Unit III

Processor

- The processing unit
  - *Central processing unit (CPU)*
  - *The term "central" is not as appropriate today – as c*omputers often include several processing units
  - Use the term ***processor***

- To achieve high performance, make various functional units of a processor operate in parallel as much as possible :
  - ***Pipelined organization*** where the execution of an instruction is started before the execution of the preceding instruction is completed
  - ***Superscalar operation****, is to fetch and start the execution of several instructions* at the same time

Fundamental Concepts

- Program :
  - Computing task
  - Series of operations
  - Specified by a sequence of machine-language instructions

- Instruction :
  - Processor fetches instruction
    - Fetch from successive location until branch or jump
  - Specified by PC
    - Keep track of next instruction
    - After instruction fetch it is updated to point to next instruction
      - Pc=pc+1
      - Branch : PC= target address
  - *Instruction register, IR,*
    - *Fetched instruction is placed here*
    - *Hold until execution is complete*
    - *Control circuit interpret or decode*

Fundamental Concepts

- *Instruction fetch phase*
  – Fetching an instruction and loading it into the IR
    1. IR←[[PC]]
    2. PC←[PC] + 4
    3. Carry out the operation specified by the instruction in the IR
- *Instruction execution phase*
  – *Performing the operation specified in the instruction*
    1. Read the contents of a given memory location and load them into a processor register.
    2. Read data from one or more processor registers.
    3. Perform an arithmetic or logic operation and place the result into a processor register.
    4. Store data from a processor register into a given memory location.

A Basic MIPS Implementation

- Simple subset, shows most aspects
  - The memory-reference instructions
    - *load word (lw) and store word (sw)*
  - The arithmetic-logical instructions
    - add, sub, AND, OR, and slt
  - Control transfer
    - *branch equal (beq) and jump (j),*

An Overview of the Implementation

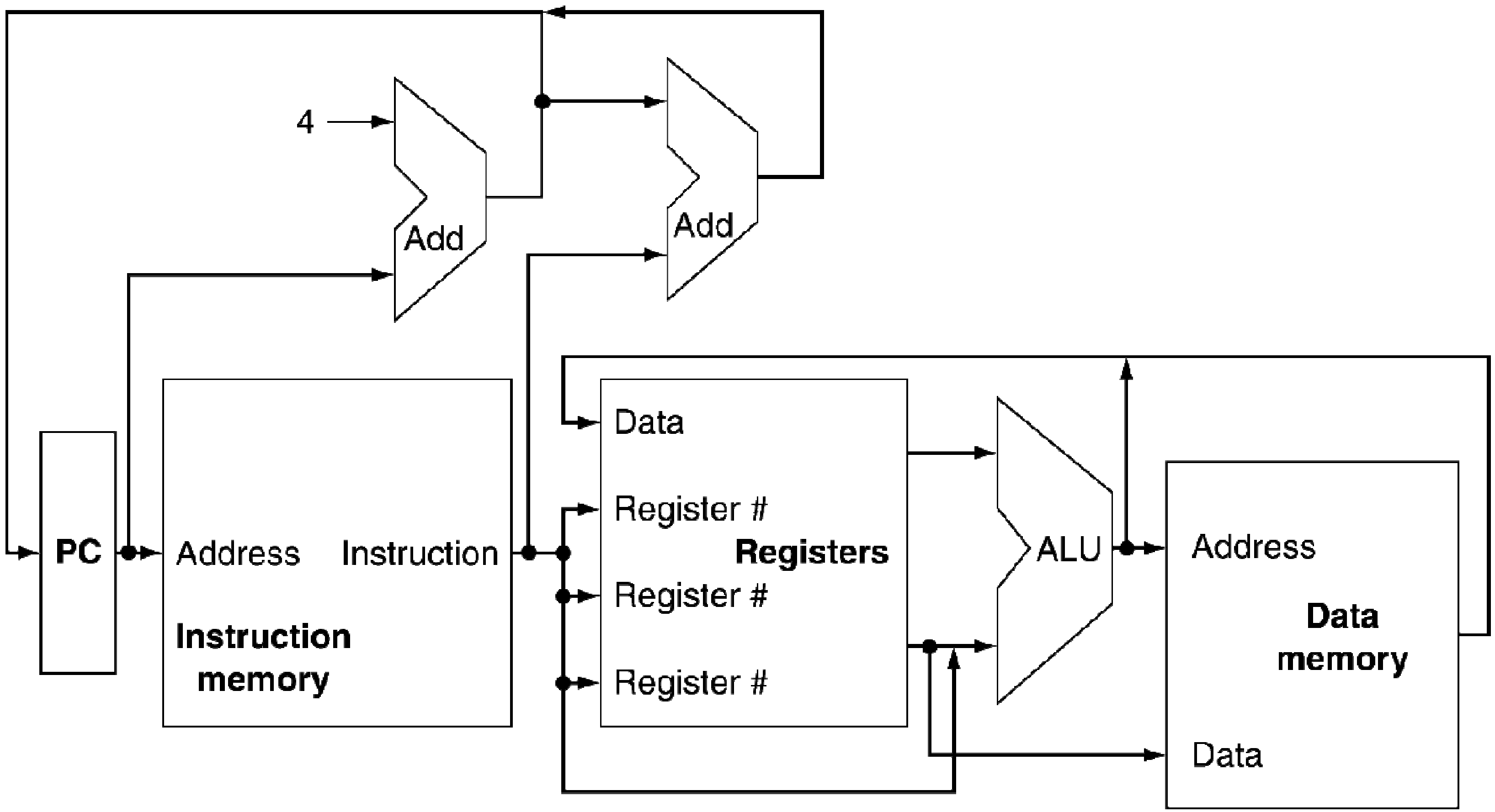
- First two steps are identical for all instruction type

1. Send the *program counter (PC) to the memory that contains the code and* fetch the instruction from that memory
   - IR←[[PC]]
2. Read one or two registers
   - Use fields of the instruction to select the registers
     - Load word instruction, need to read only one register
     - Most other instructions require reading two registers

An Overview of the Implementation

3. *Perform ALU operation (except jump)*
   - *add – to perform operation*
   - *lw – to calculate address*
   - *beq – to compare*
4. *This step differ*
   - *add –write data to register*
   - *lw – read data to register*
   - *beq- change or increment PC*

