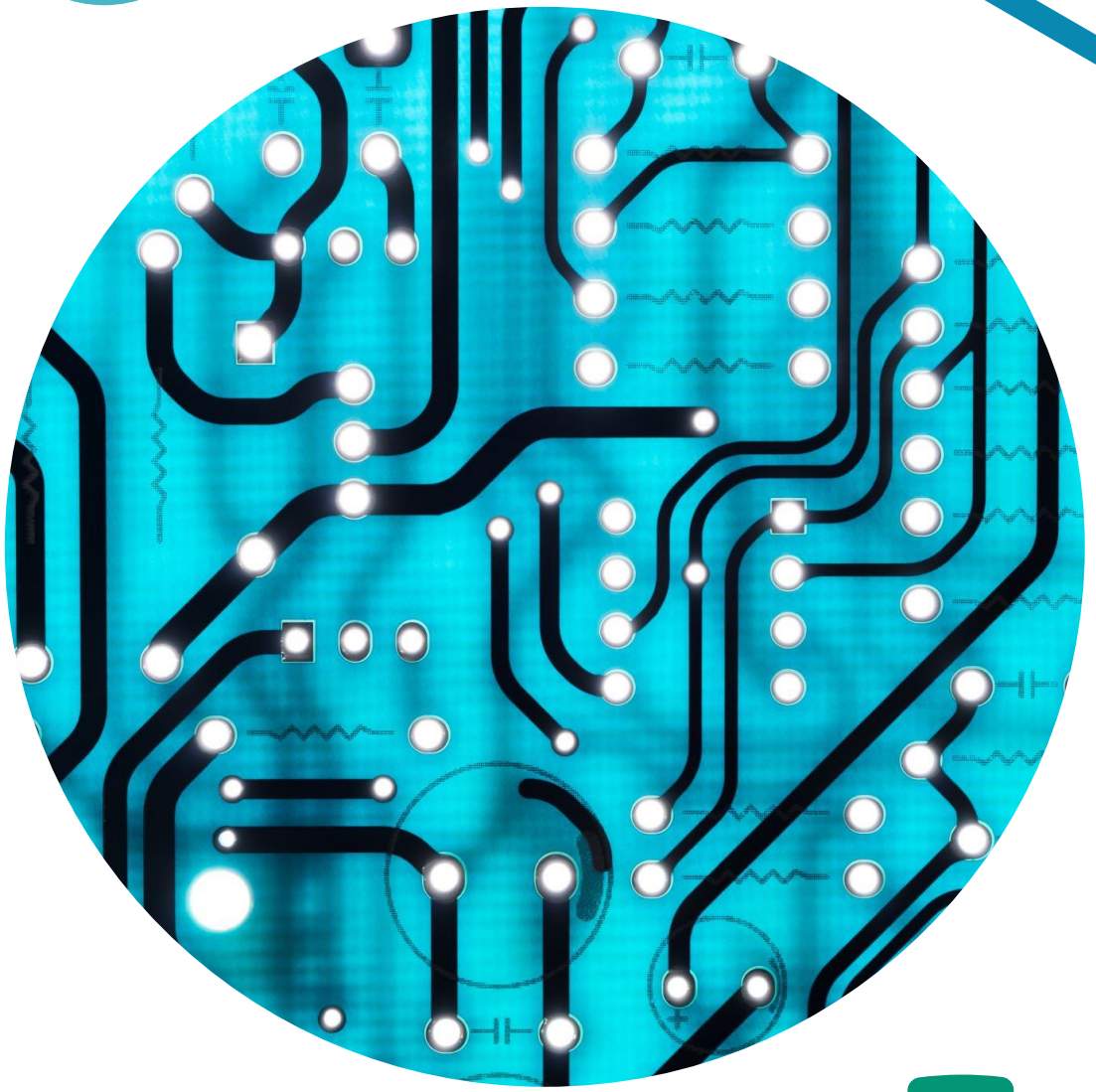


Building a Datapath

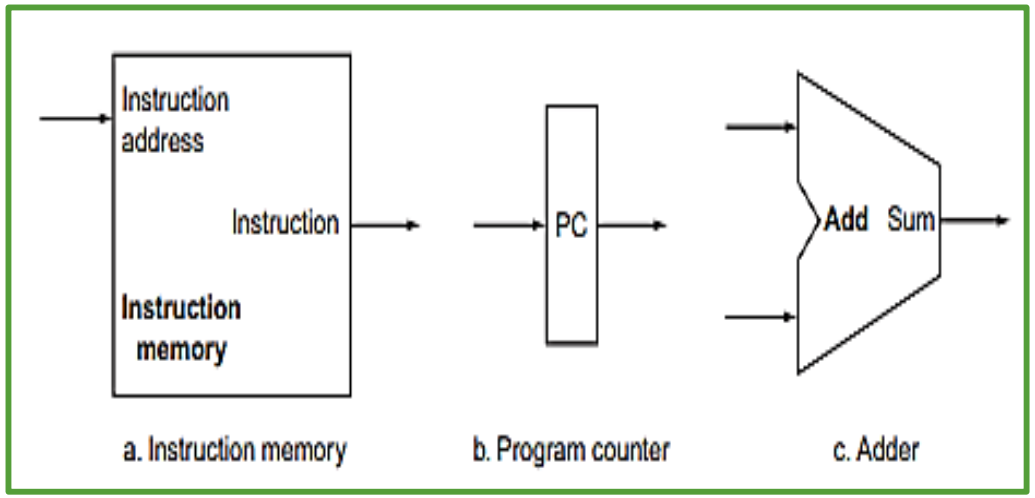
Module 4 | Deepa Mathews

Datapath Elements

A Datapath element is a unit used to operate on or hold data within a processor.

