Computer Organization and Software Systems

Session 6

Dr. Lucy J. Gudino
WILP & Department of CS & IS

BITS Pilani
Pilani Campus

MIPS Architecture

- MIPS follows RISC principles and based on Harvard Architecture
- MIPS architecture is a register architecture
- MIPS ISA is Load/Store architecture
- R2000 is a 32-bit processor
- Uses fixed length instruction format.

MIPS Register Set

Types of Registers

- 32 general-purpose registers ($0 – $31)
- Program Counter (PC)
- Two special Purpose register (HI and LO)
  - Used to hold the results of integer *multiply* and *divide* instruction.
  - integer *multiply* operation, HI and LO register hold the 64-bit result.
  - integer *divide* operation, the 32-bit quotient is stored in the LO and the remainder in the HI register.

General-Purpose Register Usage Convention

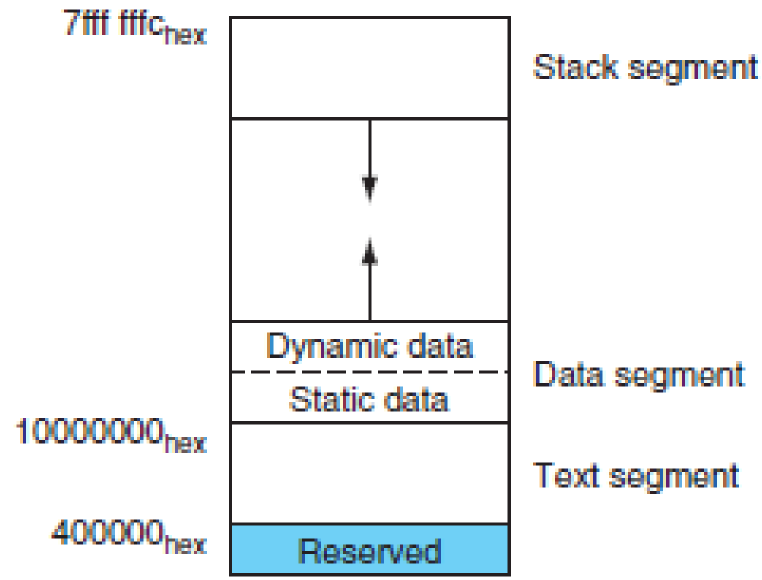
>>

Back

Register name	Number	Intended usage
Zero	0	Constant 0
at	1	Reserved for assembler
v0,v1	2,3	Values for function results and Exp evaluation
a0,a1,a2,a3	4-7	Arguments 1-4
t0-t7	8-15	Temporary (not preserved across call)
s0-s7	16-23	Saved temporary(preserved across call)
t8,t9	24,25	Temporary (not preserved across call)
k0,k1	26,27	Reserved for OS kernel
gp	28	Pointer to Global area
sp	29	Stack Pointer
fp	30	Frame pointer(if needed);Otherwise, a saved register $s8
ra	31	Return address(used by a procedure call)

Memory Usage

- A program's address space consists of three parts:
  - Code/Text segment
  - Data segment
  - Stack segment



Instruction Format

- Fixed-length instruction format
- 32-bits long
- Three different instruction formats:

  1. Immediate (I-type)
  2. Jump (J-type)
  3. Register (R-type)
- Meaning of various fields in the instruction format
  - op: Opcode
  - rs: The first register source operand
  - rt: The second register source operand
  - rd: The register destination operand
  - shamt / sa: shift amount
  - funct: function code

Instruction Format

[Register File]

R-Type:

31 – 26	25 – 21	20 – 16	15 – 11	10 – 6	5 – 0
op	**rs**	**rt**	**rd**	**shamt**	**funct**
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

I-Type:

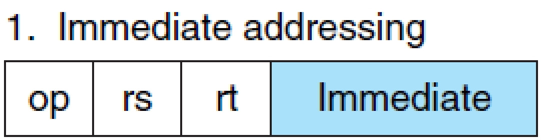
31 – 26	25 – 21	20 – 16	15 – 0
op	**rs**	**rt**	**immediate**
6 bits	5 bits	5 bits	16 bits

J-Type:

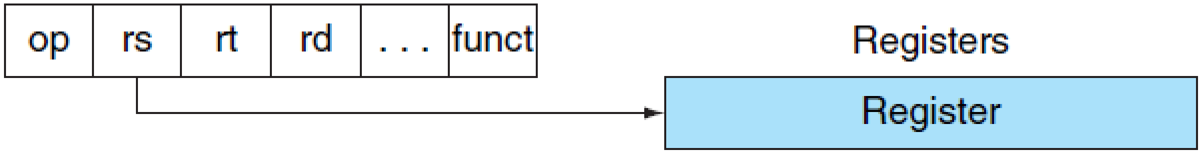
31 – 26	25 – 0
op	**target address**
6 bits	26 bits