CPU Overview



omits two important aspects of instruction execution

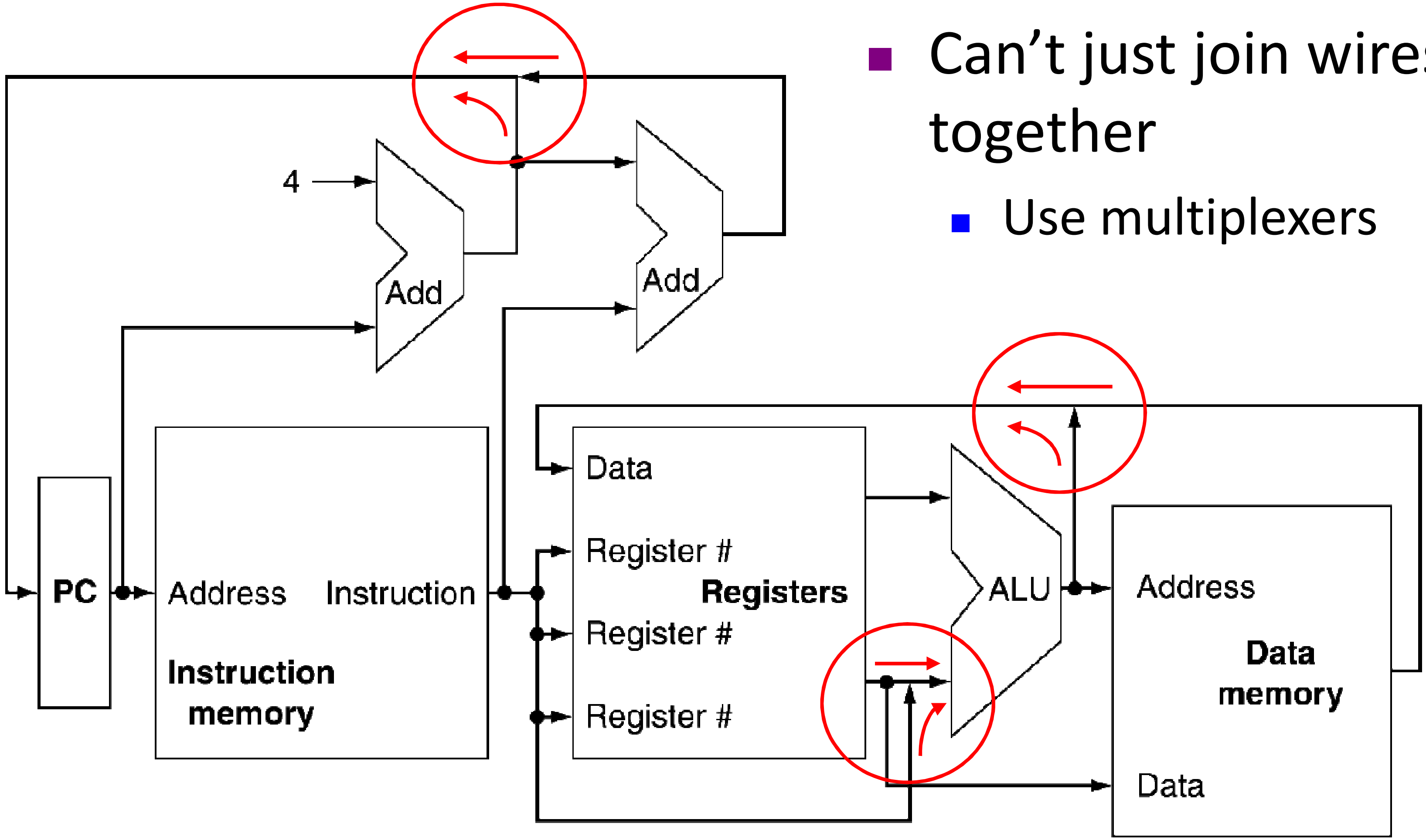
•***Multiplexor***

•***control unit***

CPU Overview

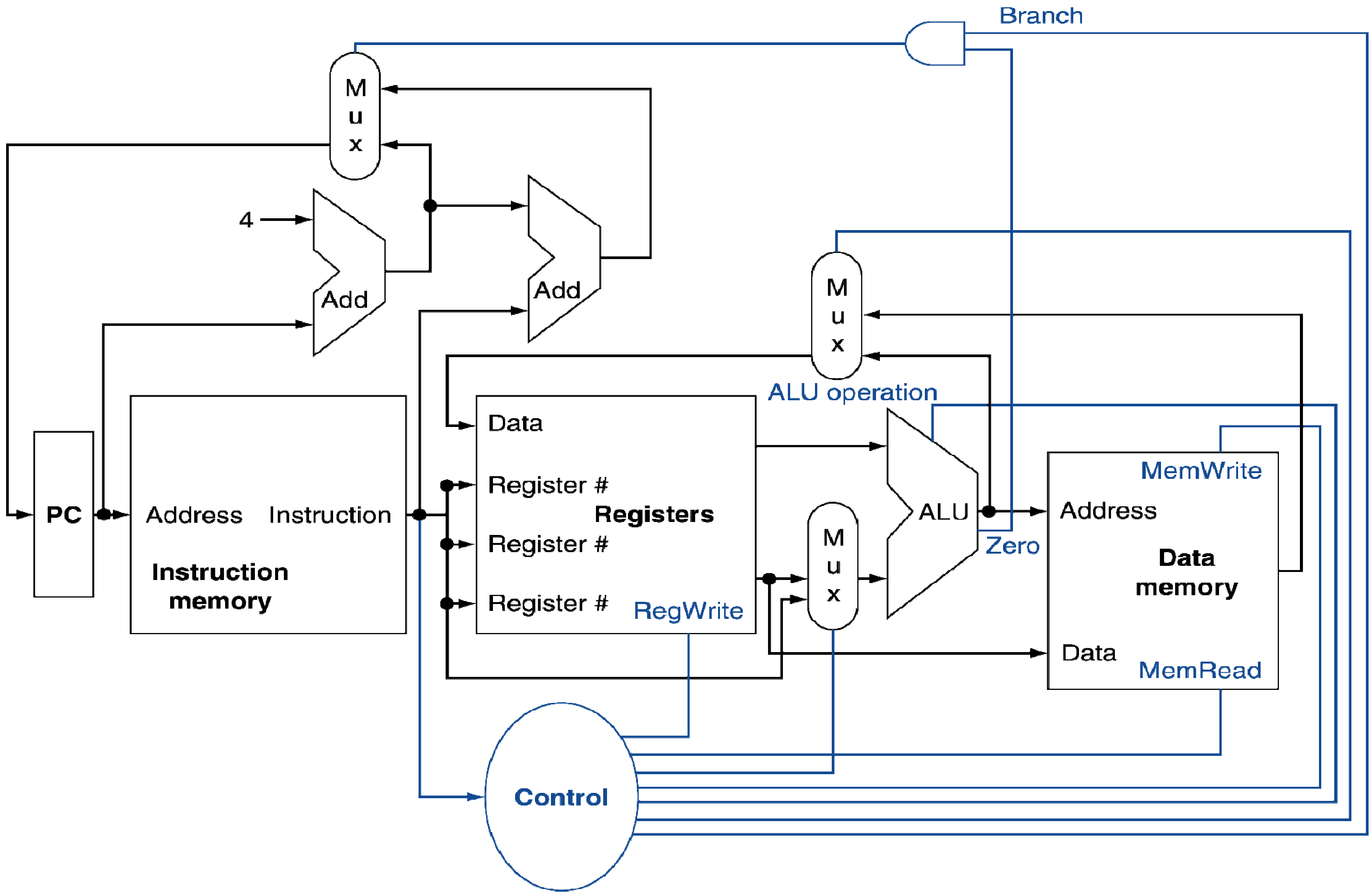
- Add : add$s1,$s2,$s3
  - PC : Address : Instruction: PC=PC+4
  - reg1,reg2 : ALU : reg3
- Lw: lw $s1,20($s2)
  - PC : Address : Instruction: PC=PC+4
  - reg1,imm : ALU: address: data memory: reg2
- Beq: beq $s1,$s2,2
  - PC : Address : Instruction: PC=PC+4
  - reg1, reg2 : ALU:
  - Zero : PC=PC + 4*2

Multiplexers



- Can't just join wires together
  - Use multiplexers

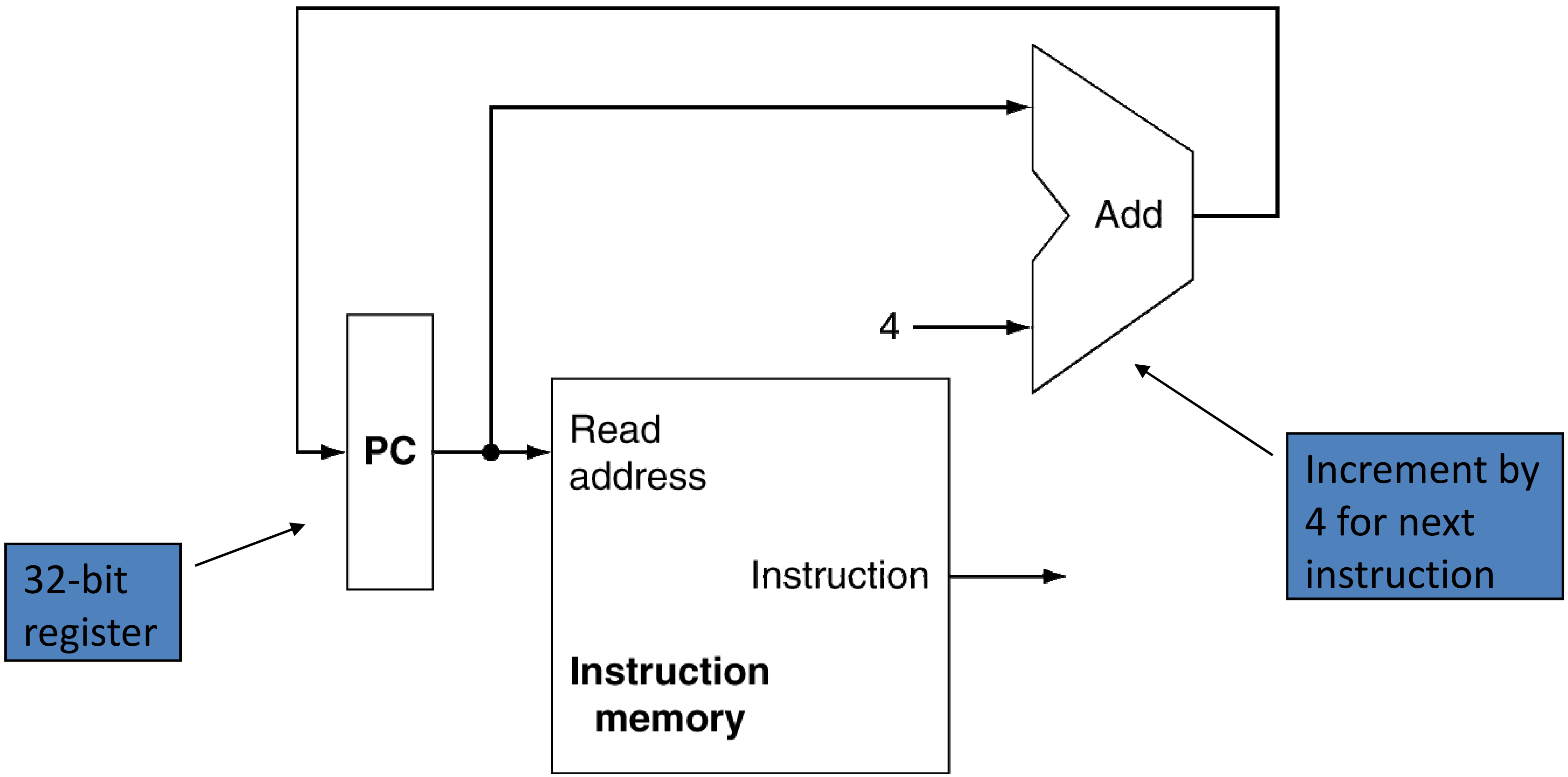
Control


Branch
M
u
x
4
Add
Add
M
u
x
ALU operation
Data
Register #
Registers
Register #
Register #
RegWrite
PC
Address
Instruction
Instruction memory
M
u
x
ALU
Zero
MemWrite
Address
Data memory
Data
MemRead
Control

Building a Datapath

- Datapath
  – Elements that process data and addresses in the CPU
    - Registers, ALUs, mux’s, memories, …

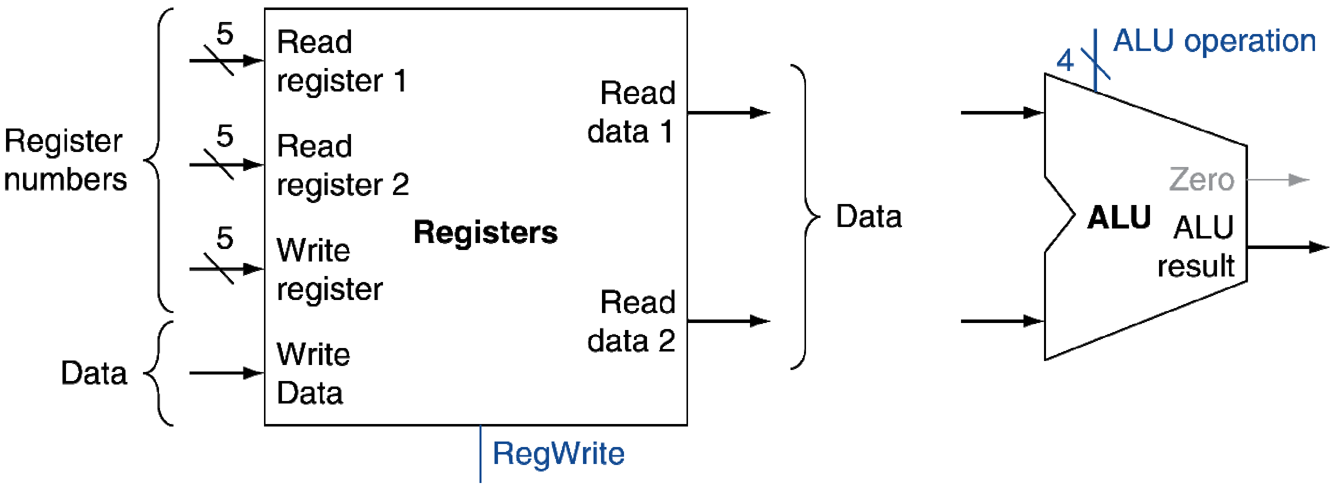
Instruction Fetch



R-Format Instructions

- The processor's 32 general-purpose registers are stored in a structure called a **register file.**
- Read two register operands
- Perform arithmetic/logical operation
- Write register result

