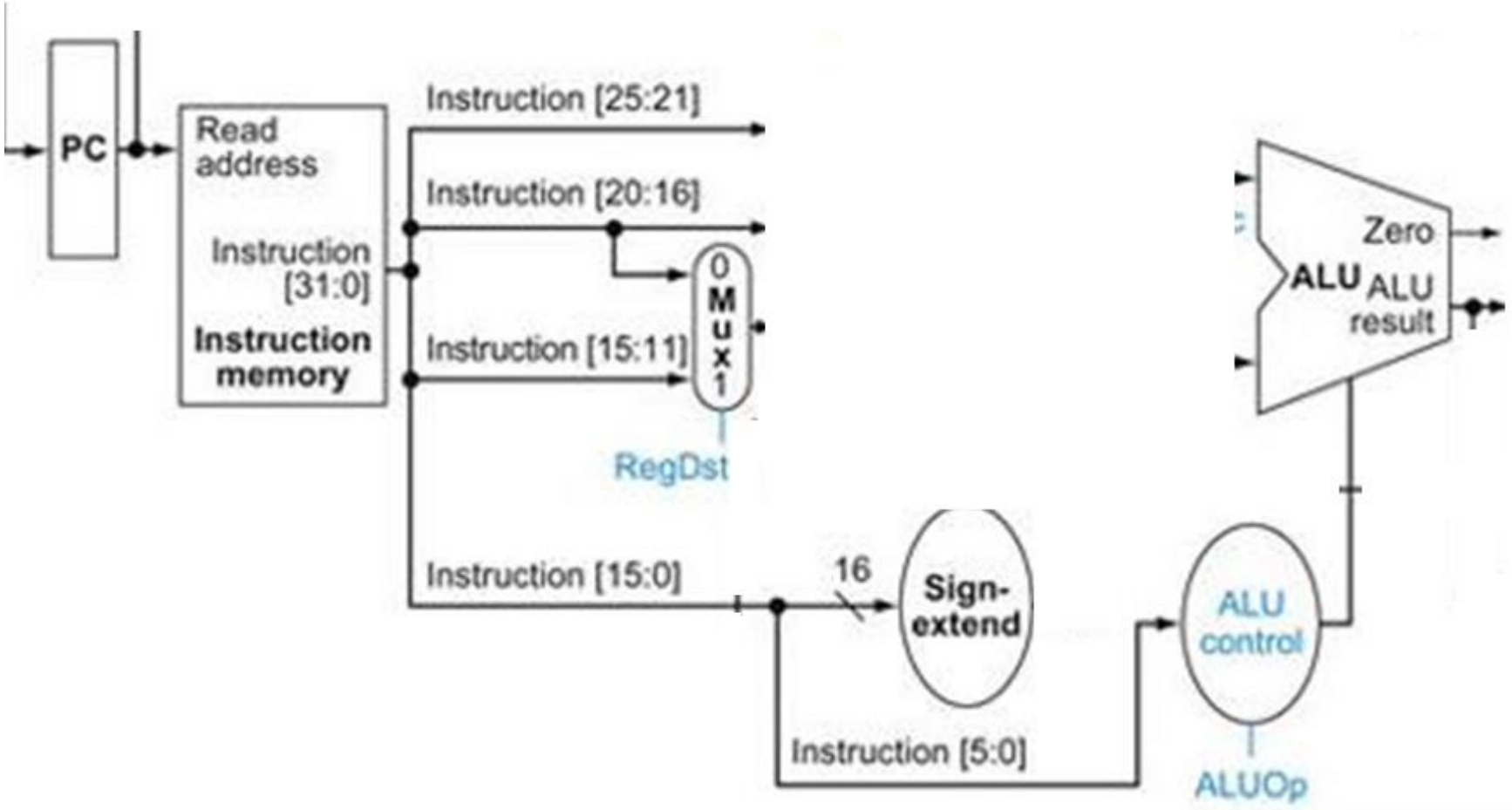
The Processor

Dr. Rajib Ranjan Maiti

CSIS, BITS-Pilani, Hyderabad

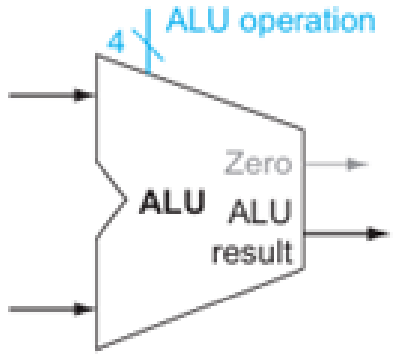
A Simple Implementation Scheme


PC
Read address
Instruction [31:0]
Instruction memory
Instruction [25:21]
Instruction [20:16]
Instruction [15:11]
0
M
u
x
1
RegDst
Instruction [15:0]
16
Sign-extend
Instruction [5:0]
ALU control
ALUOp
Zero
ALU
ALU result

An implementation of our MIPS subset

- The MIPS **ALU defines** the 6 following combinations of four control inputs:

ALU control lines	Function

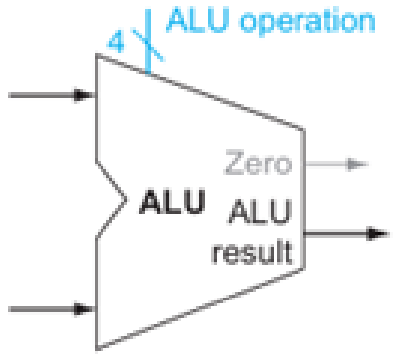


- Depending on the **instruction class**, the ALU needs to perform one of the first five functions
- For the R-type instructions,
  - the ALU needs to perform one of the five actions (AND, OR, subtract, add, or set on less than),
  - The actions depends on **6-bit funct** (or function) value

An implementation of our MIPS subset

- The MIPS **ALU defines** the 6 following combinations of four control inputs:

ALU control lines	Function
0000	AND

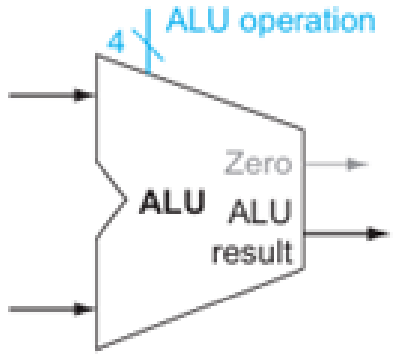


- Depending on the instruction class, the ALU needs to perform one of the first five functions
- For the R-type instructions,
  - the ALU needs to perform one of the five actions (AND, OR, subtract, add, or set on less than),
  - The actions depends on **6-bit funct** (or function) value

An implementation of our MIPS subset

- The MIPS **ALU defines** the 6 following combinations of four control inputs:

ALU control lines	Function
0000	AND
0001	OR

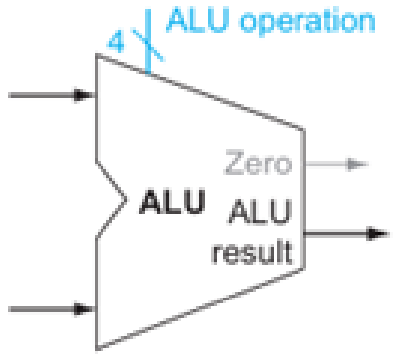


- Depending on the instruction class, the ALU needs to perform one of the first five functions
- For the R-type instructions,
  - the ALU needs to perform one of the five actions (AND, OR, subtract, add, or set on less than),
  - The actions depends on **6-bit funct** (or function) value

An implementation of our MIPS subset

- The MIPS **ALU defines** the 6 following combinations of four control inputs:

ALU control lines	Function
0000	AND
0001	OR
0010	Add

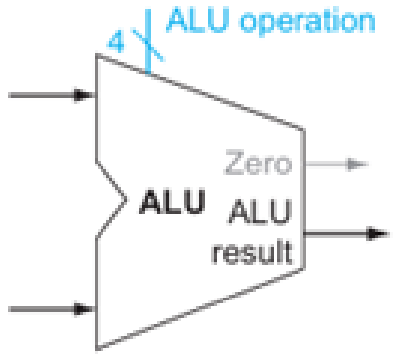


- Depending on the instruction class, the ALU needs to perform one of the first five functions
- For the R-type instructions,
  - the ALU needs to perform one of the five actions (AND, OR, subtract, add, or set on less than),
  - The actions depends on **6-bit funct** (or function) value

An implementation of our MIPS subset

- The MIPS **ALU defines** the 6 following combinations of four control inputs:

ALU control lines	Function
0000	AND
0001	OR
0010	Add
0110	Subtract

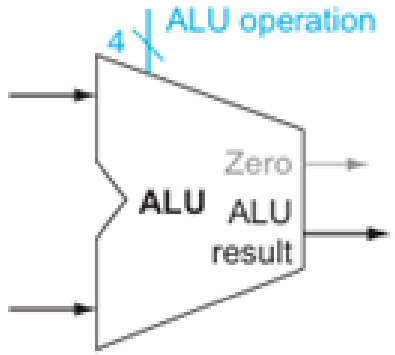


- Depending on the instruction class, the ALU needs to perform one of the first five functions
- For the R-type instructions,
  - the ALU needs to perform one of the five actions (AND, OR, subtract, add, or set on less than),
  - The actions depends on **6-bit funct** (or function) value

An implementation of our MIPS subset

- The MIPS **ALU defines** the 6 following combinations of four control inputs:

ALU control lines	Function
0000	AND
0001	OR
0010	Add
0110	Subtract
0111	set on less than

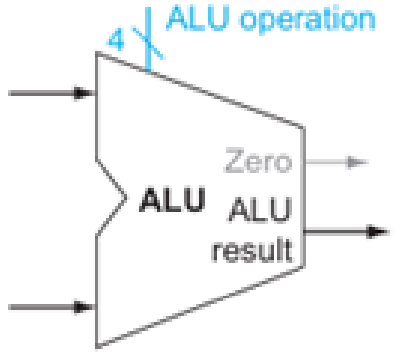


- Depending on the instruction class, the ALU needs to perform one of the first five functions
- For the R-type instructions,
  - the ALU needs to perform one of the five actions (AND, OR, subtract, add, or set on less than),
  - The actions depends on **6-bit funct** (or function) value