Addressing modes

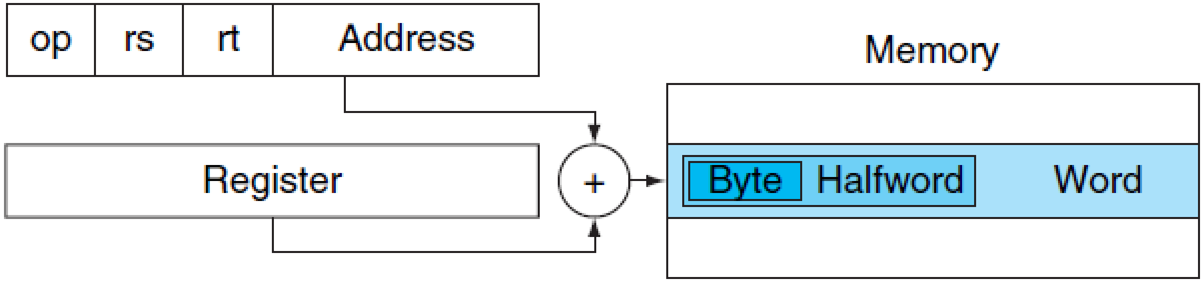
- Immediate Addressing Mode



- Register Addressing Mode

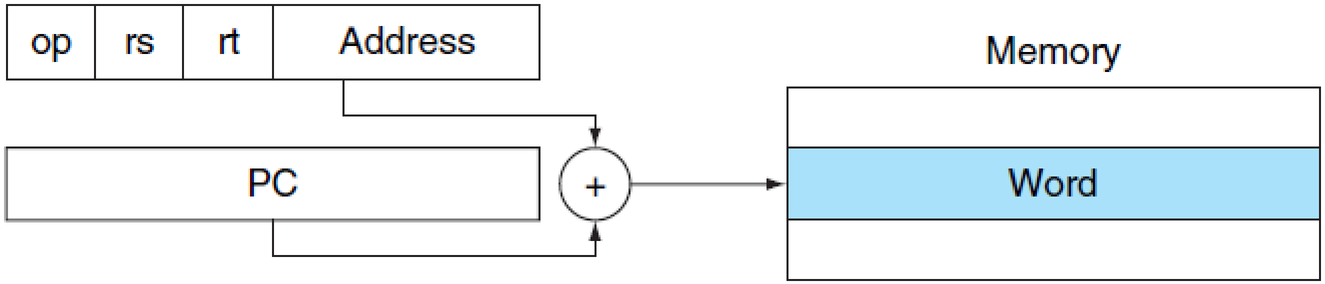


- Base or Displacement Addressing Mode

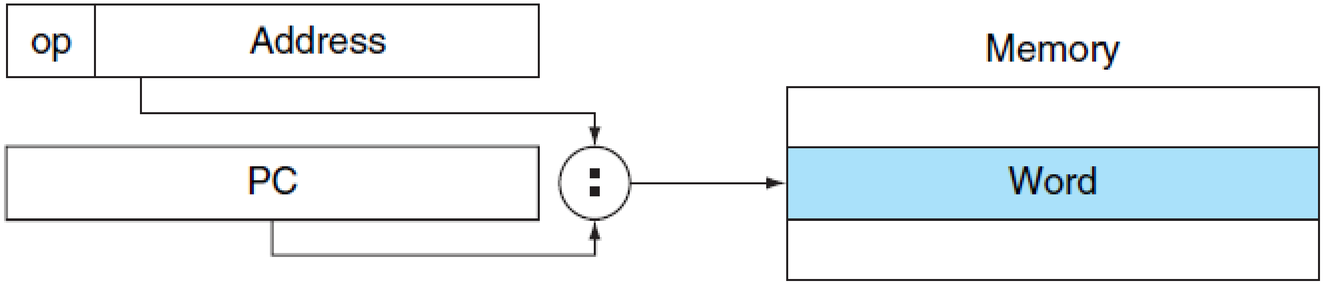


Addressing modes

- PC- Relative Addressing Mode



- Pseudo Direct Addressing Mode



MIPS Instruction Set

- Arithmetic Instructions (ADD, SUB etc.)
- Logical Instructions (OR, AND, SHIFT, etc.)
- Data Transfer Instructions (Load and Store)
- Decision Making Instructions (J, BNE, BEQ, etc.)
- Stack Related Instructions (PUSH and POP)

A Basic MIPS Implementation

- Basic implementation includes a subset of core MIPS instruction set
  - Memory reference instruction
    - lw and sw
  - Arithmetic and logical instructions
    - add, sub, and, or and slt
  - Branch instructions
    - beq and j

Memory Reference Instruction - lw

- copies data from memory to register
- Format : lw reg, address
- Example : lw \$s1, 100(\$s2)
- $\$s1 \leftarrow memory[100 + \$s2]$
- Alignment restriction
- MIPS is Big endian
- Spilling registers

Double Word							
Word				Word			
Halfword		Halfword		Halfword		Halfword	
Byte	Byte	Byte	Byte	Byte	Byte	Byte	Byte
0	1	2	3	4	5	6	7

	s2	s1	
0x23	rs	rt	Offset
6	5	5	16

Memory Reference Instruction - sw

- Store Instruction
- copies data from register to memory
- Format : sw reg, address
- Example : sw \$s1, 100(\$s2)

memory[100 +\$s2]← \$s1

	s2	s1	
0x2b	rs	rt	Offset
6	5	5	16

Arithmetic Instructions :add

add des, src1, src2

Addressing mode: register

Example: add \$s3, \$s1, \$s2

meaning : \$s3 = \$s1 + \$s2

opcode : 0 , function: 0x20

sa :0

	\$s1	\$s2	\$s3		
0	rs	rt	rd	0	0x20
6	5	5	5	5	6

Arithmetic Instructions :sub

sub des, src1, src2

Addressing mode: register

Example: sub \$s3, \$s1, \$s2

meaning : \$s3 = \$s1 - \$s2

opcode : 0 , function: 0x22

sa :0

	\$s1	\$s2	\$s3		
0	rs	rt	rd	0	0x22
6	5	5	5	5	6

Logical instructions :and

and des, src1, src2

- Addressing mode: register
- Example: and \$s3, \$s1, \$s2
  - meaning : \$s3 = \$s1 & \$s2 (bit by bit)
  - opcode : 0 , function: 0x24
  - sa :0

	\$s1	\$s2	\$s3		
0	rs	rt	rd	0	0x24
6	5	5	5	5	6

Logical instructions :or

or des, src1, src2

- Addressing mode: register
- Example: or $s3, $s1, $s2
  - meaning : $s3 = $s1 | $s2 (bit by bit)
  - opcode : 0 , function: 0x25
  - sa :0

	$s1	$s2	$s3		
0	rs	rt	rd	0	0x25
6	5	5	5	5	6

Logical instructions :slt

set on less than : slt

Format: slt des, src1, src2

set des = 1, if src1 < src2

Example:

slt $s3, $s1, $s2

	$s1	$s2	$s3		
0	rs	rt	rd	0	0x2a
6	5	5	5	5	6

Set register rd to 1 if rs is less than rt, and to 0 otherwise

Branch Instructions: beq and j

beq : branch if equal

- format : beq $s1, $s2, label
- If $s1 = $s2 then branch to address specified as Label

4	rs	rt	Offset
6	5	5	16

j : jump unconditional

- format : j label

2	target
6	26

Back

Summary

- lw $s1, 100($s2)

0x23	rs	rt	Offset
6	5	5	16

- sw $s1, 100($s2)

0x2b	rs	rt	Offset
6	5	5	16

- add $s1, $s2, $s3

0	rs	rt	rd	0	0x20
6	5	5	5	5	6

- sub $s1, $s2, $s3

0	rs	rt	rd	0	0x22
6	5	5	5	5	6

- and $s1, $s2, $s3

0	rs	rt	rd	0	0x24
6	5	5	5	5	6

- or $s1, $s2, $s3

0	rs	rt	rd	0	0x25
6	5	5	5	5	6

- slt $t0, $s1, $s2

0	rs	rt	rd	0	0x2a
6	5	5	5	5	6

- beq $s1, $s2, label

4	rs	rt	Offset
6	5	5	16

- j label

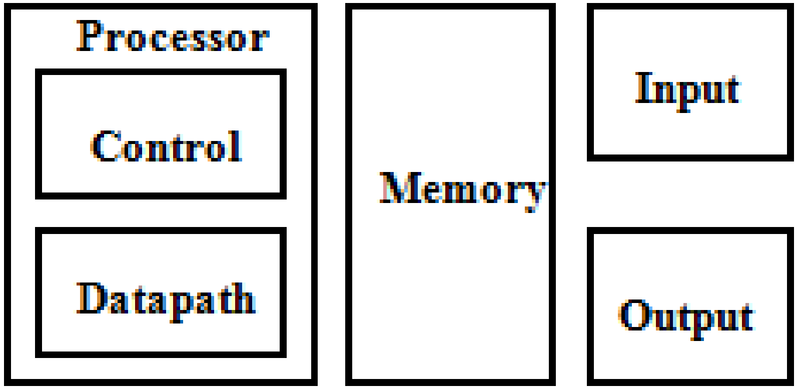
2	target
6	26

BITS Pilani
Pilani Campus

Data Path Design

Introduction

- The Five Classic Components of a Computer
  - Datapath: The component of the processor that performs data processing operations
  - Control: The component of the processor that commands the datapath, memory and I/O devices according to the instructions of the memory
  - Memory
  - Input device
  - Output device