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits



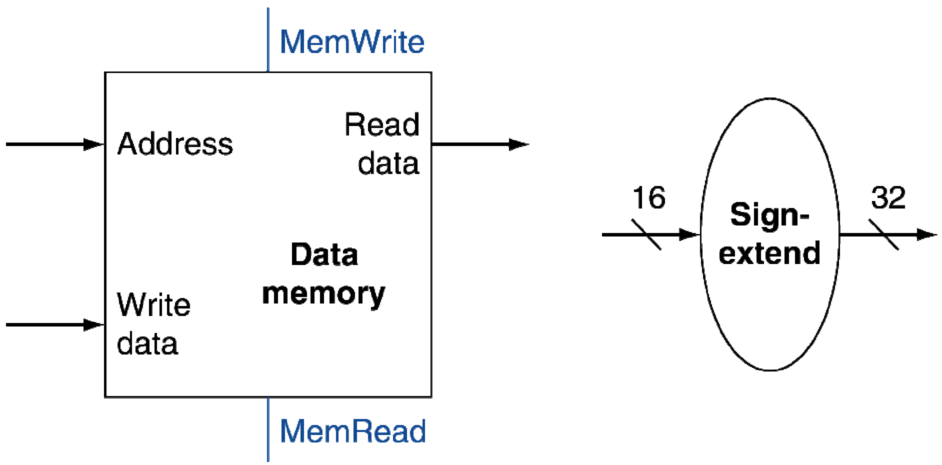
a. Registers

b. ALU

Load/Store Instructions

- Read register operands
- Calculate address using 16-bit offset
  - Use ALU, but sign-extend offset
- Load: Read memory and update register
- Store: Write register value to memory

op	rs	rt	constant or address
6 bits	5 bits	5 bits	16 bits



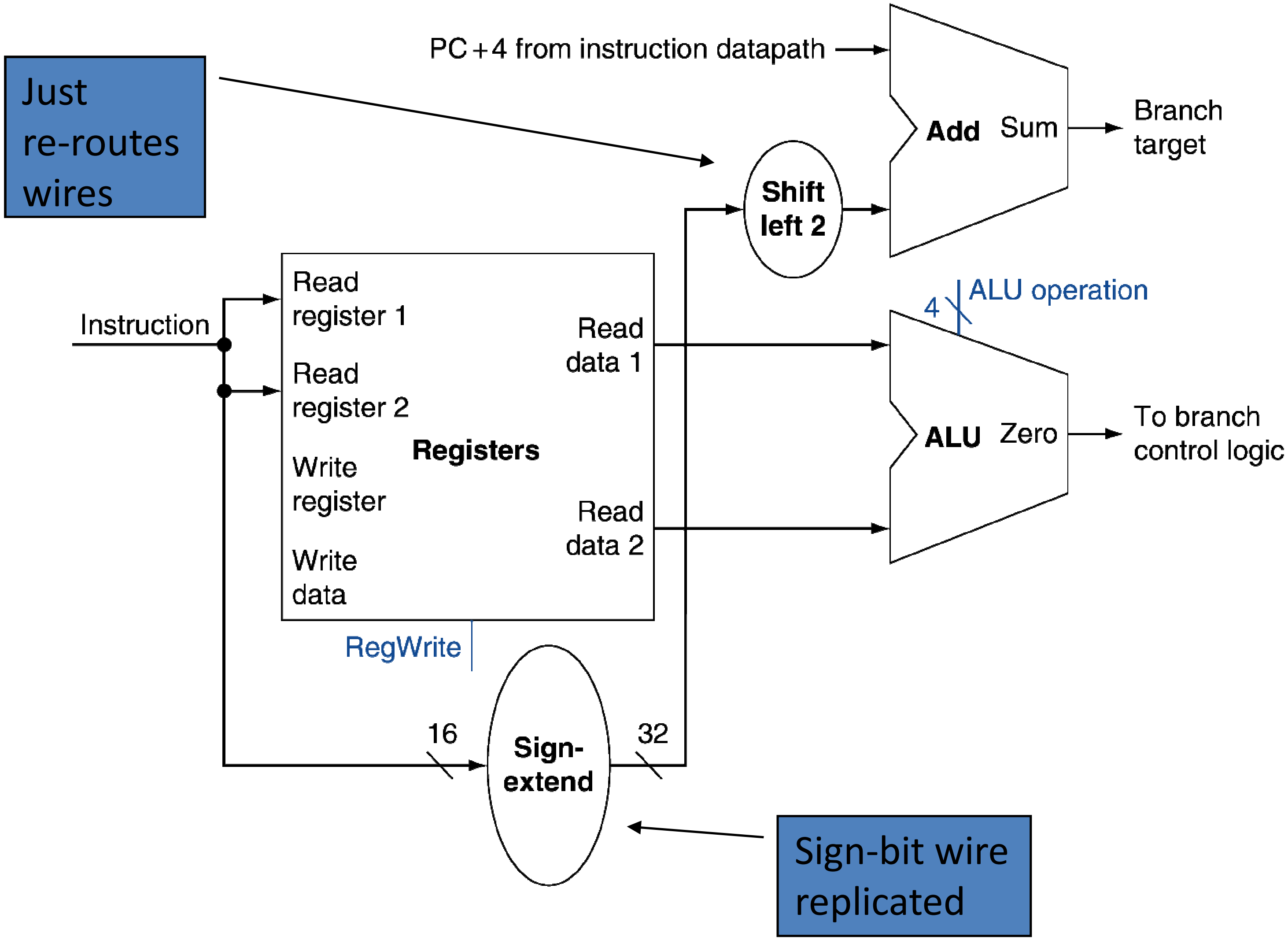
a. Data memory unit

b. Sign extension unit

Branch Instructions

- Read register operands
- Compare operands
  - Use ALU, subtract and check Zero output
- Calculate target address
  - Sign-extend displacement
  - Shift left 2 places (word displacement)
  - Add to PC + 4
    - Already calculated by instruction fetch

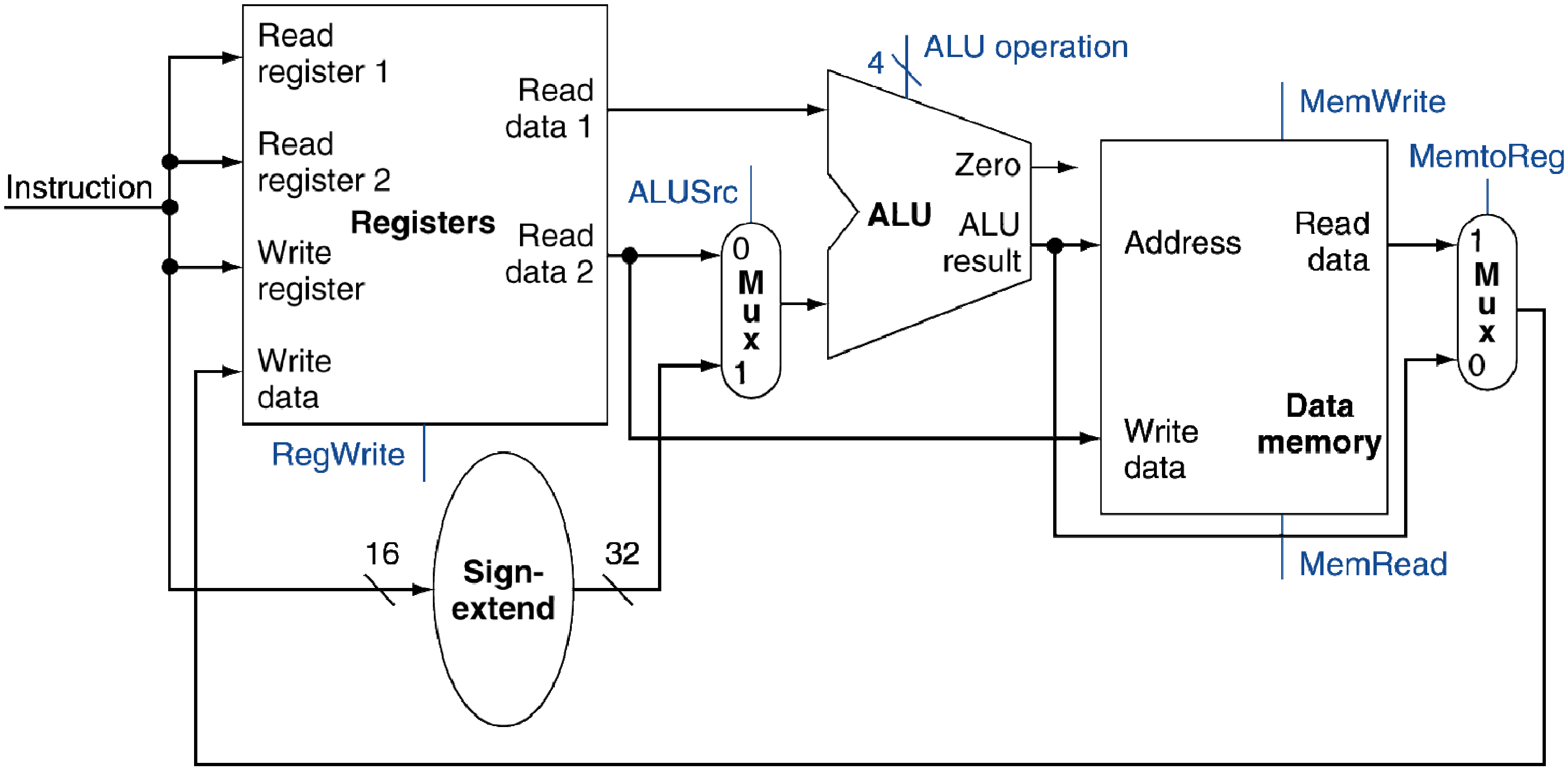
Branch Instructions



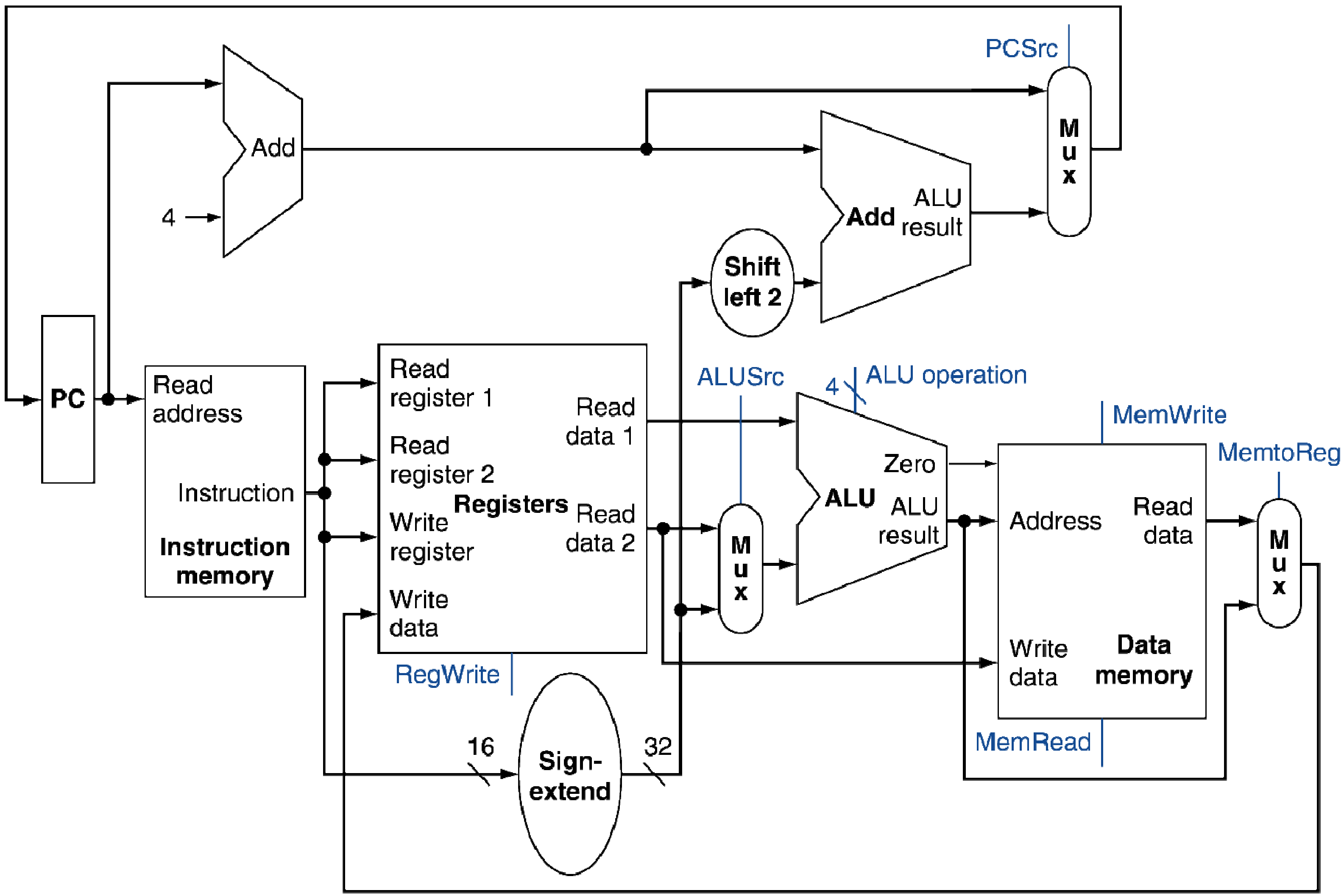
Composing the Elements

- First-cut data path does an instruction in one clock cycle
  - Each datapath element can only do one function at a time
  - Hence, we need separate instruction and data memories
- Use multiplexers where alternate data sources are used for different instructions