An implementation of our MIPS subset

- The MIPS **ALU defines** the 6 following combinations of four control inputs:

ALU control lines	Function
0000	AND
0001	OR
0010	Add
0110	Subtract
0111	set on less than
1100	NOR



- Depending on the instruction class, the ALU needs to perform one of the first five functions
- For the R-type instructions,
  - the ALU needs to perform one of the five actions (AND, OR, subtract, add, or set on less than),
  - The actions depends on **6-bit funct** (or function) value

Instruction opcode	ALUOp	Instruction operation	Funct field	Desired ALU action	ALU control Input
LW	00	load word	XXXXXX	add	0010
SW	00	store word	XXXXXX	add	0010
Branch equal	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
R-type	10	subtract	100010	subtract	0110
R-type	10	AND	100100	AND	0000
R-type	10	OR	100101	OR	0001
R-type	10	set on less than	101010	set on less than	0111

Designing the Main Control Unit

- let's identify the fields of an instruction and the control lines that are needed for the datapath we constructed

Field	0	rs	rt	rd	shamt	funct
Bit positions	31:26	25:21	20:16	15:11	10:6	5:0

a. R-type instruction

Field	35 or 43	rs	rt	address
Bit positions	31:26	25:21	20:16	15:0

b. Load or store instruction

Field	4	rs	rt	address
Bit positions	31:26	25:21	20:16	15:0

c. Branch instruction

Review the formats of the three instruction classes

- The op field, also called the **opcode**,
  - always contained in bits 31:26

Review the formats of the three instruction classes

- The op field, also called the **opcode**,
  - always contained in bits 31:26
- The **two registers** to be read
  - always specified by rs (25:21) and rt (20:16) fields
  - Same for the R-type, branch equal, and store

Review the formats of the three instruction classes

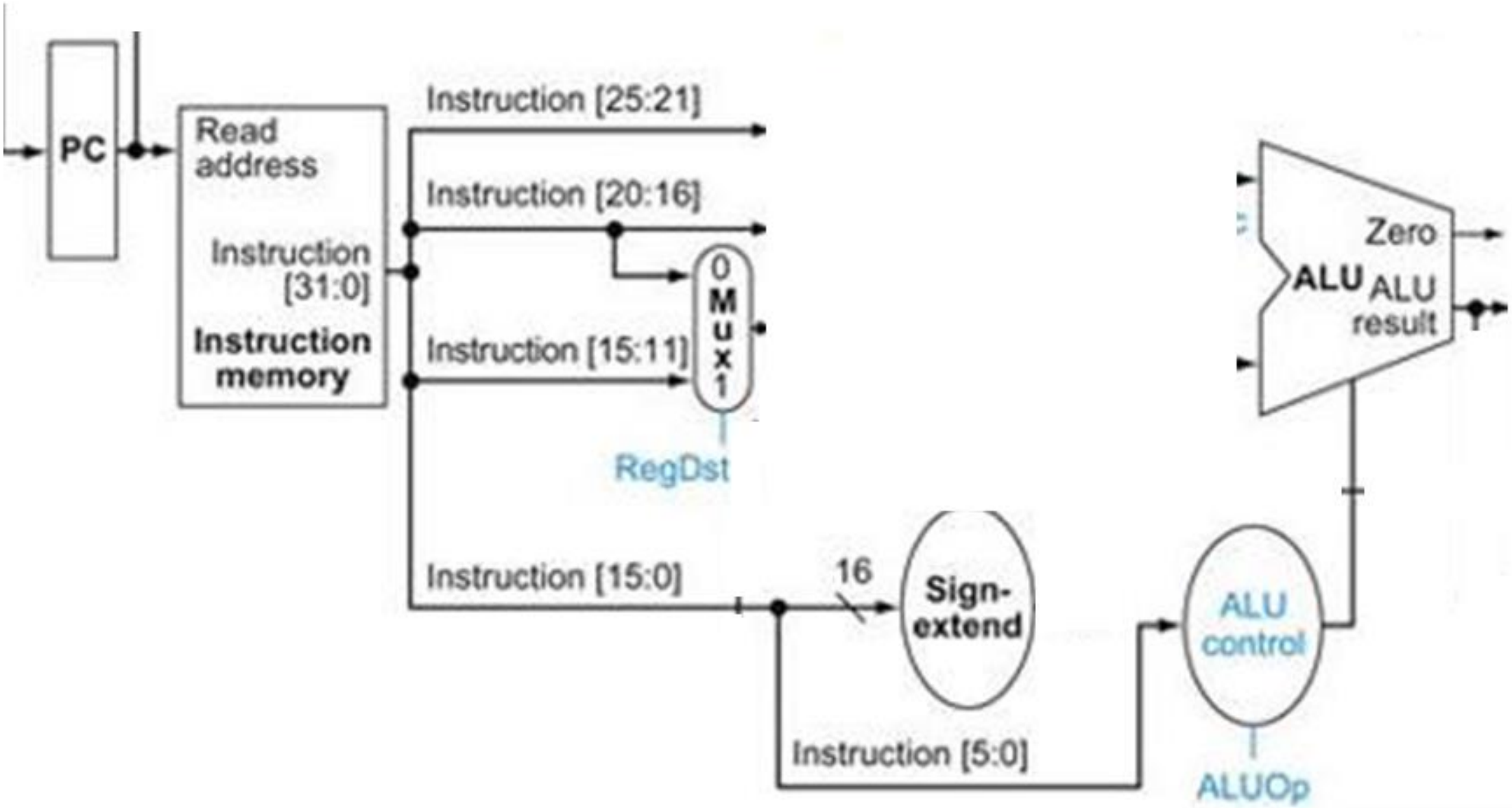
- The op field, also called the **opcode**,
  - always contained in bits 31:26
- The **two registers** to be read
  - always specified by rs (25:21) and rt (20:16) fields
  - Same for the R-type, branch equal, and store
- The **base register** for load and store instructions
  - always in rs (25:21)

Review the formats of the three instruction classes

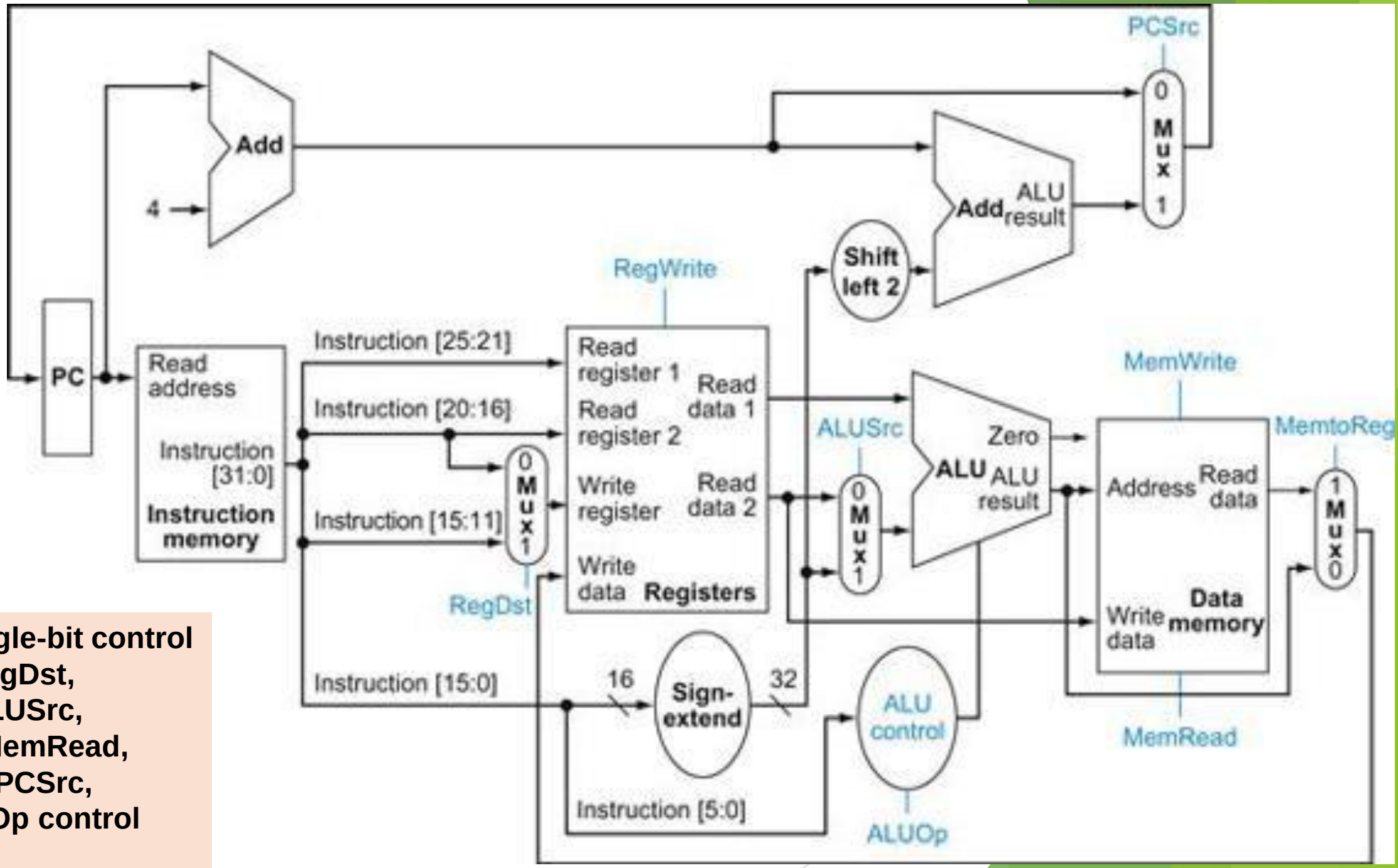
- The op field, also called the **opcode**,
  - always contained in bits 31:26
- The **two registers** to be read
  - always specified by rs (25:21) and rt (20:16) fields
  - Same for the R-type, branch equal, and store
- The **base register** for load and store instructions
  - always in rs (25:21)
- The **16-bit off set** for branch equal, load, and store
  - always in positions 15:0

Review the formats of the three instruction classes

- The op field, also called the **opcode**,
  - always contained in bits 31:26
- The **two registers** to be read
  - always specified by rs (25:21) and rt (20:16) fields
  - Same for the R-type, branch equal, and store
- The **base register** for load and store instructions
  - always in rs (25:21)
- The **16-bit off set** for branch equal, load, and store
  - always in positions 15:0
- The **destination register**
  - **one of two places**
  - For a **load** it is in bit positions **20:16 (rt),**
  - for an **R-type** instruction it is in bit positions **15:11 (rd)**
  - Therefore, we will need to add a multiplexor


PC
Read address
Instruction [31:0]
Instruction memory
Instruction [25:21]
Instruction [20:16]
Instruction [15:11]
0
M
u
x
1
RegDst
Instruction [15:0]
16
Sign-extend
Instruction [5:0]
ALU
Zero
ALU result
ALU control
ALUOp

The Datapath



→ seven single-bit control lines, i.e., RegDst, RegWrite, ALUSrc, MemWrite, MemRead, MemToReg, PCSrc,
→ 2-bit ALUOp control signal

7 control lines

Signal name	Effect when deasserted	Effect when asserted
RegDst	The register destination number for the Write register comes from the rt field (bits 20:16).	The register destination number for the Write register comes from the rd field (bits 15:11).