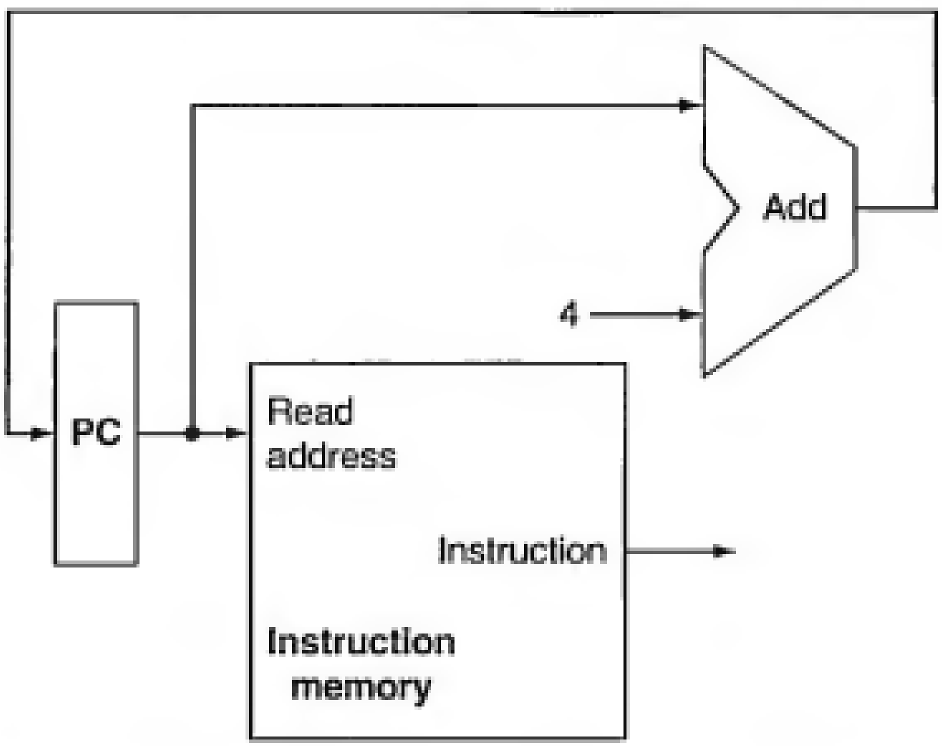


Building a datapath

- Datapath elements :memory – instruction/data, register file, ALU, Adders, Multiplexers….
- Demonstrate Datapath implementation for the following instruction:
  - lw and sw
  - add, sub, and, or and slt
  - beq and j
- Instruction Cycle = Fetch + Execute

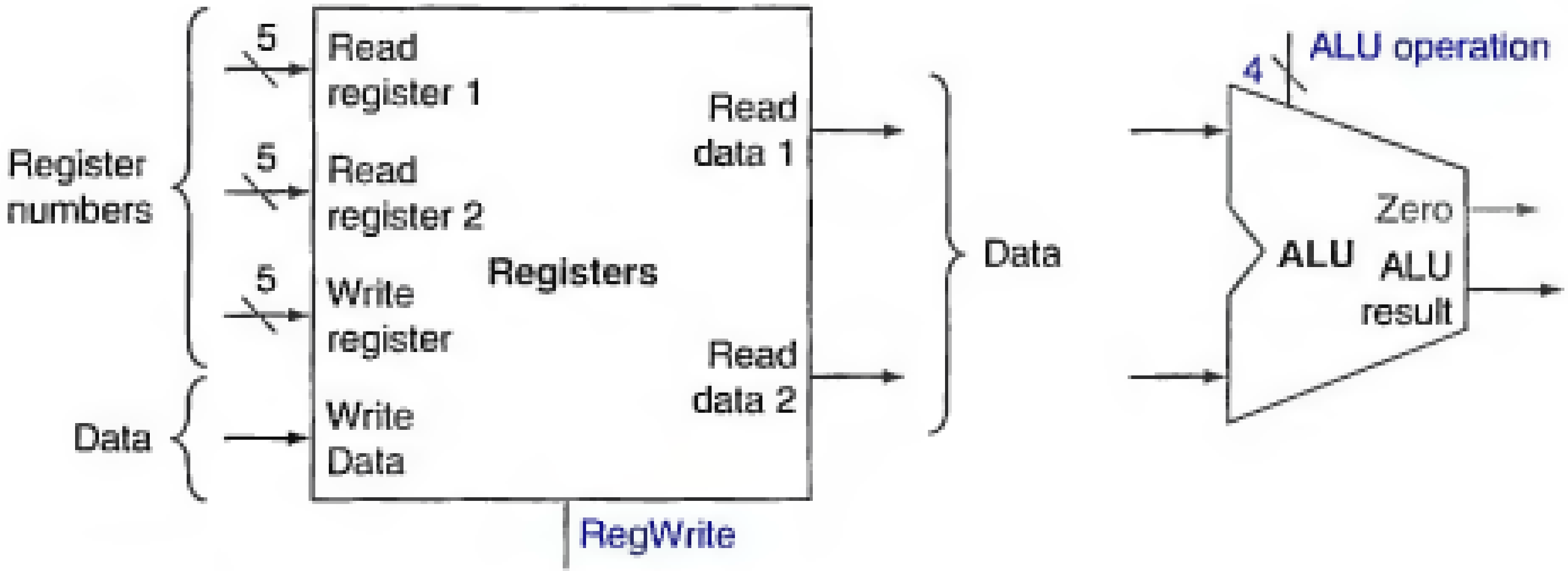
Do this…

write a datapath used for fetching instructions and incrementing the program counter



Data Path for R-type instruction

[Instruction Summary]



a. Registers

b. ALU

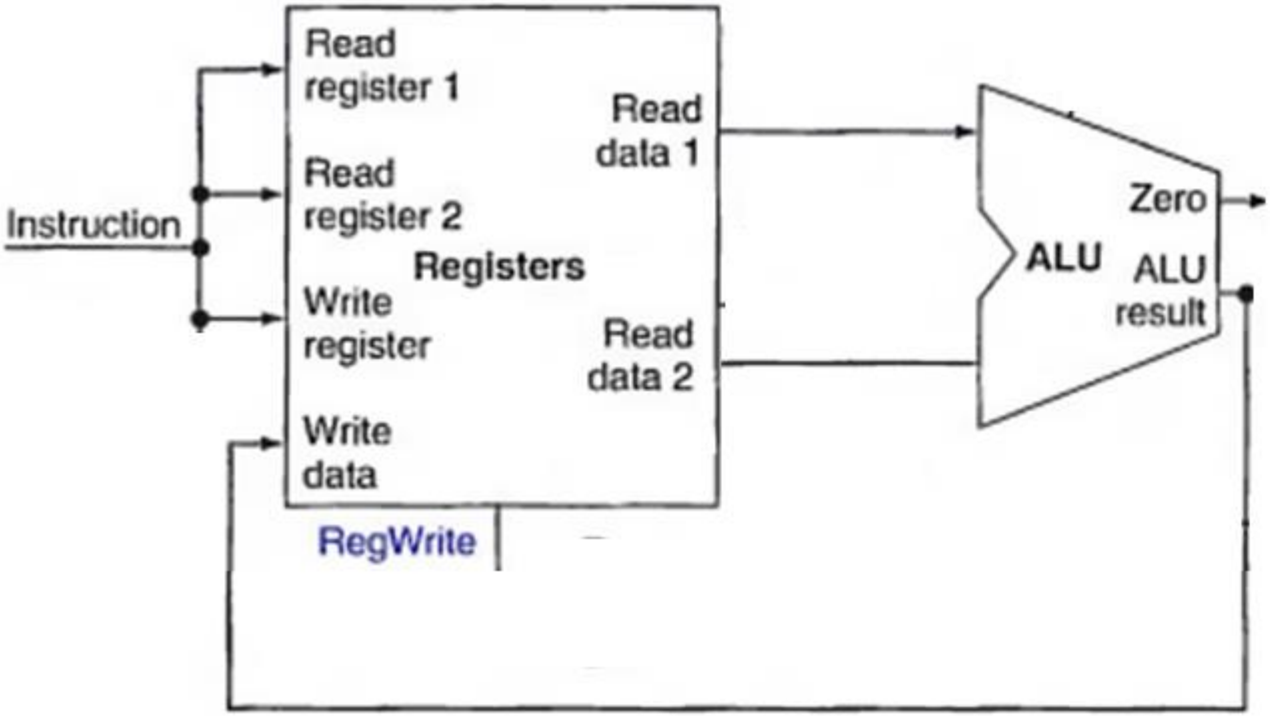
Examples:
add, sub, and, or, slt → rs, rt, rd

