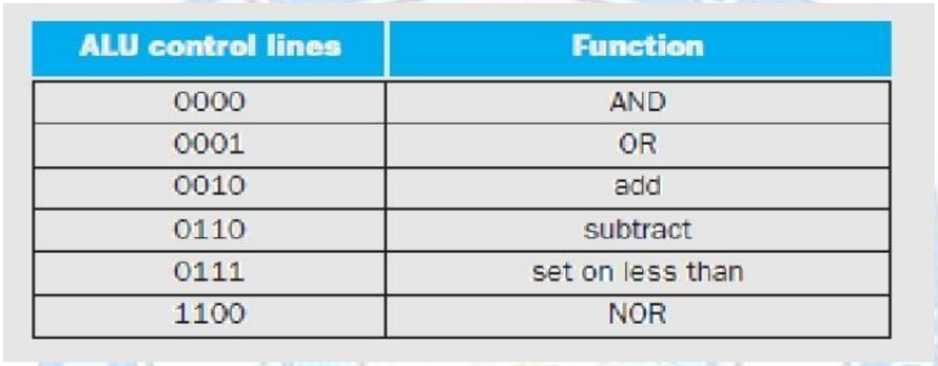
DESIGNING A CONTROL IMPLEMENTATION SCHEME

This simple implementation covers load word (lw), store word (sw), branch equal(beq), and the arithmetic-logical instructions add, sub, AND, OR, and set on less than. **The ALU Control**

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



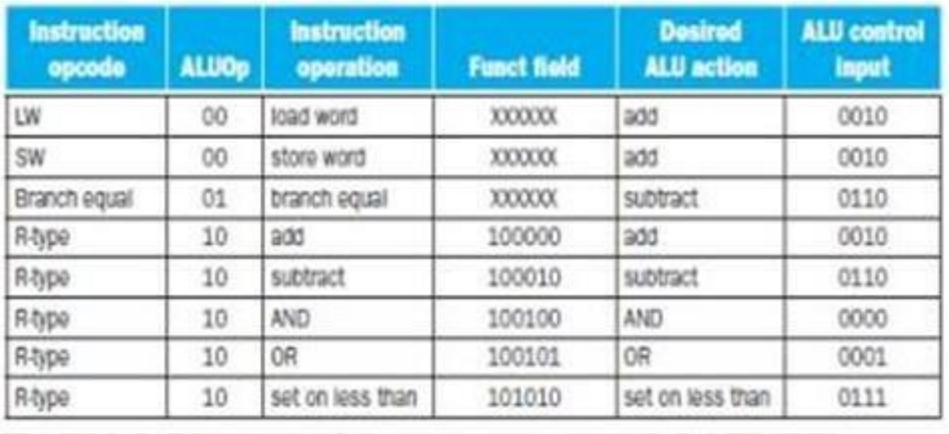
ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR

Table 3.1: ALU control signals

Depending on the instruction class, the ALU will need to perform one of these first five functions. For branch equal, the ALU must perform a subtraction. We can generate the 4-bit ALU control input using a small control unit that has as inputs the function field of the instruction and a 2-bit control field, which we call ALUOp.

ALUOp indicates whether the operation to be performed should be add

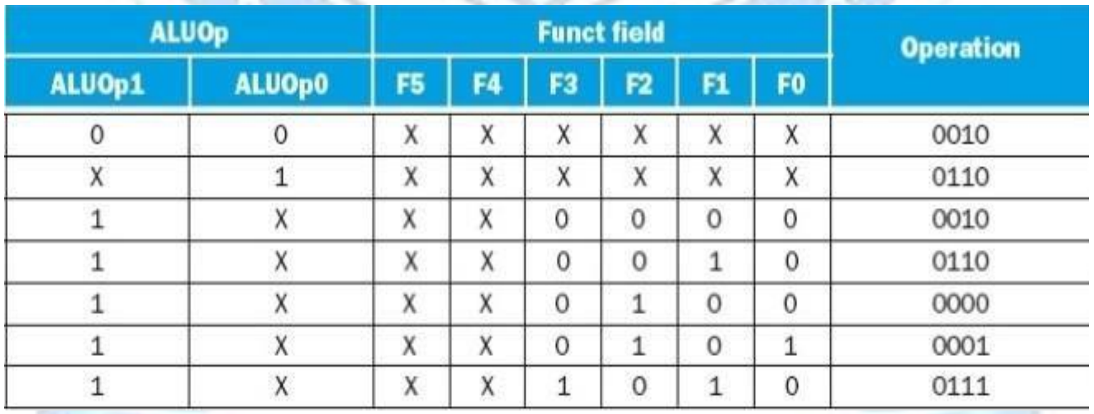
(00) for loads and stores, subtract (01) for beq, or determined by the operation encoded in the funct field (10). The output of the ALU control unit is a 4-bit signal that directly controls the ALU by generating one of the 4-bit combinationsshown previously. In Figure 3.2, we show how to set the ALU control inputs based on the 2-bit ALUOp control and the 6-bit function code. Later in this chapter we will see how the ALUOp bits are generated from the main control unit.