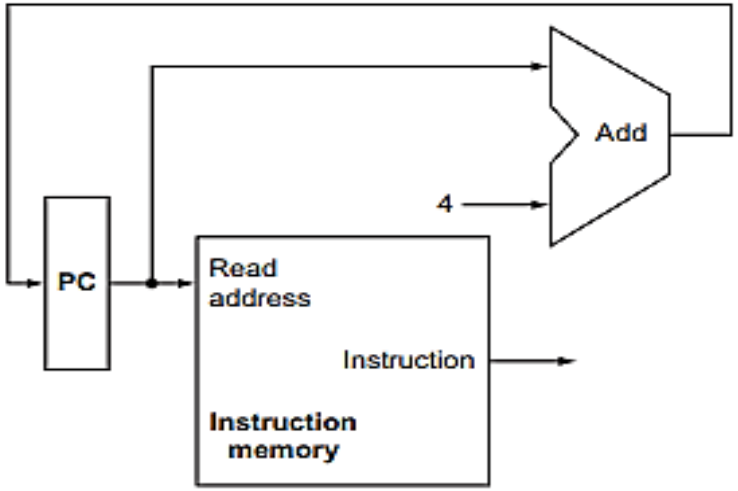
The datapath elements that each instruction required are:

- **Memory Unit :** store instructions of a program and supply them from the given address.
- **Program Counter (PC)** is a register that holds the address of the instruction.
- **Register File** holds the processor's 32 GPRs
- **Adder** is required to increment the PC to the address of the next instruction.
- **ALU unit** to perform ALU operations



Datapath used for FETCHING instructions and incrementing PC

- To execute any instruction, start by fetching the instruction from memory. The elements required are PC, Instruction Memory and an Adder.
- PC should be incremented so that it points at the next instruction, 4 bytes later.

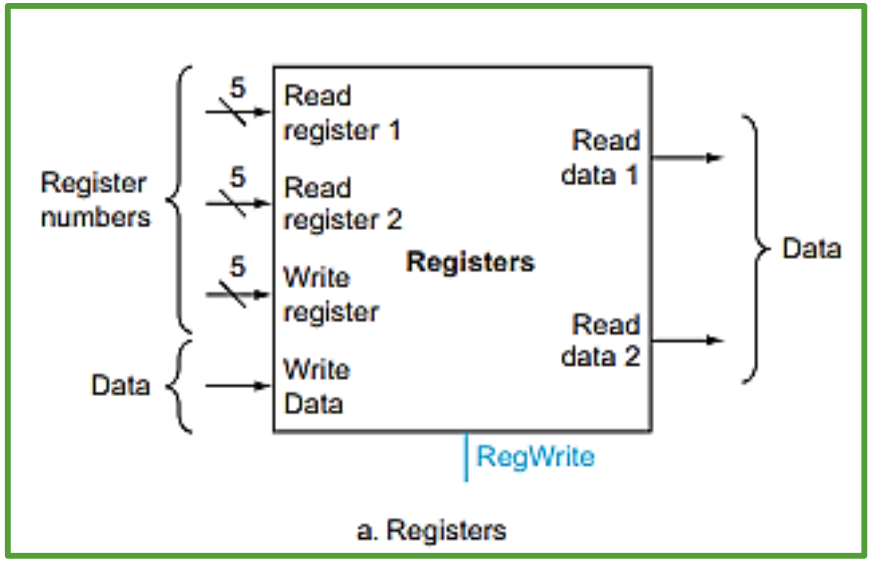


A portion of the datapath used for fetching instructions and incrementing the program counter. The fetched instruction is used by other parts of the datapath.

Datapath used for read/write Register file

Elements required are Register File, ALU

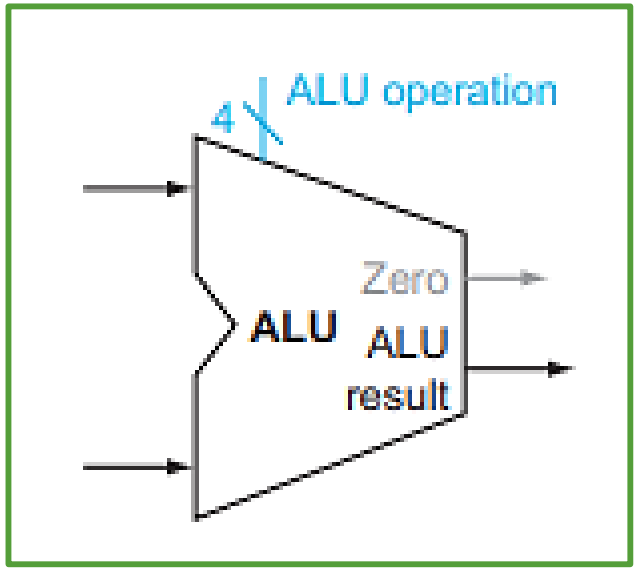
- R-format instructions have 3 register operands- 2 source & 1 destination register.
- For each data word **to be read** from registers, we need **an input to register file** that specifies register number to be read and **an output from the register file** that will carry the value that has been read from the registers.
- **To write a data word**, **2 inputs**: one to specify **register number** to be written and one to supply the **data to be written** into register. Writes are controlled by **RegWrite** control signal, which must be asserted for a write to occur at clock edge.