add $s1, $s2, $s3

0	rs	rt	rd	0	0x20
6	5	5	5	5	6

Data Path for R-type instruction…

Instruction Summary



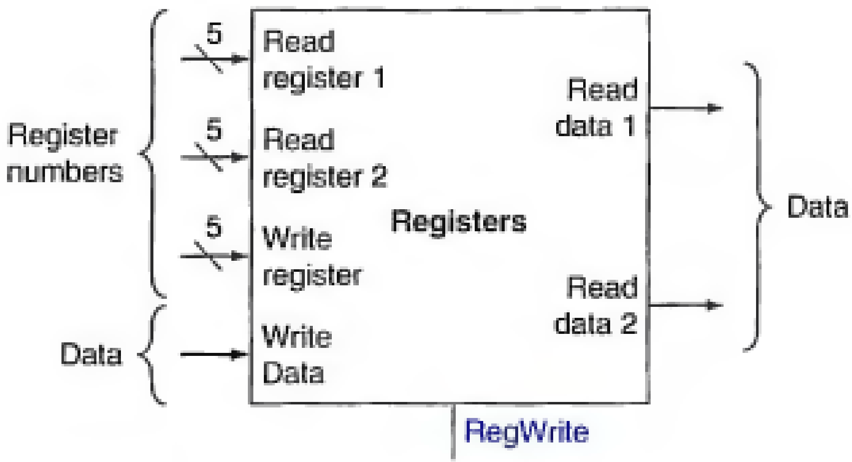
•add $s1, $s2, $s3

The datapath for the memory instructions and the R-type instructions.

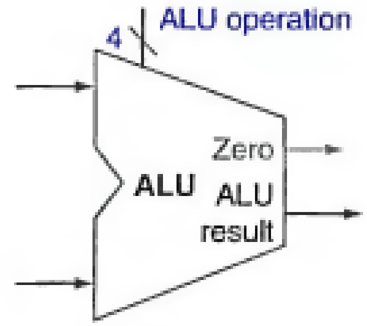
31–26	25–21	20–16	15–11	10–6	5–0
op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Data Path for lw and sw instructions

Instruction Summary



a. Registers



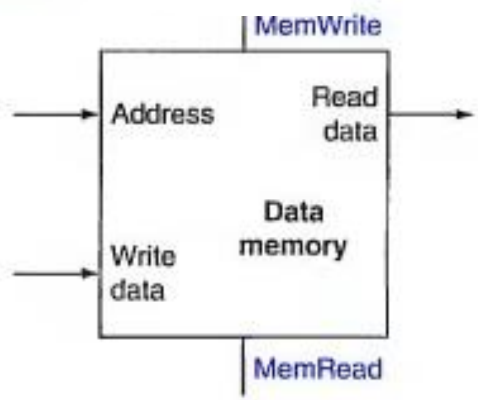
b. ALU

lw $s1, 100($s2)

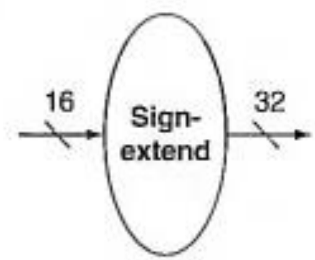
0x23	rs	rt	Offset
6	5	5	16

sw $s1, 100($s2)

0x2b	rs	rt	Offset
6	5	5	16



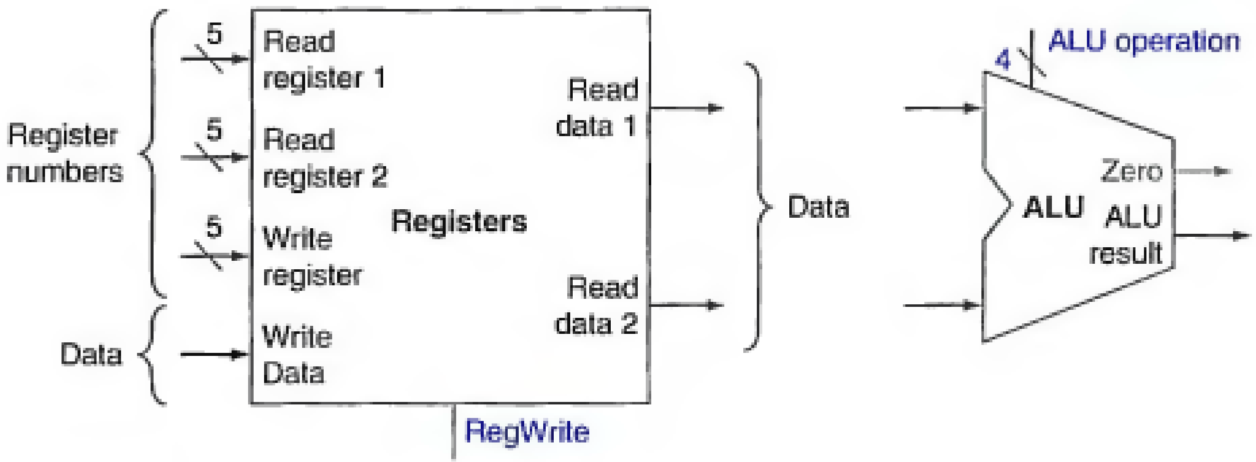
a. Data memory unit



b. Sign extension unit

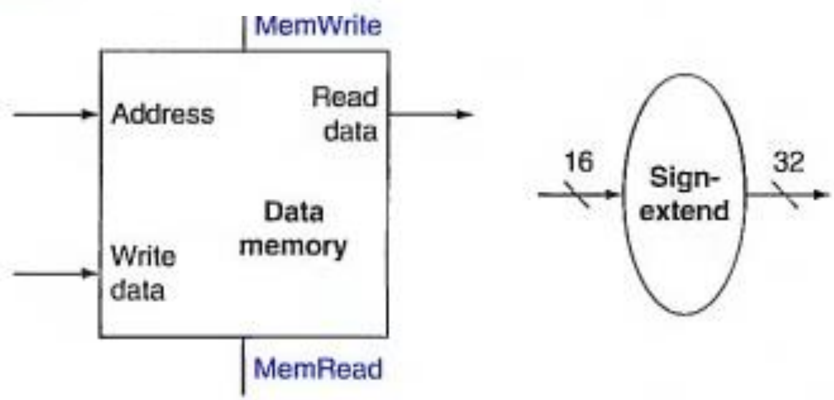
Data Path for lw and sw instructions

Instruction Summary



a. Registers

b. ALU



a. Data memory unit

b. Sign extension unit

lw $s1, 100($s2)