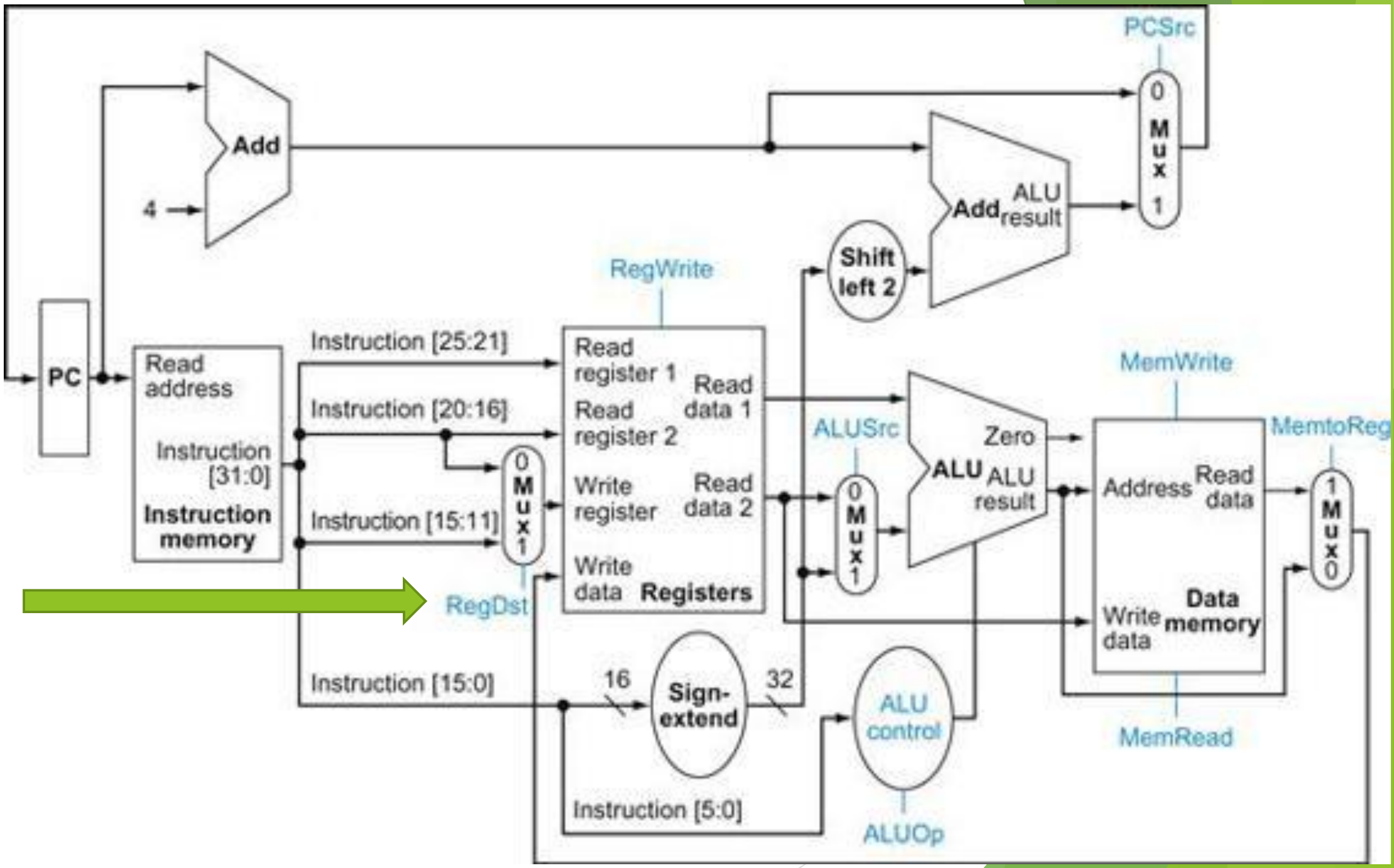
Field	35 or 43	rs	rt	address
Bit positions	31:26	25:21	20:16	15:0

b. Load or store instruction

Field	0	rs	rt	rd	shamt	funct
Bit positions	31:26	25:21	20:16	15:11	10:6	5:0

a. R-type instruction

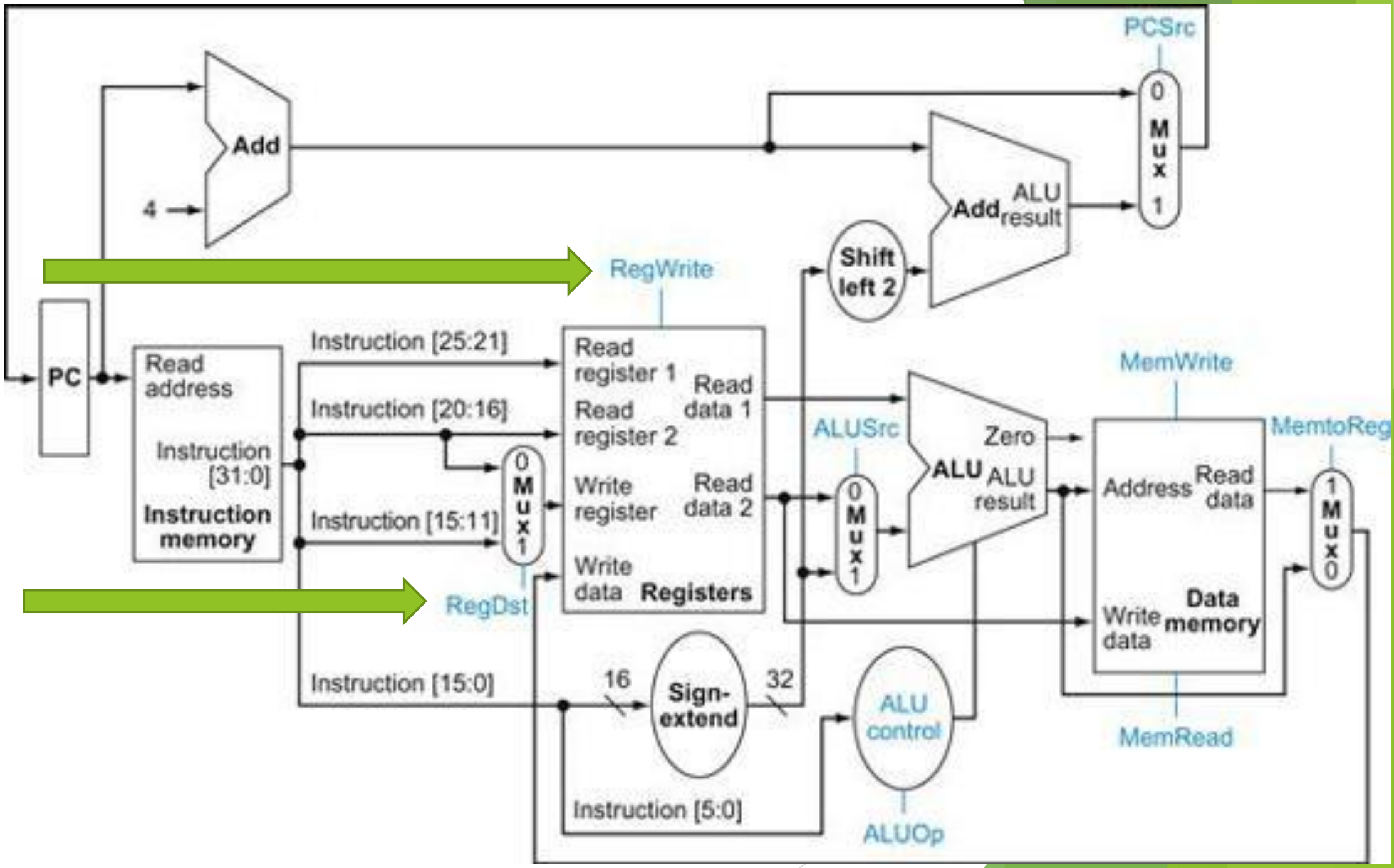
The Datapath



7 control lines

Signal name	Effect when deasserted	Effect when asserted
RegDst	The register destination number for the Write register comes from the rt field (bits 20:16).	The register destination number for the Write register comes from the rd field (bits 15:11).
RegWrite	None.	The register on the Write register input is written with the value on the Write data input.

