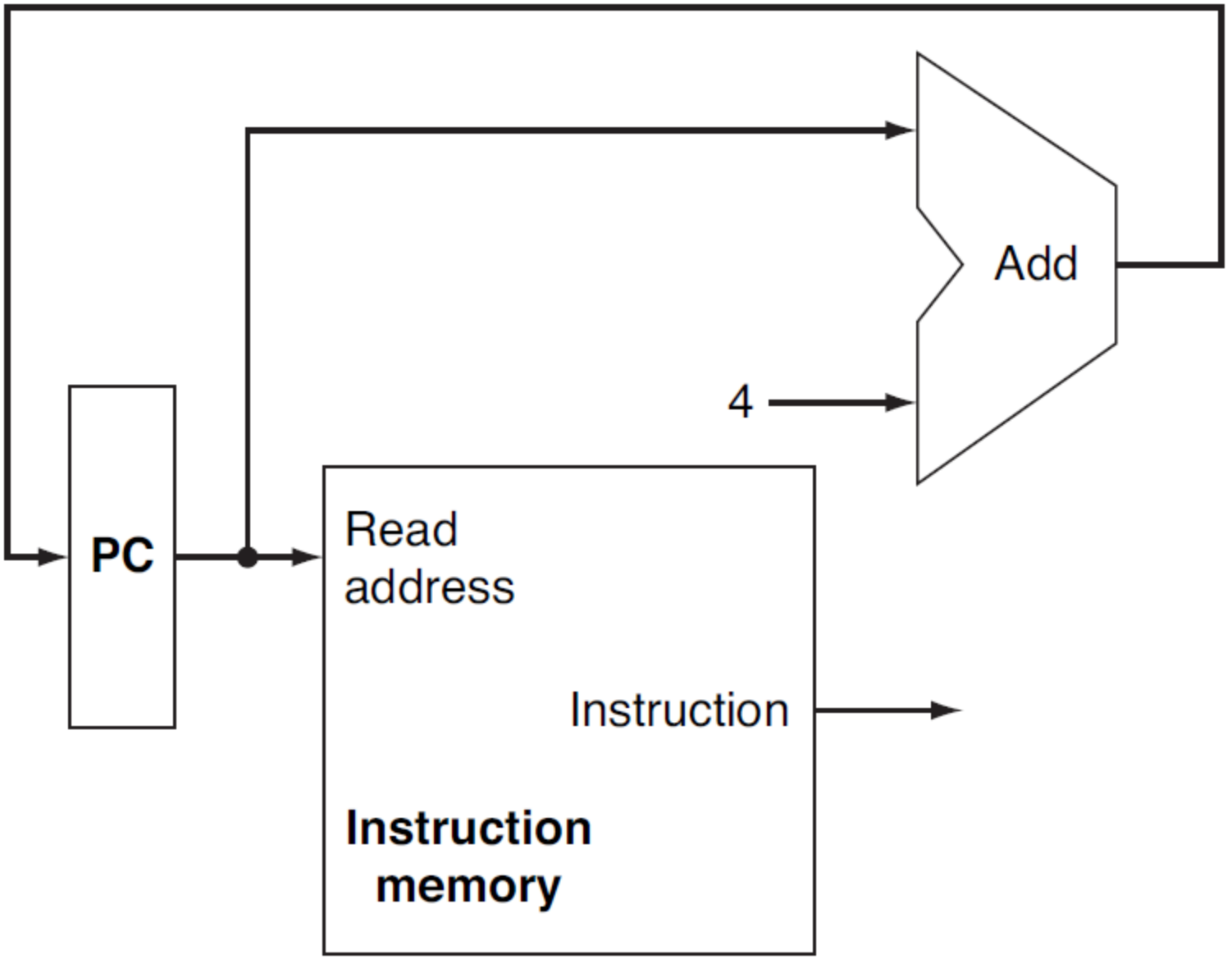
- Read Program Counter
- Fetch instruction from Instruction memory pointed to by the PC
- Increment PC

Instruction Fetch – Elements

Instruction Fetch – Elements



Instruction Fetch



ALU Instructions – Operations

ADD R1, R2, R3

ALU Instructions – Operations

- Read R2 and R3 from Register file
  - Send 2 and 3 to RF
  - RF reads contents of R2 and R3

ADD R1, R2, R3

ALU Instructions – Operations

- Read R2 and R3 from Register file
  - Send 2 and 3 to RF
  - RF reads contents of R2 and R3
- Add contents of R2 and R3 in the ALU

ADD R1, R2, R3

ALU Instructions – Operations

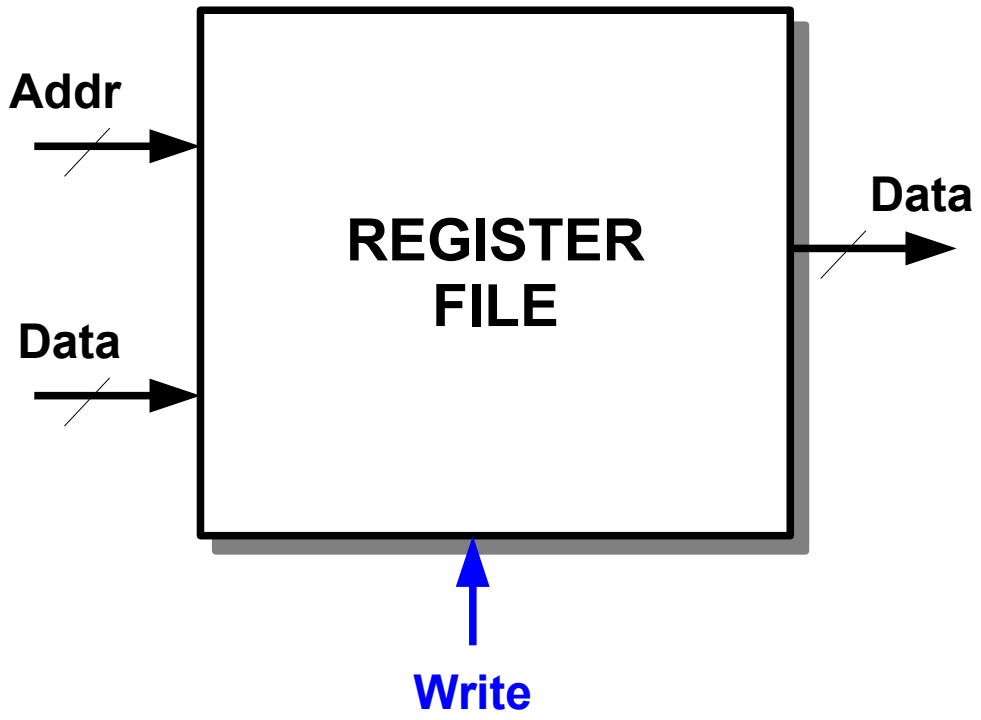
- Read R2 and R3 from Register file
  - Send 2 and 3 to RF
  - RF reads contents of R2 and R3
- Add contents of R2 and R3 in the ALU
- Feed the sum output to the RF; Ask it to write into R1

ADD R1, R2, R3

ALU Operations – Elements

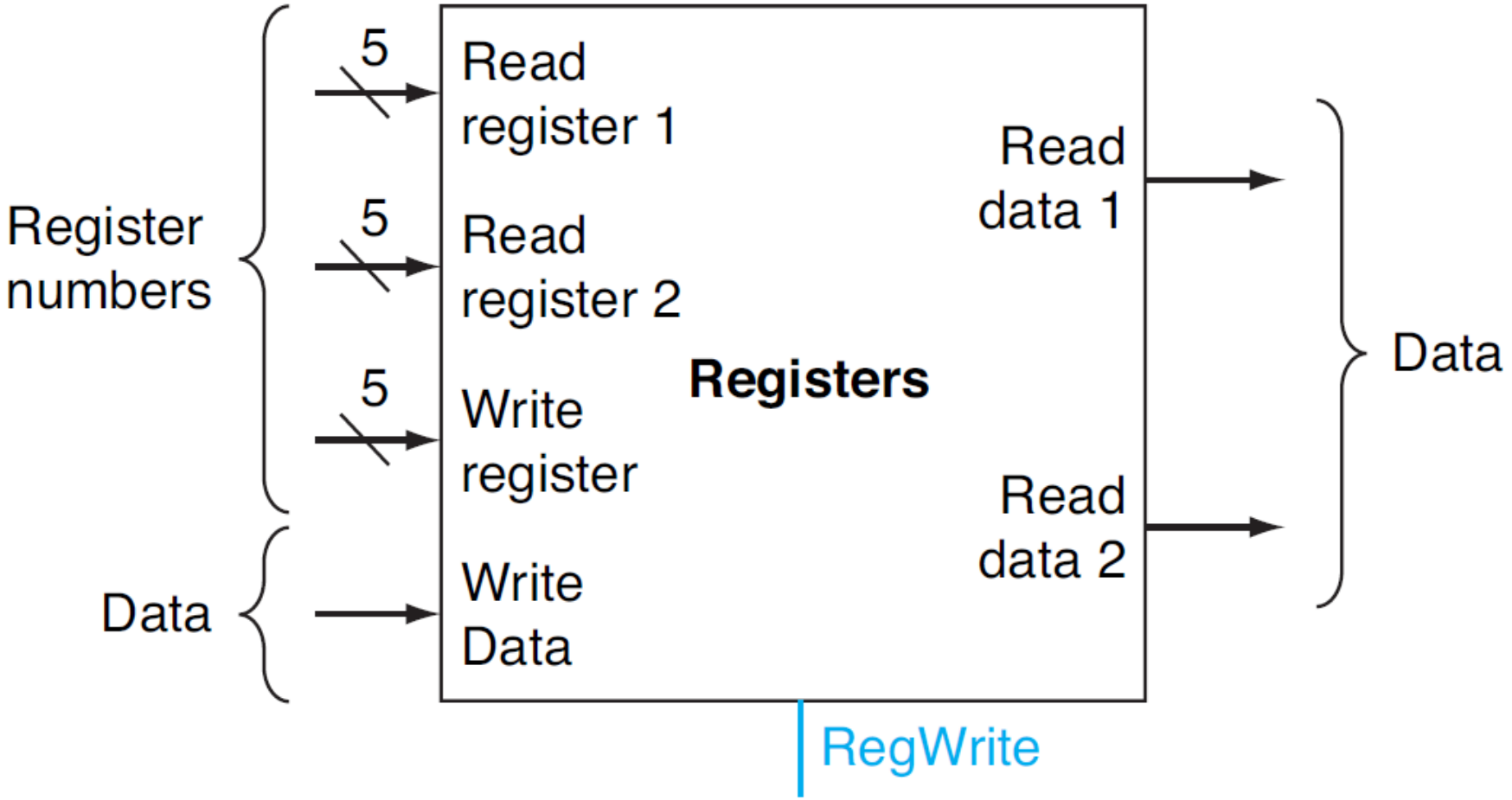
ADD R1, R2, R3

ALU Operations – Elements



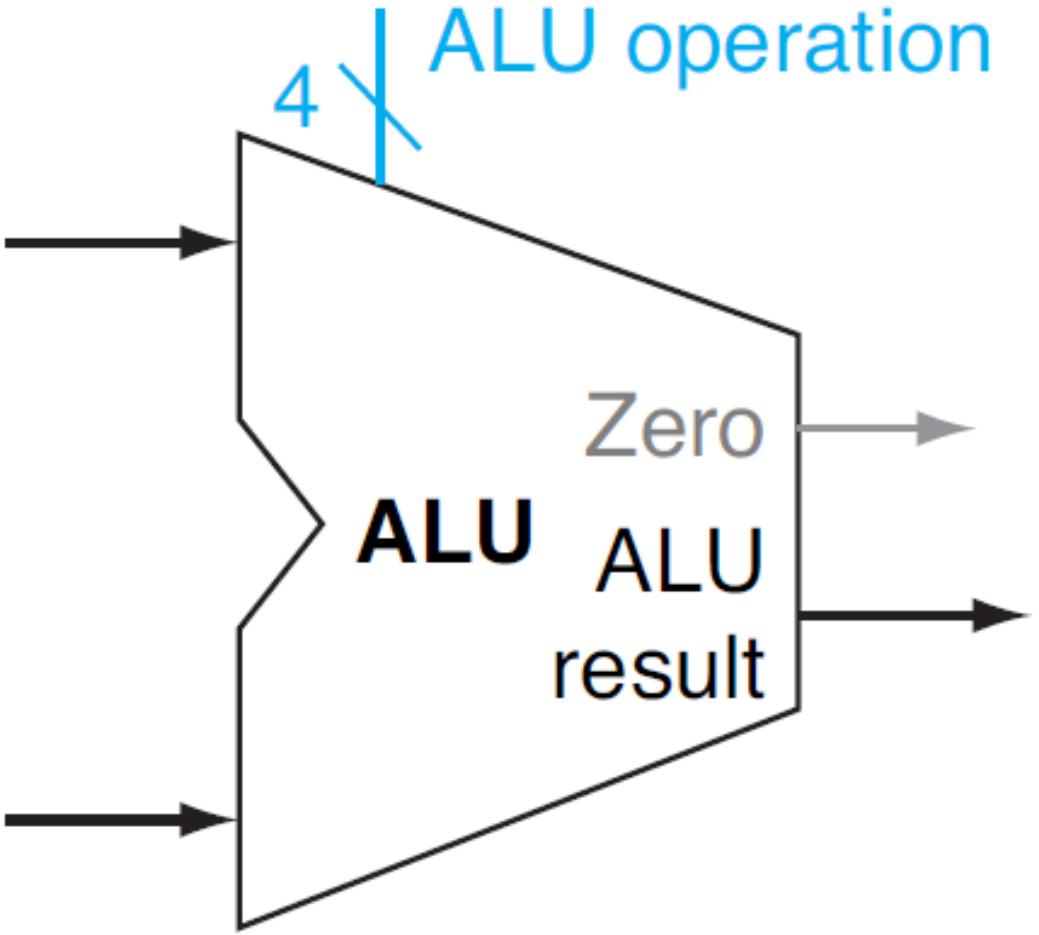
ADD R1, R2, R3

ALU Operations – Elements



ADD R1, R2, R3

ALU Operations – Elements



ADD R1, R2, R3

ALU Operations – Datapath

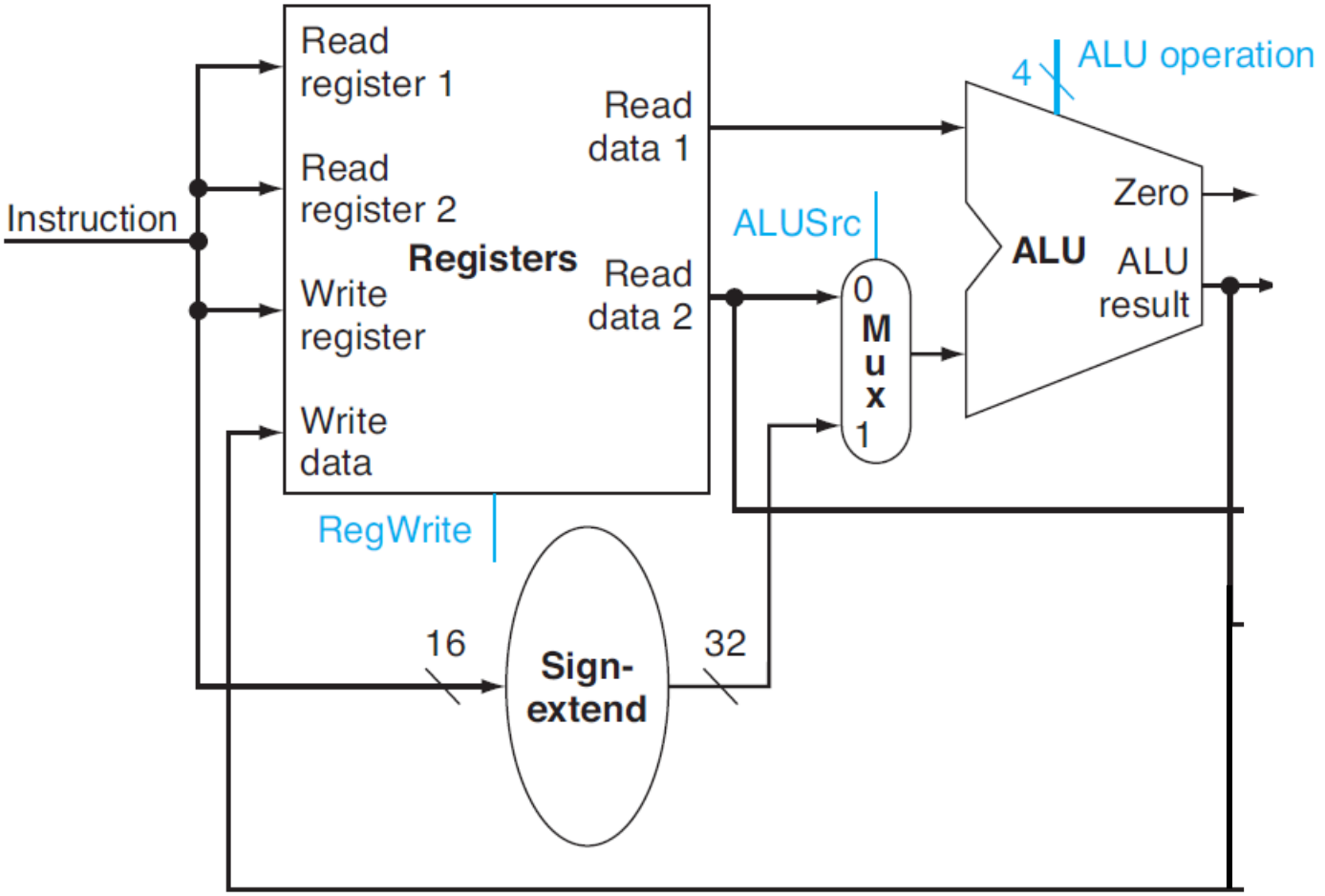
ADD R1, R2, R3

ALU Operations – Datapath

- How will the design change for ADDI?

ADDI R1, R2, -13

ALU Operations – Datapath



ADDI R1, R2, -13

Loads and Stores – Actions

LW R1, -8(R2)

Loads and Stores – Actions

- Calculate full address
  - Sum of -8 (offset) and contents of R2 (base)
  - Size of offset? Size of contents of R2?