R-Type/Load/Store Datapath



Full Datapath : add,lw,beq



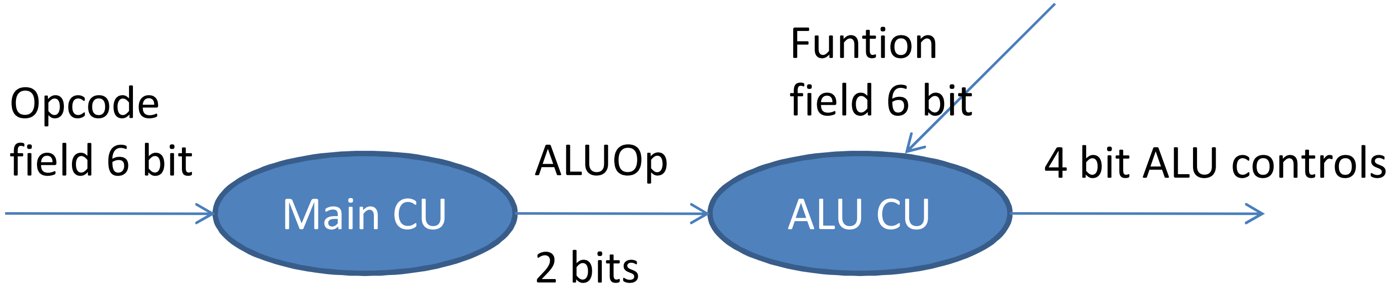
The simple datapath for the core MIPS architecture combines the elements required by different instruction classes.

A Simple Implementation Scheme

Control Unit

- Nine Control signals are generated by the main control unit using opcode from the Instruction
  - RegDst
  - ALUSrc
  - MemtoReg
  - RegWrite
  - MemRead
  - MemWrite
  - Branch
  - Jump
  - ALUOp1 ALUOp0
- ALUControl (4) bits are generated by the ALUCU using two bit ALUOp and function field from the instruction

- Multiple levels of decoding
  - reduce the size of the main control unit.
  - increase the speed of the control unit

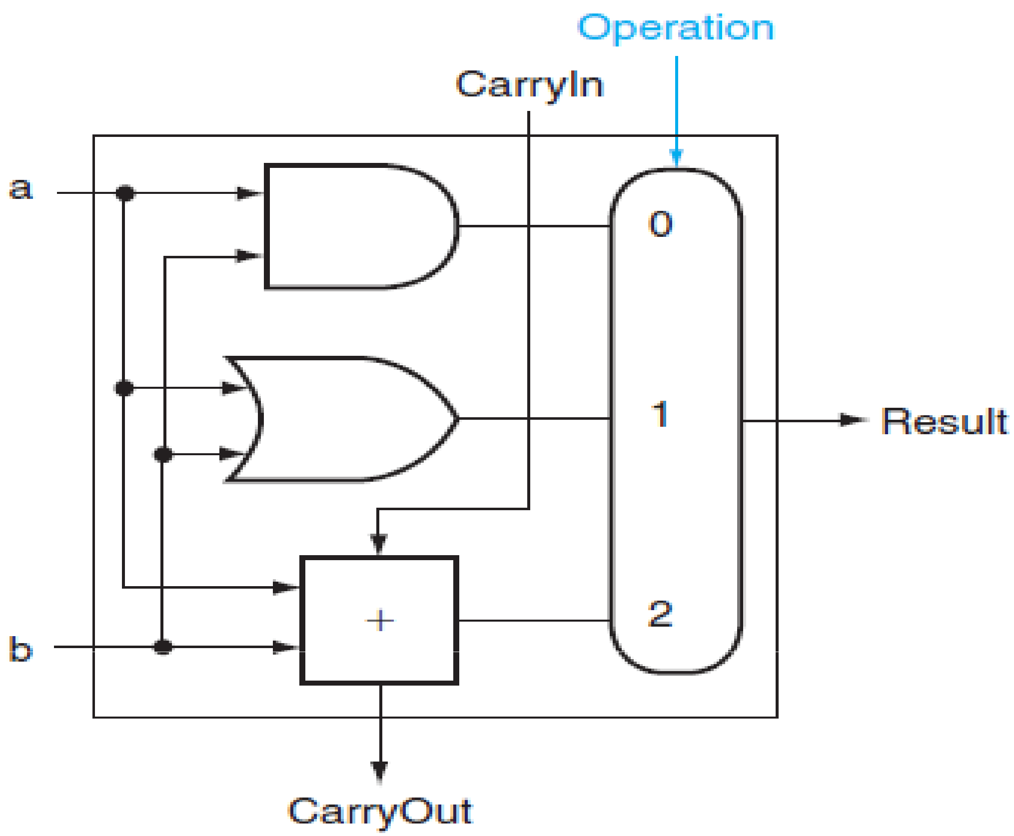


A Simple Implementation Scheme

ALU Control

- ALU used for
  - Load/Store: F = add (to compute memory address) (00)
  - Branch: F = subtract (to check if register content are equal) (01)
  - R-type: F = and, or, add, sub, slt (depends on funct field) (10)

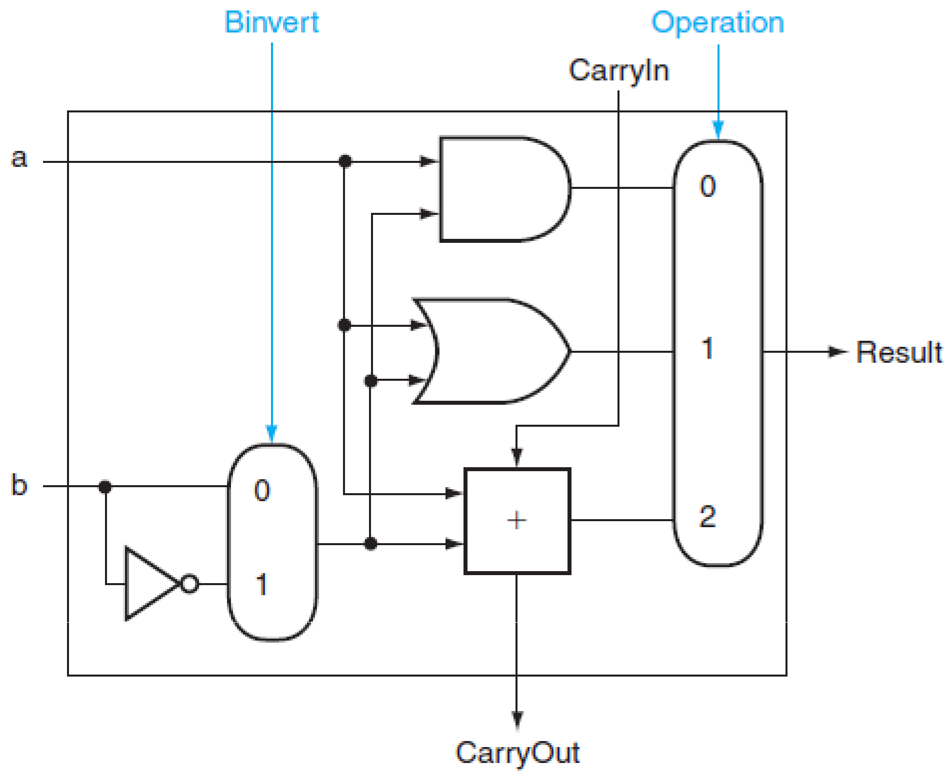
ALU control	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set-on-less-than
1100	NOR



ALU Control

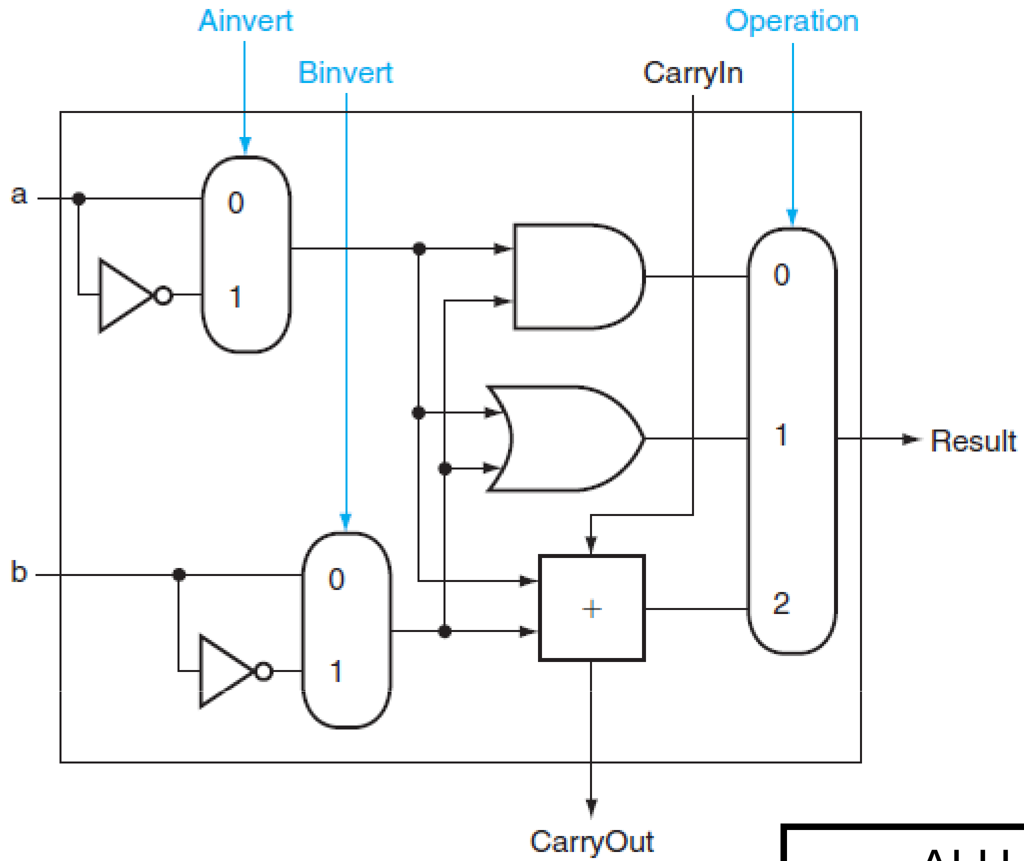
ALU control	Function
00	AND
01	OR
10	add
10	subtract