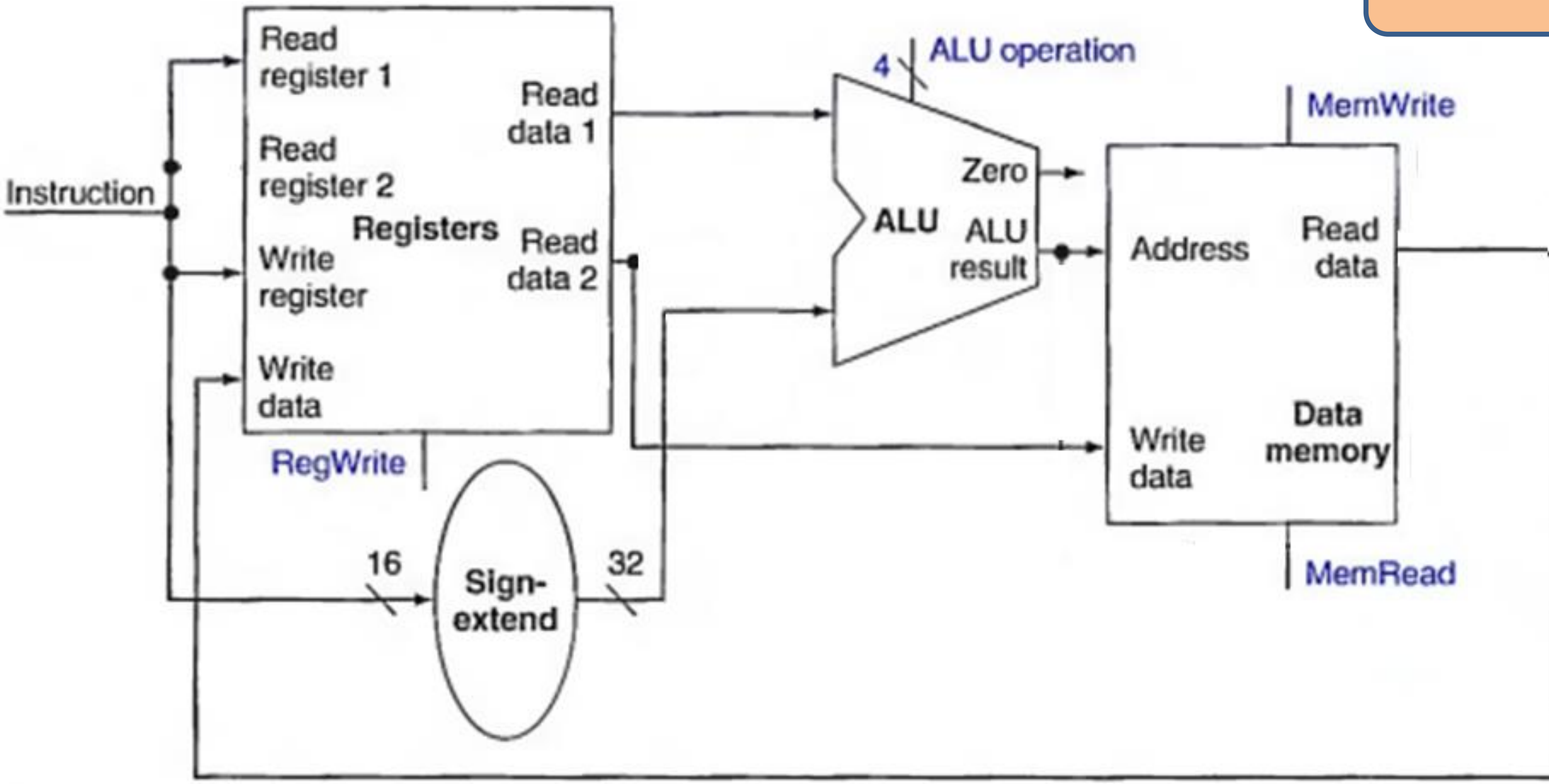
0x23	rs	rt	Offset
6	5	5	16

sw $s1, 100($s2)

0x2b	rs	rt	Offset
6	5	5	16

Data Path for lw instructions

Previous

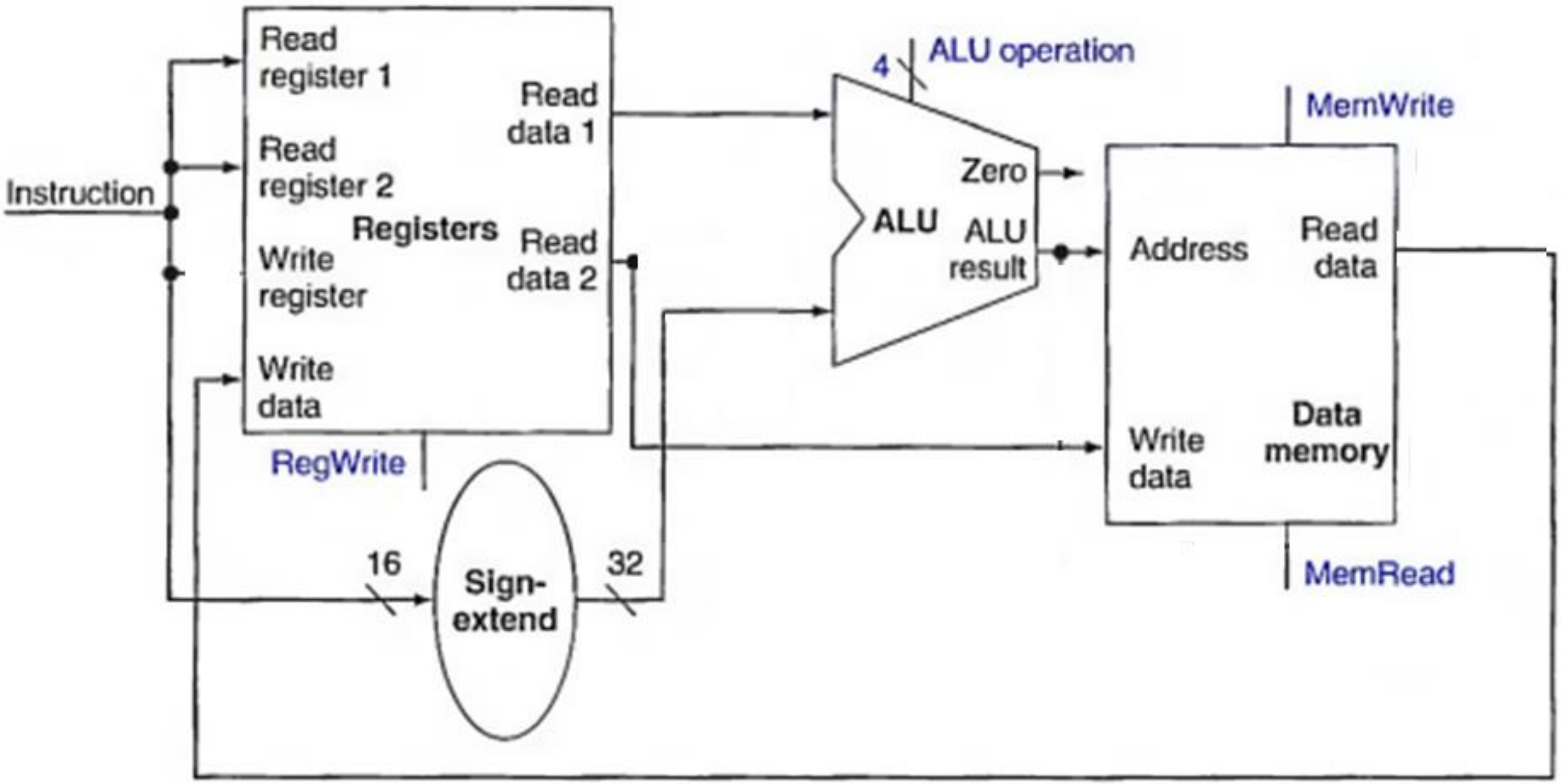


lw $s1, 100($s2)