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

The ALU Op control bits and the different function codes for the R-typeinstruction.

Using multiple levels of control can reduce the size of the main control unit. Using several smaller control units may also potentially increase the speed of the control unit. There are several different ways to implement the mapping from the 2-bit ALU Op field and the 6-bit function field to the four ALU operation control bits. Table 3.3; shows the truth table how the 4-bit ALU control is set depending on these two input fields. Since the full truth table is very large (28 = 256 entries), we show only the truth table entries for which the ALU control must have a specific value

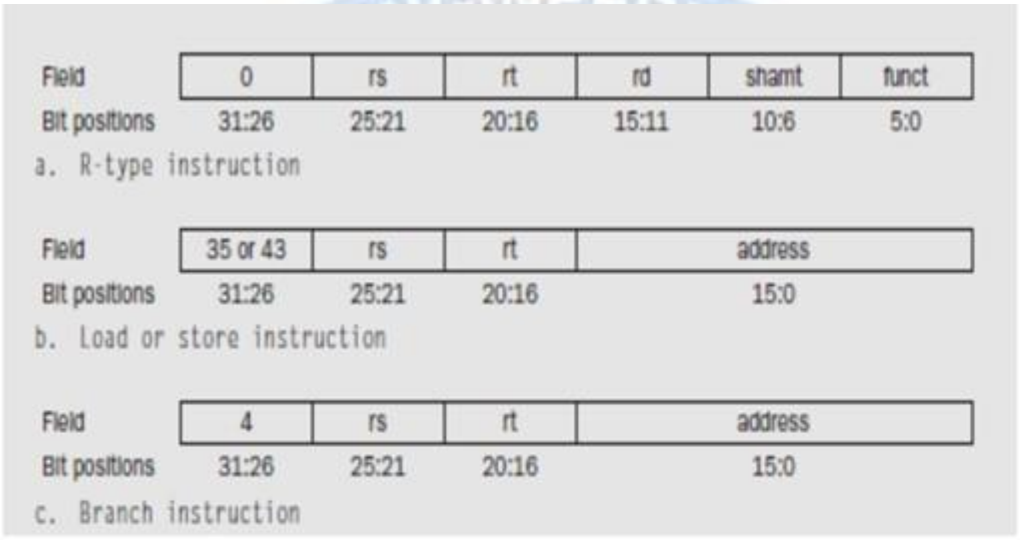


ALUOp		Funct field						Operation
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	
0	0	X	X	X	X	X	X	0010
X	1	X	X	X	X	X	X	0110
1	X	X	X	0	0	0	0	0010
1	X	X	X	0	0	1	0	0110
1	X	X	X	0	1	0	0	0000
1	X	X	X	0	1	0	1	0001
1	X	X	X	1	0	1	0	0111

The truth table for the 4 ALU control bits (called Operation)

Designing the Main Control Unit

To understand how to connect the fields of an instruction to the datapath, it is useful to review the formats of the three instruction classes: the R-type, branch, and load-store instructions. Figure 3.8 shows these formats.



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

The three instruction classes (R-type, load and store, and branch) usetwo different instruction formats.

There are several major observations about this instruction format that we will rely on:

- The op field, is called the opcode, is always contained in bits 31:26. Refer to this field as Op[5:0].
- The two registers to be read are always specified by the rs and rt fields, at positions 25:21 and 20:16. This is true for the R-type instructions, branch equal, and store.
- The base register for load and store instructions is always in bit positions 25:21 (rs).
- The 16-bit off set for branch equal, load, and store is always in positions 15:0.

The destination register is in one of two places. For a load it is in bit positions 20:16 (rt), while for an R-type instruction it is in bit positions 15:11 (rd).

Thus, we will need to add a multiplexor to select which field of the instruction is used to indicate the register number to be written. The first design principle simplicity favors regularity—pays off here in specifying control.