ALU Control



ALU control	Function
000	AND
001	OR
010	add
110	subtract

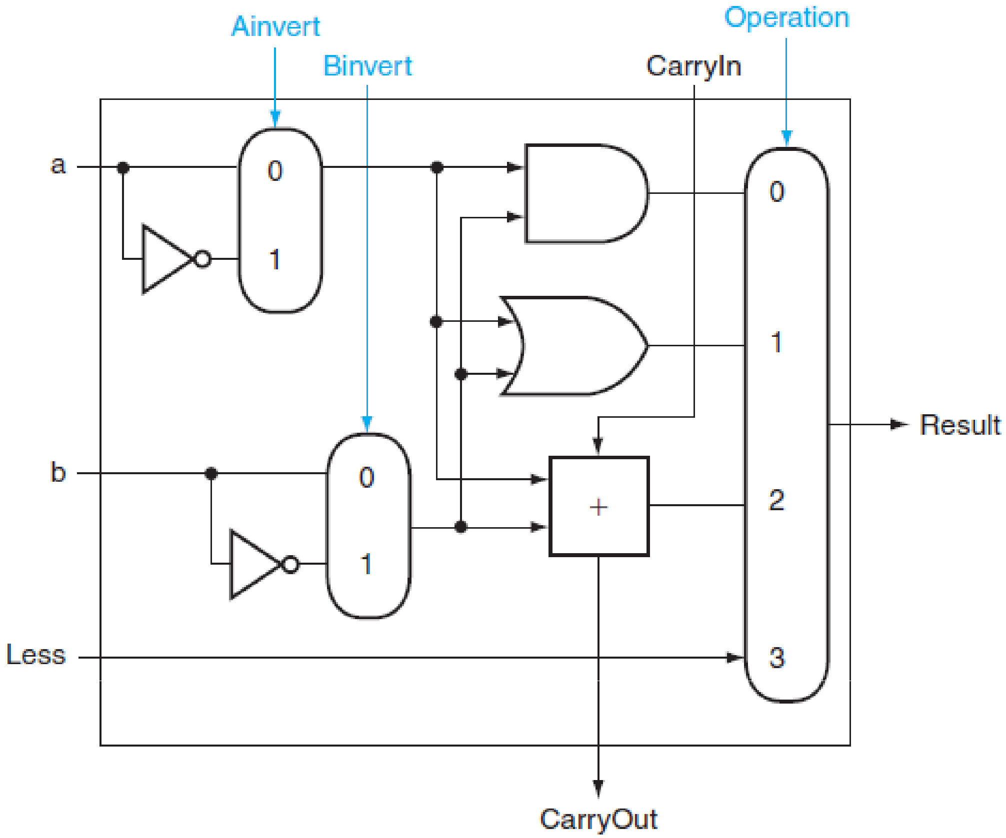
ALU Control



A	1101	A`	0010
B	1001	B`	0110
A | B	1101	A` & B`	0010
(A | B)`	0010		
Demorgans : a NOR b = (a or b)` = a’ AND b’.			

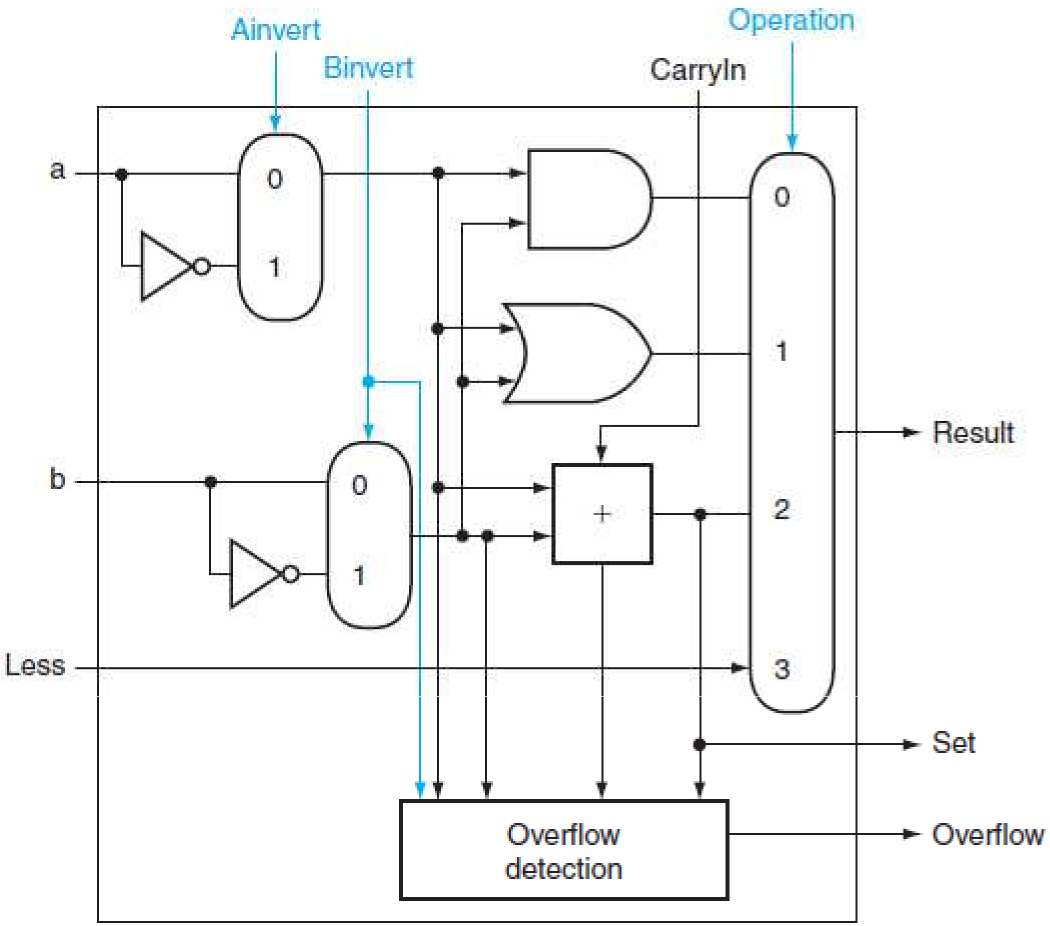
ALU control	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set-on-less-than
1100	NOR

ALU Control



ALU control	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set-on-less-than
1100	NOR

ALU Control



- MSB Bit

If A-B = -ive then
A<B is true // set 1
Else
A<B is false // set 0

A=4 -> 0100
B=5 -> 1011
A-B -> 1111 (-ive)
Set =1=MSB it

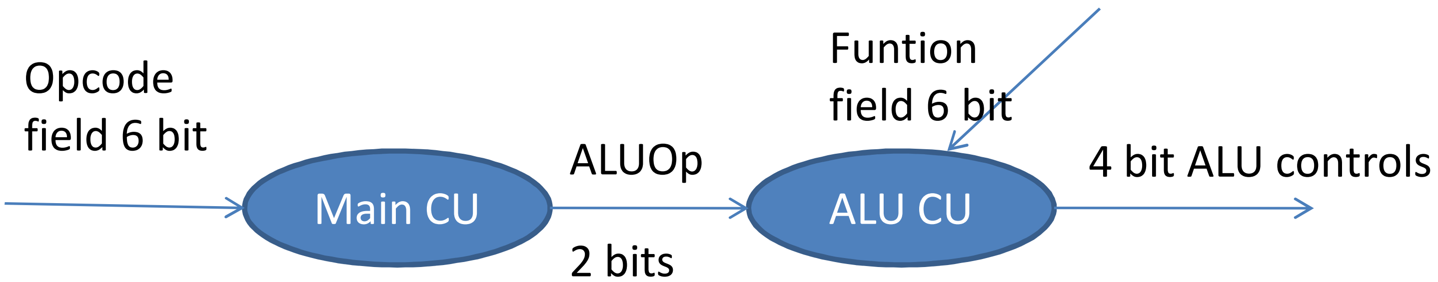
A=5 -> 0101
B=4 -> 1100
A-B -> 0001 (+ive)
set=0 =MSB bit

ALU Control

- 4 bit ALU controls are generated by a small control unit based on value of the 6-bit funct field and 2 bit control field ALUOp

Instruction opcode	ALUOp
LW	00
SW	00
Branch equal	01
R-type	10

ALU control	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set-on-less-than
1100	NOR



ALU Control

- Assume 2-bit ALUOp derived from opcode
  - Only for R type the ALU control depends on funct field
  - Combinational logic derives ALU control
  - K-map(6 bit – 64 combinations and four bit output function)

opcode	ALUOp	Operation	Funct	ALU function	ALU control
lw	00	load word	XXXXXX	add	0010
sw	00	store word	XXXXXX	add	0010
beq	01	branch equal	XXXXXX	subtract	0110
R-type	10	Add (32)	100000	add	0010
		Subtract (34)	100010	subtract	0110
		AND (36)	100100	AND	0000
		OR (37)	100101	OR	0001
		set-on-less-than (42)	101010	set-on-less-than	0111

ALU Control

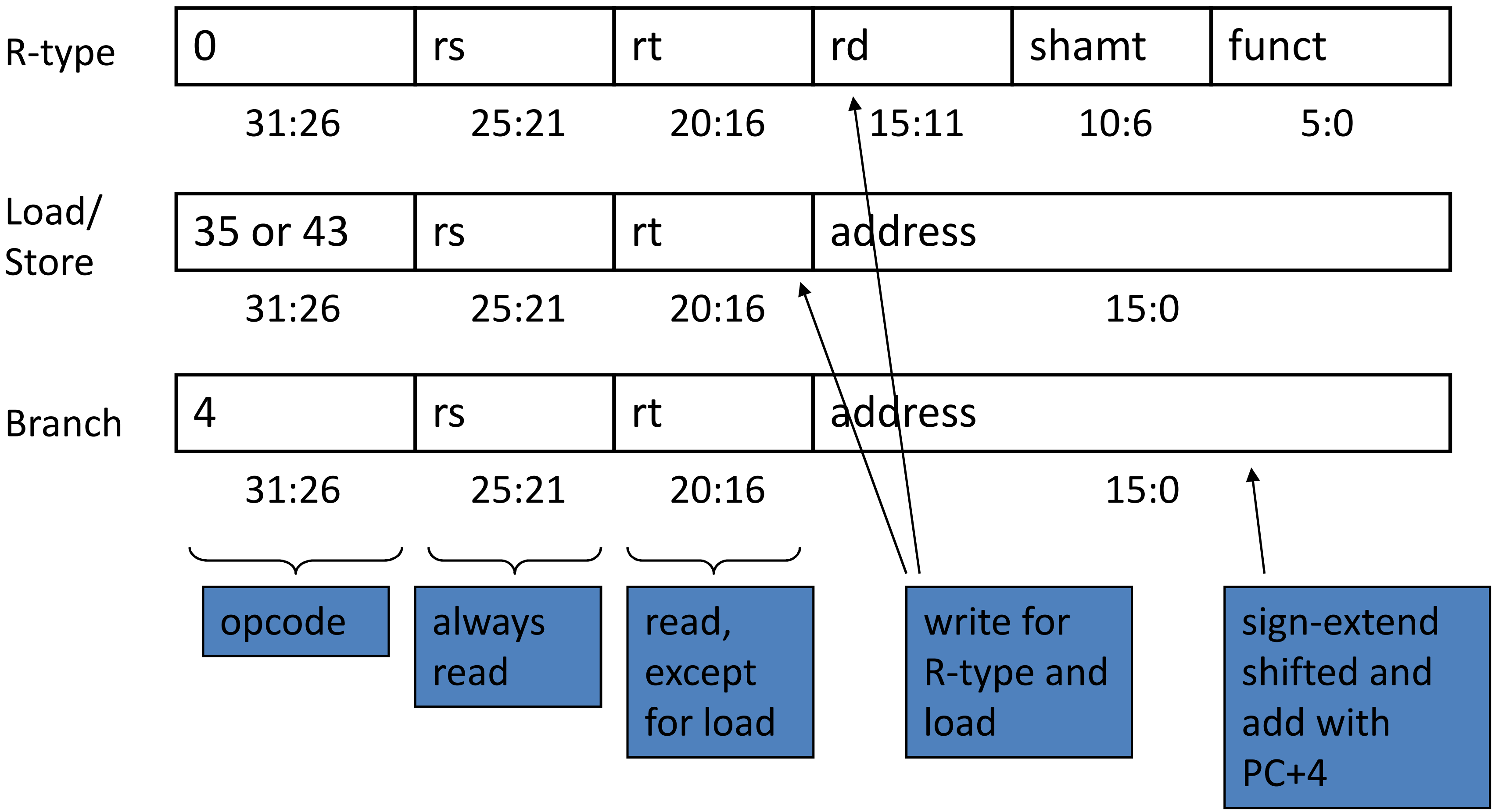
- Create truth table for the interesting combinations of the function code field and the ALUOp bits
- Once the truth table has been constructed, it can be optimized and then turned into gates