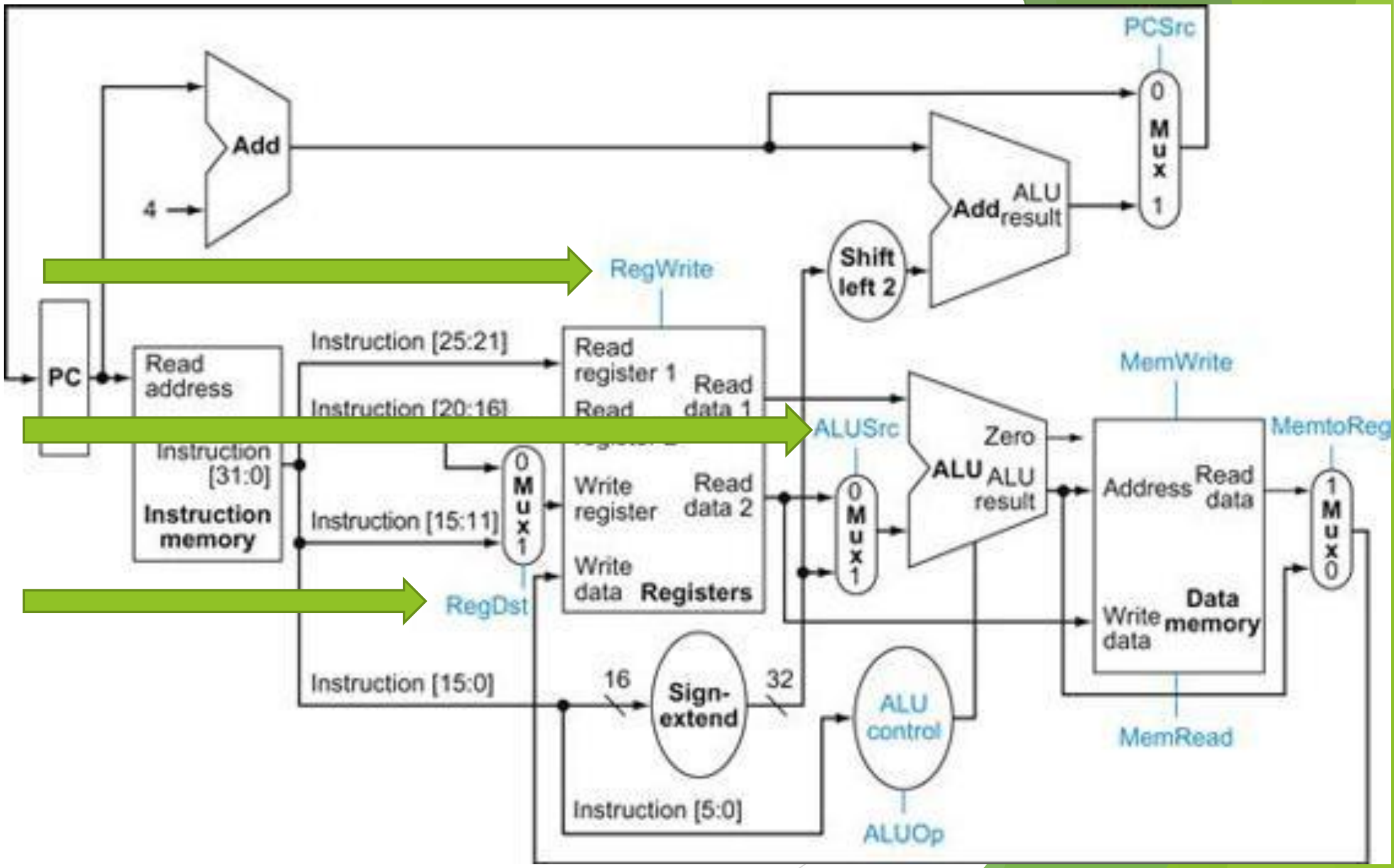
The Datapath



7 control lines

Signal name	Effect when deasserted	Effect when asserted
RegDst	The register destination number for the Write register comes from the rt field (bits 20:16).	The register destination number for the Write register comes from the rd field (bits 15:11).
RegWrite	None.	The register on the Write register input is written with the value on the Write data input.
ALUSrc	The second ALU operand comes from the second register file output (Read data 2).	The second ALU operand is the sign-extended, lower 16 bits of the instruction.

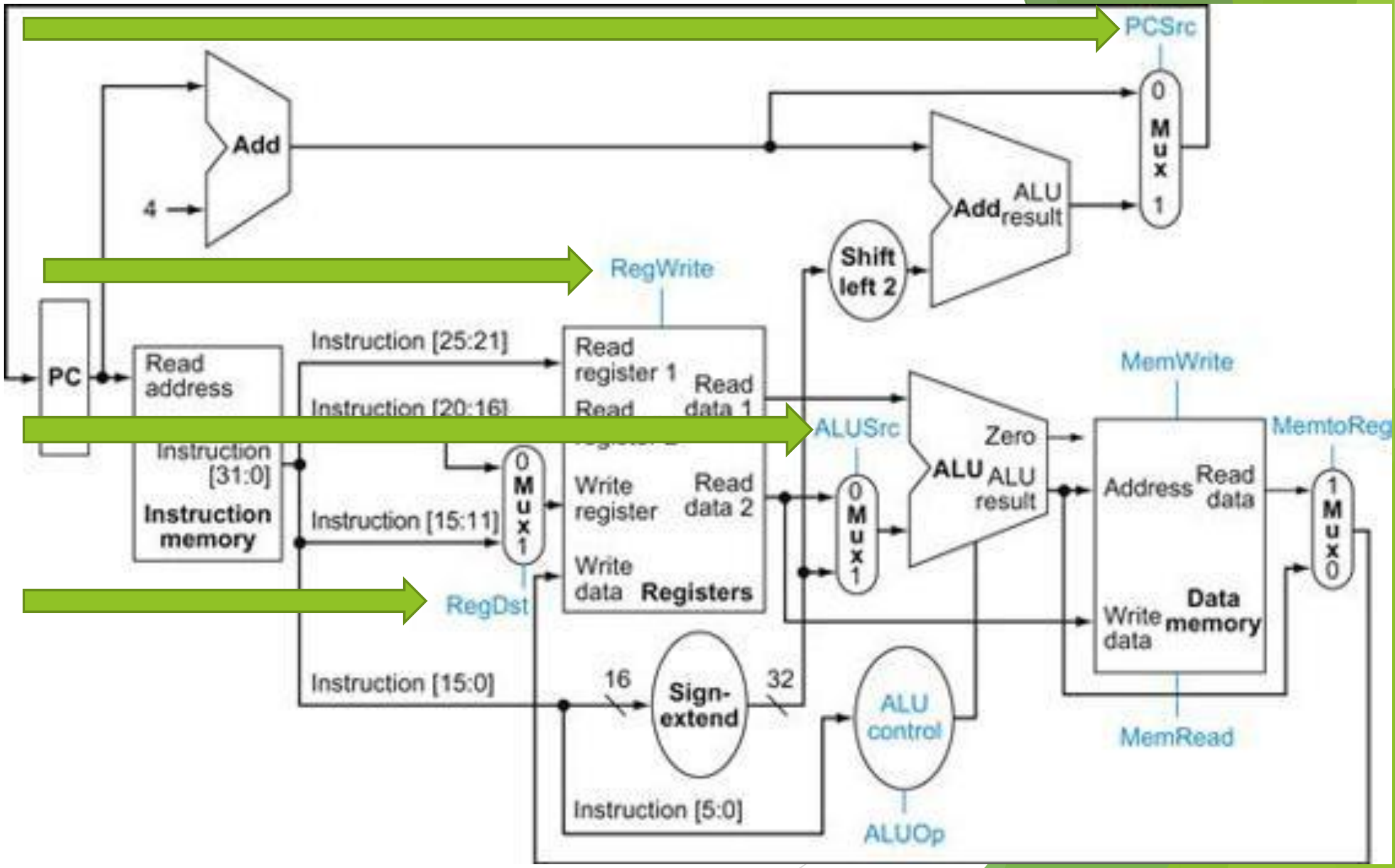
The Datapath



7 control lines

Signal name	Effect when deasserted	Effect when asserted
RegDst	The register destination number for the Write register comes from the rt field (bits 20:16).	The register destination number for the Write register comes from the rd field (bits 15:11).
RegWrite	None.	The register on the Write register input is written with the value on the Write data input.
ALUSrc	The second ALU operand comes from the second register file output (Read data 2).	The second ALU operand is the sign-extended, lower 16 bits of the instruction.
PCSrc	The PC is replaced by the output of the adder that computes the value of PC + 4.	The PC is replaced by the output of the adder that computes the branch target.