LW R1, -8(R2)

Loads and Stores – Actions

- Calculate full address
  - Sum of -8 (offset) and contents of R2 (base)
  - Size of offset? Size of contents of R2?
- Send the address to Data memory

LW R1, -8(R2)

Loads and Stores – Actions

- Calculate full address
  - Sum of -8 (offset) and contents of R2 (base)
  - Size of offset? Size of contents of R2?
- Send the address to Data memory
- DM reads out the contents of Mem[R2+(-8)]

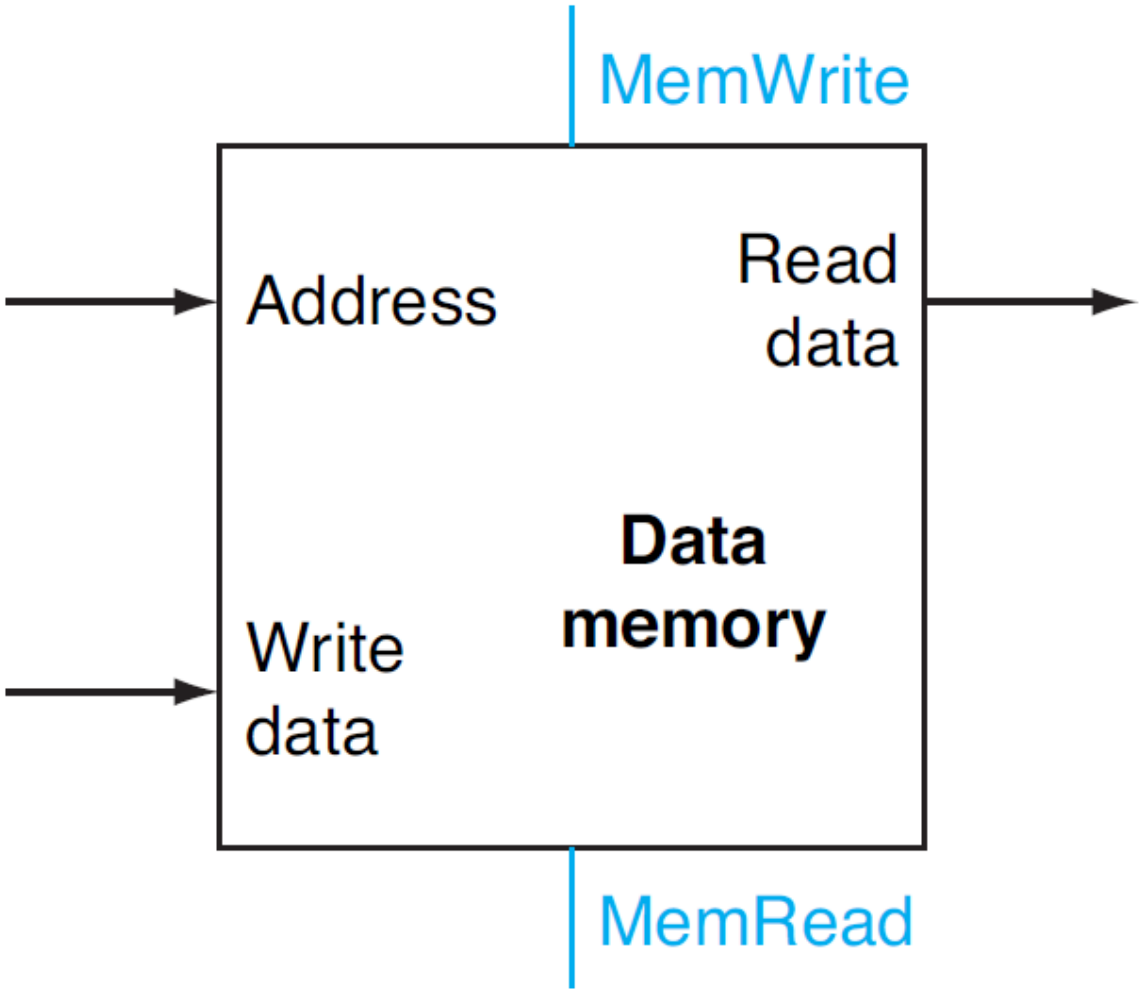
LW R1, -8(R2)

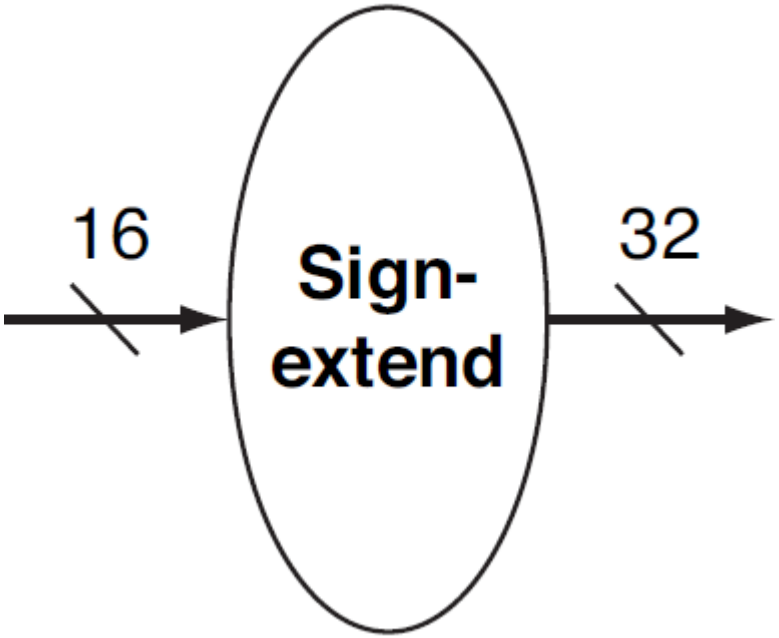
Loads and Stores – Actions

- Calculate full address
  - Sum of -8 (offset) and contents of R2 (base)
  - Size of offset? Size of contents of R2?
- Send the address to Data memory
- DM reads out the contents of Mem[R2+(-8)]
- Feed the value from memory to the RF; Ask it to write the value into R1

LW R1, -8(R2)

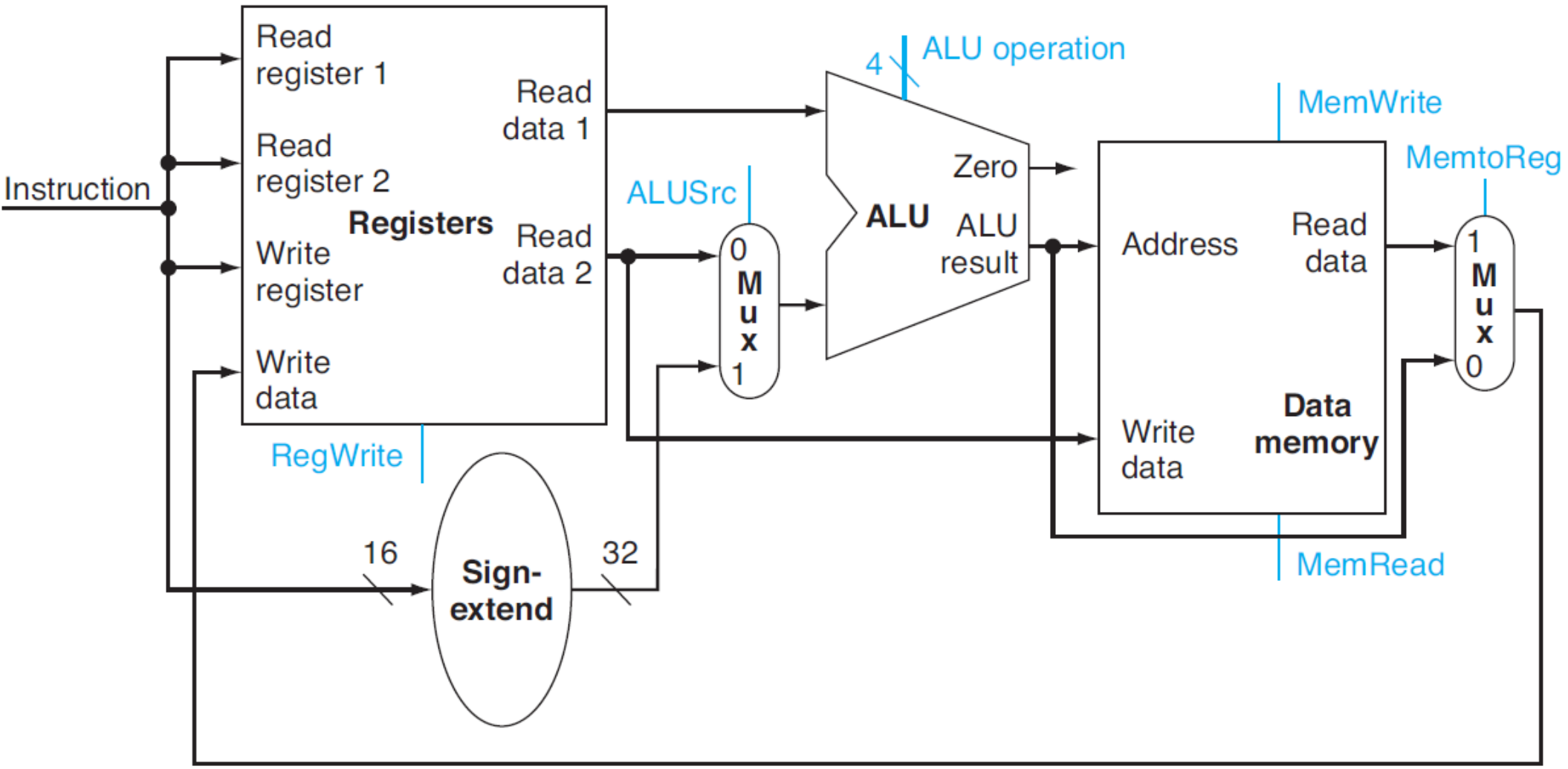
Loads and Stores – Elements



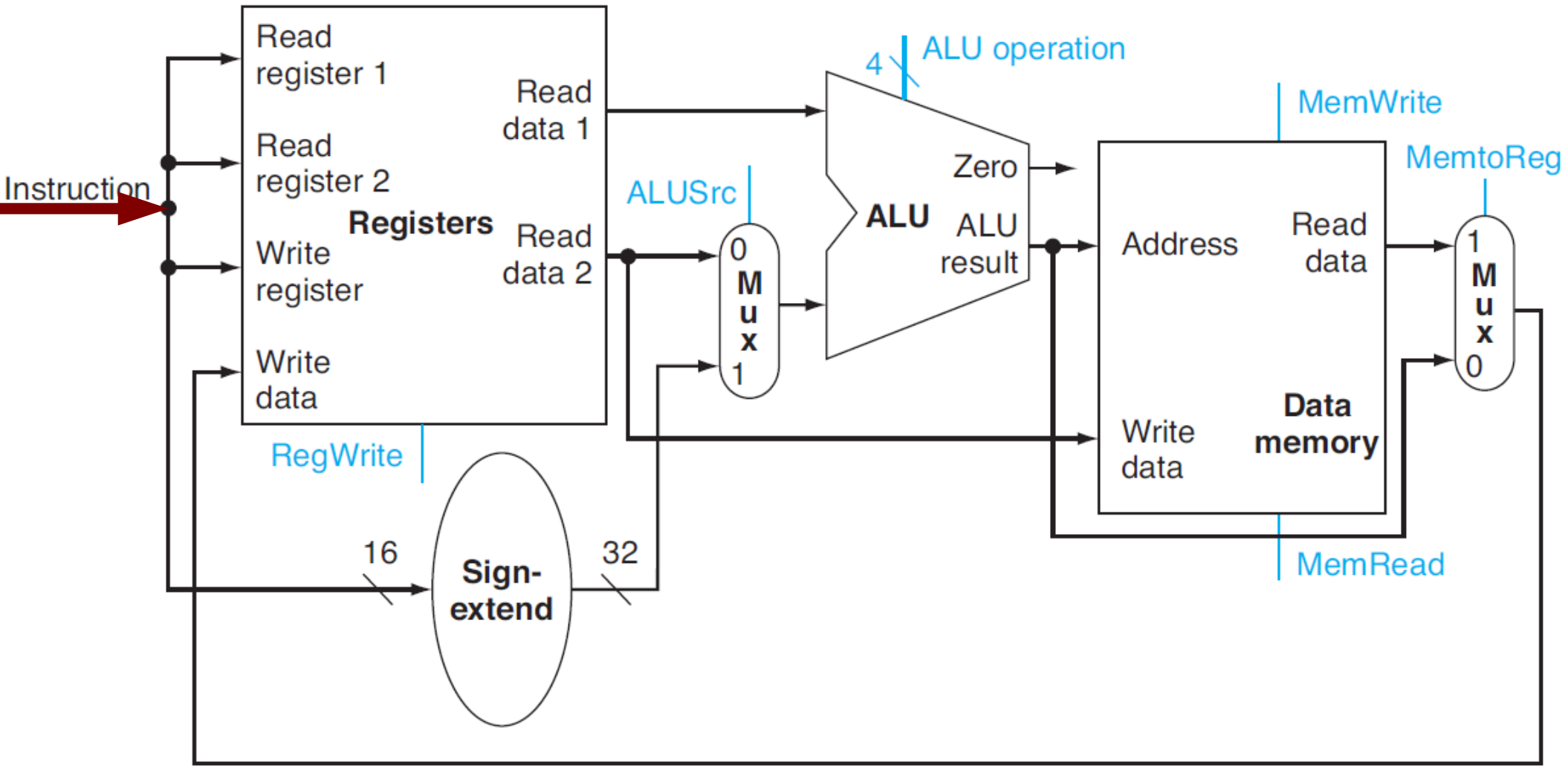


LW R1, -8(R2)

Memory and R-type Instructions

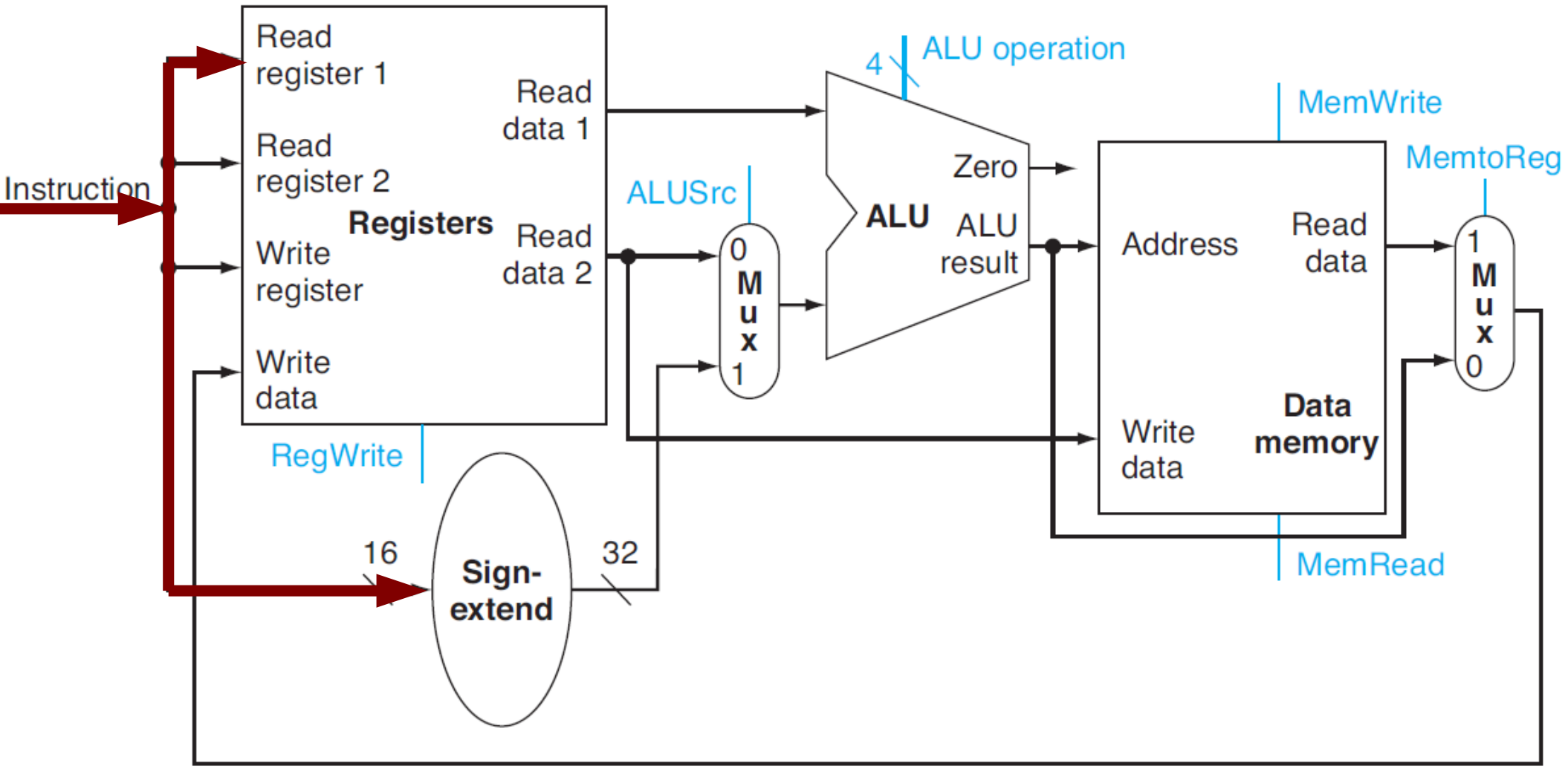


Memory Instruction – Load



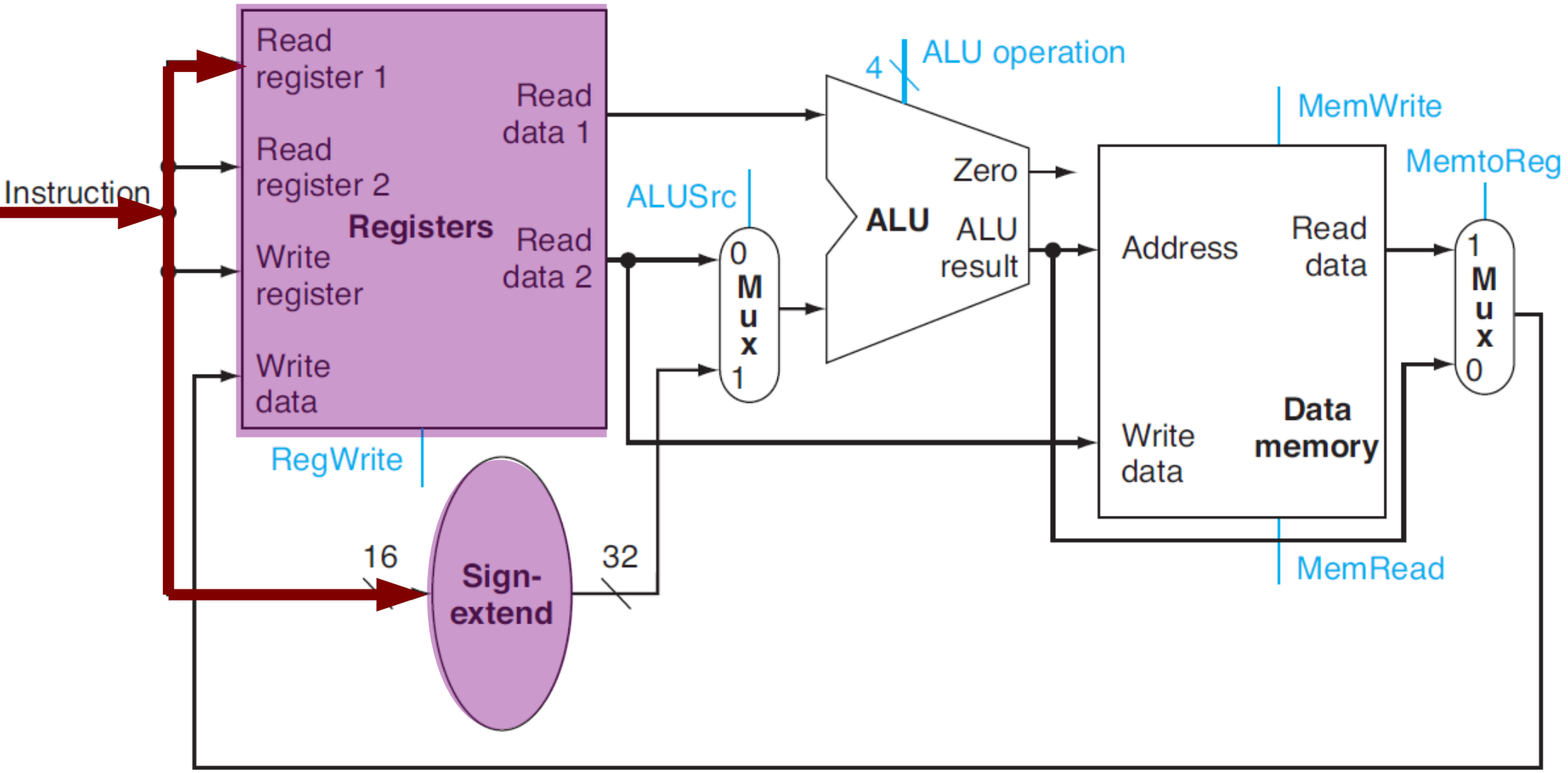
LW R1, -8(R2)