- 00 : lw/sw
- 01 : beq X1
- 10 :R type 1X
- 11 dont care

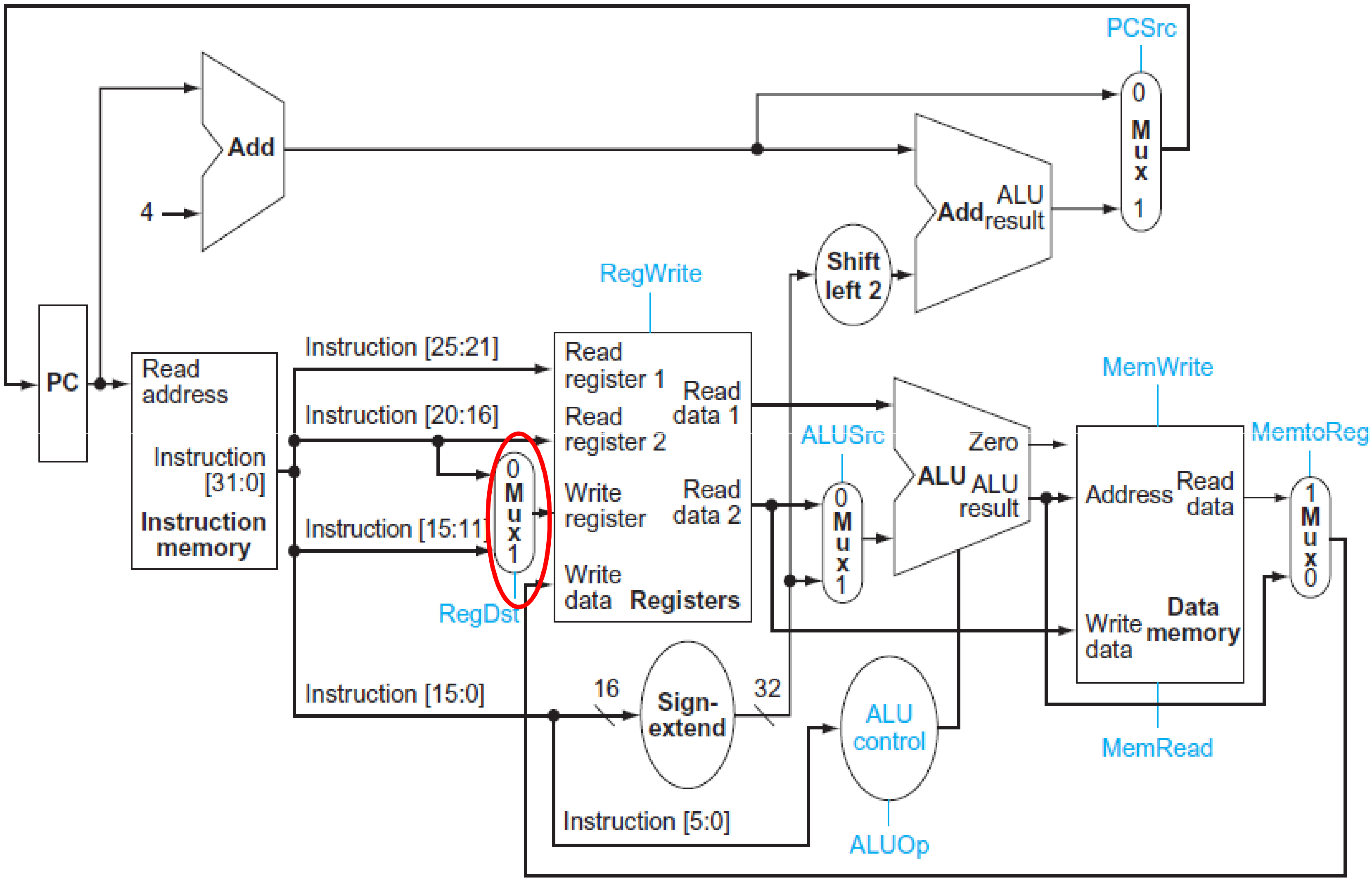
ALUOp		Funct field						
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	Operation
0	0	X	X	X	X	X	X	0010
X	1	X	X	X	X	X	X	0110
1	X	X	X	0	0	0	0	0010
1	X	X	X	0	0	1	0	0110
1	X	X	X	0	1	0	0	0000
1	X	X	X	0	1	0	1	0001
1	X	X	X	1	0	1	0	0111

The Main Control Unit

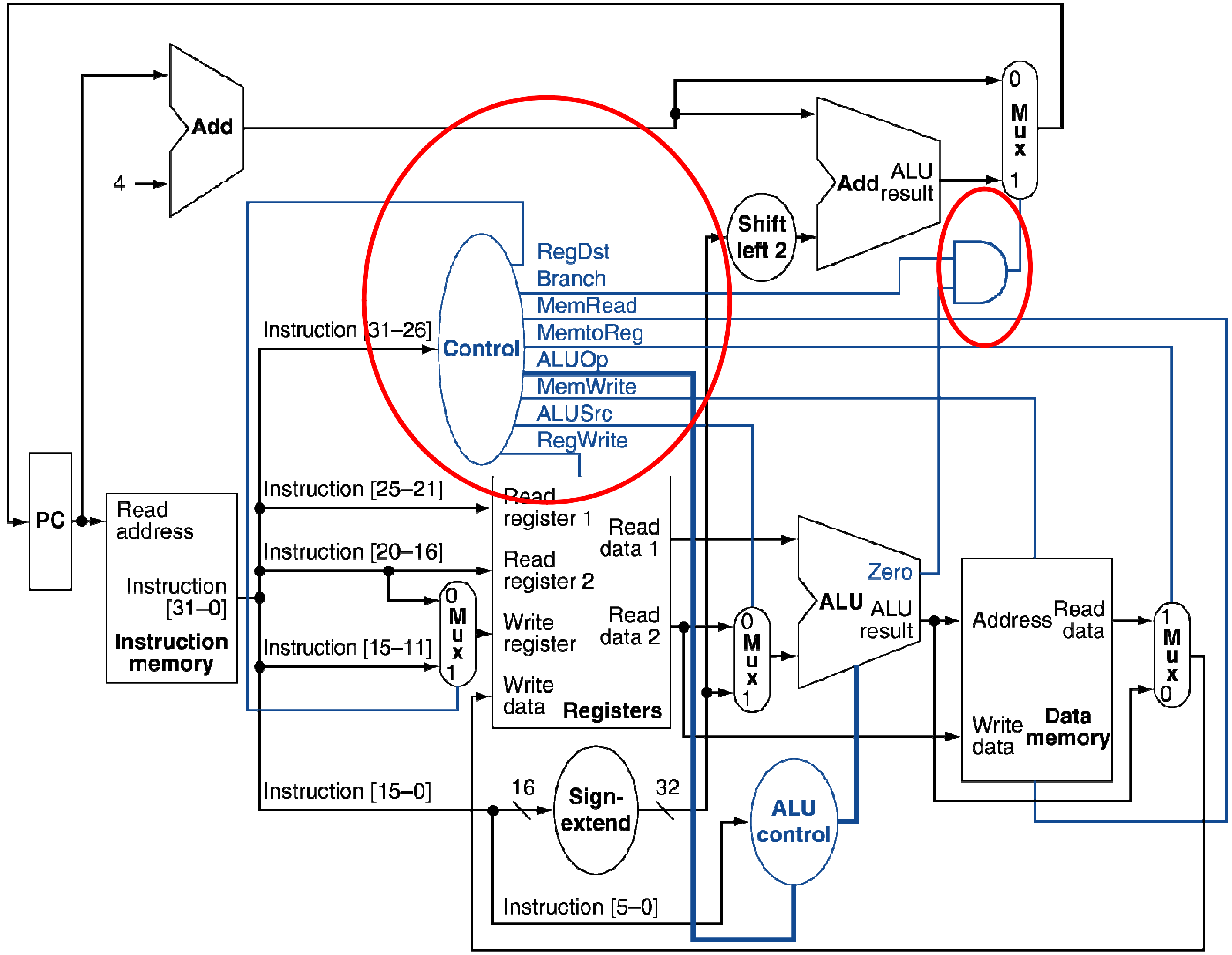
- Control signals derived from instruction