s2 s1

0x23	rs	rt	Offset
6	5	5	16

Data Path for sw instructions

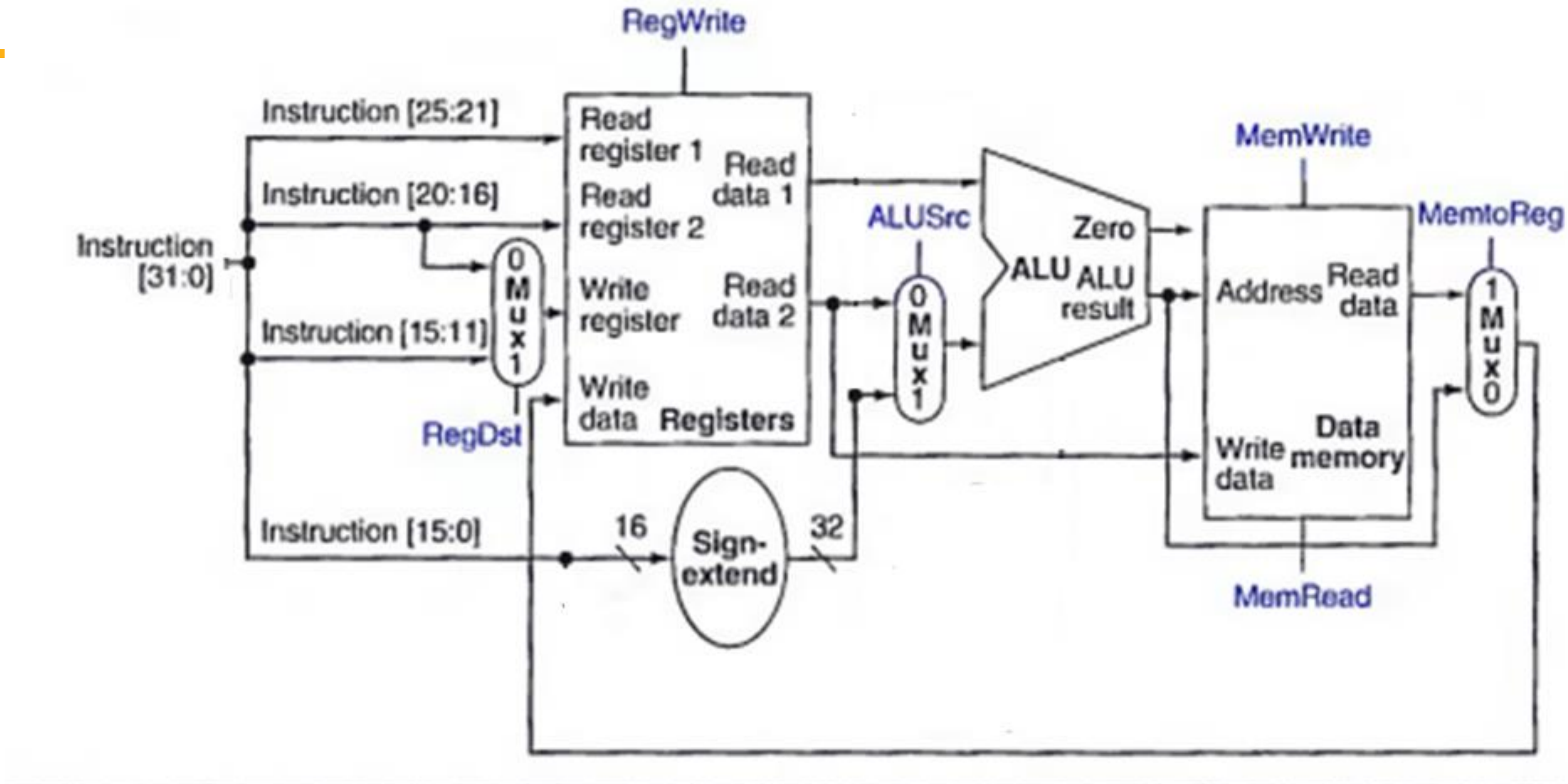


sw $s1, 100($s2)

s2 s1

0x2b	rs	rt	Offset
6	5	5	16

Combined Datapath for R-Type and I-Type instructions



The datapath with all necessary multiplexors and all control lines identified.

R-Type

0	rs	rt	rd	0	0x20
6	5	5	5	5	6

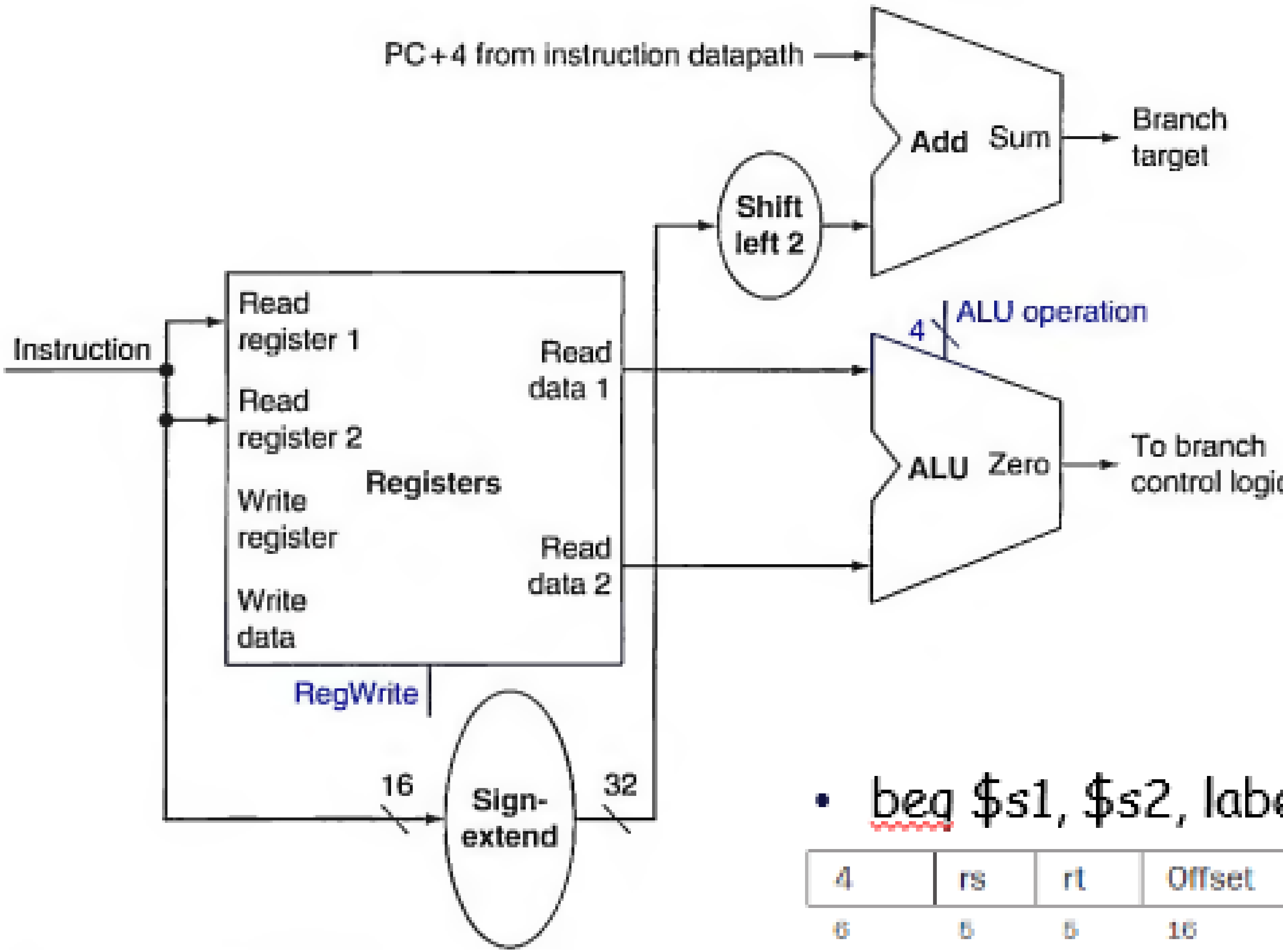
I-Type

0x2b	rs	rt	Offset
6	5	5	16

beq instruction

- beq $s1, $s2, offset
- Two important points to be noted
  - The instruction set architecture defines that the base for the branch address calculation is the address of the instruction following the branch
  - the architecture also states that the offset field is shifted left 2 bits
- branch taken or branch not taken