Memory Instruction – Load



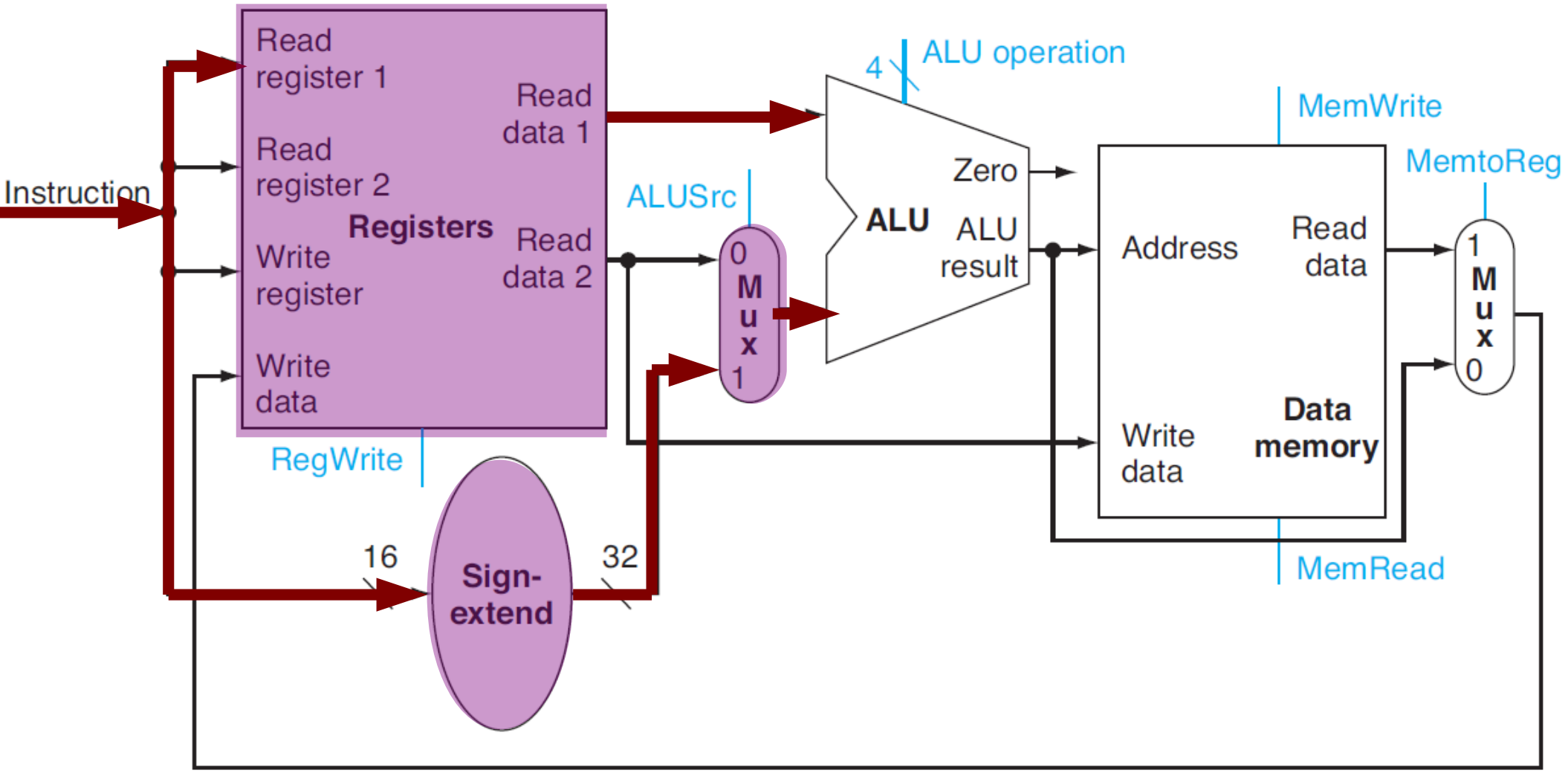
LW R1, -8(R2)

Memory Instruction – Load



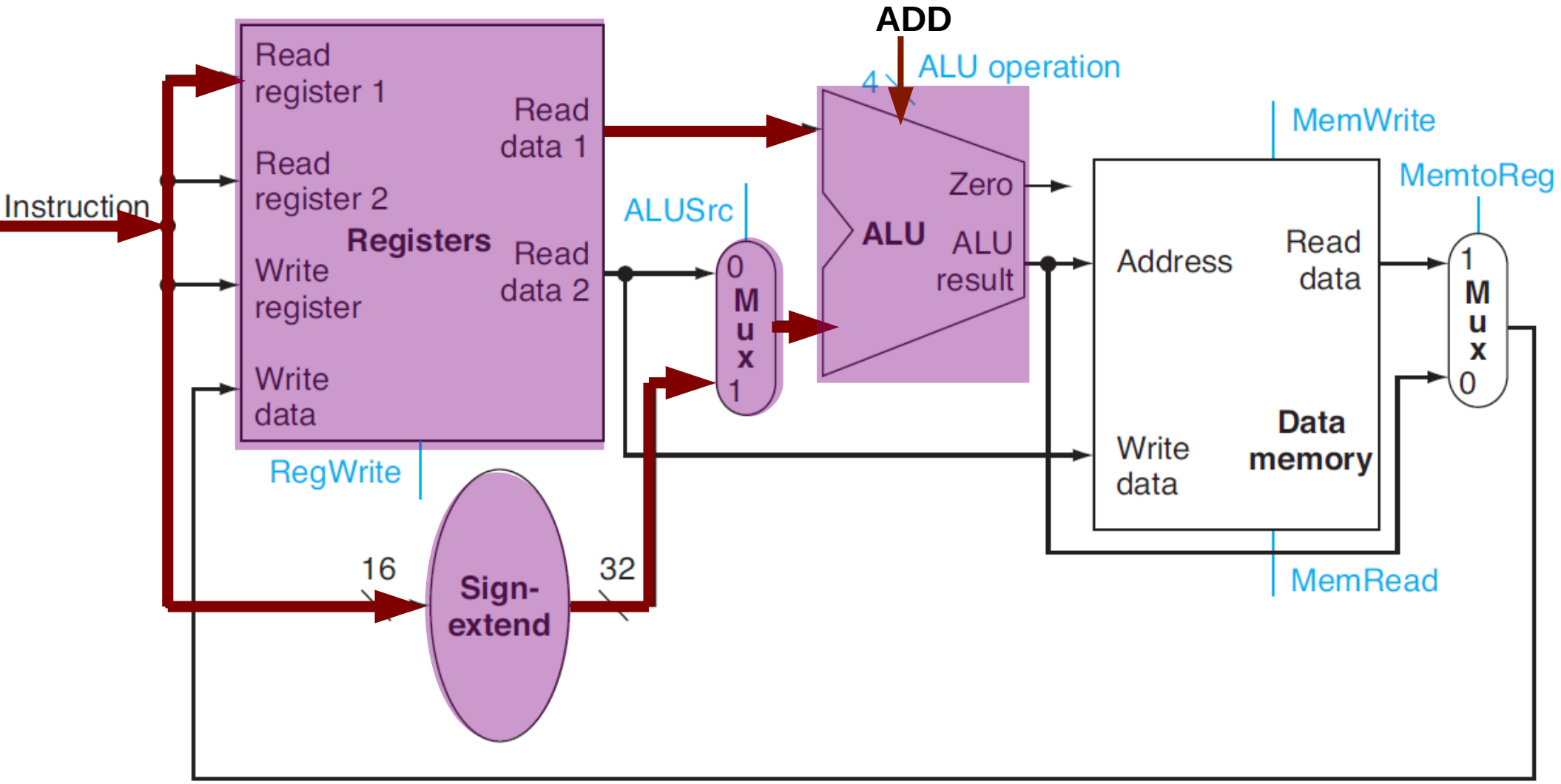
LW R1, -8(R2)

Memory Instruction – Load



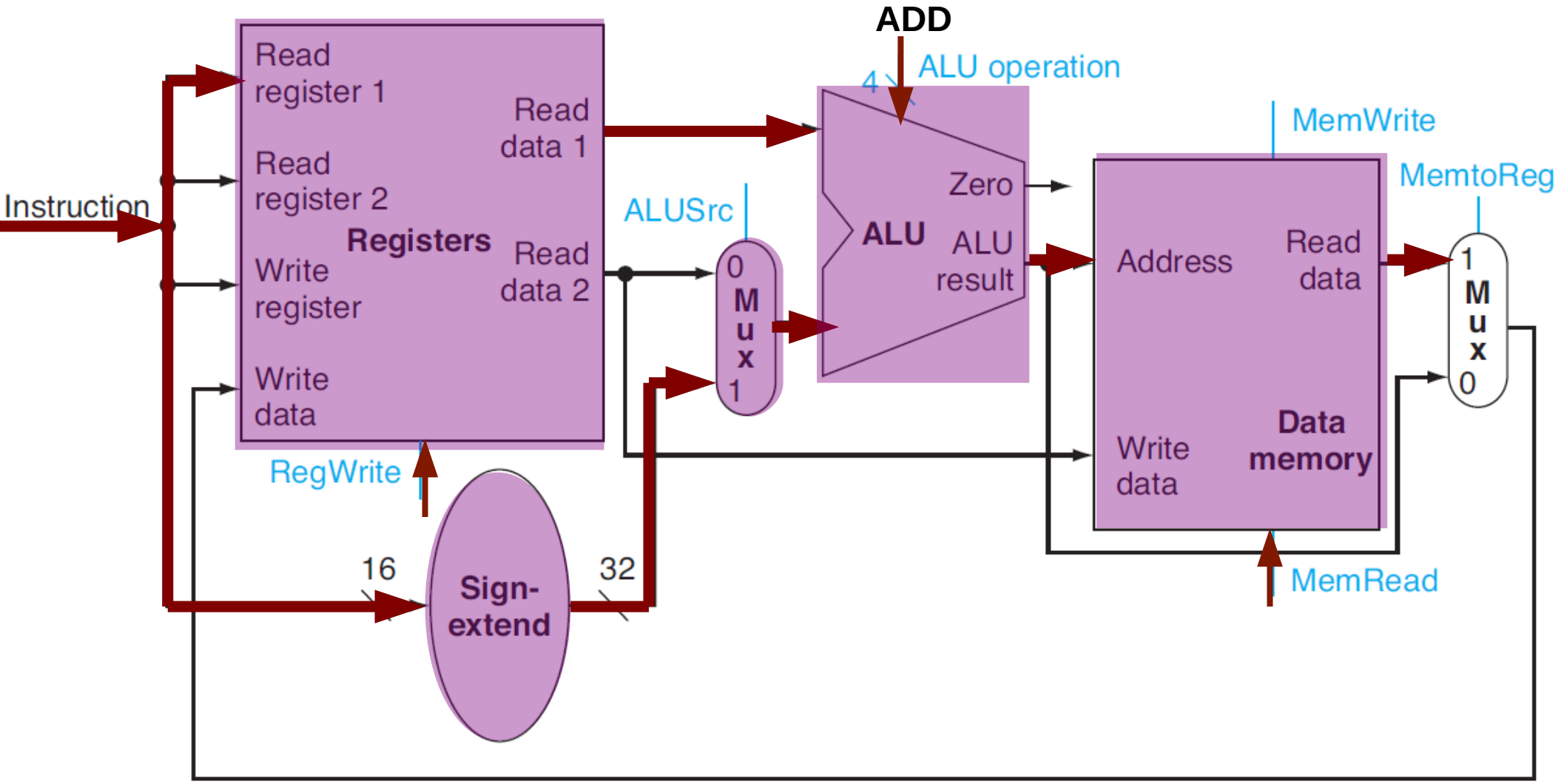
LW R1, -8(R2)

Memory Instruction – Load



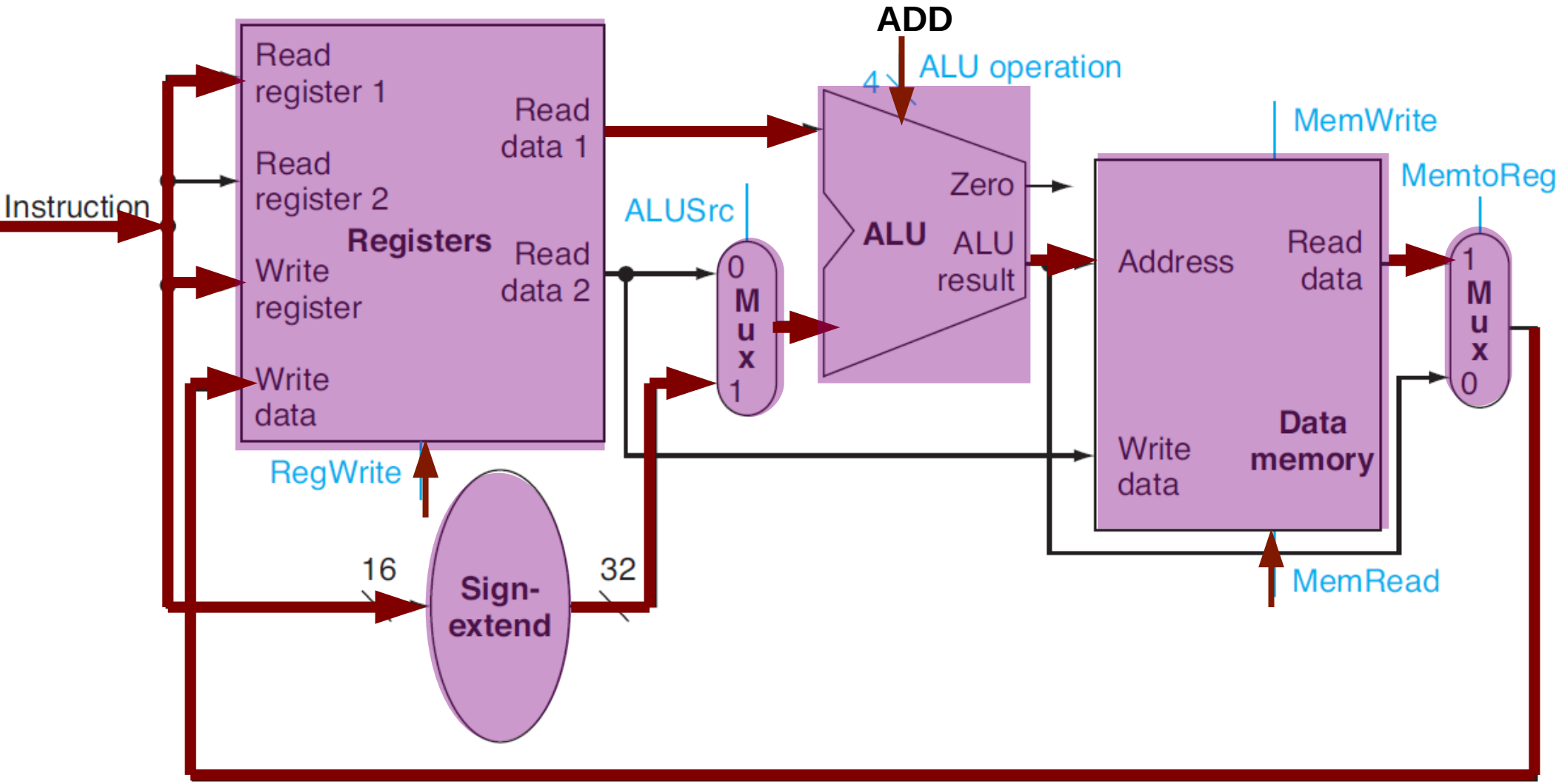
LW R1, -8(R2)

Memory Instruction – Load



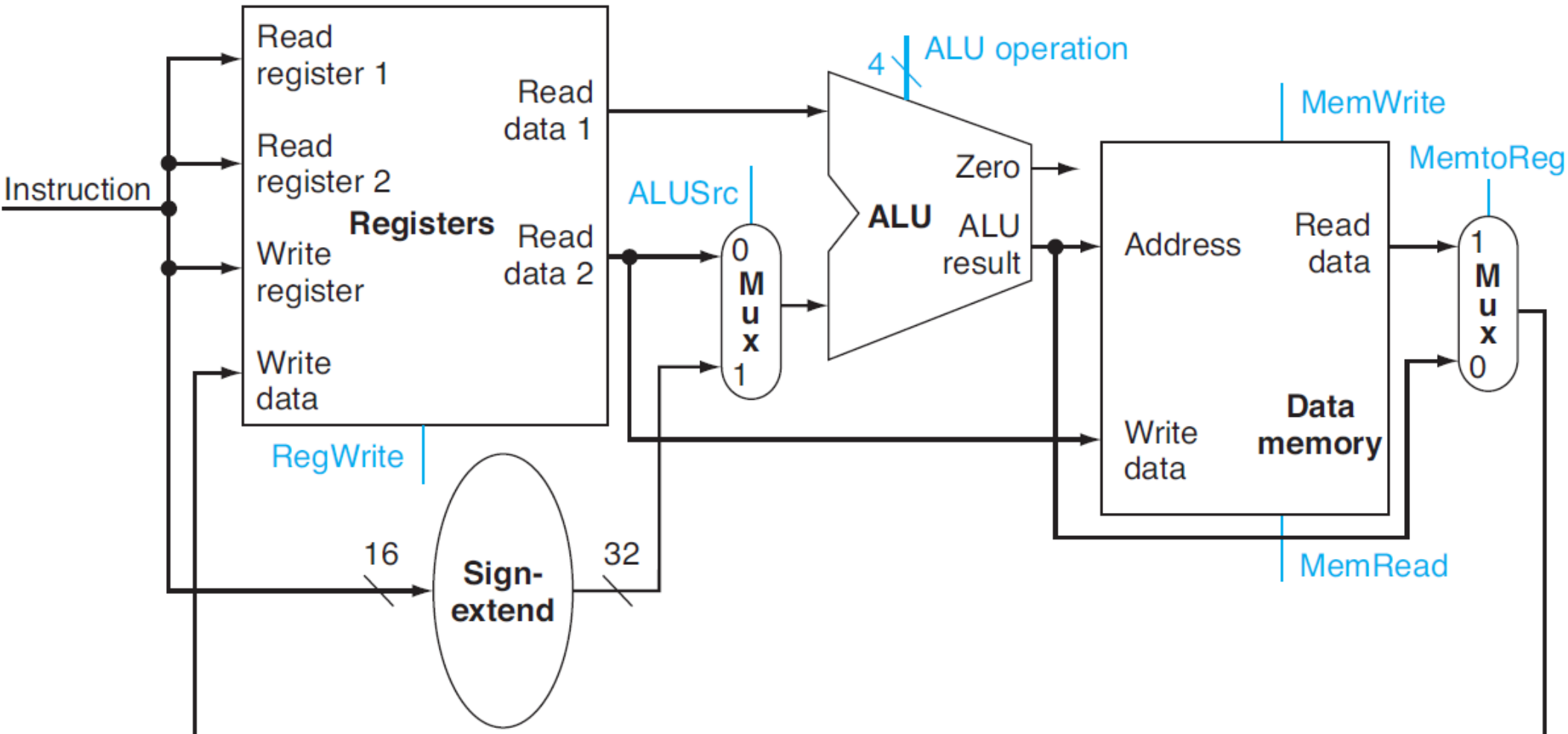
LW R1, -8(R2)