DATAPATH USED FOR READ/WRITE REGISTER FILE

Elements required are Register File, ALU

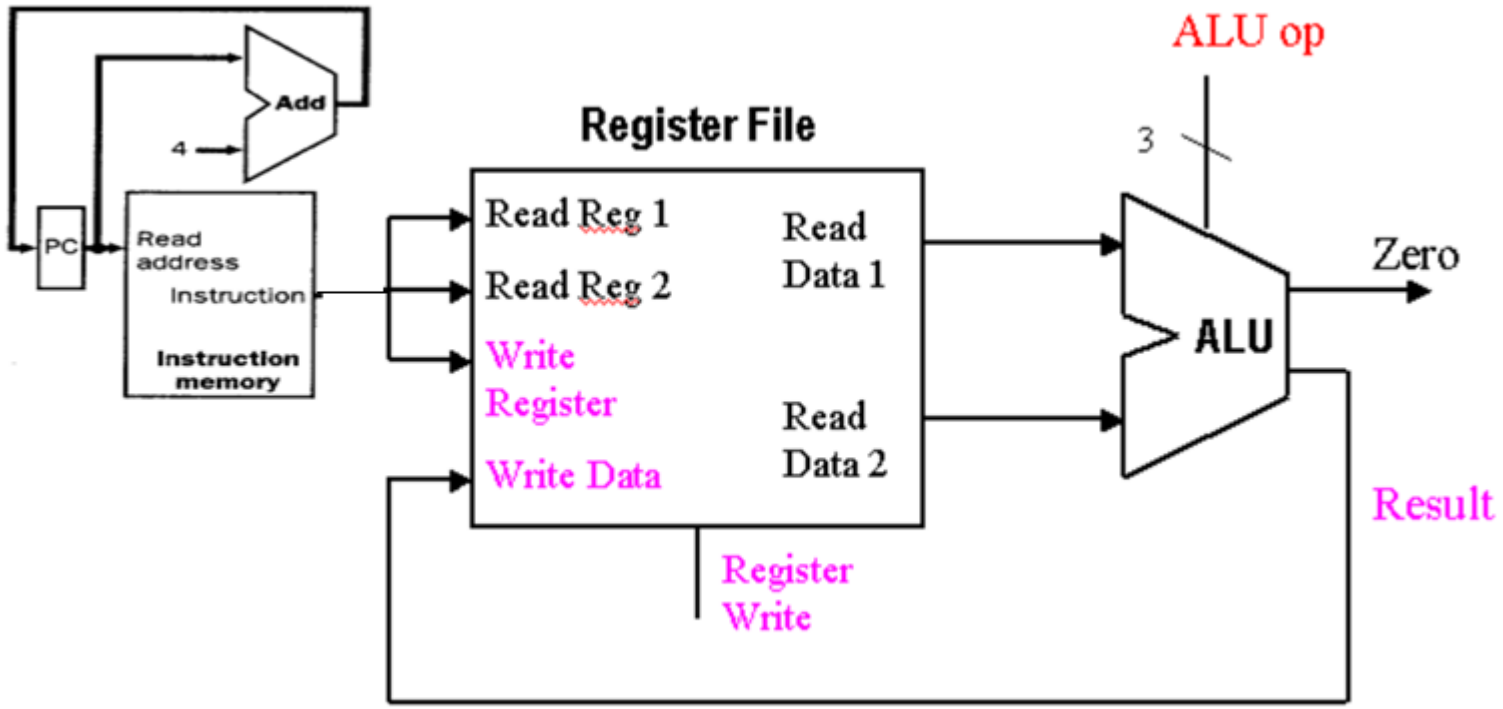
- The operation to be performed by the ALU is controlled with the **ALU operation signal,** which will be of 4 bits wide.
- The ALU, which takes two 32-bit inputs and produces a 32-bit result, as well as a 1-bit signal if the result is 0.
- The **Zero detection output** of the ALU is used to implement branches.



DATAPATH USED FOR READ/WRITE REGISTER FILE

Elements required are Register File, ALU

Format : add $s1,$s2,S3

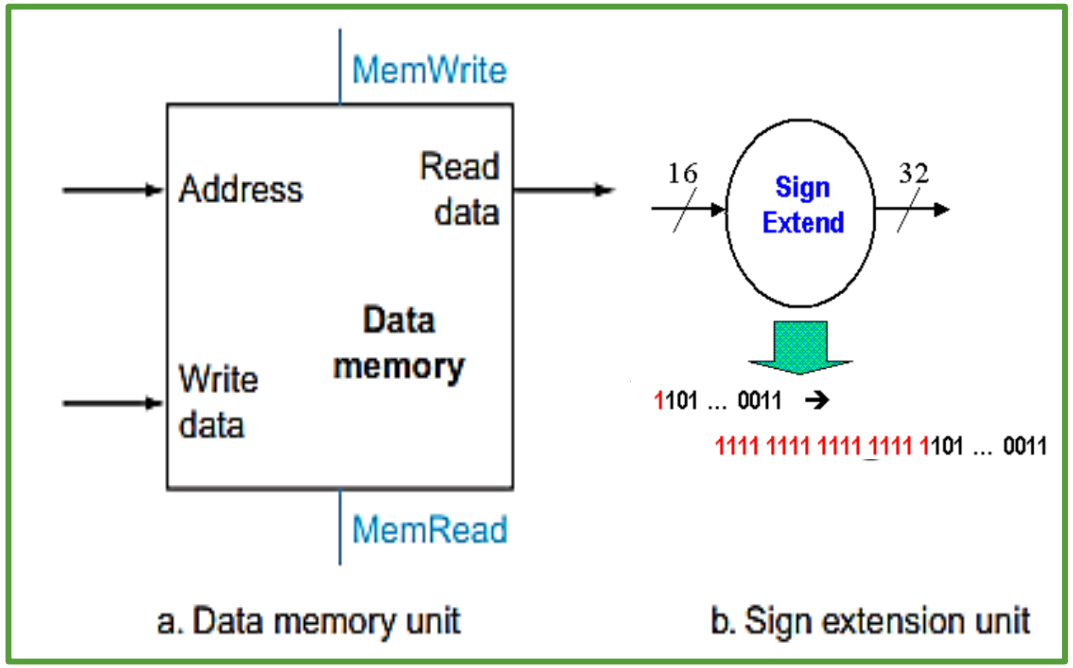


Datapath used for load/store Instructions

Elements required are Register File, ALU, Sign-extend unit, Data Memory Unit

The **load word and store word** instructions compute a memory address **by adding the base register, to the 16-bit signed offset field** contained in the instruction.