Datapath for beq instruction



- beq $s1, $s2, label

4	rs	rt	Offset
6	5	5	16

Jump instruction

j <26 bit jump offset>

the jump instruction operates by replacing the lower 28 bits of the PC with the lower 26 bits of the instruction shifted left by 2 bits

- j label

2	target
6	26

Creating a single cycle datapath

Simplest datapath executes all instructions in one clock cycle

- No element in the datapath can be used more than once
- Any element needed more than once must be duplicated

Control unit design

Previous

The ALU control

- used by load and store instructions
- used by r-type instruction : AND, OR, add, sub and slt
  - opcode : zero
  - operation performed based on "function field"

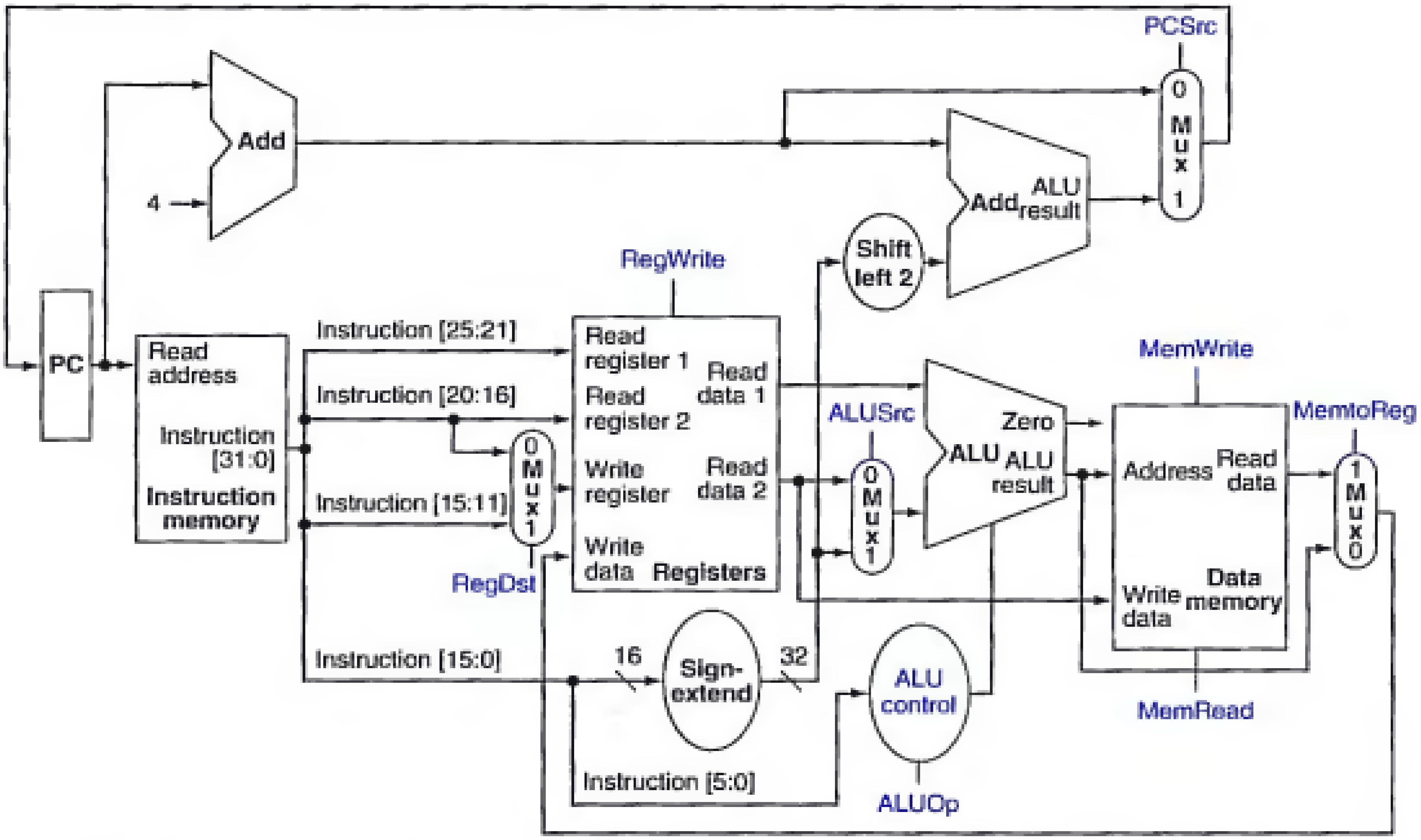
ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR

31–26	26–21	21–16	16–11	11–6	6–0
op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Contd…

- ALU control unit generates 4 bit ALUOperation control signal based on
  - function field
  - 2 bit control field → ALUOp
    - 00 → lw and sw
    - 01→ beq (sub)
    - 10→ add, subtract, AND, OR and slt
- Main Control unit generates 7 control signals : RegDst, RegWrite, ALUSrc, PCSrc, MemRead, MemWrite and MemtoReg

Datapath



The datapath with all necessary multiplexors and all control lines identified.

Effects of 7 control signals

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.