Memory Instruction – Load



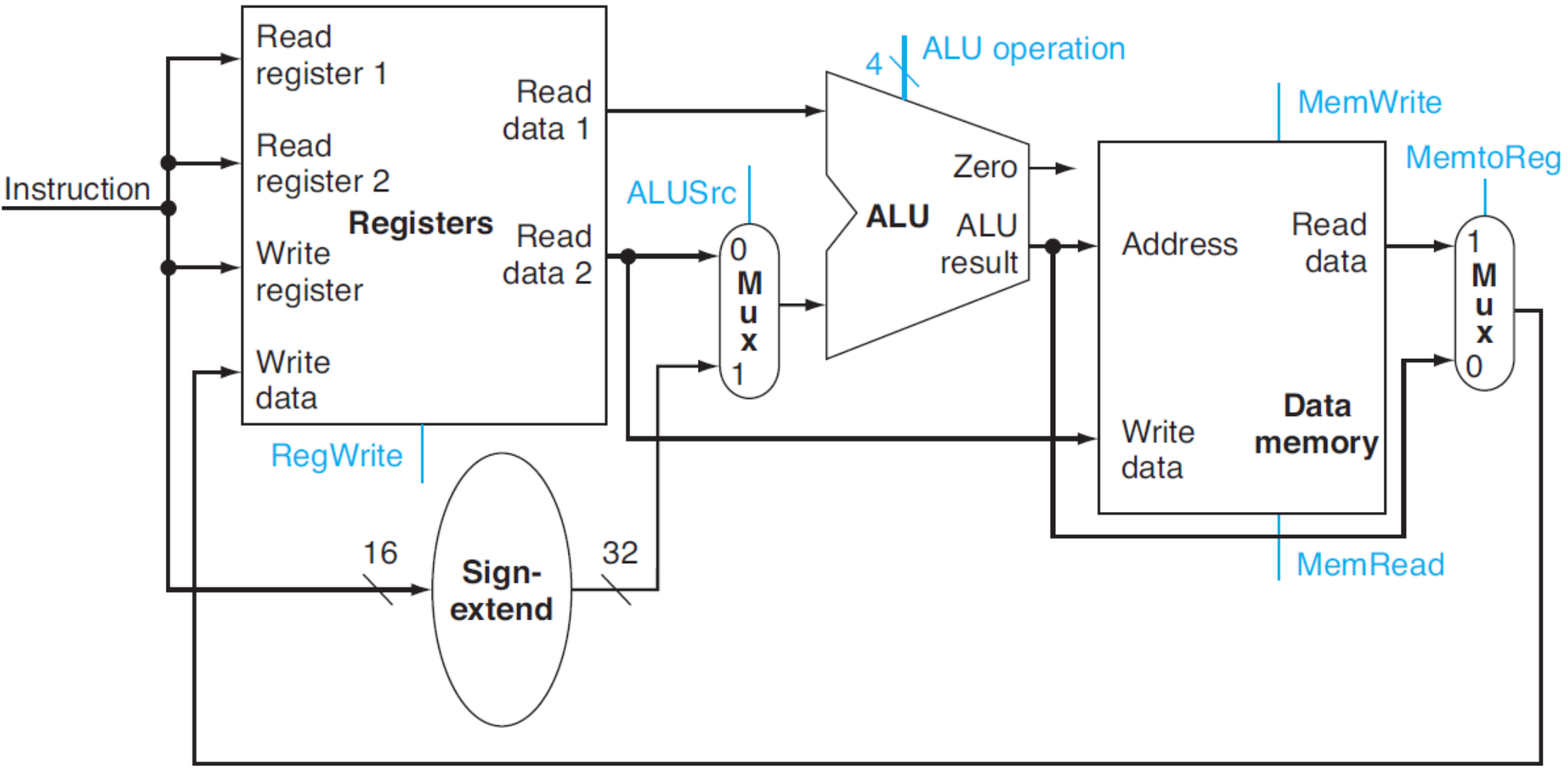
LW R1, -8(R2)

Memory Instruction – Load



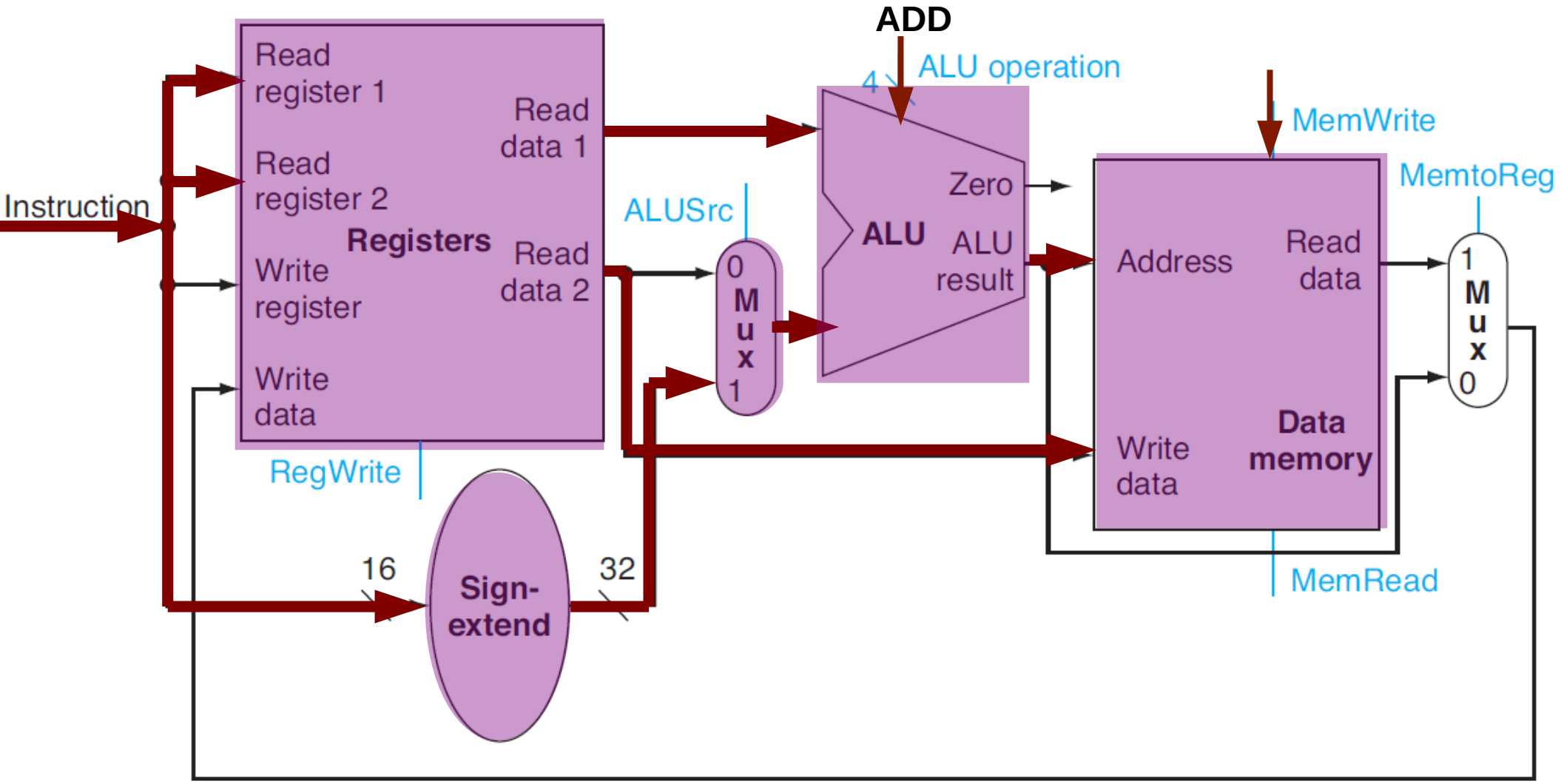
Control Signals: RegWrite=1; ALUSrc=1; ALUoperation=ADD; MemRead=1;MemWrite=0; MemToReg=1;

Memory Instruction – Store



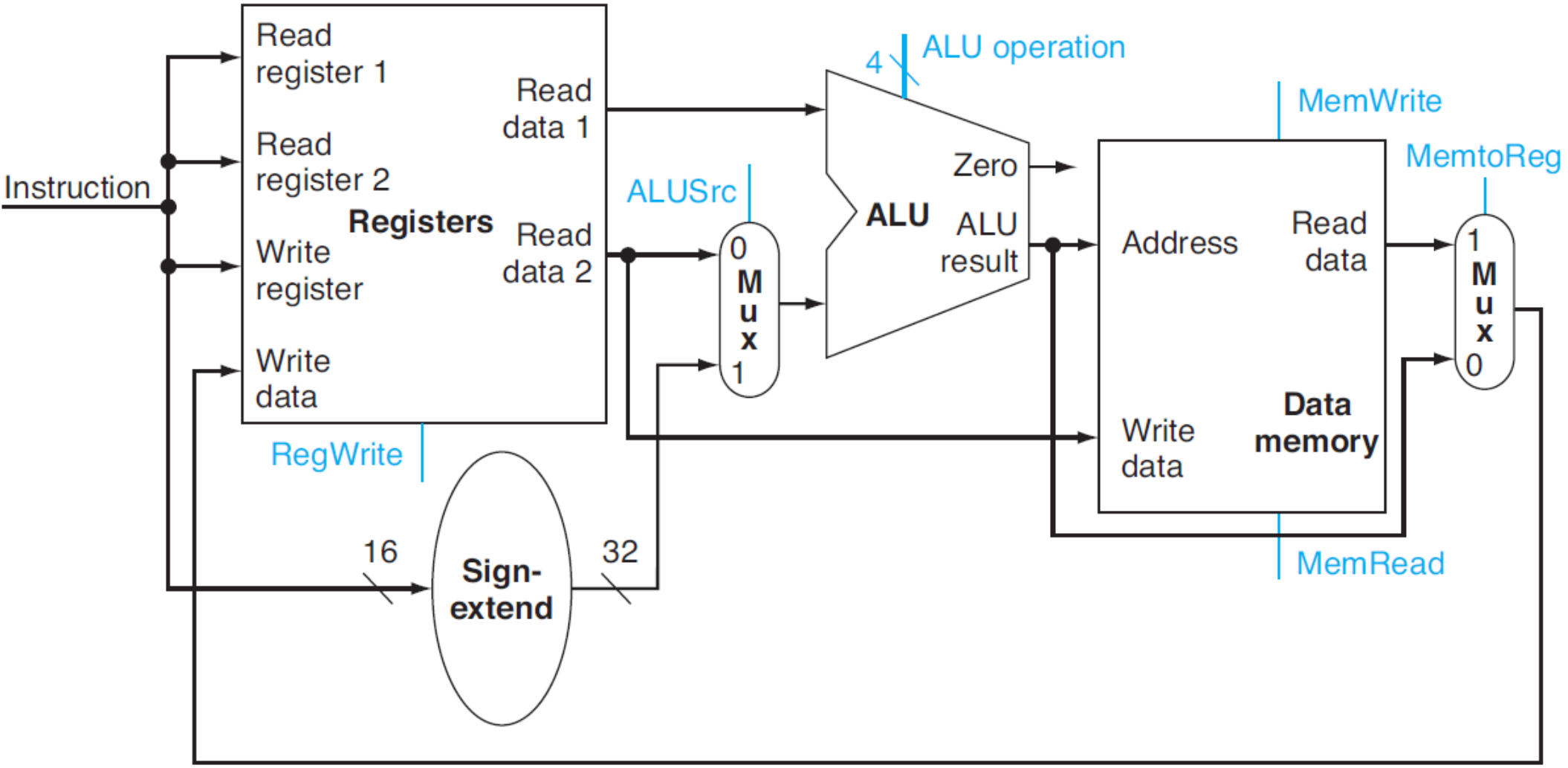
SW R1, -8(R2)

Memory Instruction – Store



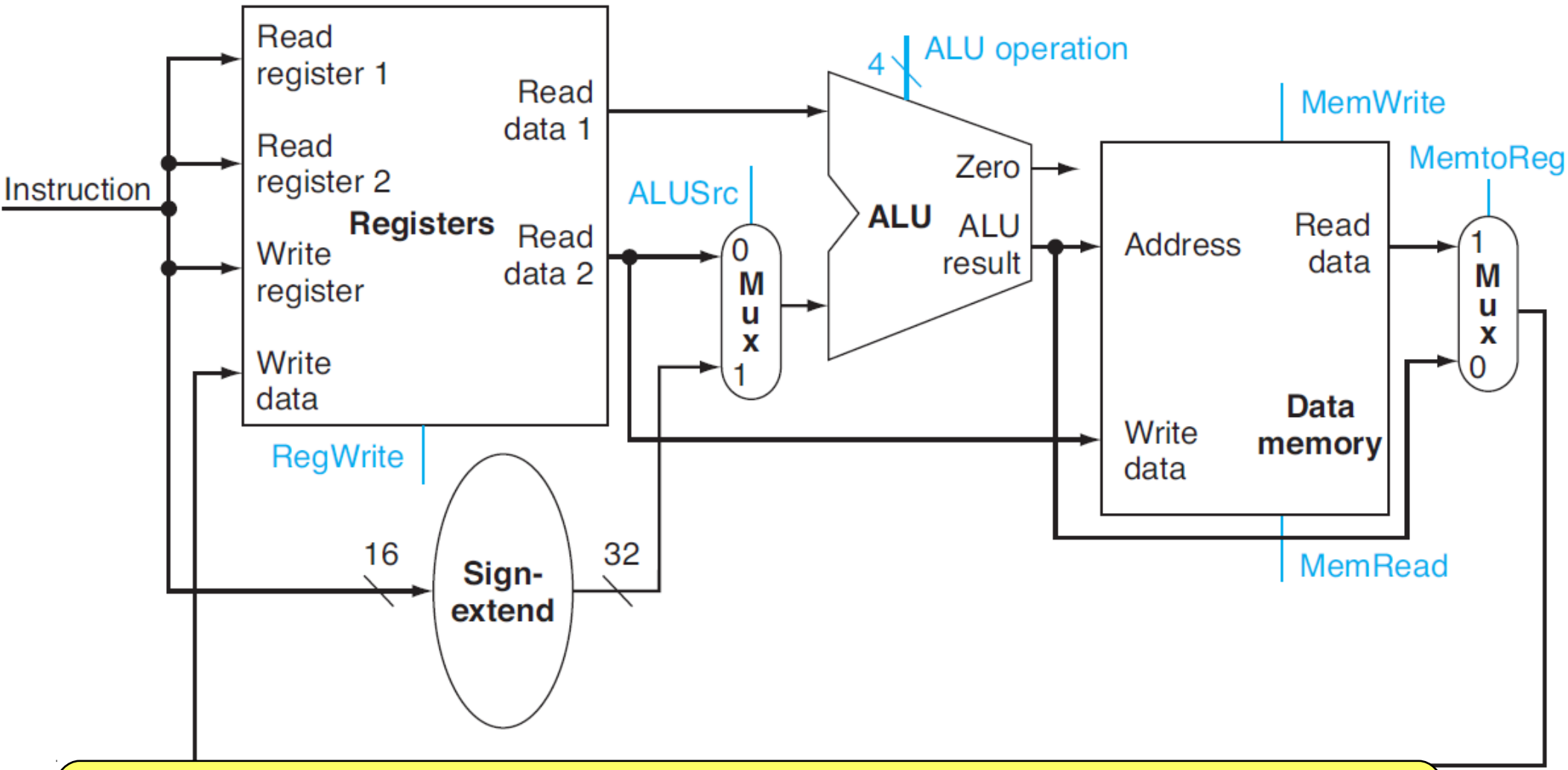
SW R1, -8(R2)

Memory Instruction – Store



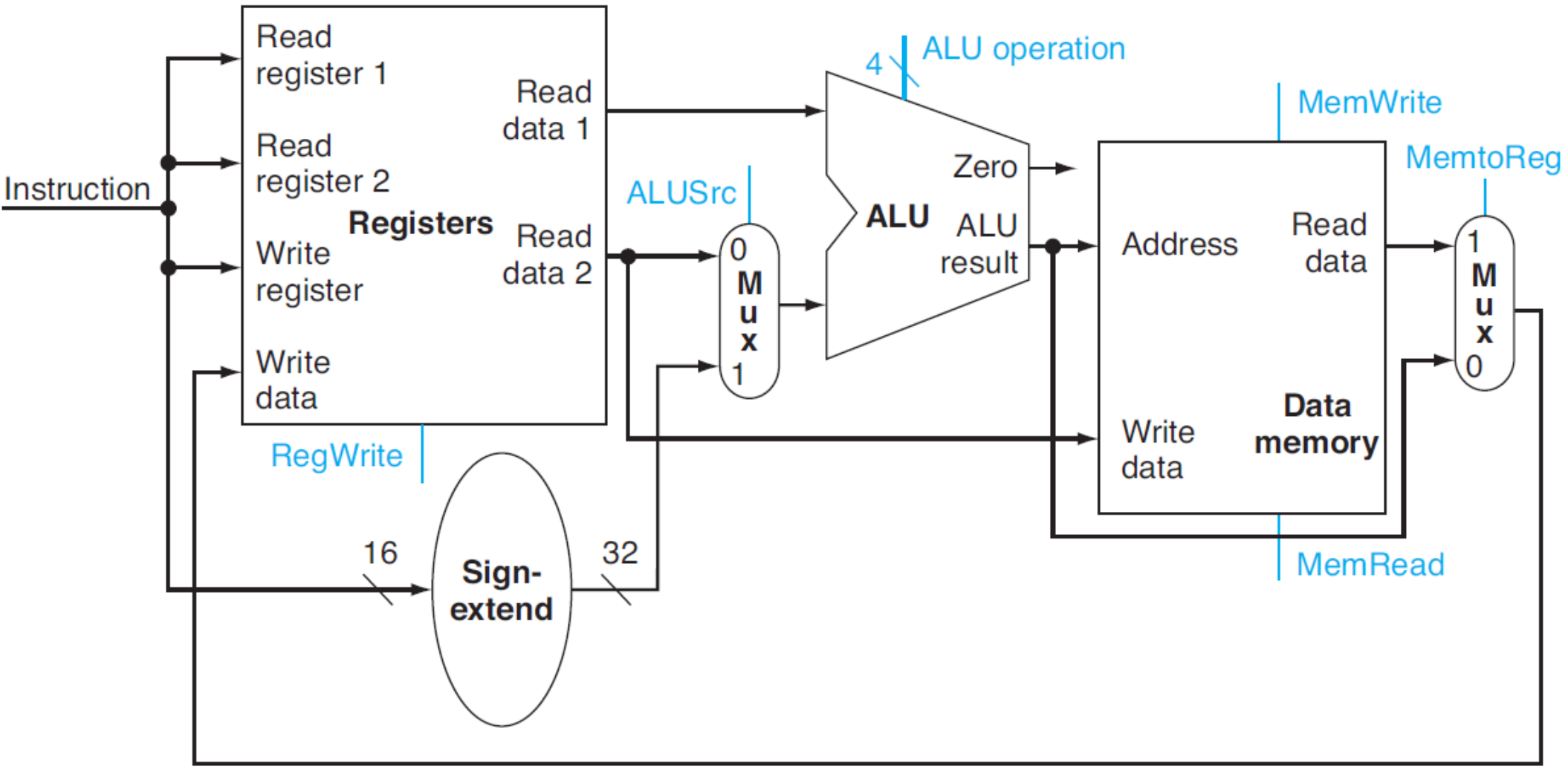
Control Signals:

Memory Instruction – Store



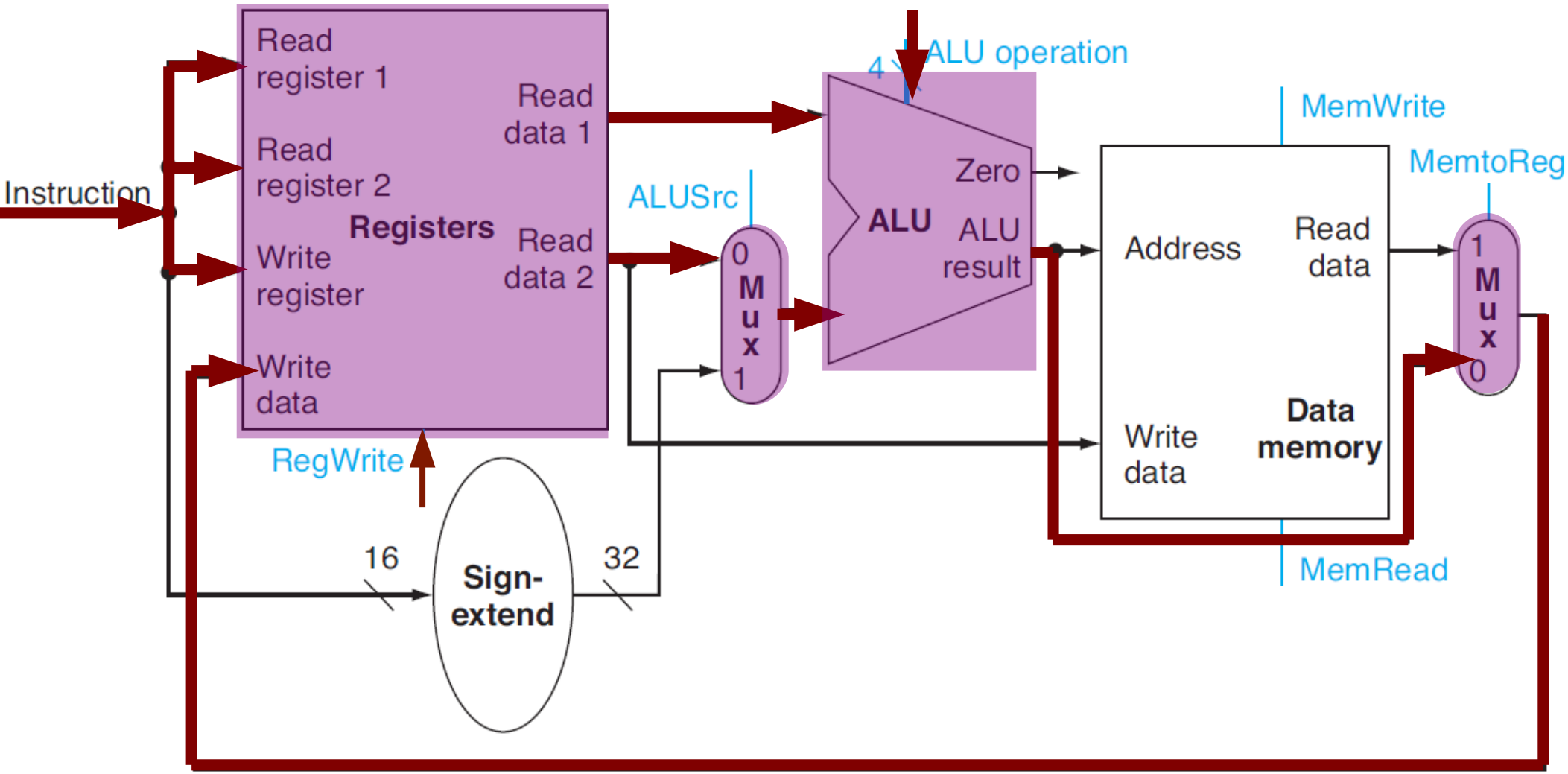
Control Signals: RegWrite=0; ALUSrc=1; ALUoperation=ADD; MemRead=0;MemWrite=1; MemToReg=X;

R Type Instruction – ADD



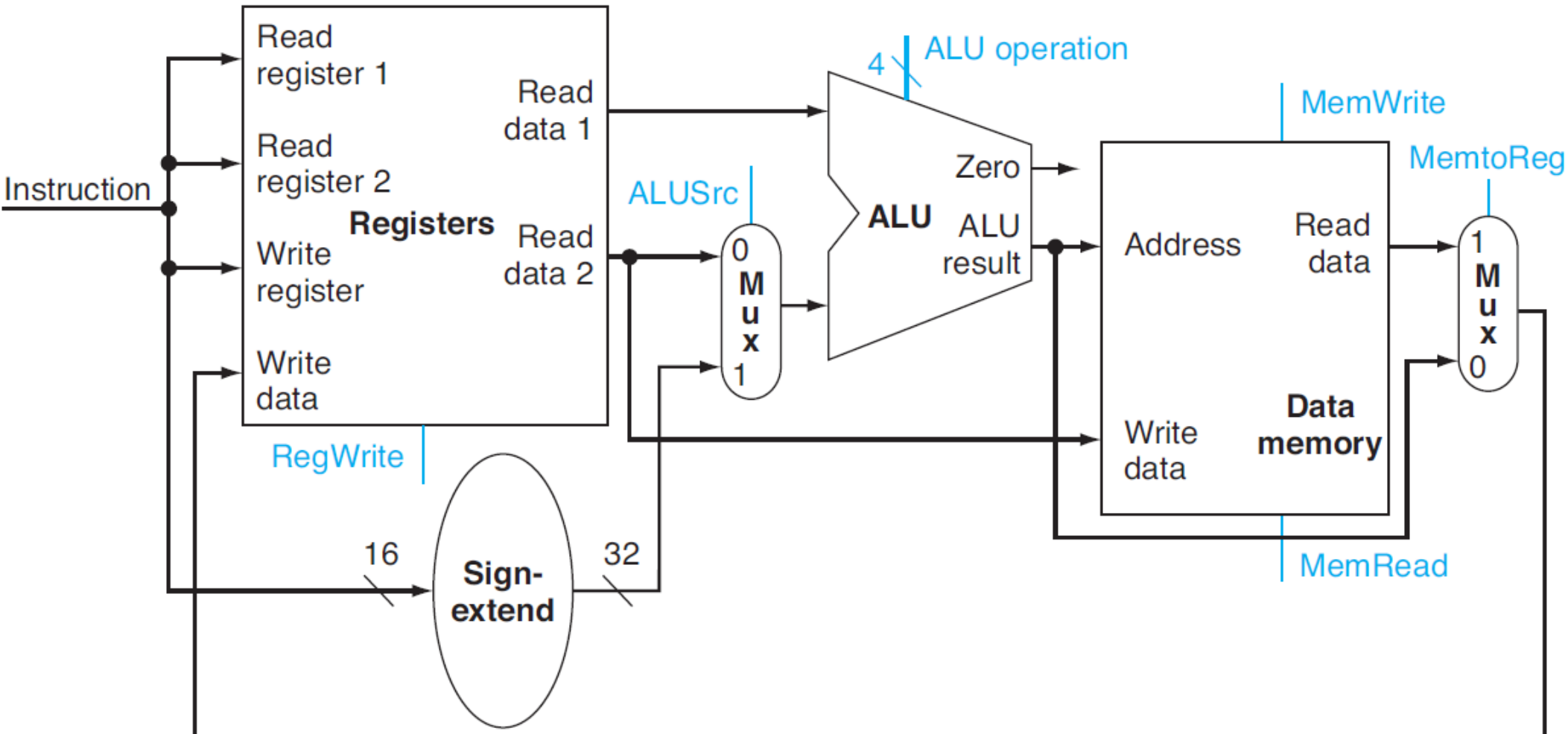
ADD R1, R2, R3

R Type Instruction – ADD



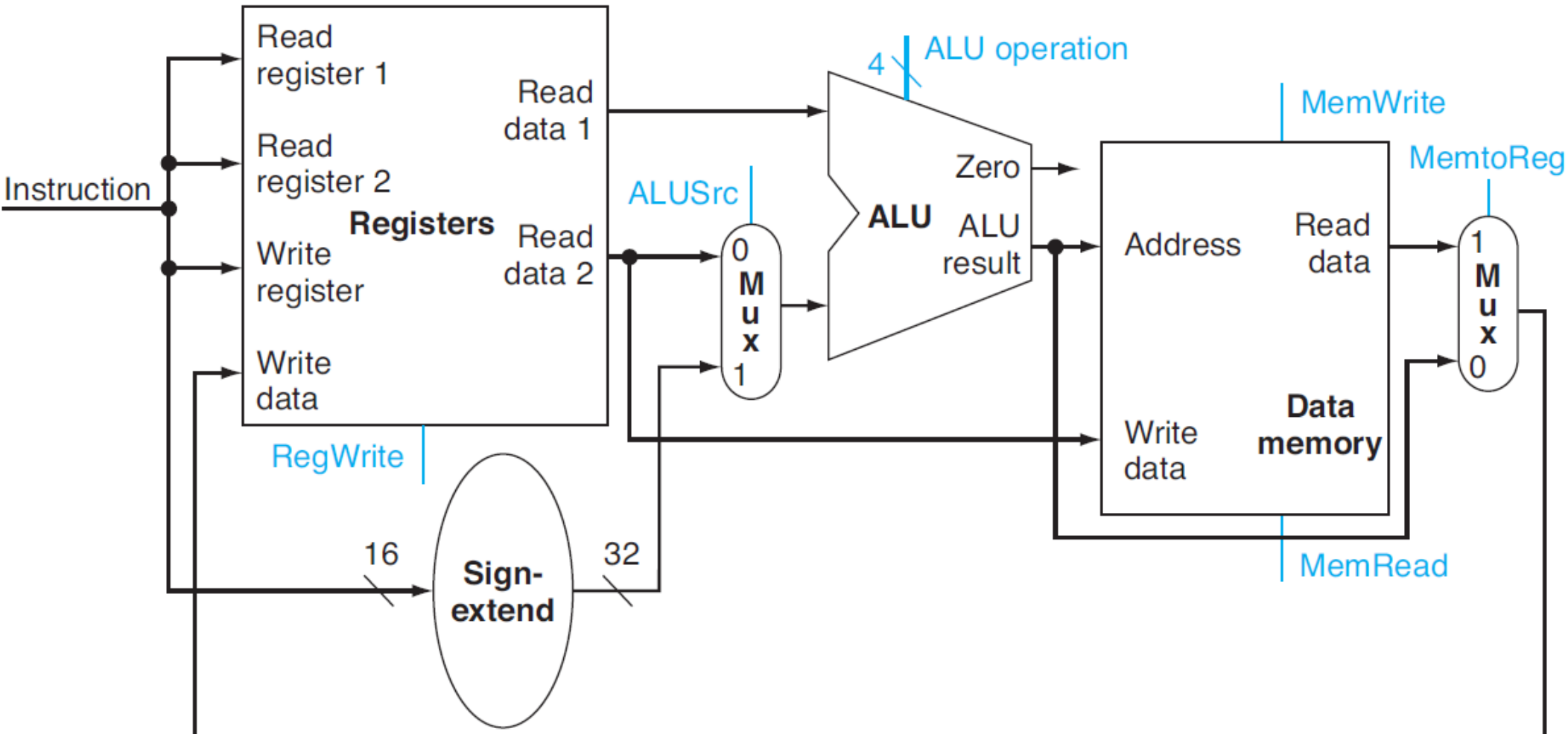
ADD R1, R2, R3

R Type Instruction – ADD



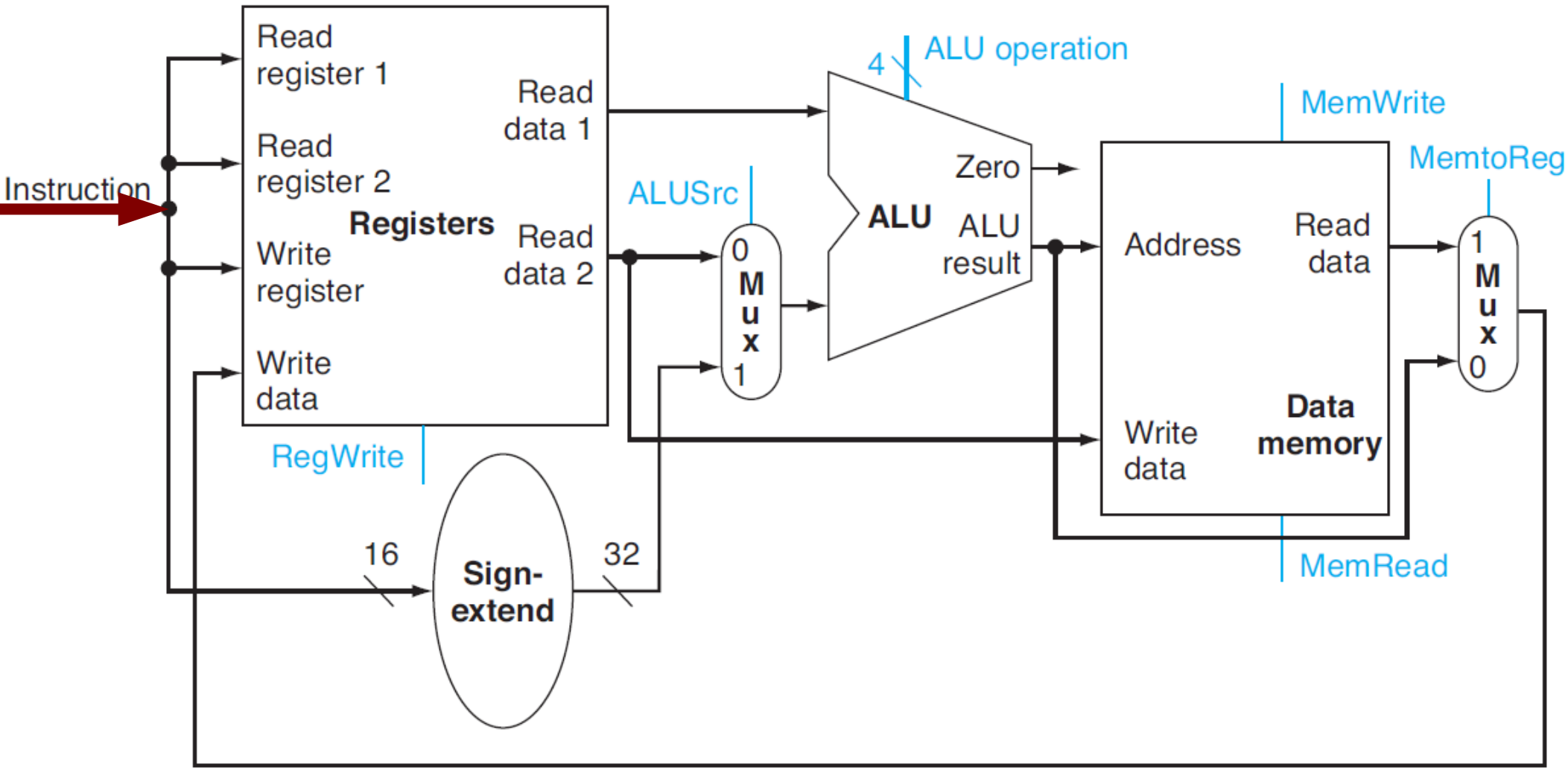
Control Signals:

R Type Instruction – ADD



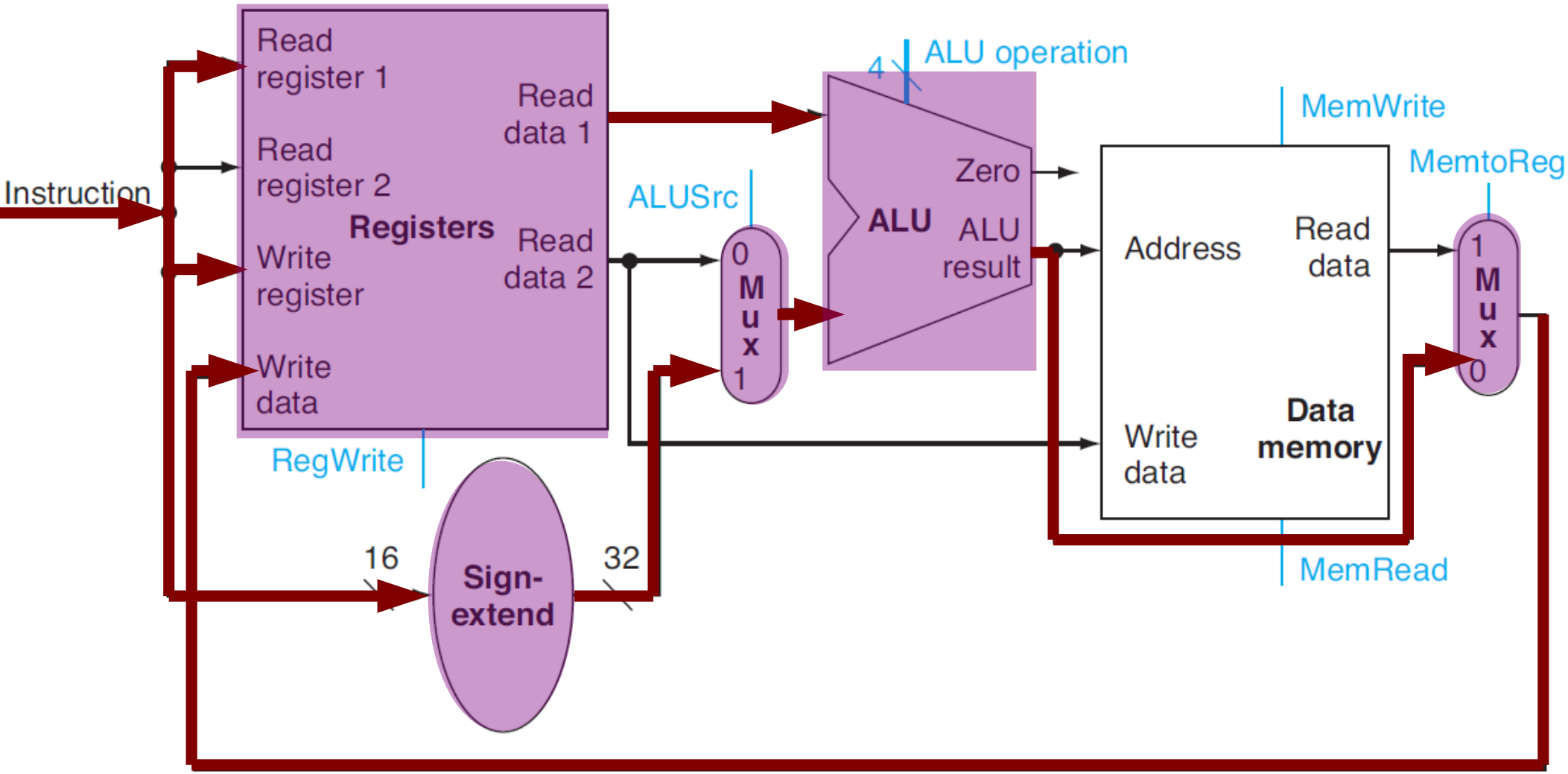
Control Signals: RegWrite=1; ALUSrc=0; ALUoperation=ADD; MemRead=X;MemWrite=X; MemToReg=0;