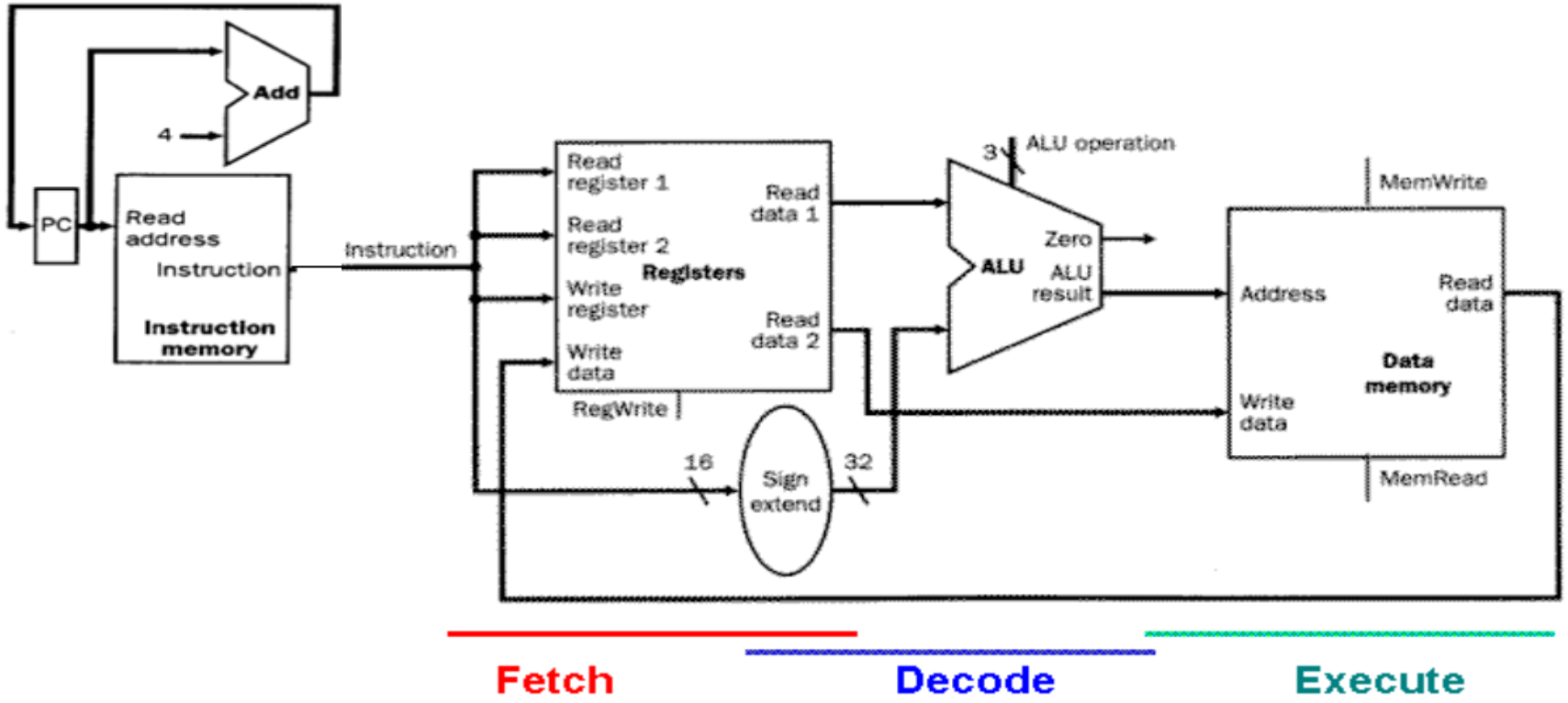
- Thus, we need both the **register file** and the **ALU**.
- A unit to **sign-extend** the 16-bit offset field in the instruction to a 32-bit signed value is required, and
- A **data memory unit** to read from or write to. This data memory has both read and write control signals, an address input, and an input for the data to be written into memory.



Datapath used for load/store Instructions

Elements required are Register File, ALU, Sign-extend unit, Data Memory Unit

- lw $s0,12($s1)
- sw $s0,16($s1)



Datapath used for Branch Instructions

Elements required are Register File, ALU, Sign-extend unit, Adder

The **beq instruction** has three operands, **two registers** that are compared for equality, **and a 16-bit offset** used to compute the branch target address relative to the branch instruction address. Its form is **beq $t1,$t2,offset.**

To implement this instruction,

- first need to **compute the branch target address** by adding the sign-extended offset field of the instruction to the PC; so a **Sign-extend Unit and an adder circuit** is required
- Also we need the **register file,** and the **ALU unit** for doing the comparison. The two registers are compared to determine whether the next instruction is the instruction that follows sequentially or the instruction is at the branch target address. When the condition is true, branch is taken and the branch target address becomes the new PC; else branch is not taken and so the PC is updated with PC+4.