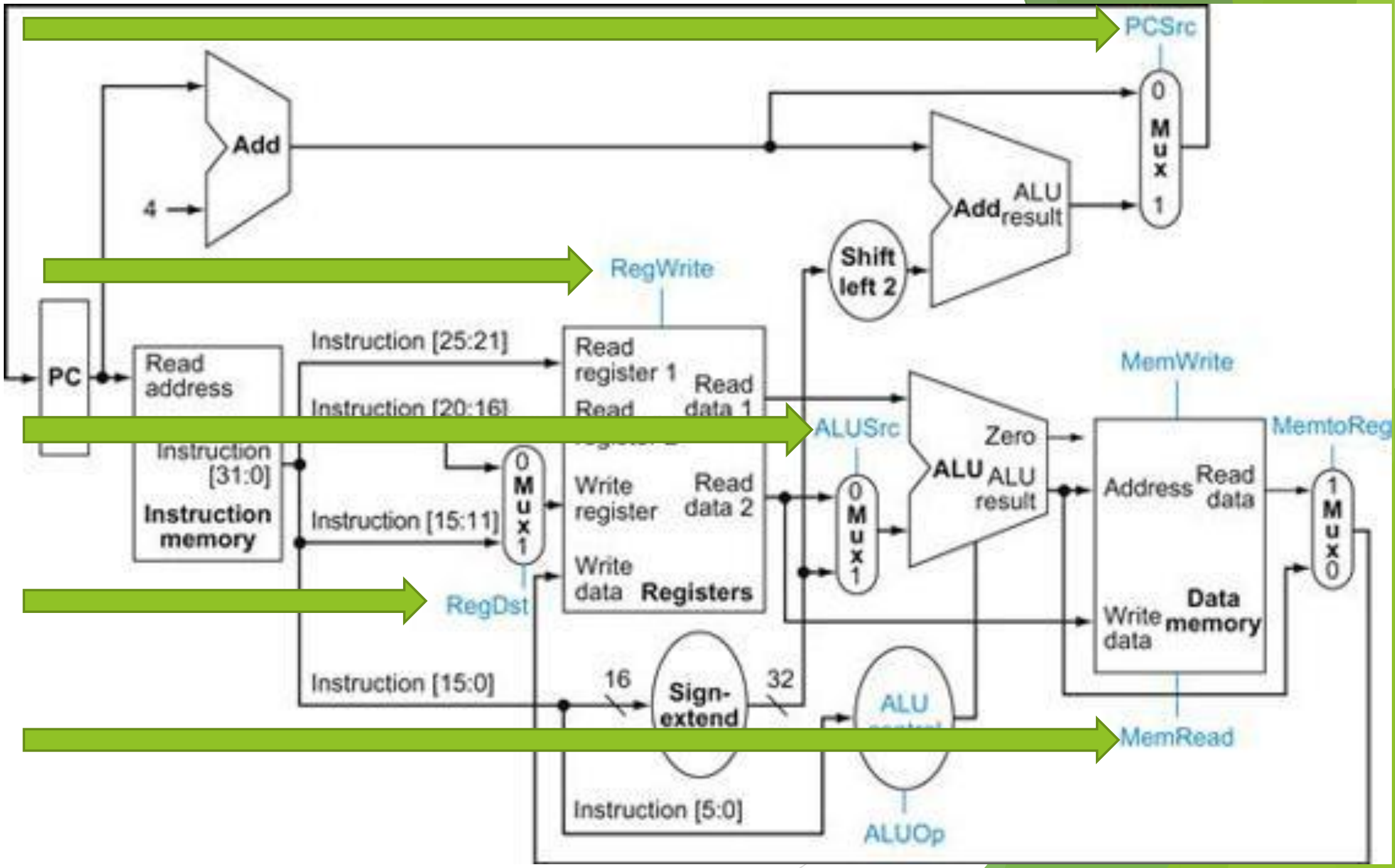
The Datapath



7 control lines

Signal name	Effect when deasserted	Effect when asserted
RegDst	The register destination number for the Write register comes from the rt field (bits 20:16).	The register destination number for the Write register comes from the rd field (bits 15:11).
RegWrite	None.	The register on the Write register input is written with the value on the Write data input.
ALUSrc	The second ALU operand comes from the second register file output (Read data 2).	The second ALU operand is the sign-extended, lower 16 bits of the instruction.
PCSrc	The PC is replaced by the output of the adder that computes the value of PC + 4.	The PC is replaced by the output of the adder that computes the branch target.
MemRead	None.	Data memory contents designated by the address input are put on the Read data output.

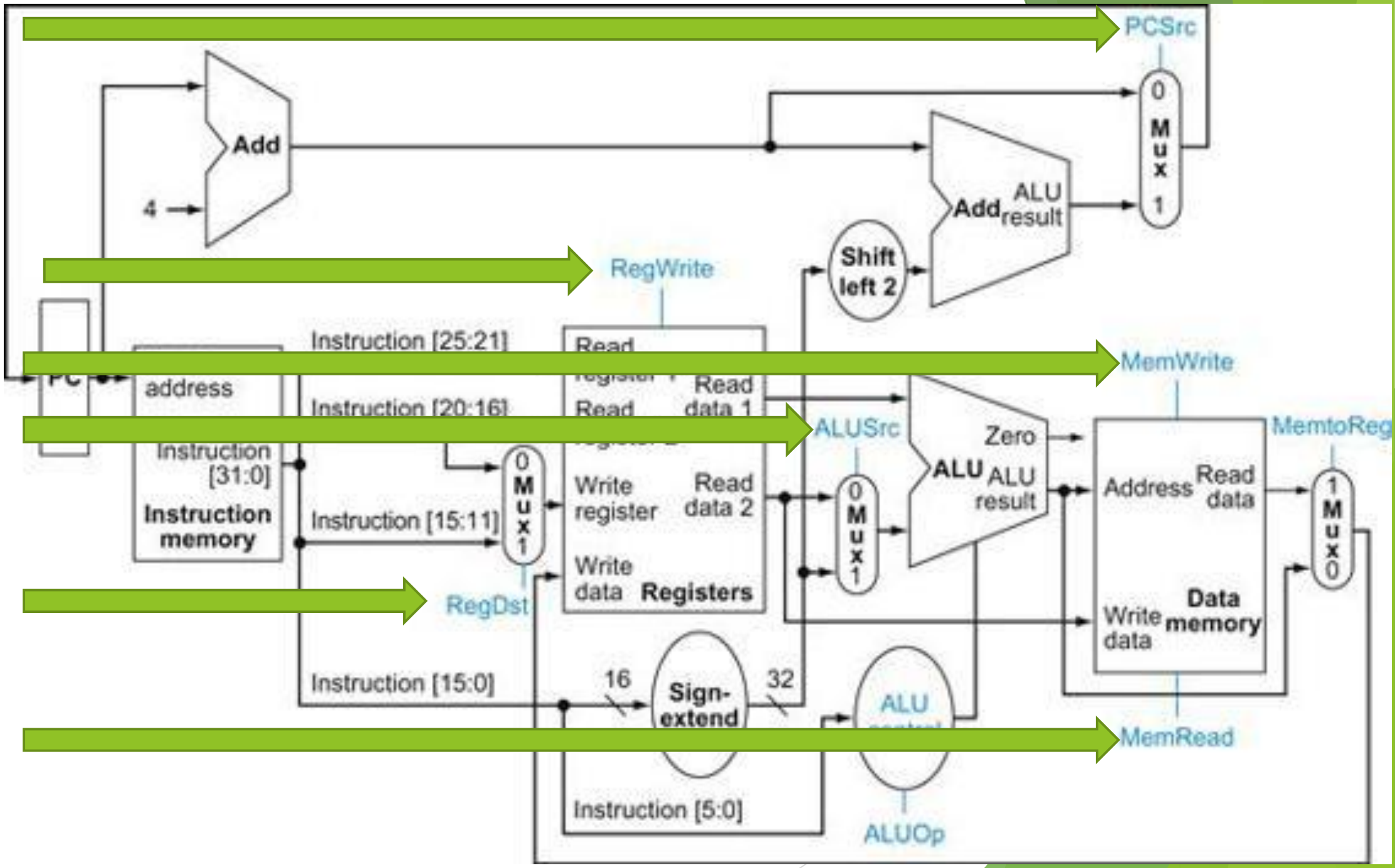
The Datapath



7 control lines

Signal name	Effect when deasserted	Effect when asserted
RegDst	The register destination number for the Write register comes from the rt field (bits 20:16).	The register destination number for the Write register comes from the rd field (bits 15:11).
RegWrite	None.	The register on the Write register input is written with the value on the Write data input.
ALUSrc	The second ALU operand comes from the second register file output (Read data 2).	The second ALU operand is the sign-extended, lower 16 bits of the instruction.
PCSrc	The PC is replaced by the output of the adder that computes the value of PC + 4.	The PC is replaced by the output of the adder that computes the branch target.
MemRead	None.	Data memory contents designated by the address input are put on the Read data output.
MemWrite	None.	Data memory contents designated by the address input are replaced by the value on the Write data input.