I Type Instruction – ADDI



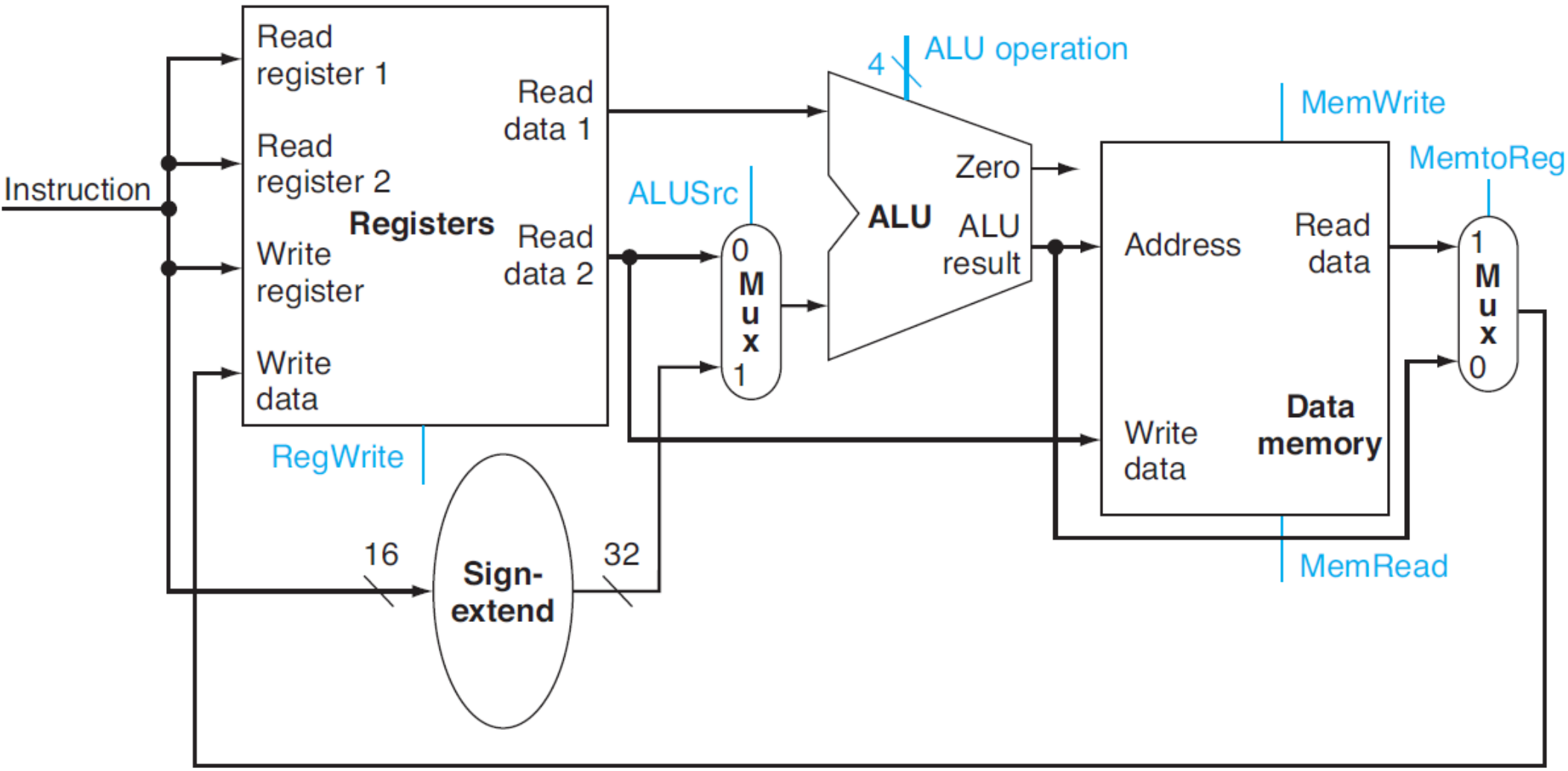
ADDI R1, R2, 13

I Type Instruction – ADDI



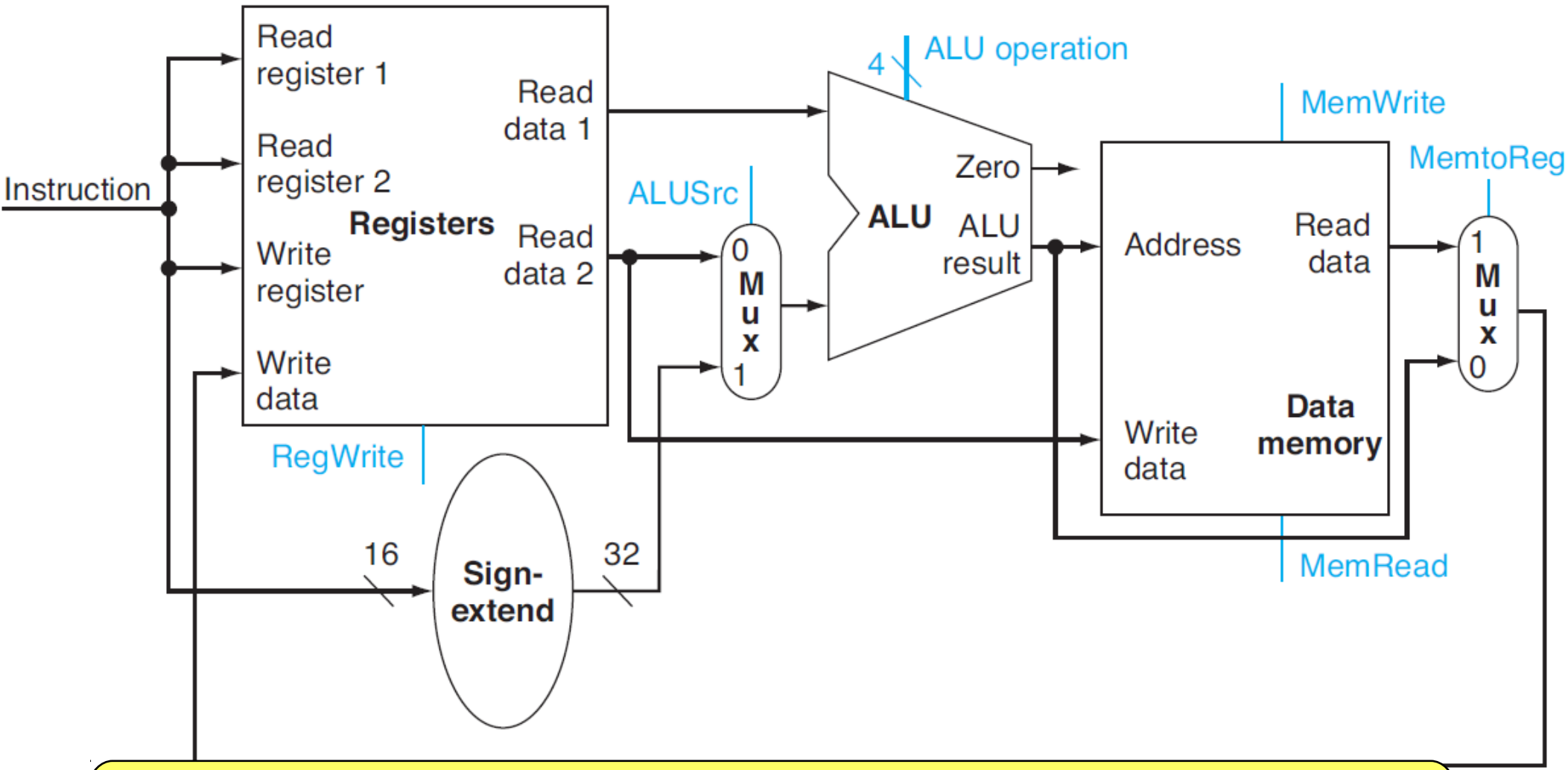
ADDI R1, R2, 13

I Type Instruction – ADDI



Control Signals:

I Type Instruction – ADDI



Control Signals: RegWrite=1; ALUSrc=1; ALUoperation=ADD; MemRead=X;MemWrite=X; MemToReg=0;

BEQ – Actions

BEQ R1, R2, -16

BEQ – Actions

- Read R1 and R2 from Register file
  - Send 1 and 2 to RF
  - RF reads contents of R1 and R2

BEQ R1, R2, -16

BEQ – Actions

- Read R1 and R2 from Register file
  - Send 1 and 2 to RF
  - RF reads contents of R1 and R2
- Send to ALU; Ask it to Subtract

BEQ R1, R2, -16

BEQ – Actions

- Read R1 and R2 from Register file
  - Send 1 and 2 to RF
  - RF reads contents of R1 and R2
- Send to ALU; Ask it to Subtract
- Read out Zero flag from ALU

BEQ R1, R2, -16

BEQ – Actions

- Read R1 and R2 from Register file
  - Send 1 and 2 to RF
  - RF reads contents of R1 and R2
- Send to ALU; Ask it to Subtract
- Read out Zero flag from ALU
- If Z flag == 0; then PC = (PC + 4) – 16

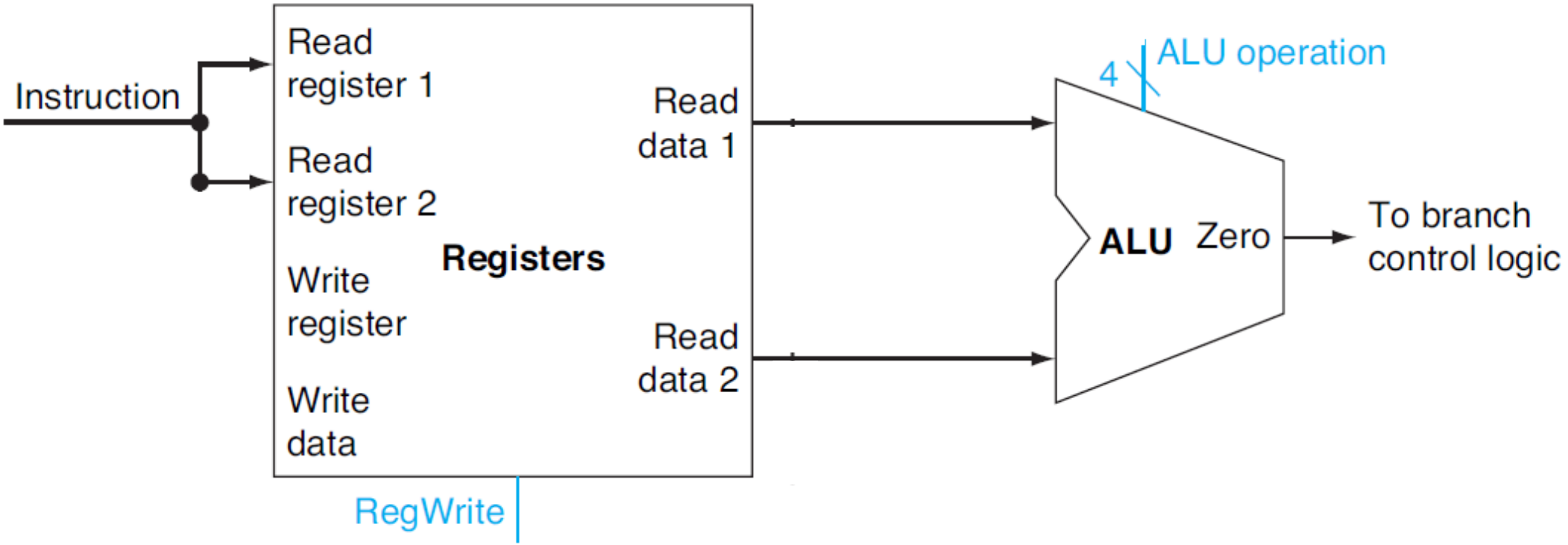
BEQ R1, R2, -16

BEQ – Actions

- Read R1 and R2 from Register file
  - Send 1 and 2 to RF
  - RF reads contents of R1 and R2
- Send to ALU; Ask it to Subtract
- Read out Zero flag from ALU
- If Z flag == 0; then PC = (PC + 4) – 16
- Else if Z flag == 1; then PC = PC + 4