Datapath used for Branch Instructions

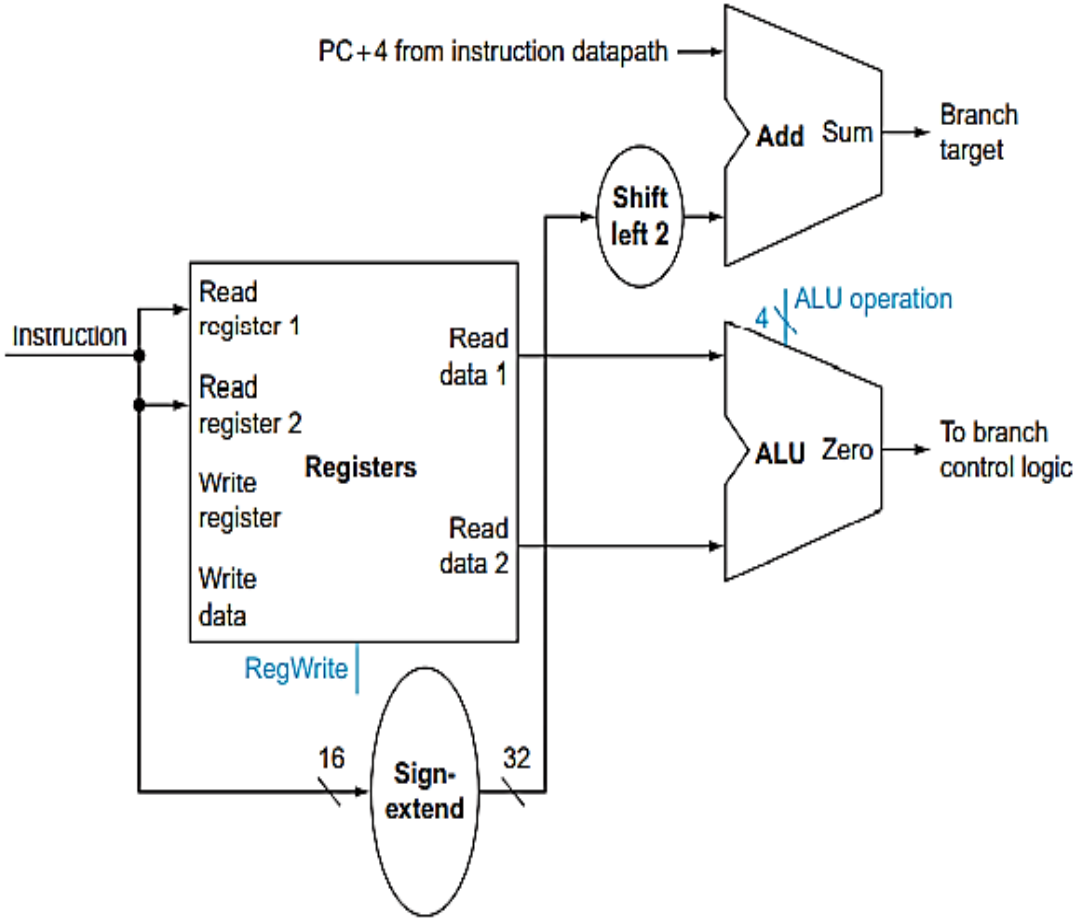
Elements required are Register File, ALU, Sign-extend unit, Adder

To **compute branch target address**,

PC = PC+4+4*offset

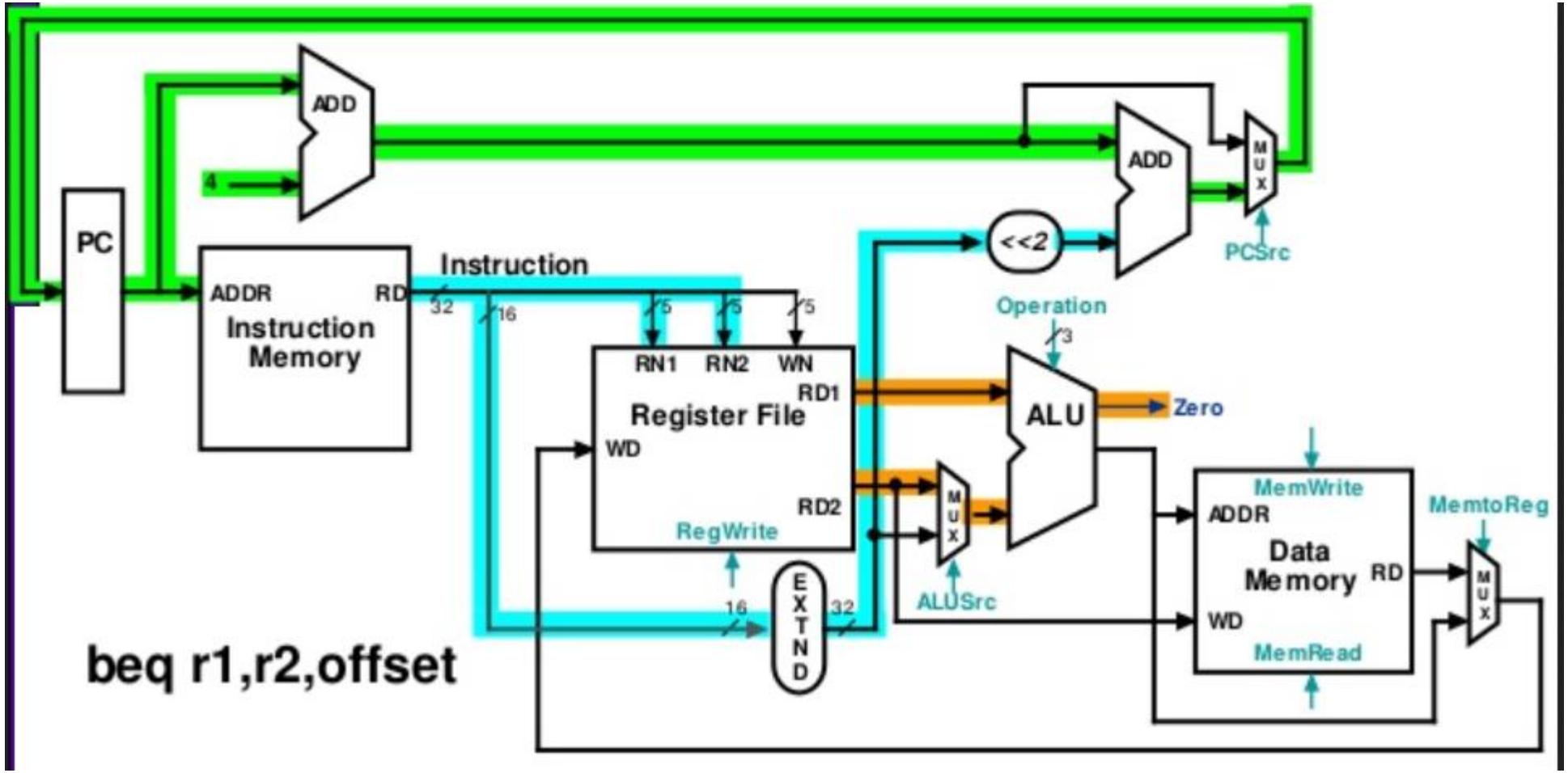
- To **perform the compare**, register file is used to supply the 2 operands. The 2 register operands are send to the ALU with the control set to do a subtract. If the Zero signal out of the ALU unit is asserted, then the two values are equal.
- By *taking the branch*, the ISA specification means that the ALU adds a sign-extended offset to the program counter (PC). The offset is shifted left 2 bits to allow for word alignment (since $2^2 = 4$, and words are comprised of 4 bytes).