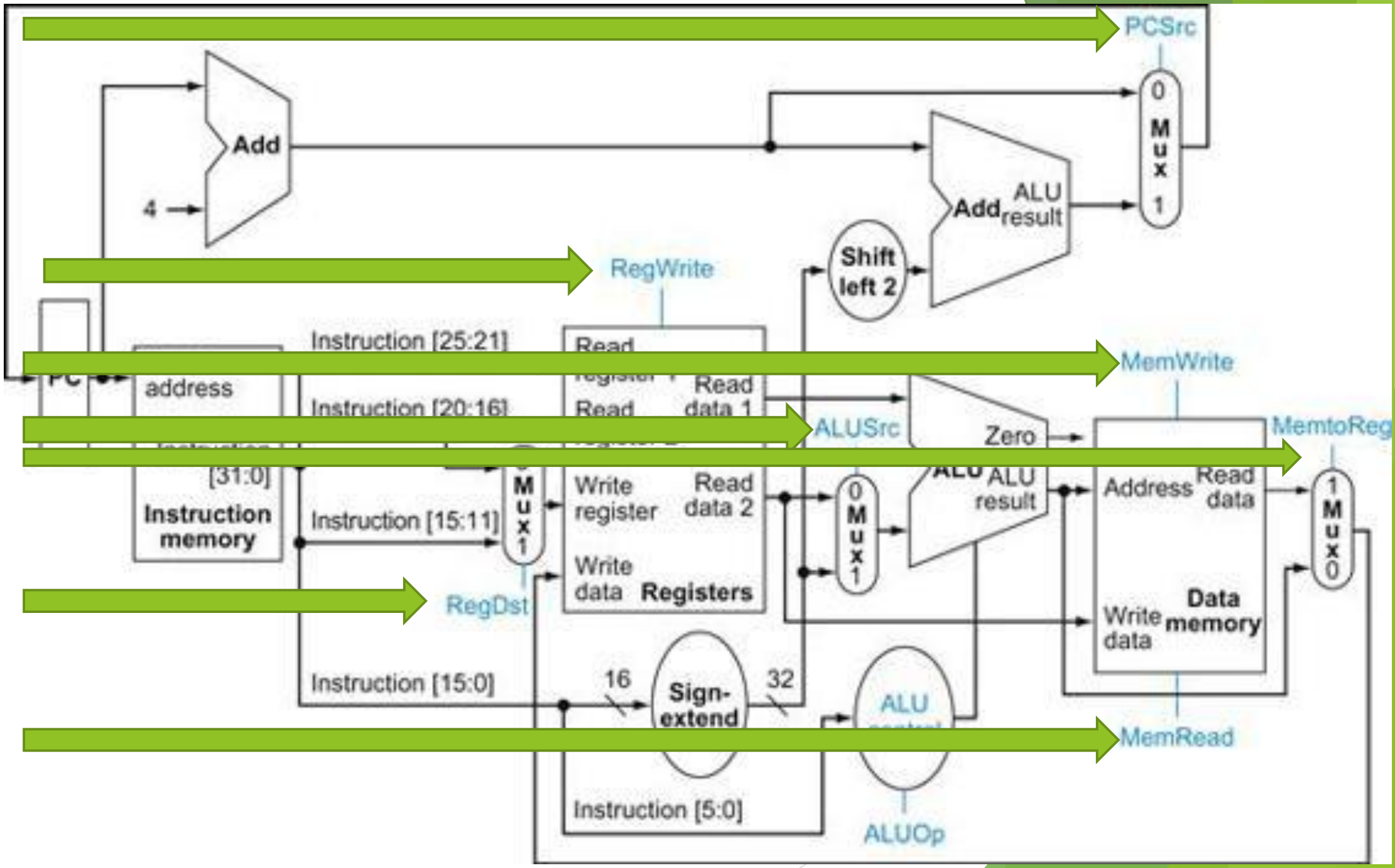
The Datapath



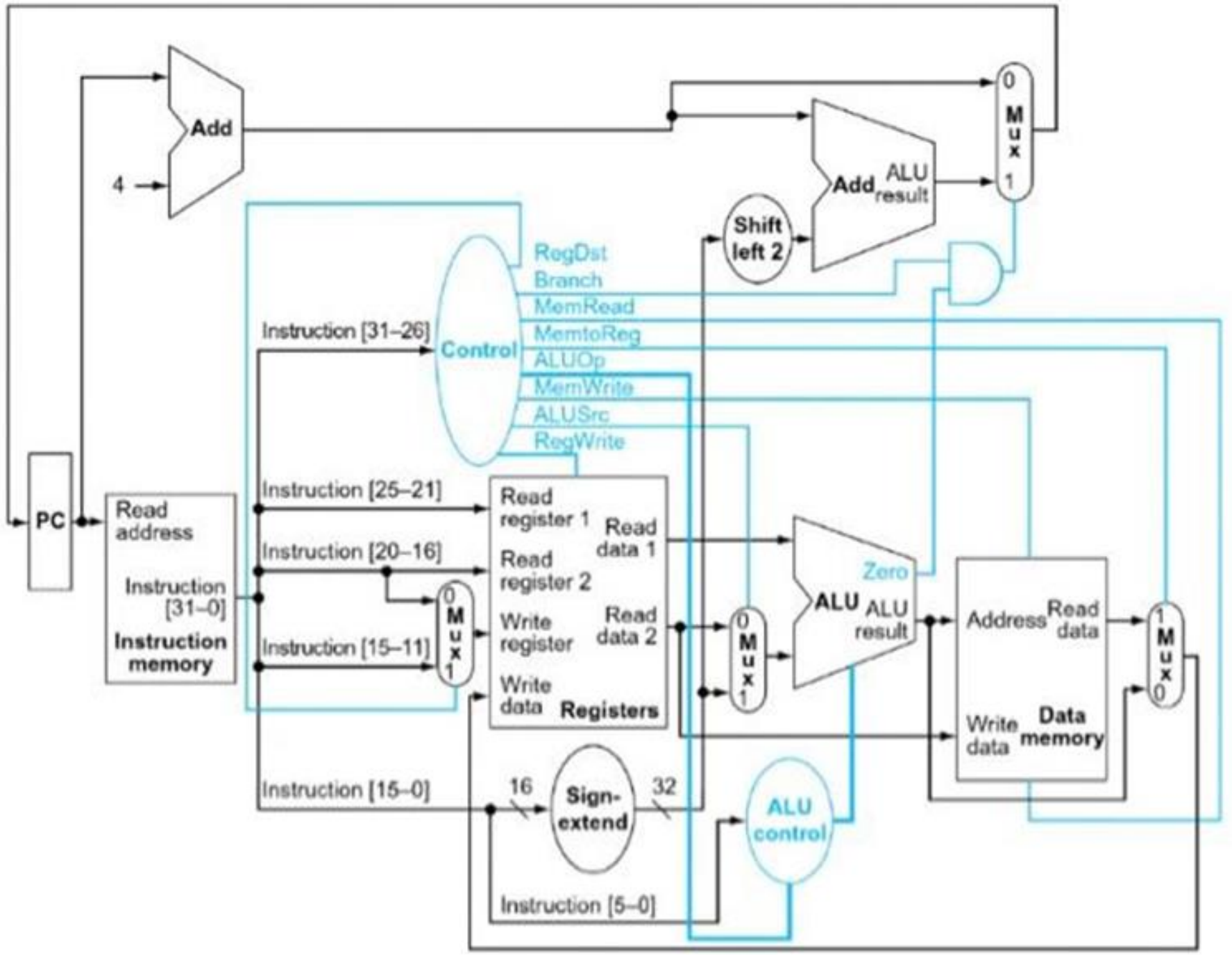
7 control lines

Signal name	Effect when deasserted	Effect when asserted
RegDst	The register destination number for the Write register comes from the rt field (bits 20:16).	The register destination number for the Write register comes from the rd field (bits 15:11).
RegWrite	None.	The register on the Write register input is written with the value on the Write data input.
ALUSrc	The second ALU operand comes from the second register file output (Read data 2).	The second ALU operand is the sign-extended, lower 16 bits of the instruction.
PCSrc	The PC is replaced by the output of the adder that computes the value of PC + 4.	The PC is replaced by the output of the adder that computes the branch target.
MemRead	None.	Data memory contents designated by the address input are put on the Read data output.
MemWrite	None.	Data memory contents designated by the address input are replaced by the value on the Write data input.
MemtoReg	The value fed to the register Write data input comes from the ALU.	The value fed to the register Write data input comes from the data memory.