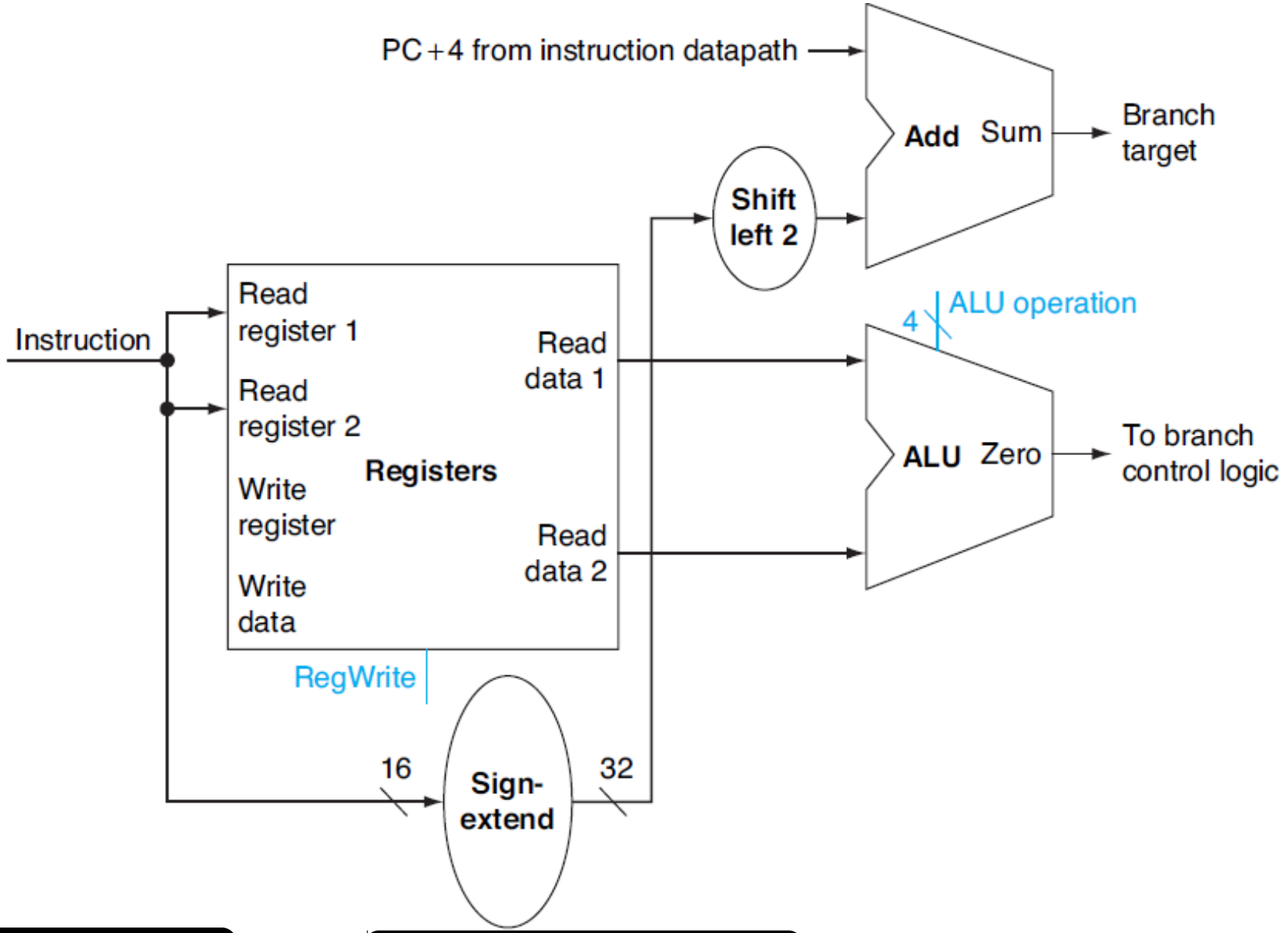
BEQ R1, R2, -16

Branches – Elements



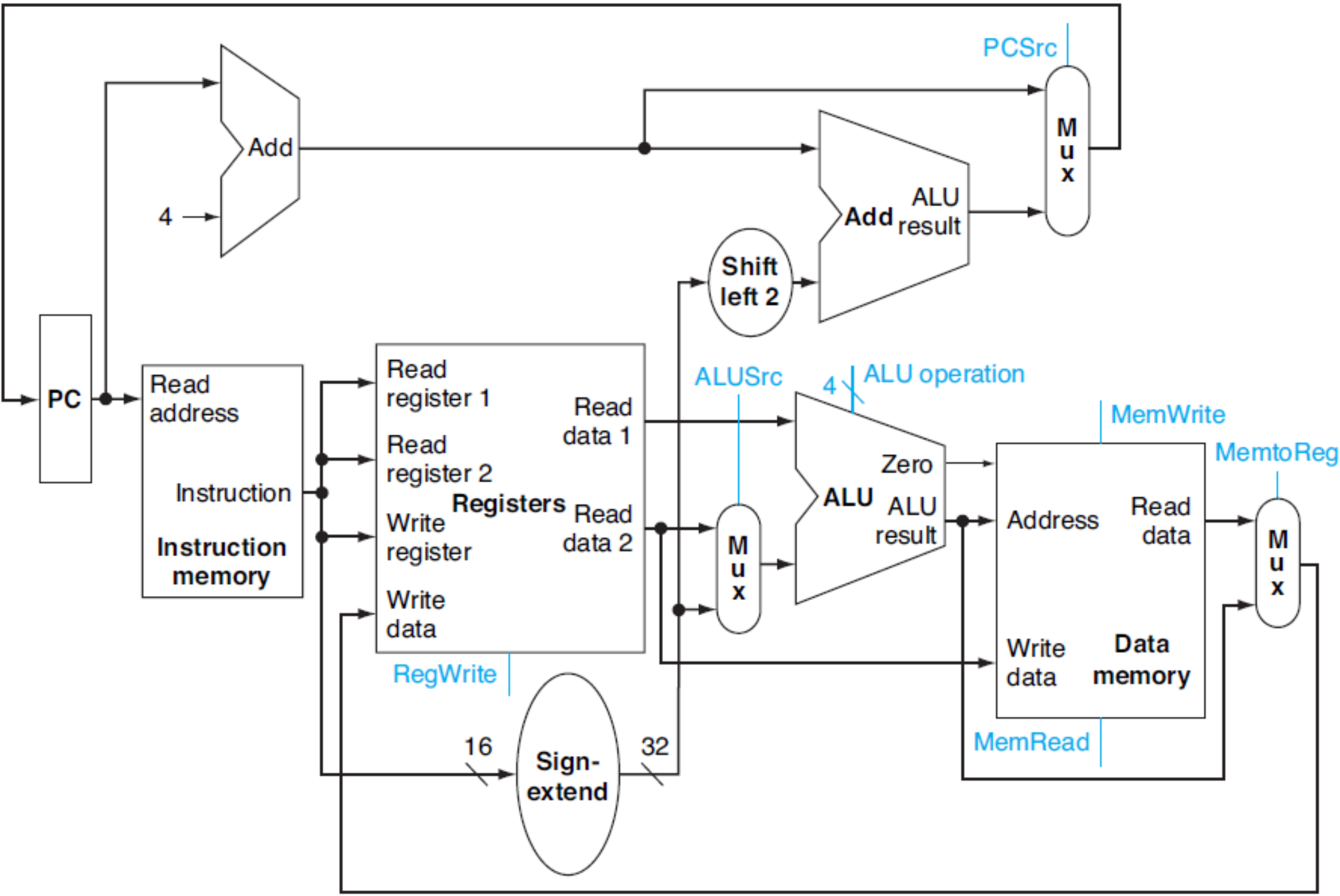
BEQ R1, R2, LABEL → **BEQ R1, R2, -16**

Branches – Elements

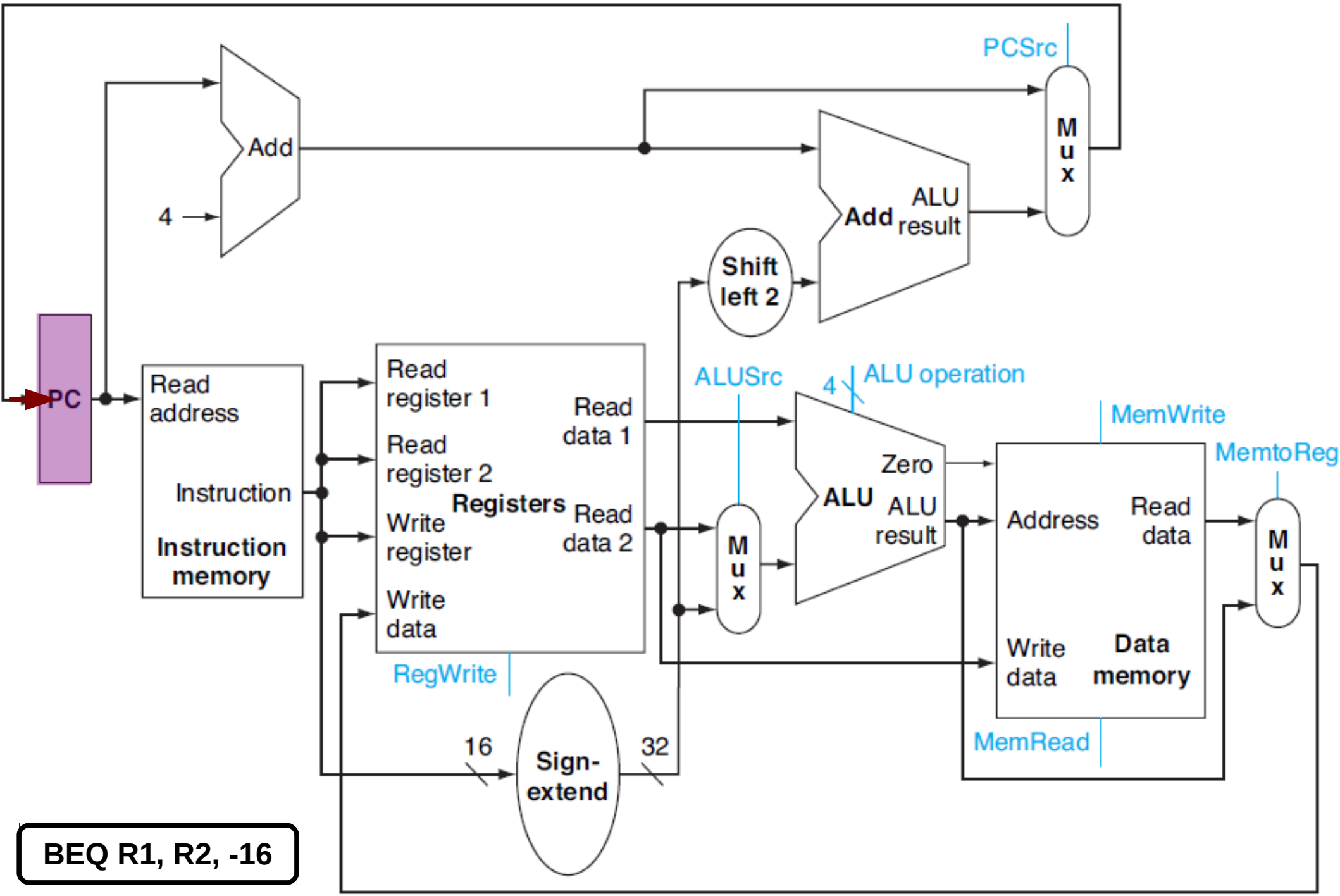


BEQ R1, R2, LABEL → **BEQ R1, R2, -16**

The MIPS Datapath

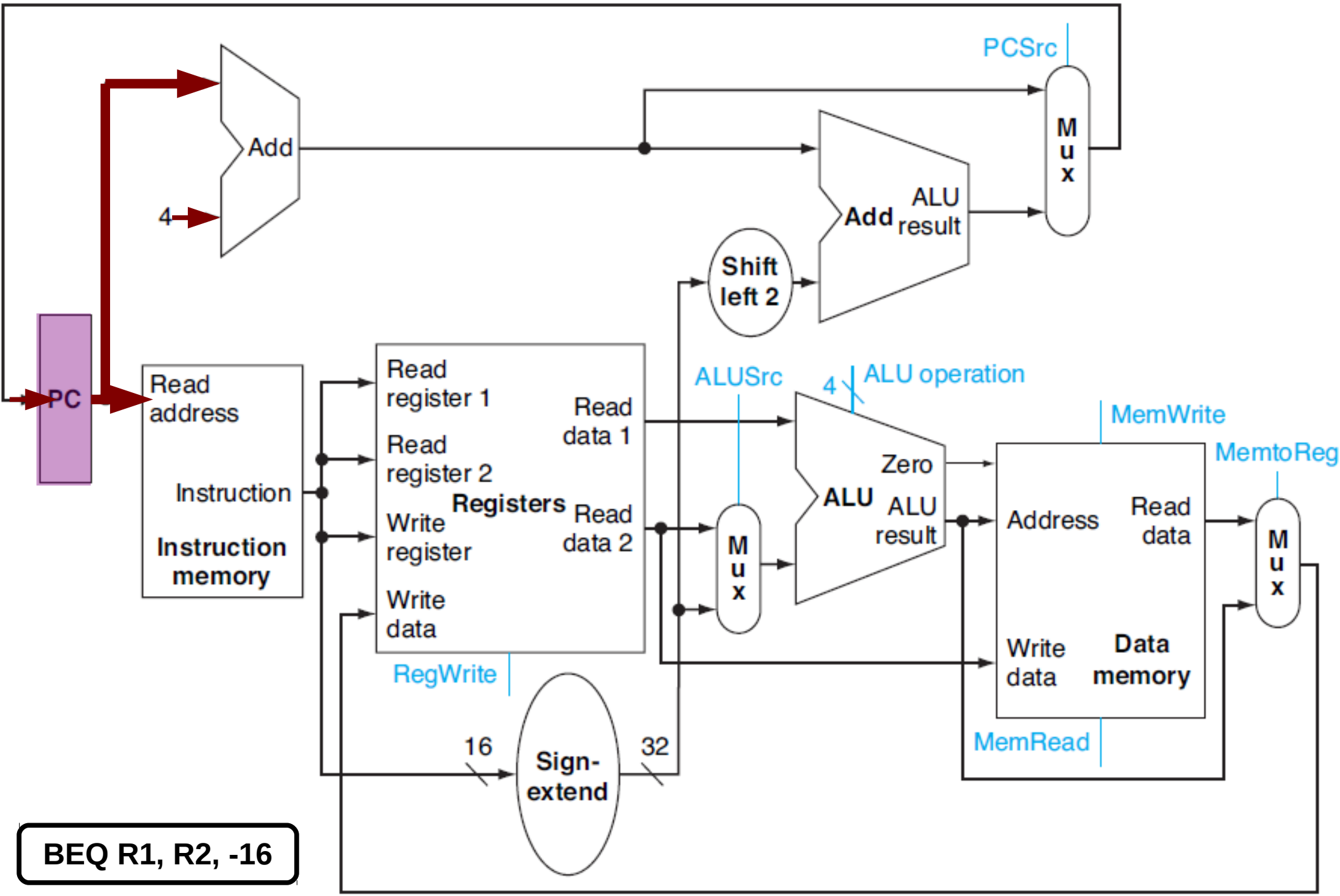


The MIPS Datapath – BEQ



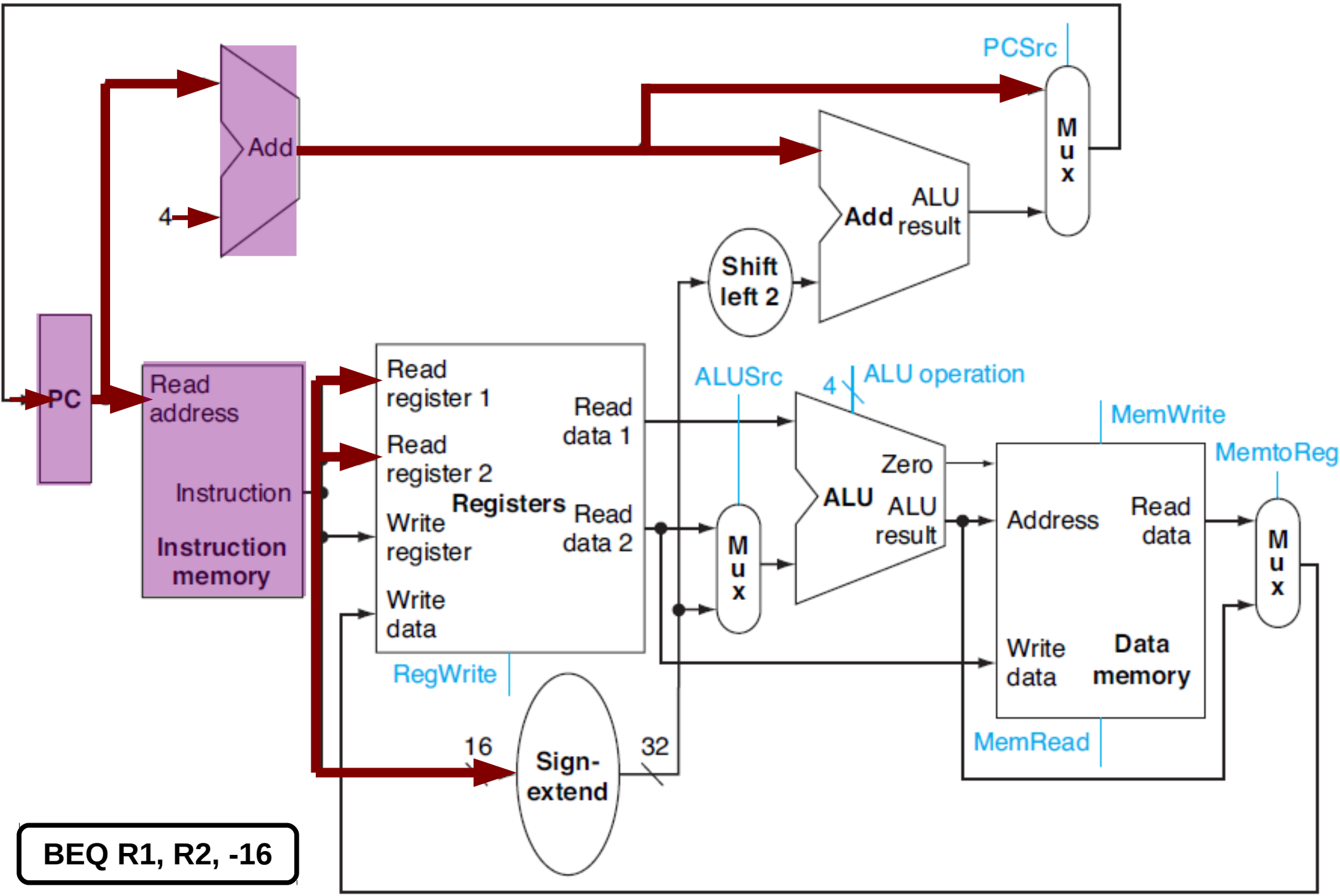
BEQ R1, R2, -16

The MIPS Datapath – BEQ



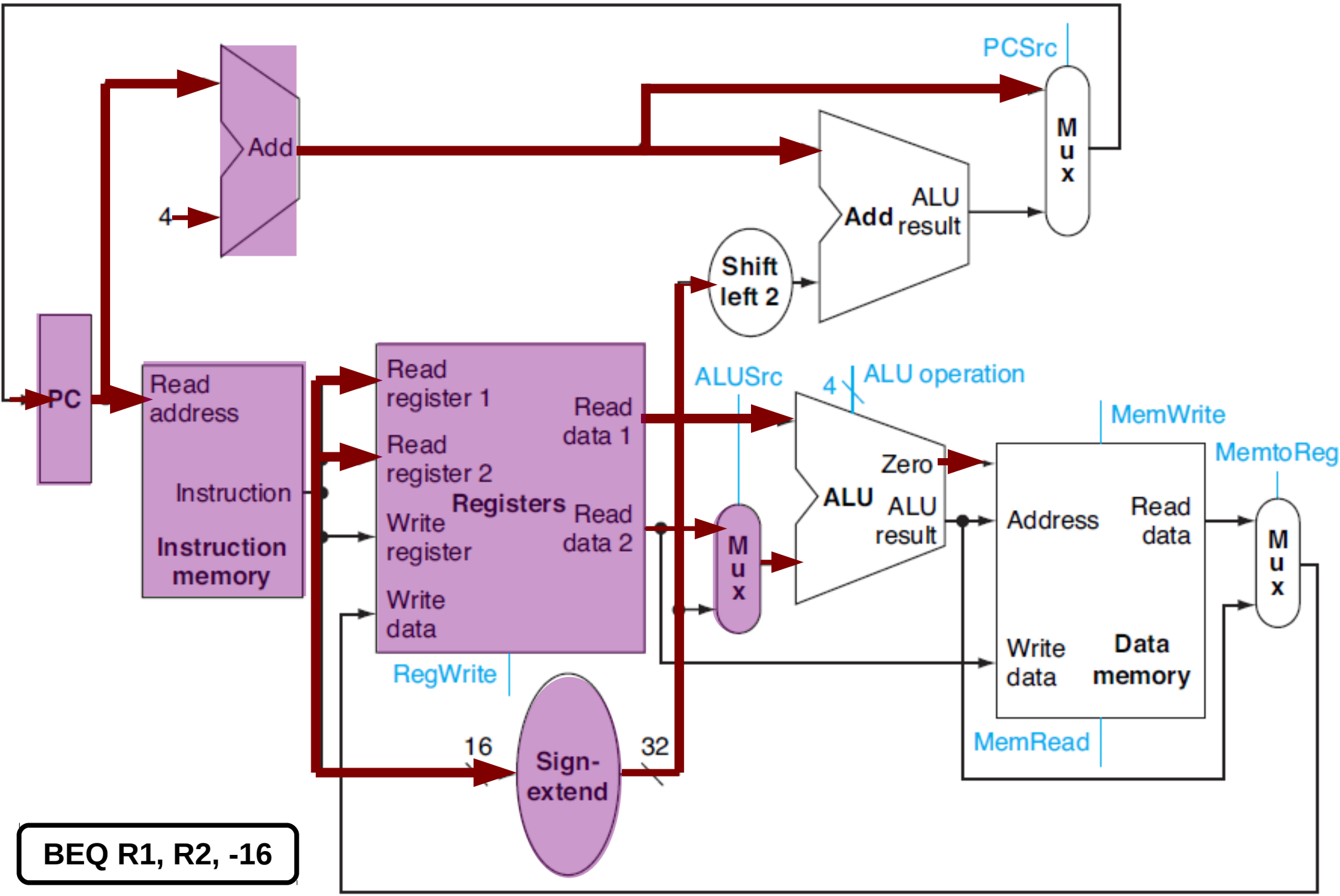
BEQ R1, R2, -16

The MIPS Datapath – BEQ



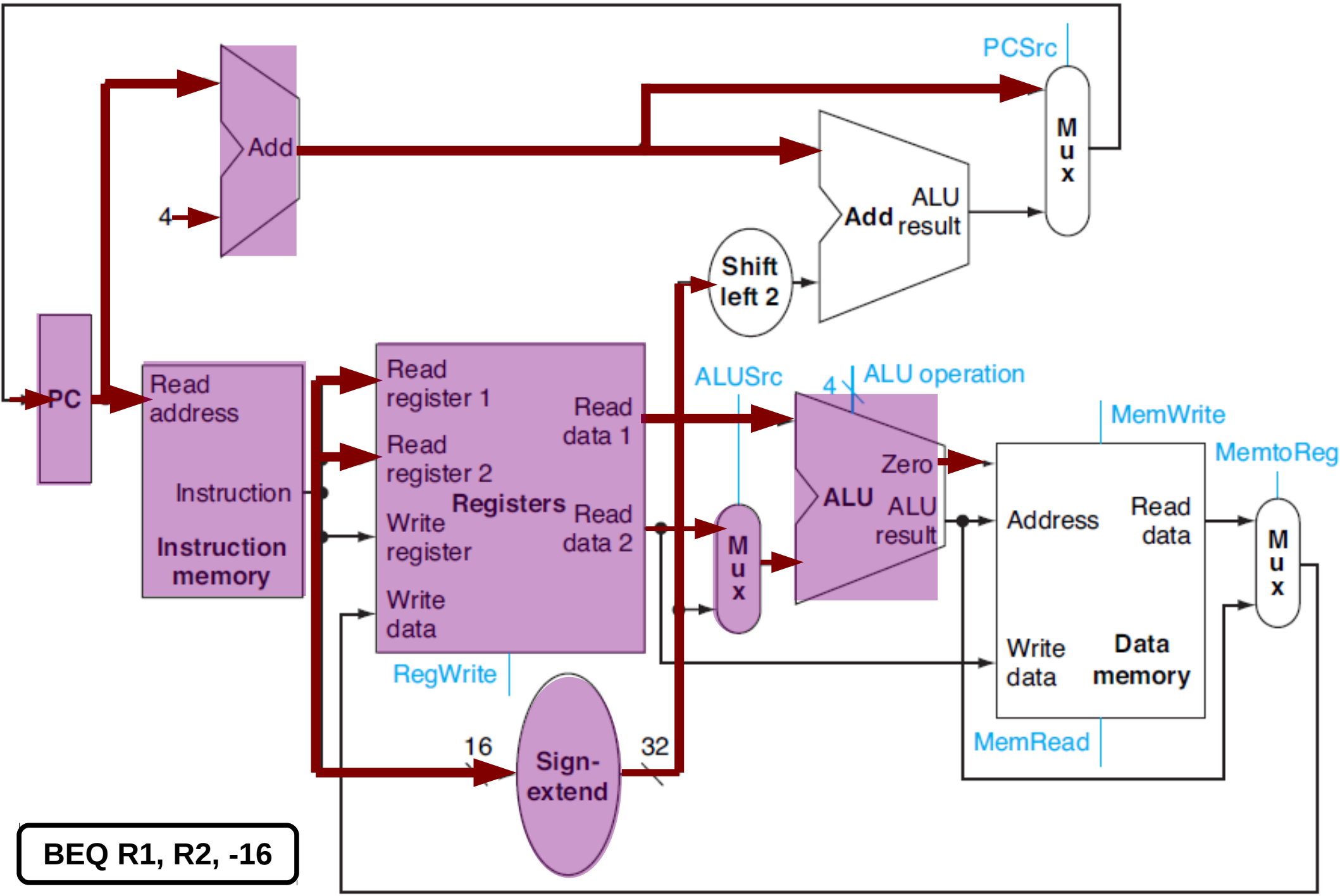
BEQ R1, R2, -16

The MIPS Datapath – BEQ



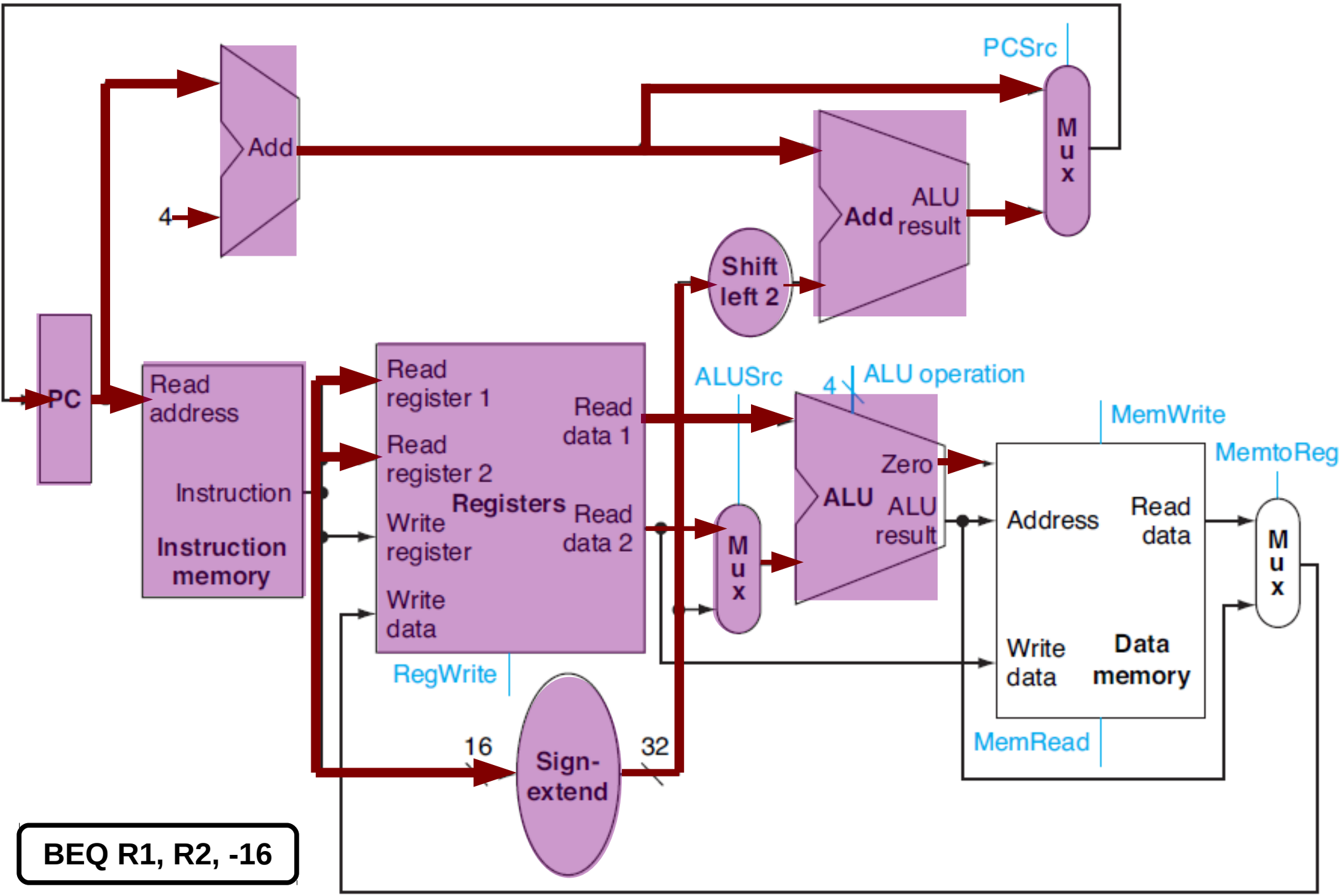
BEQ R1, R2, -16

The MIPS Datapath – BEQ



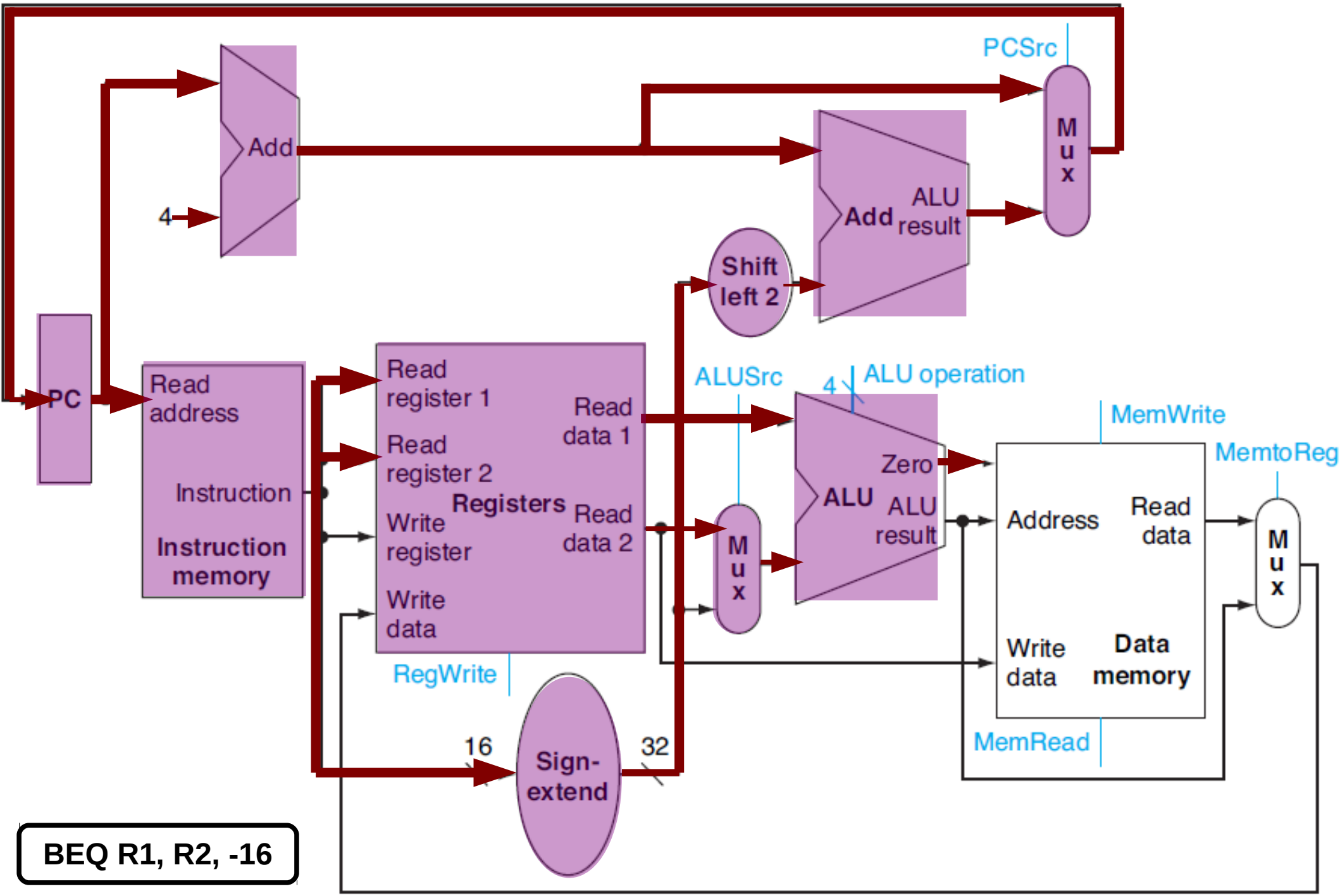
BEQ R1, R2, -16

The MIPS Datapath – BEQ



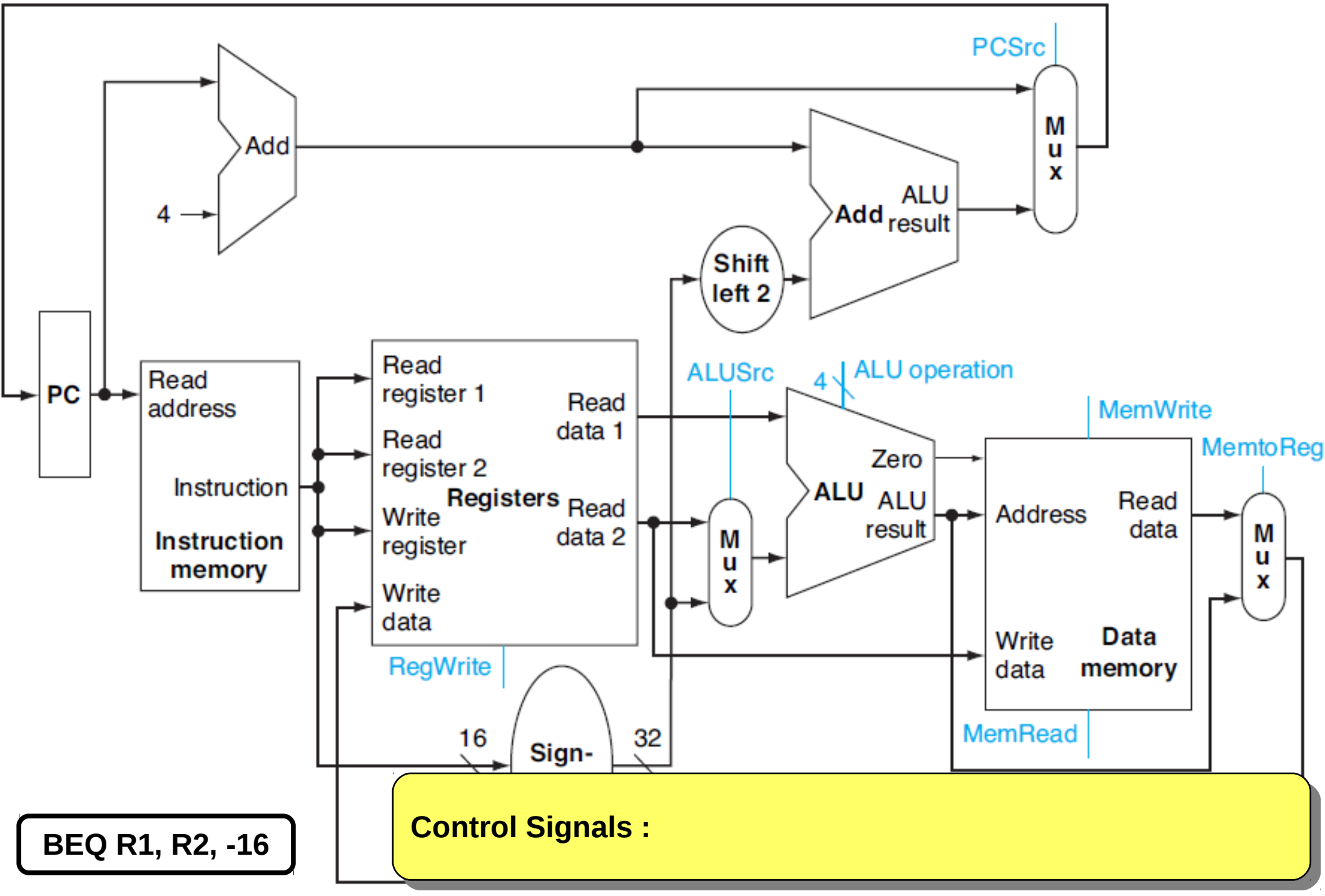
BEQ R1, R2, -16

The MIPS Datapath – BEQ



BEQ R1, R2, -16

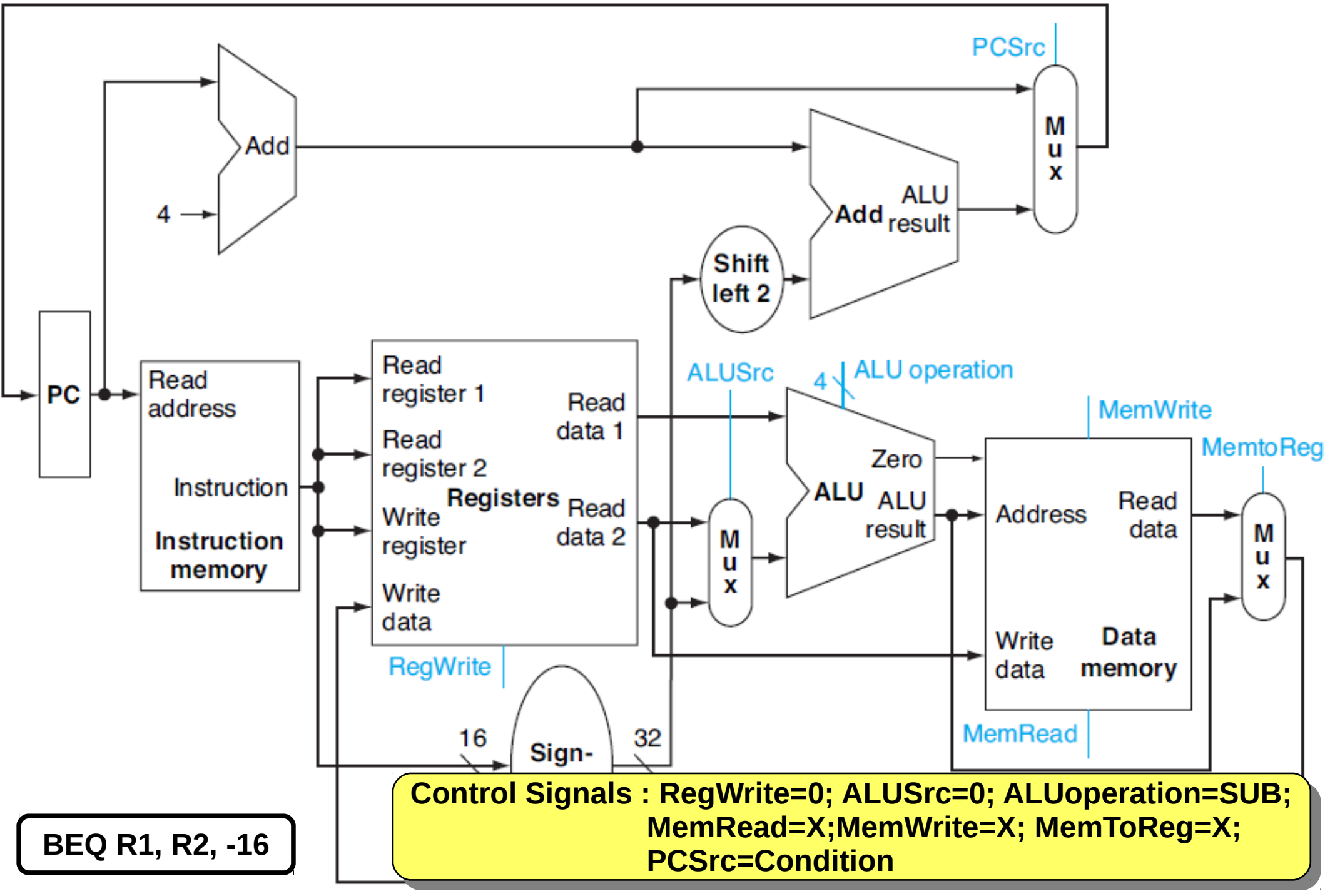
The MIPS Datapath – BEQ



BEQ R1, R2, -16

Control Signals :

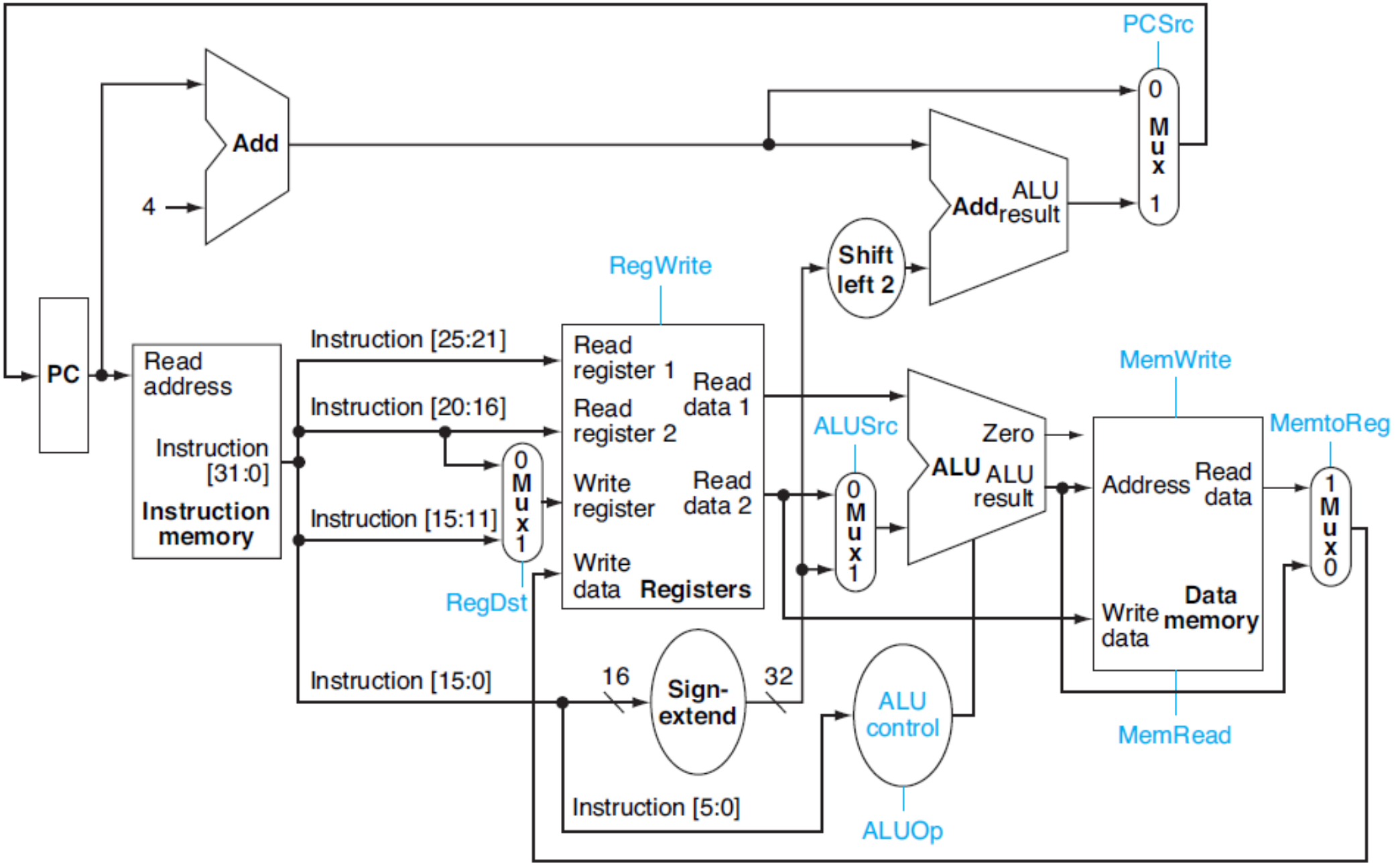
The MIPS Datapath – BEQ



BEQ R1, R2, -16

Control Signals : RegWrite=0; ALUSrc=0; ALUoperation=SUB; MemRead=X;MemWrite=X; MemToReg=X; PCSrc=Condition