Shifting left by 2 is the same as multiplying by 4



Datapath used for Branch Instructions

Elements required are Register File, ALU, Sign-extend unit, Adder



Creating a Single Datapath

- To complete the implementation of an instruction, the datapath components needed for the individual instruction classes, **are combined into a single datapath by adding control lines.**
- This simplest datapath will attempt to execute all instructions **in one clock cycle**.
- A memory is needed for storing instructions and data separately.
- To share a datapath element between two different instruction classes, multiple connections to the input of an element is needed, **using a multiplexer** and the **control signals are** used to select among the multiple inputs.

Building a Datapath - Question

Show how to **build a datapath** for the operational portion of the memory-reference and arithmetic-logical instructions that uses a single register file and a single ALU to handle both types of instructions, adding any necessary multiplexers.

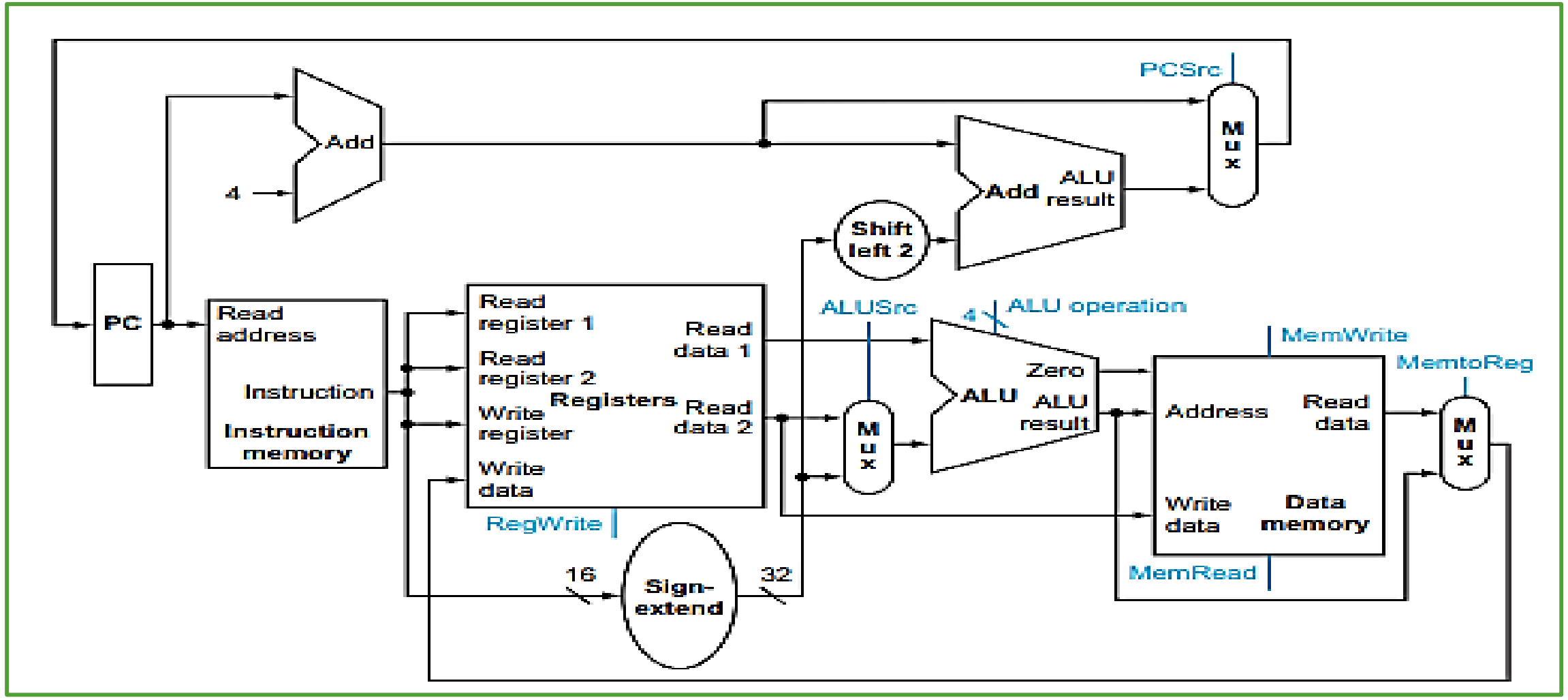
- A single datapath for the core MIPS architecture is build by **adding the datapath for instruction fetch, the datapath from R-type and memory instructions, and the datapath for branches.**