The Datapath



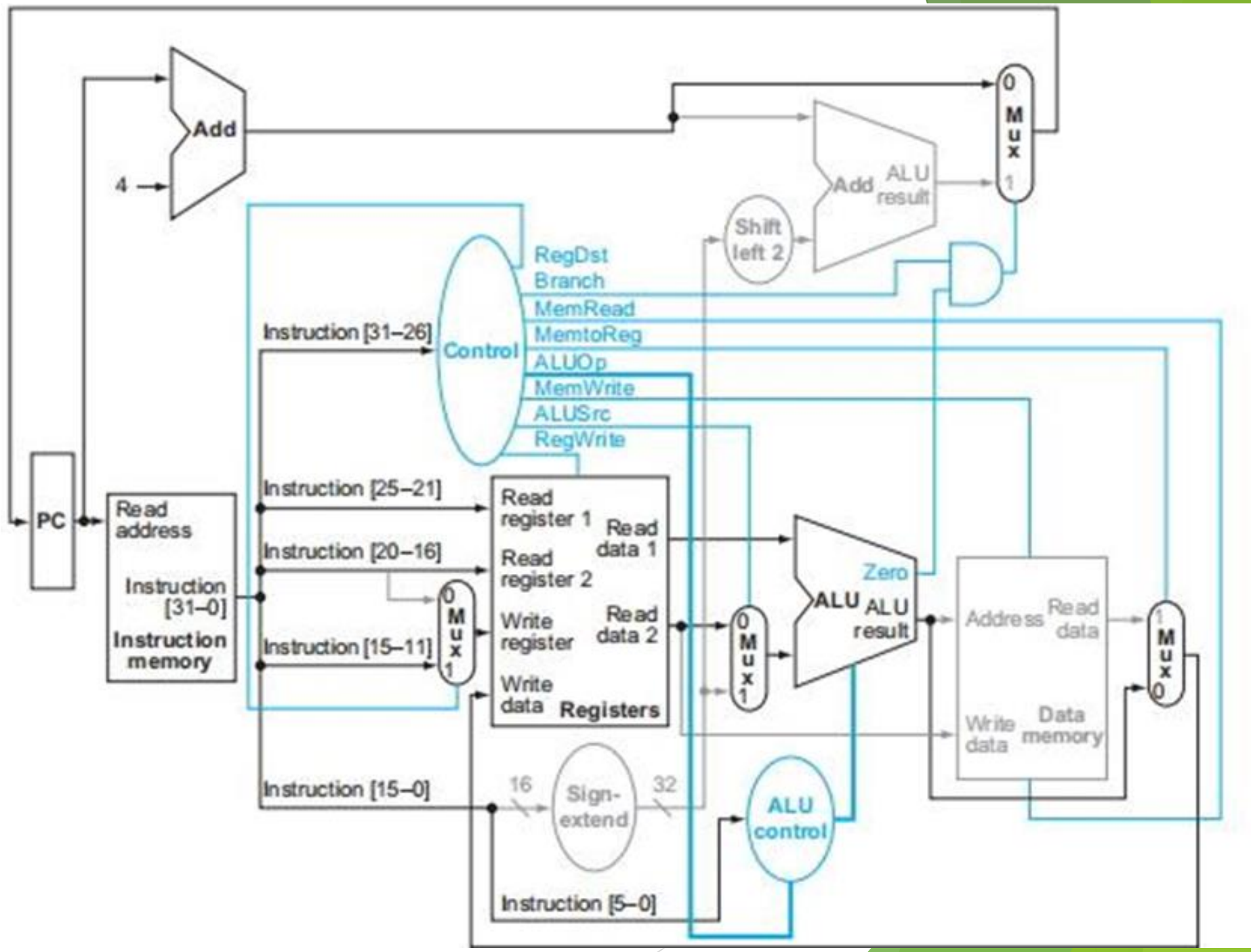
Datapath with control lines



Everything occurs in one clock cycle

Datapath for each class of instructions

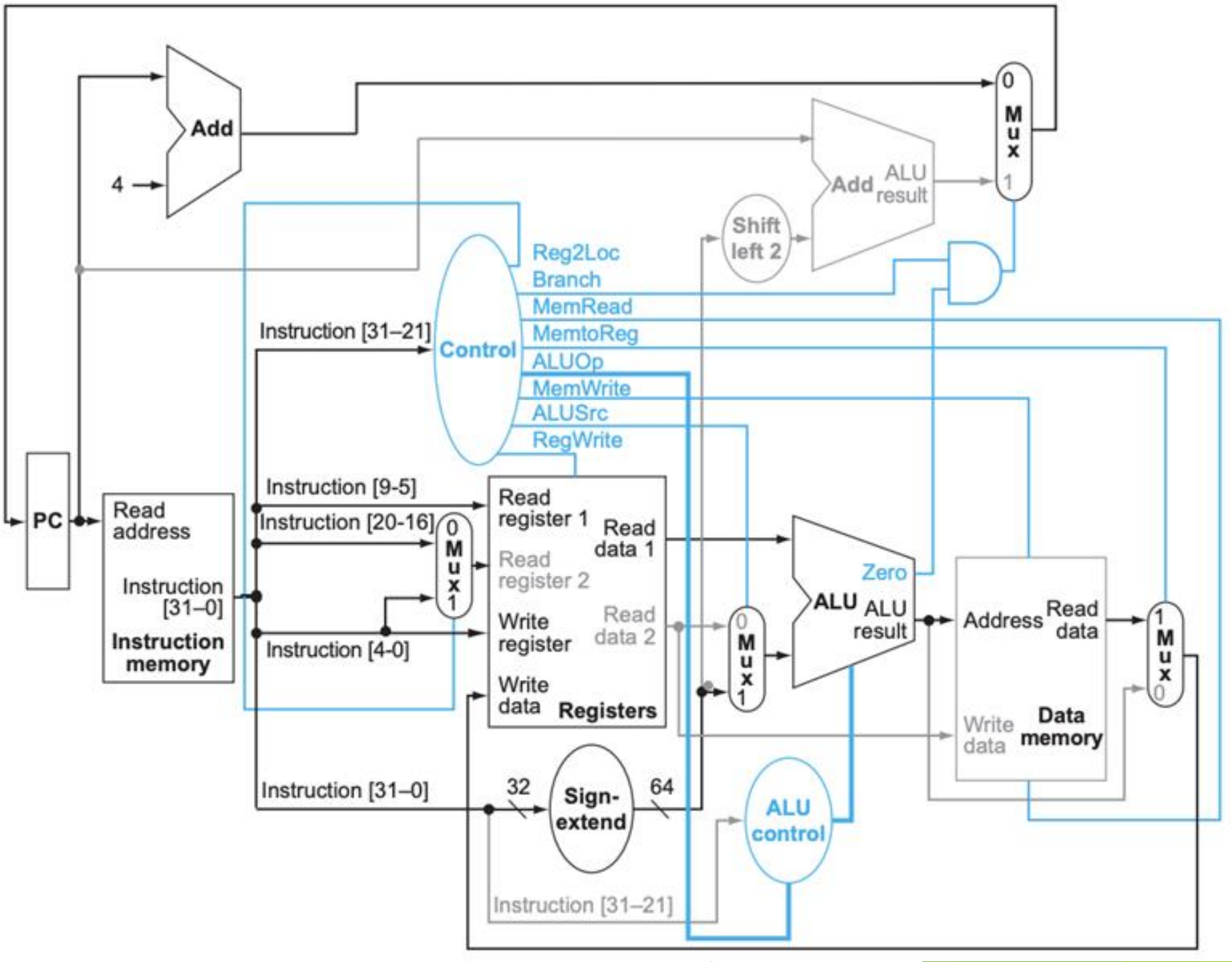
Datapath for R-Type instruction



Operation of the Datapath

- How about the execution of a load word, such as
  - lw $t1, offset($t2)
- a load instruction is operating in **five steps**
  1. An instruction is fetched from the instruction memory, and the PC is incremented
  2. A register ($t2) value is read from the register file
  3. The ALU computes the sum of the value read from the register file and the sign-extended, lower 16 bits of the instruction (offset)
  4. The sum from the ALU is used as the address for the data memory
  5. The data from the memory unit is written into the register file; the register destination is given by bits 20:16 of the instruction ($t1)

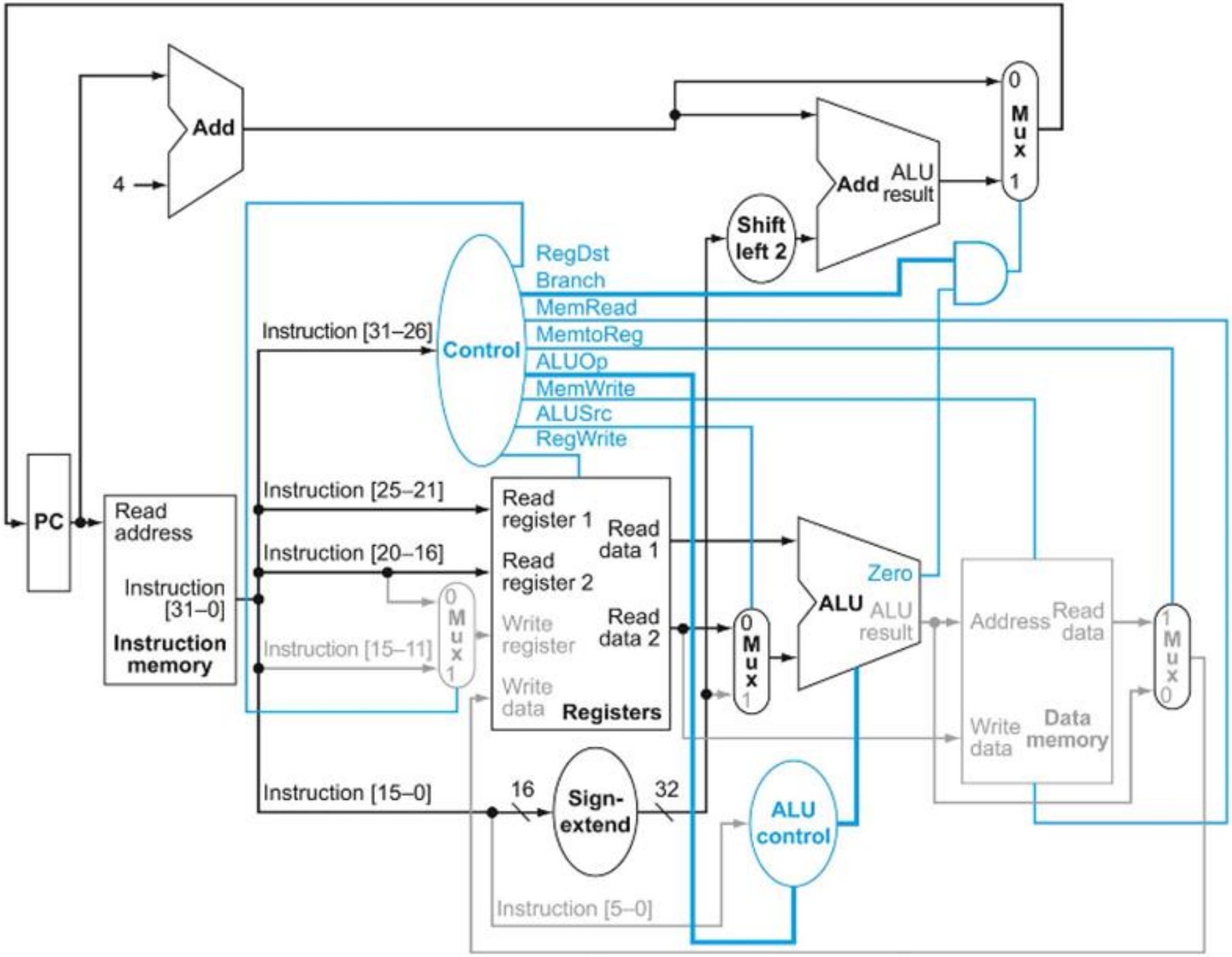
Data path for lw t1, 4(t2)



Operation of the Datapath

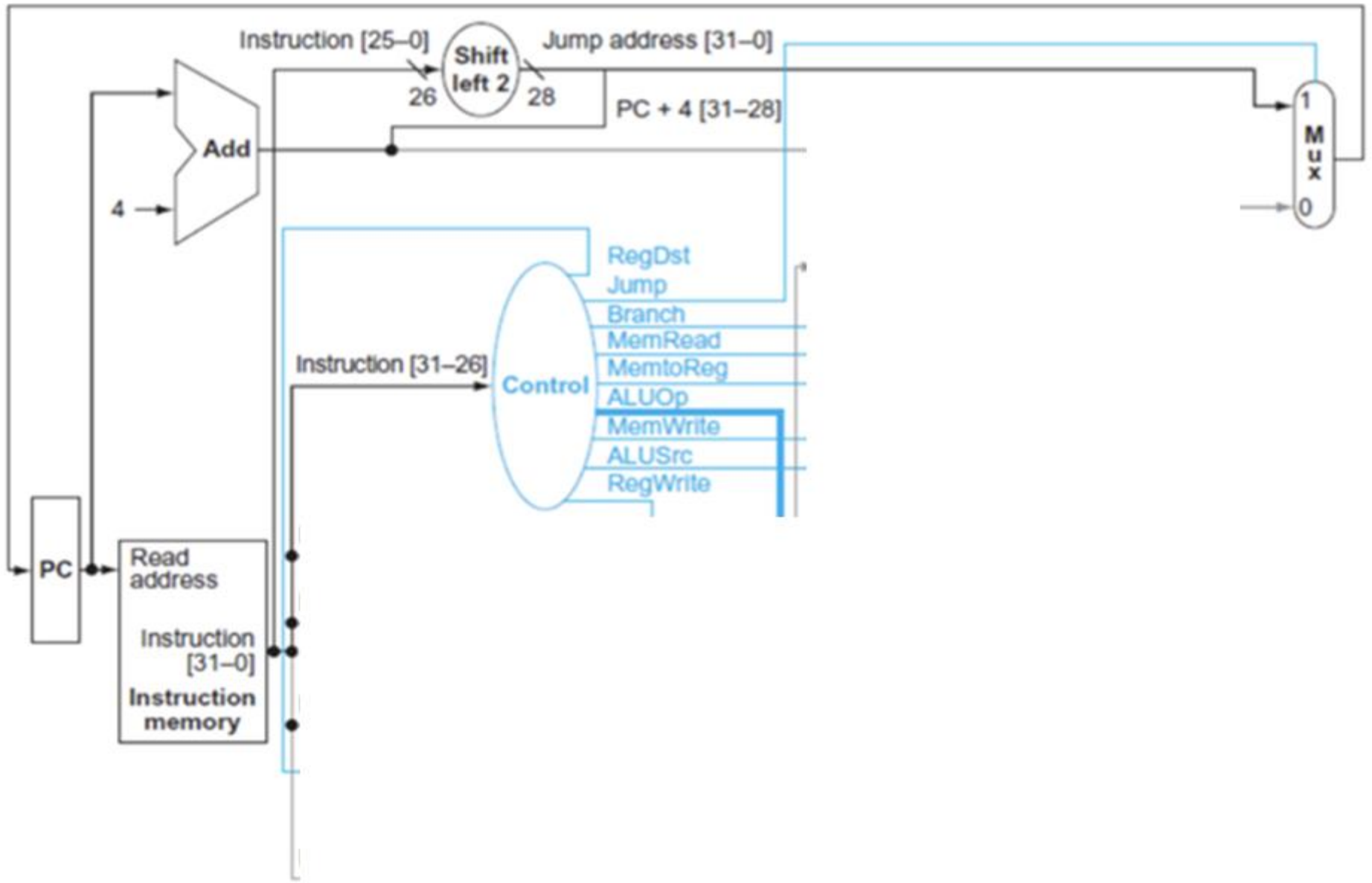
- How about the execution of a beq, such as
  - **beq $t1, $t2, offset**
- a **beq** instruction is operating in **four steps**
  1. An instruction is fetched from the instruction memory, and the PC is incremented
  2. Two registers, $t1 and $t2, are read from the register file
  3. The ALU performs a subtract on the data values read from the register file
     1. The value of PC + 4 is added to the sign-extended, lower 16 bits of the instruction (offset) shifted left by two; **the result is the branch target address**
  4. The Zero result from the ALU is used to decide which adder result to store into the PC