MIPS Datapath and Control Lines



Module Outline

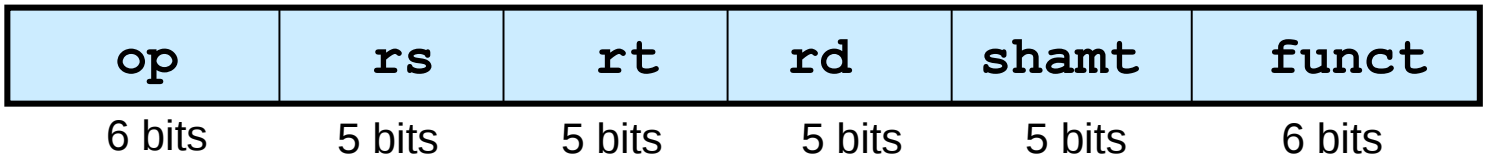
- MIPS datapath implementation
  - Register File, Instruction memory, Data memory
- Instruction interpretation and execution.
- Combinational control
- Assignment: Datapath design and Control Unit design using SystemC.

Backup

ALU Instructions

- ALU instructions (R type), Memory Transfer (effective address calculation), Branches (BEQ)

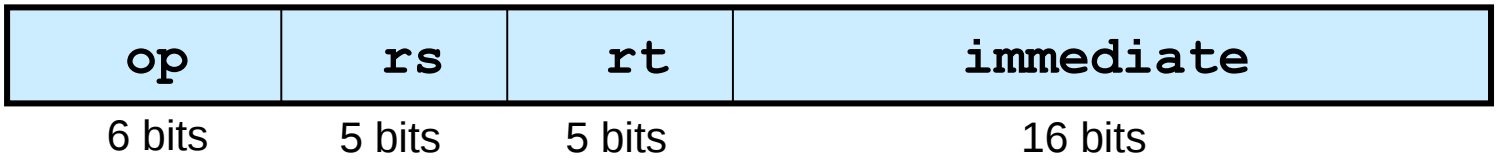
- R-type



OP rd, rs, rt

op: Opcode (class of instruction). Eg. ALU
funct: Which subunit of the ALU to activate?

- I-type



OP rt, rs, IMM

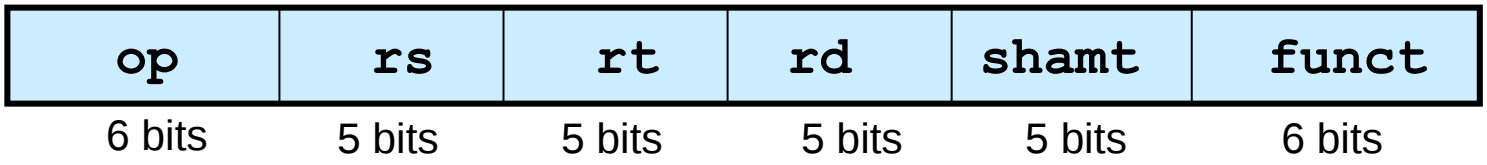
ALU Instructions

- ALU instructions (R type), Memory Transfer (effective address calculation), Branches (BEQ)
- Identified by Opcode fields and Funct fields

ALU Instructions

- ALU instructions (R type), Memory Transfer (effective address calculation), Branches (BEQ)

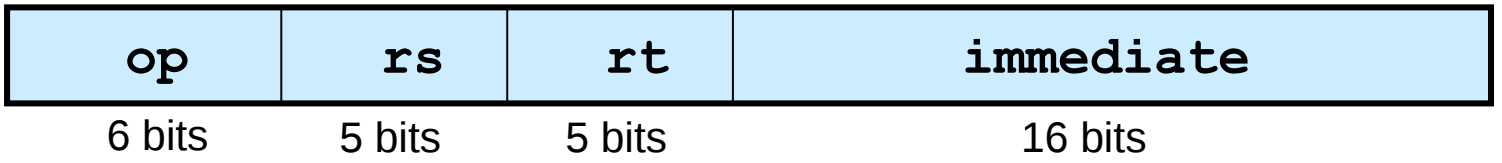
- R-type



OP rd, rs, rt

op: Opcode (class of instruction). Eg. ALU
funct: Which subunit of the ALU to activate?

- I-type



OP rt, rs, IMM