Building a Datapath

- To create a datapath with only a single register file and a single ALU, we must support two different sources for the second ALU input, as well as two different sources for the data stored into the register file. Thus, one multiplexer is placed at the ALU input and another at the data input to the register file.

- The branch instruction uses the ALU for comparison of the register operands, so an adder is used for computing the branch target address. An additional multiplexer is required to select either the sequentially following instruction address (PC + 4) or the branch target address to be written into the PC.

BUILDING A DATAPATH



A Simple Implementation Scheme
Designing ALU Control

- Depending on the instruction class, the ALU performs one of these first **5 functions**. For the R-type instructions, the ALU is used to perform one of 5 actions (AND/OR/add/subtract/set on less than), depending on the value of the 6-bit funct field in low-order bits of instruction. For load word and store word instructions, ALU is used to compute the memory address by addition. For branch equal, the ALU must perform a subtraction.

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

A Simple Implementation Scheme

We can generate the **4-bit ALU control input** using a small control unit that has inputs as the function field of the instruction and a 2-bit control field, ALUOp. ALUOp indicates the operation to be performed is add (00) for loads & stores, subtract (01) for beq, or determined by the operation encoded in the funct field (10). The function field is used only when ALUOp bits is 10.

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

Design Steps

Create a truth table for combinations of function code field and ALUOp bits. The don't-care term indicates that output does not depend on value of the input corresponding to that column.

ALUOp		Funct field						Operation
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	
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

When the ALUOp bits are 00, the ALU control bits are set to **0010**, independent of the function code.

Once truth table has been constructed, **it can be optimized and then turned into gates.**