Data path for **beq t1, t2, offset**

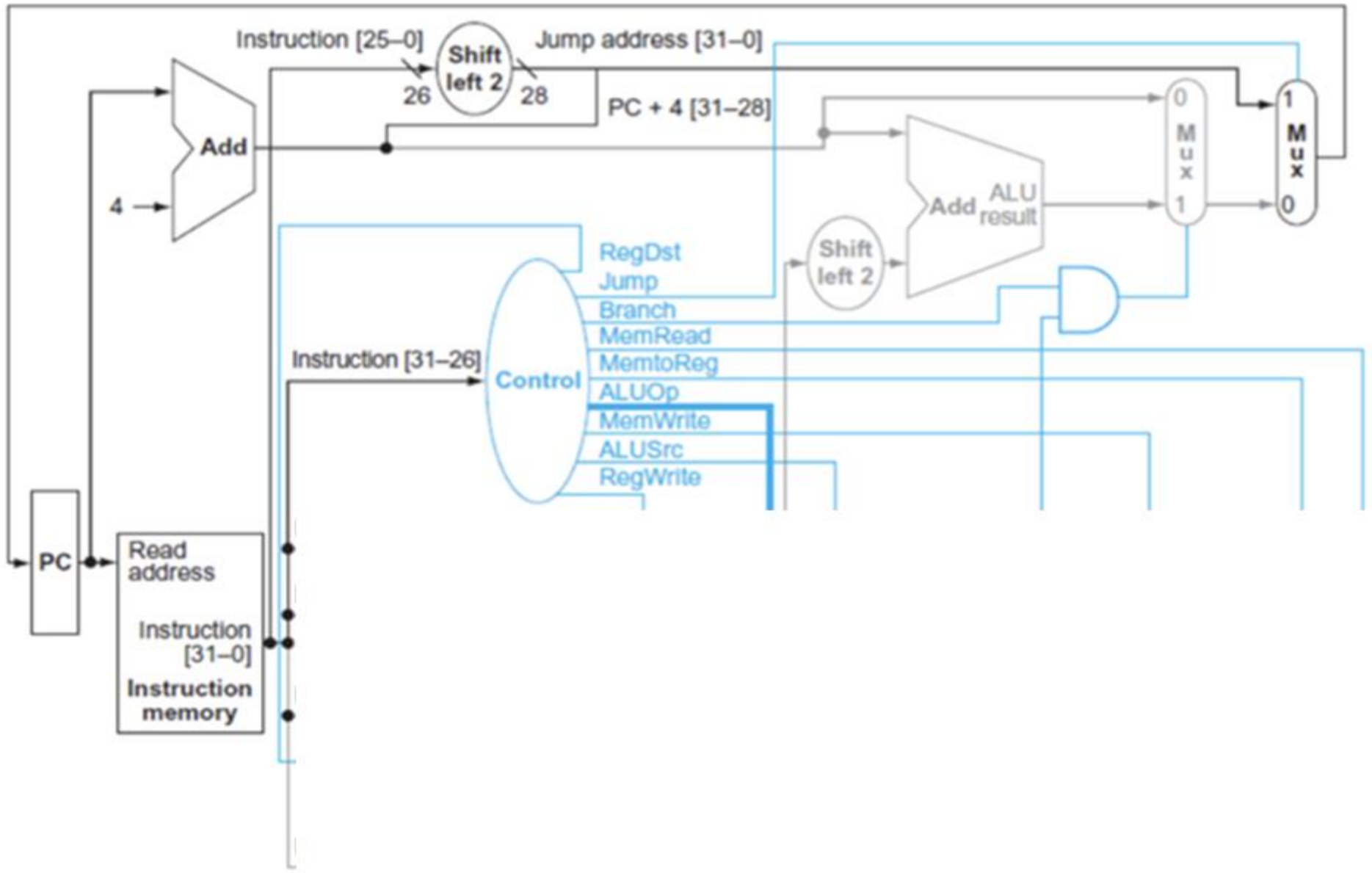


let’s add the **jump** instruction to show how the basic datapath and control can be extended

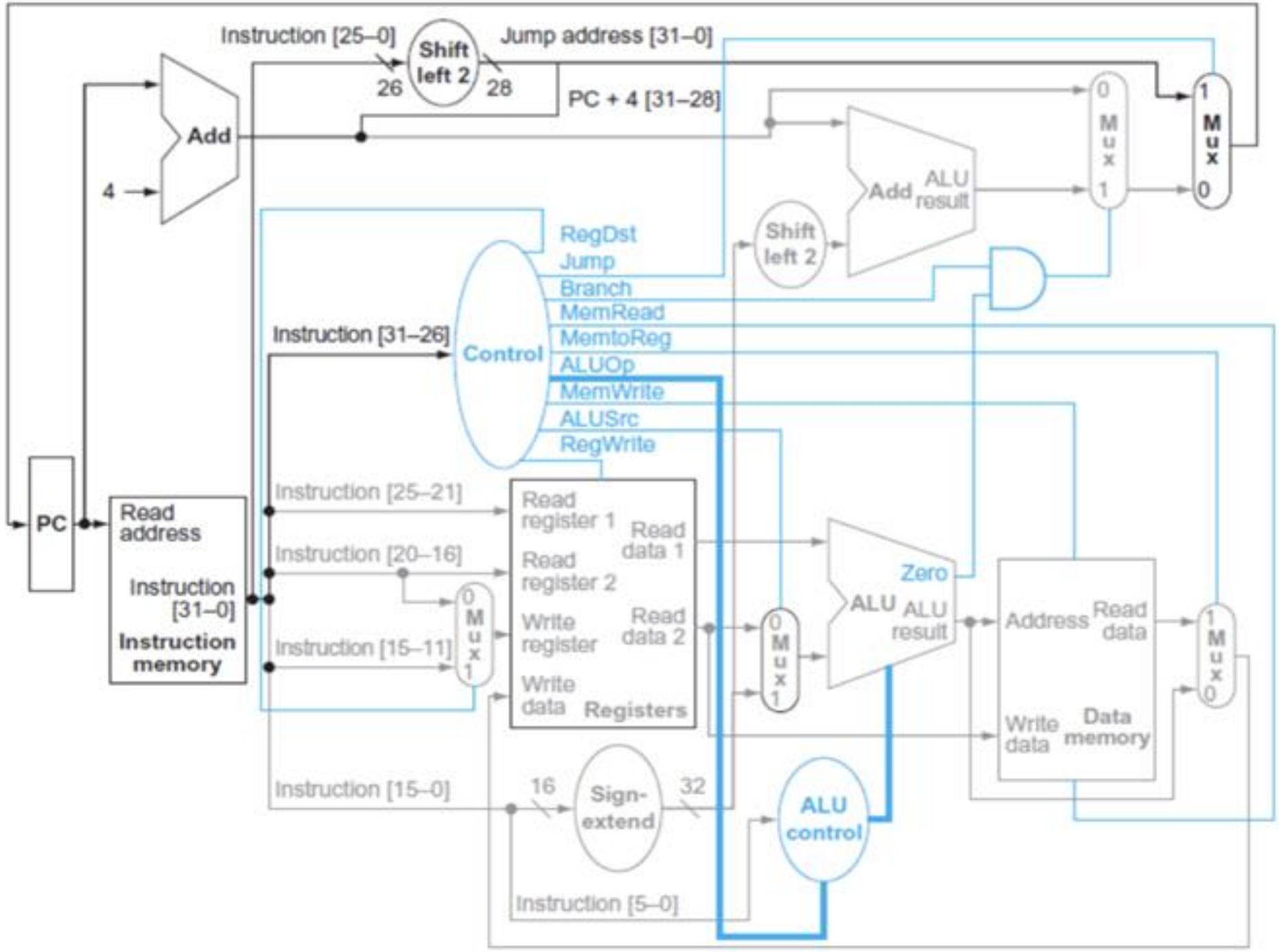
Datapa for j addres



Datapa for j addres



Datapath for j address




Thank You