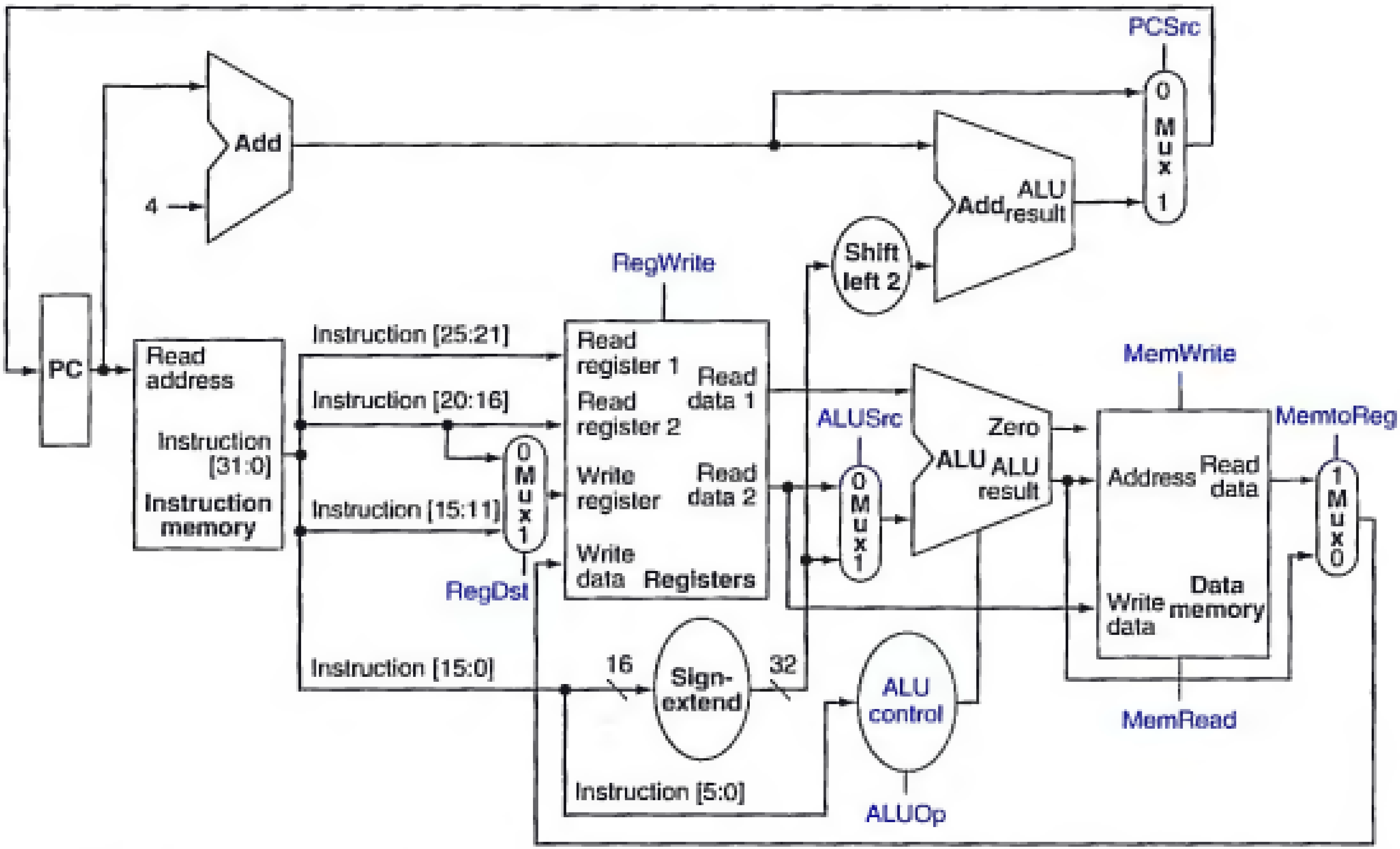
Major observations

- opcode is always contained in bits 31:26
- The two registers to be read are always specified by the rs and rt fields at positions 25:21 and 20:16
  - r-type
  - branch equal
  - store
- The base register for load and store instruction is always in bit positions 25:21 i.e. rs
- The 16 bit offset for branch equal, load and store is always in positions 15:0
- The destination register is in one of two places.
  - load instruction →rt 20:16
  - r type instruction →rd 15:11

Execution steps: add $s1, $s2, $s3

1. The instruction is fetched from _________, and the PC is incremented to ____________
2. Two registers ________ and _________ are read from the register file
3. The ALU operates on the data read from the register file, based on the _________code
4. The result from the ALU is written into the __________register in register file

Name	Format	Example						Comments
add	R	0	18	19	17	0	32	add $s1, $s2, $s3



The datapath with all necessary multiplexors and all control lines identified.

General-Purpose Register Usage Convention

>>

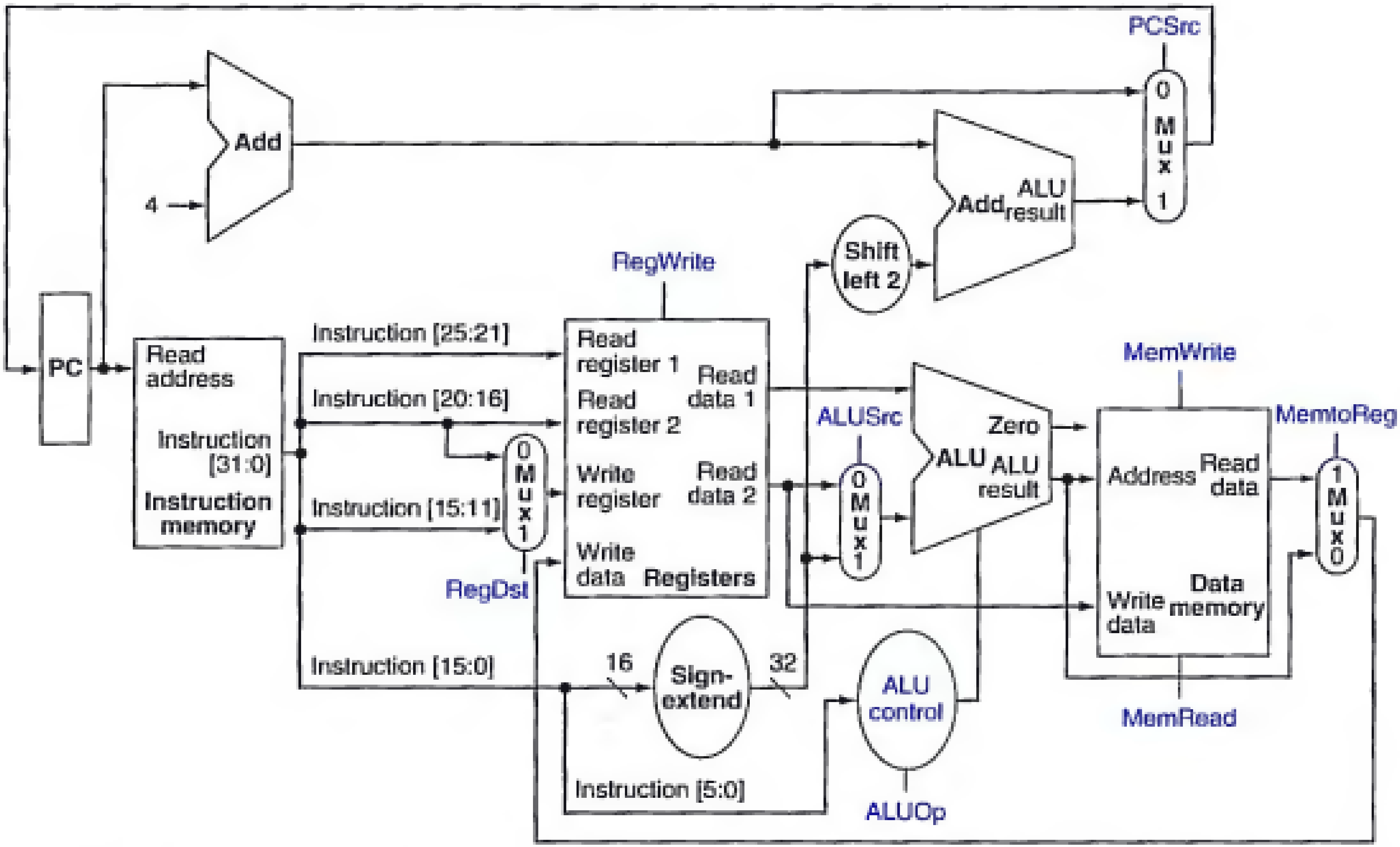
Back

Register name	Number	Intended usage
Zero	0	Constant 0
at	1	Reserved for assembler
v0,v1	2,3	Values for function results and Exp evaluation
a0,a1,a2,a3	4-7	Arguments 1-4
t0-t7	8-15	Temporary (not preserved across call)
s0-s7	16-23	Saved temporary(preserved across call)
t8,t9	24,25	Temporary (not preserved across call)
k0,k1	26,27	Reserved for OS kernel
gp	28	Pointer to Global area
sp	29	Stack Pointer
fp	30	Frame pointer(if needed);Otherwise, a saved register $s8
ra	31	Return address(used by a procedure call)

Execution steps: lw $s1, 100($s2)

1. The instruction is fetched, and the PC is incremented
2. A register ___________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
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 ______ in register file

Name	Format	Example				Comments
lw	I	35	18	17	100	lw $s1, 100($s2)



The datapath with all necessary multiplexors and all control lines identified.

Complete Data Path