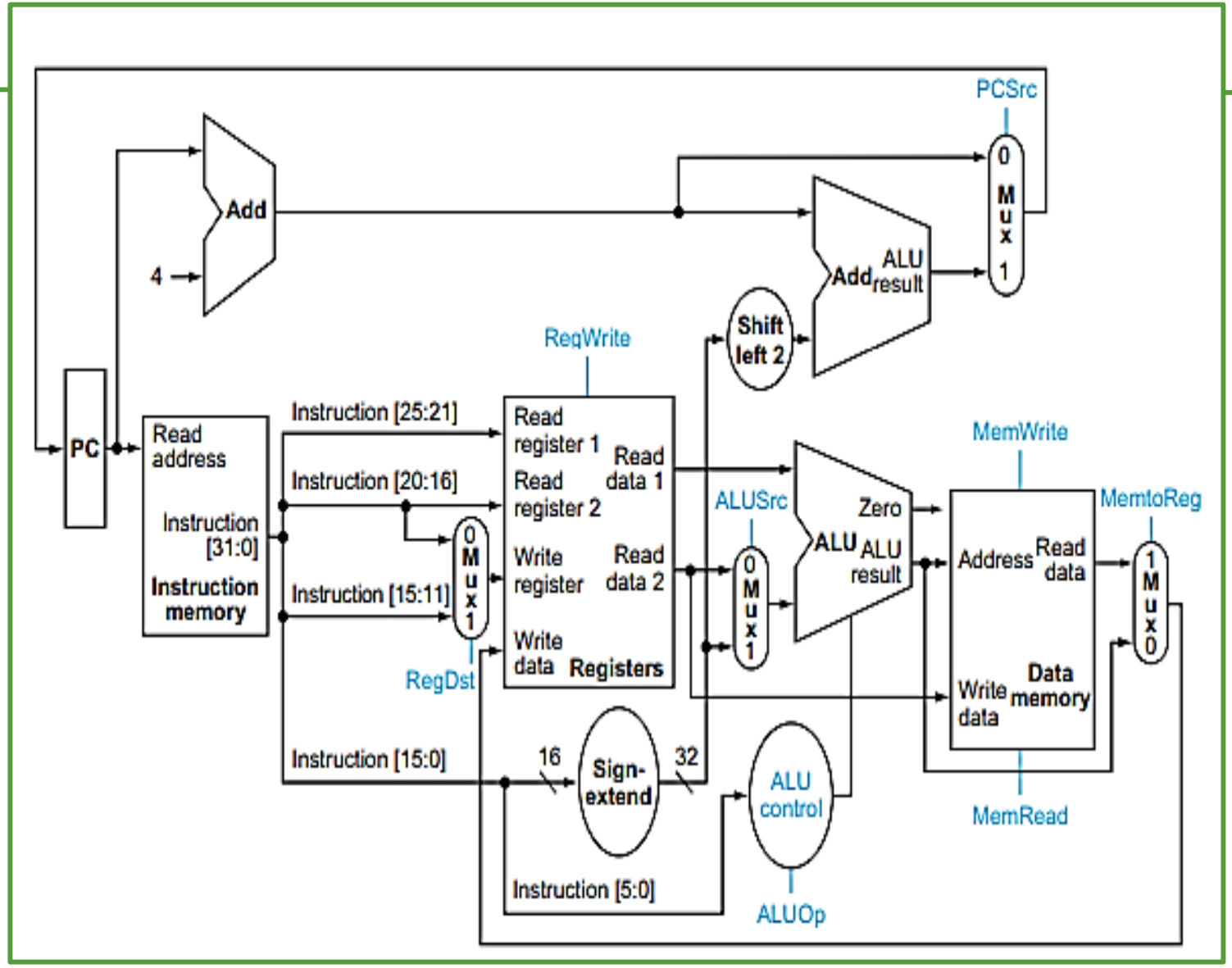
Designing the Main Control Unit

- The **ALU unit** is designed using function code and a 2-bit signal as its control inputs.
- A **control block** is required to generate appropriate control signals to identify the fields of an instruction and the control lines that are needed for the datapath.
- **Add the instruction labels** and **extra multiplexers** to the simple datapath.
- **Add the write signals** for state elements, **the read signal** for the data memory, and **the control signals** for the multiplexers. Since all the multiplexers have two inputs, they each require a single control line.

Designing the Main Control Unit

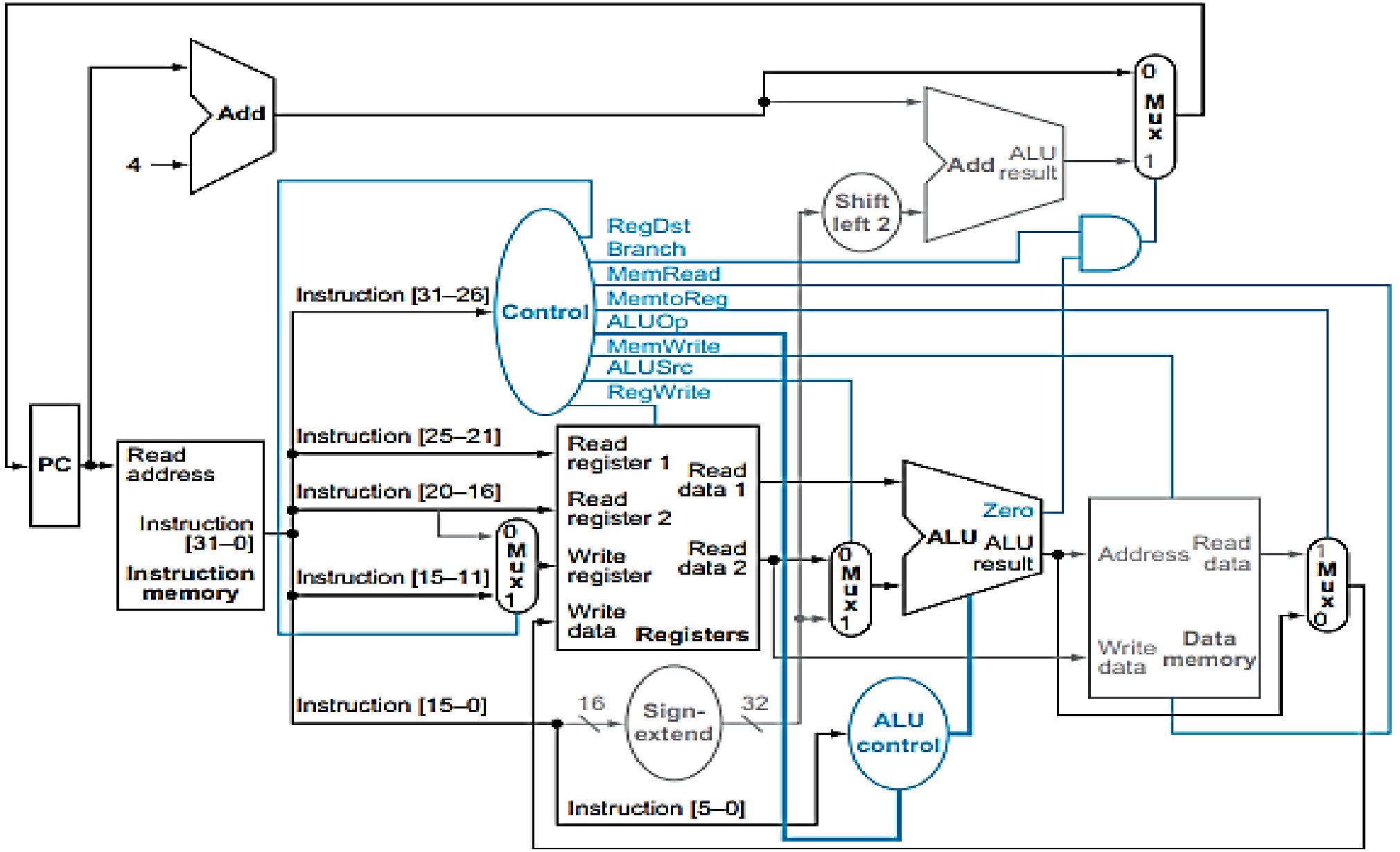
7 single-bit control lines plus the 2-bit ALUOp control signal are added.

When 1-bit control to 2 way multiplexer is asserted, the multiplexer selects input corresponding to 1. Else, if the control is deasserted, the multiplexer selects the 0 input.



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.



The datapath in operation for an R-type instruction, such as add $t1,$t2,$t3. The control lines, datapath units, and connections that are active are highlighted.