The datapath with all necessary multiplexors and all control lines



Datapath With Control



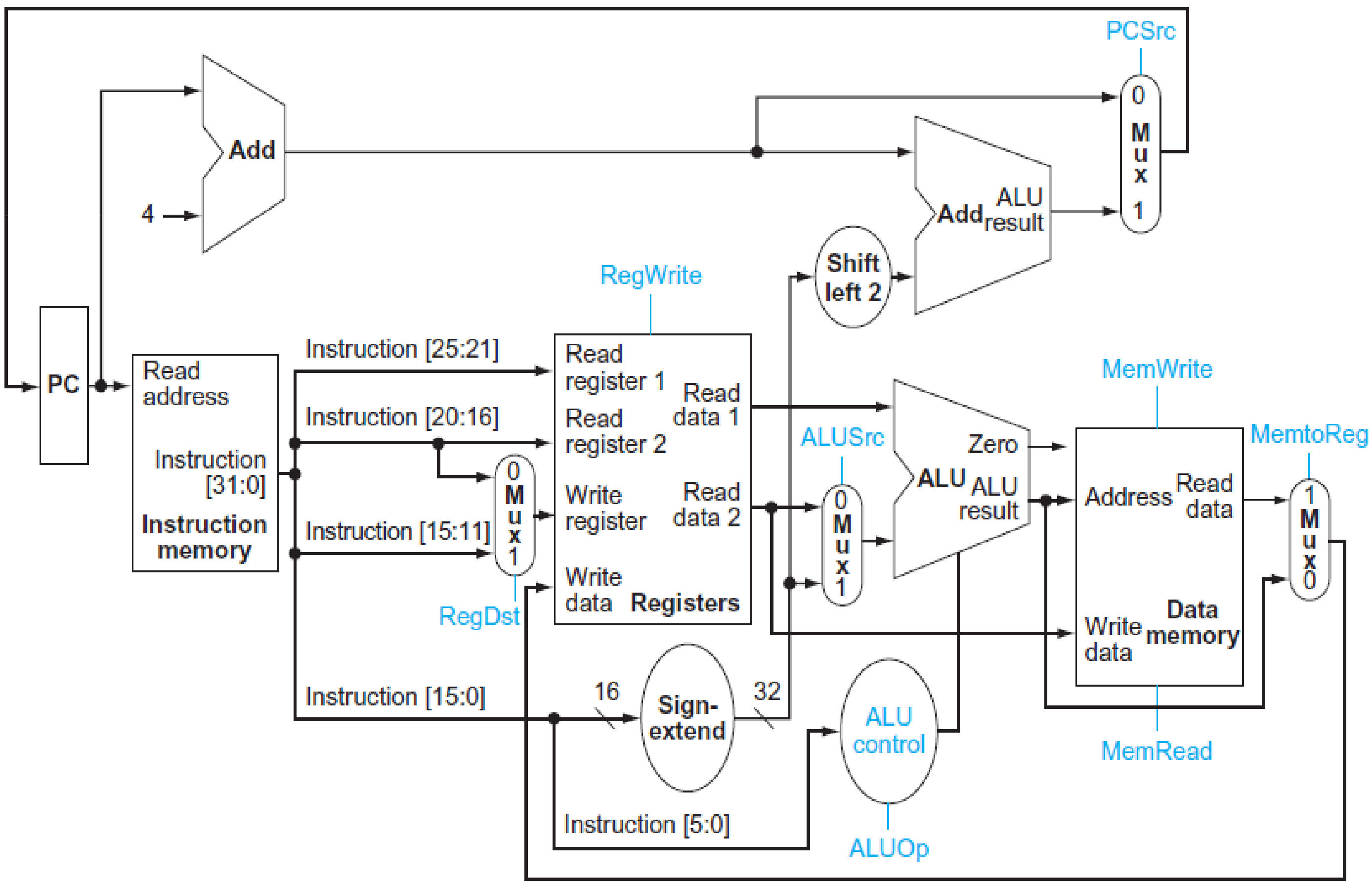
Datapath With Control

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 effect of each of the seven control signals. When the 1-bit control to a two way multiplexor is asserted, the multiplexor selects the input corresponding to 1. Otherwise, if the control is deasserted, the multiplexor selects the 0 input.

Nine control signals : AUOp 2 bit control

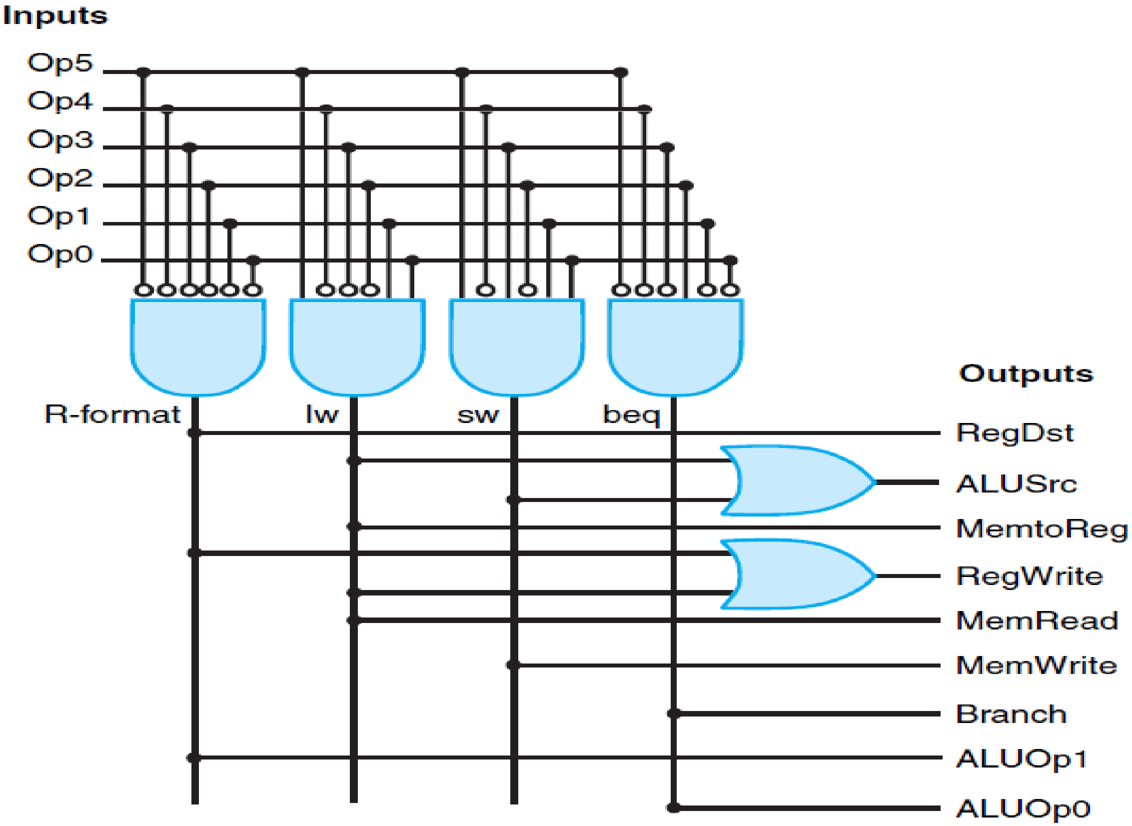
Instruction	RegDst	ALUSrc	Memto-Reg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
R-format	1	0	0	1	0	0	0	1	0
`lw`	0	1	1	1	1	0	0	0	0
`sw`	X	1	X	0	0	1	0	0	0
`beq`	X	0	X	0	0	0	1	0	1



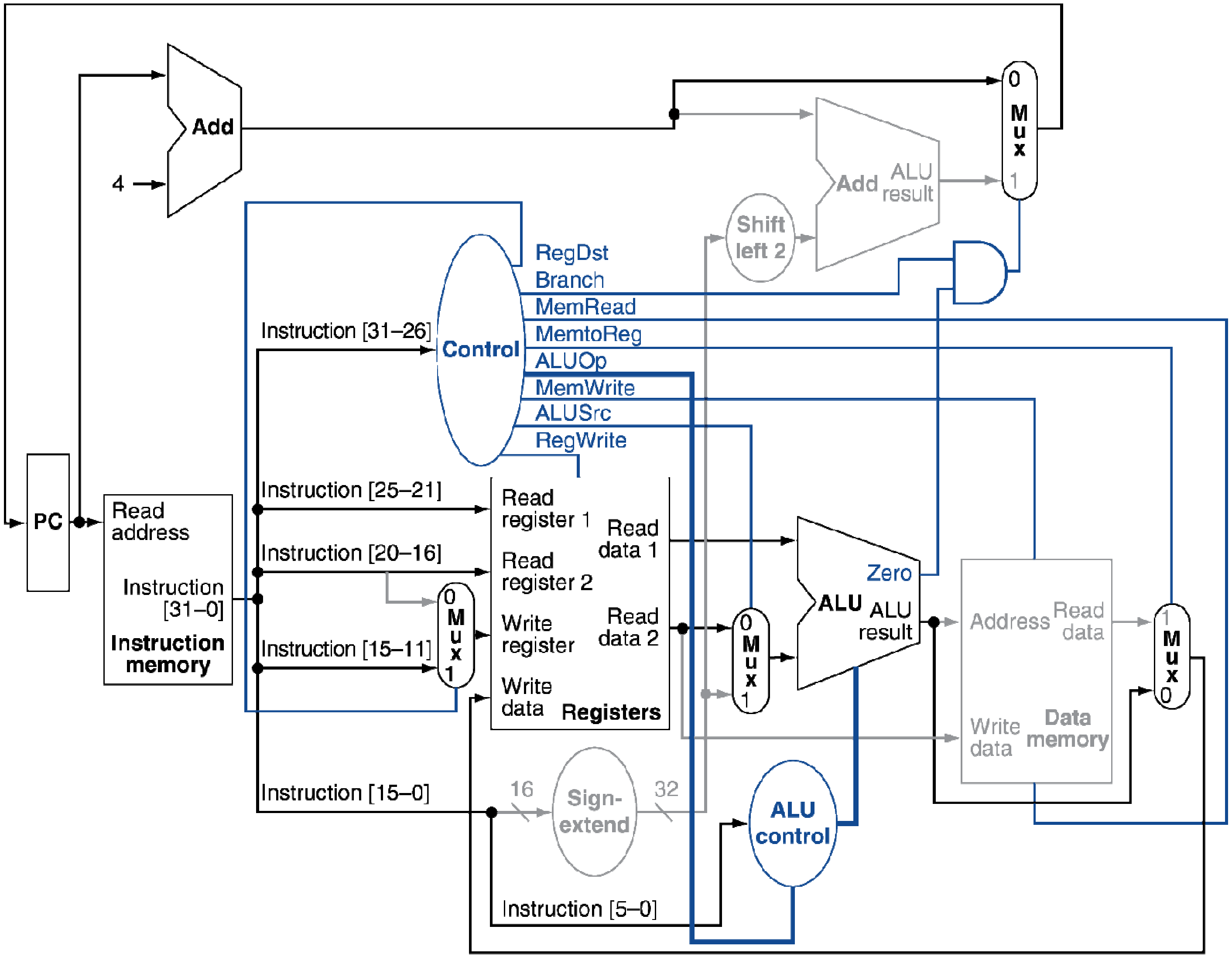
Instruction	RegDst	ALUSrc	Memto-Reg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
R-format	1	0	0	1	0	0	0	1	0
`lw`	0	1	1	1	1	0	0	0	0
`sw`	X	1	X	0	0	1	0	0	0
`beq`	X	0	X	0	0	0	1	0	1

Input or output	Signal name	R-format	`lw`	`sw`	`beq`
Inputs	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Outputs	RegDst	1	0	X	X
	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1

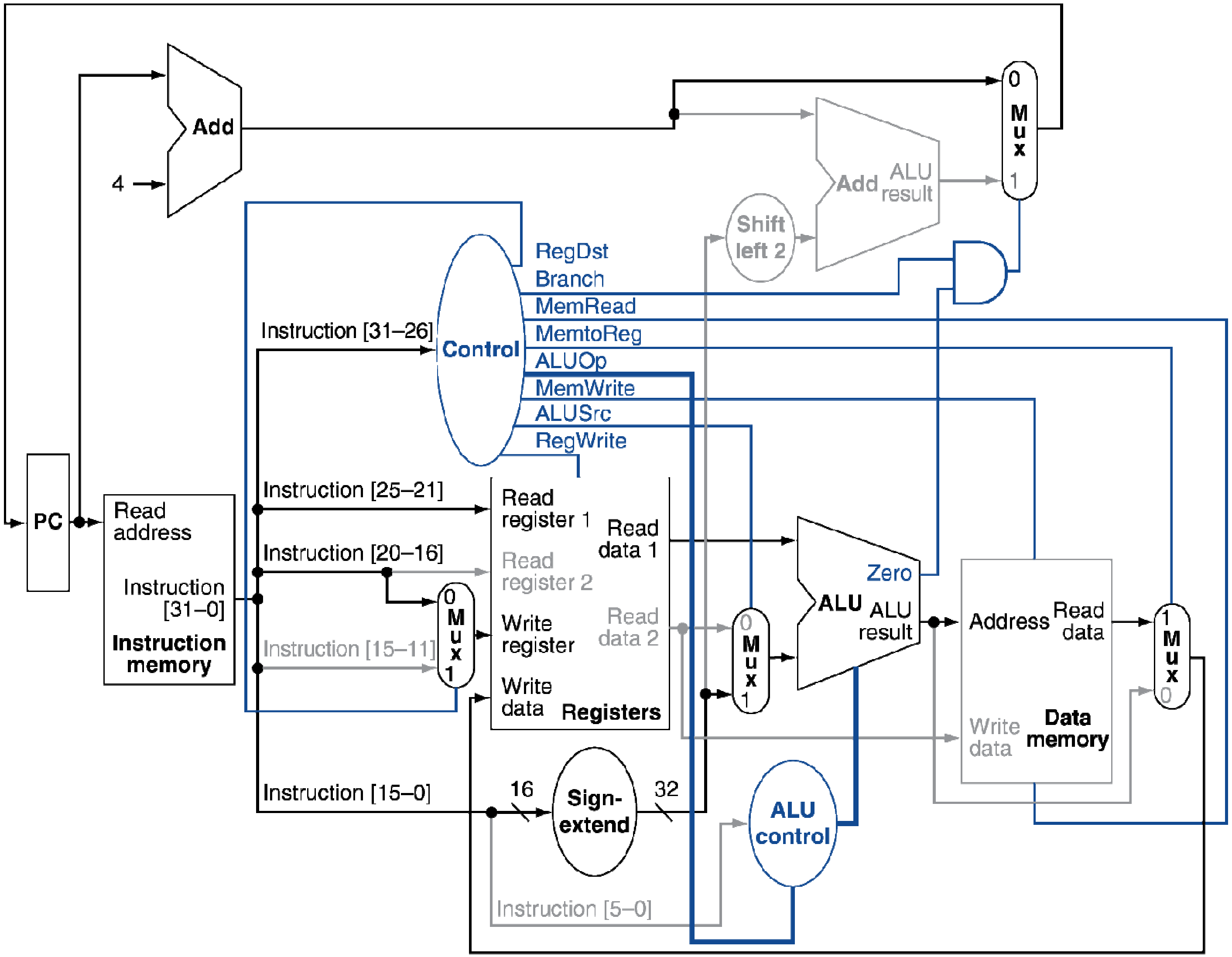
Input or output	Signal name	R-format	lw	sw	beq
Inputs	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Outputs	RegDst	1	0	X	X
	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1



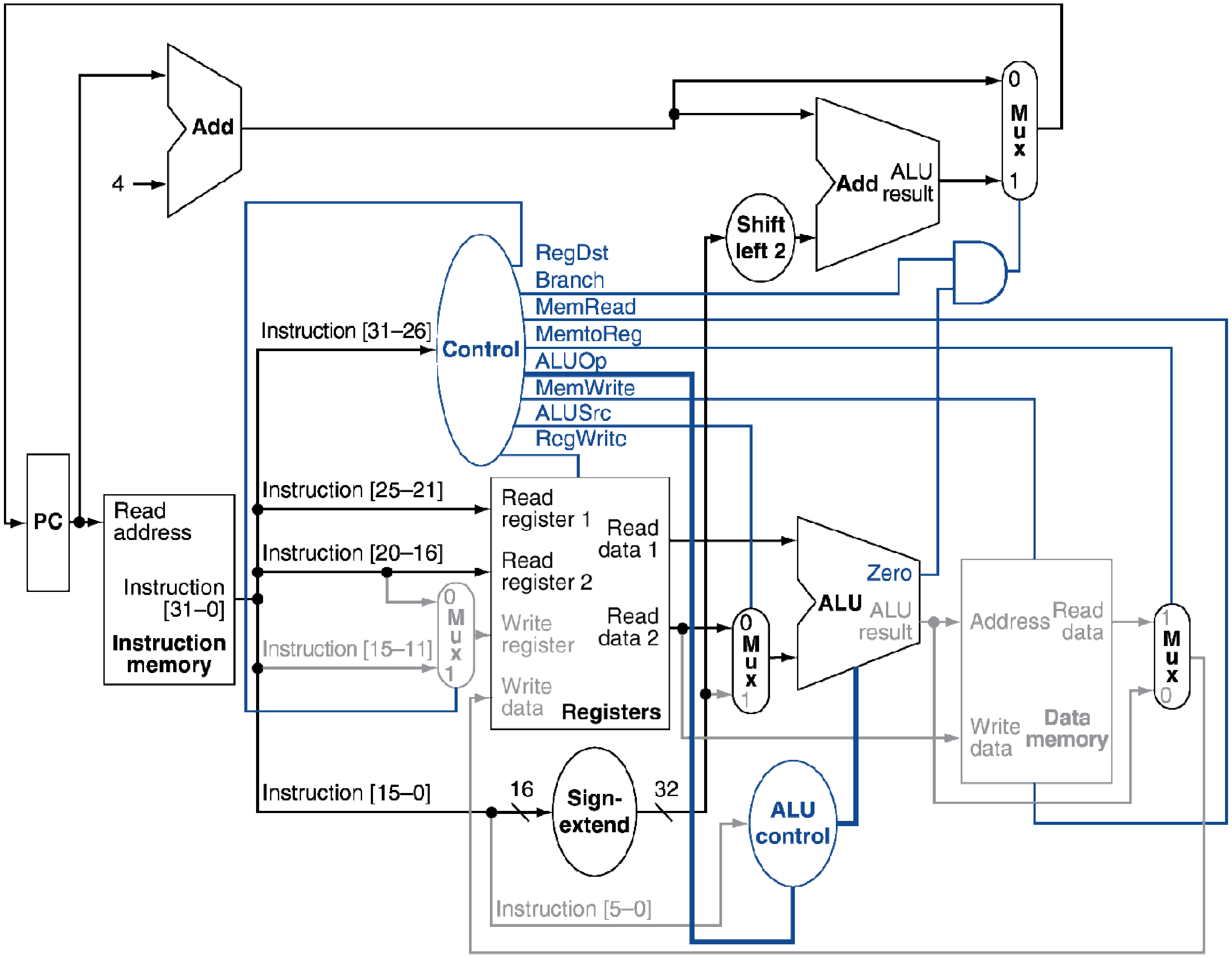
R-Type Instruction



Load Instruction



Branch-on-Equal Instruction

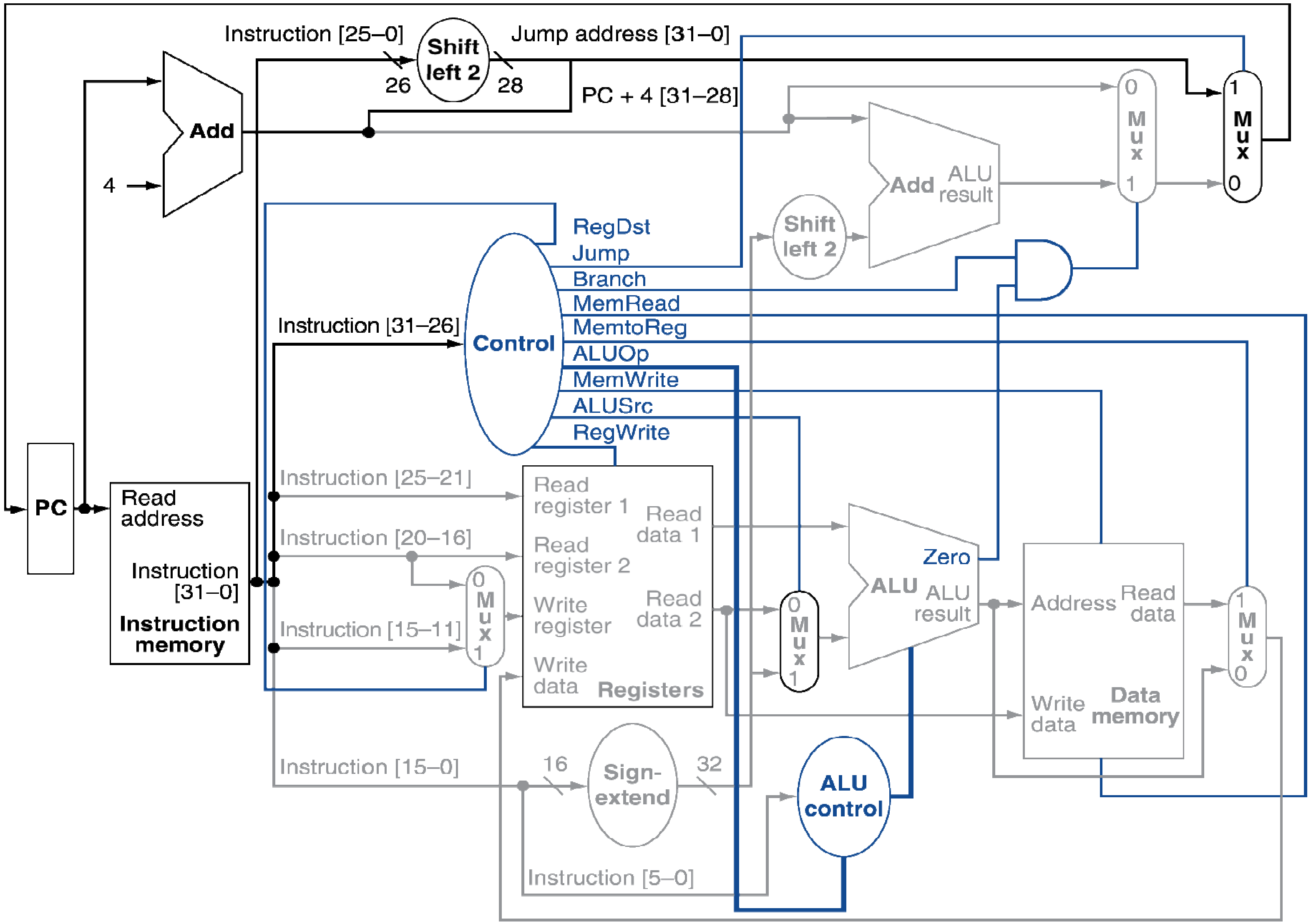


Implementing Jumps

Jump	2	address
	31:26	25:0

- Jump uses word address
- Update PC with concatenation of
  - Top 4 bits of old PC
  - 26-bit jump address
  - 00
- Need an extra control signal decoded from opcode

Datapath With Jumps Added



Performance Issues

- Longest delay determines clock period
  - Critical path: load instruction
  - Instruction memory $\rightarrow$ register file $\rightarrow$ ALU $\rightarrow$ data memory $\rightarrow$ register file
- Not feasible to vary period for different instructions
- Violates design principle
  - Making the common case fast
- We will improve performance by pipelining