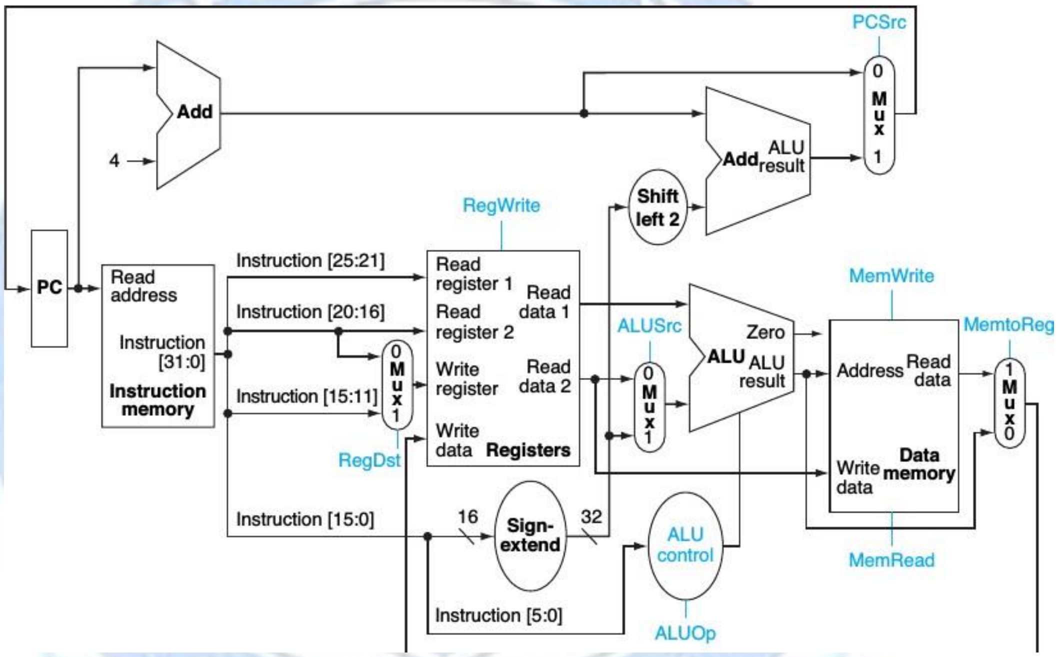
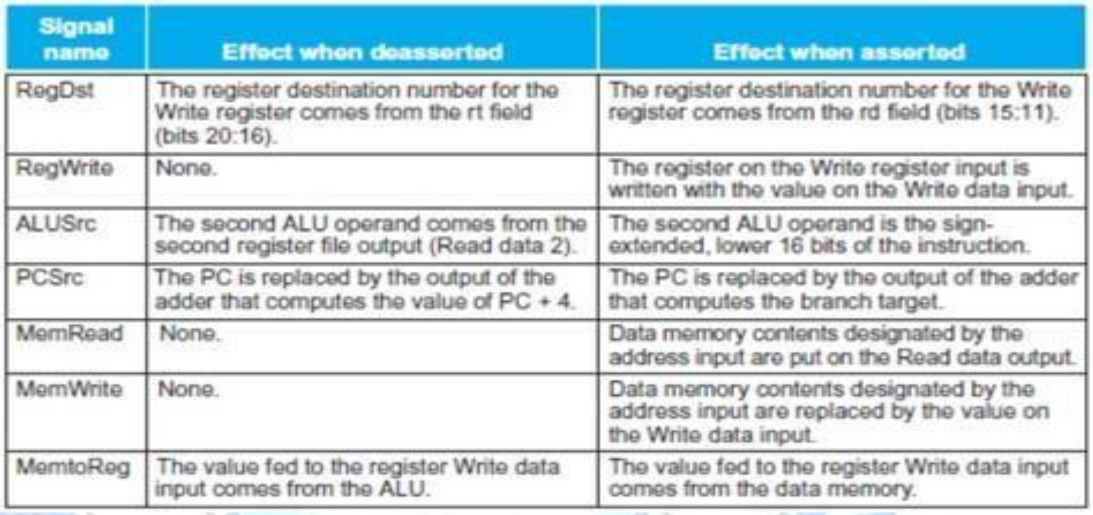


Figure 3.9 shows seven single-bit control lines plus the 2-bitALUOp control signal. We have already defined how the ALUOp control signal works, and it is useful to define what the seven other control signals do informally before we determine how to set these control signals during instruction execution.

Table 3.4 describes the function of these seven control lines.That control line should be asserted (activated or set true) if the instruction is branch on equal (a decision that the control unit can make) and the Zero output of the ALU, which is used for equality comparison, is asserted. To generate the PCSrc signal, we will need to AND together a signal from the control unit, which we call Branch, with the Zero signal out of the ALU.



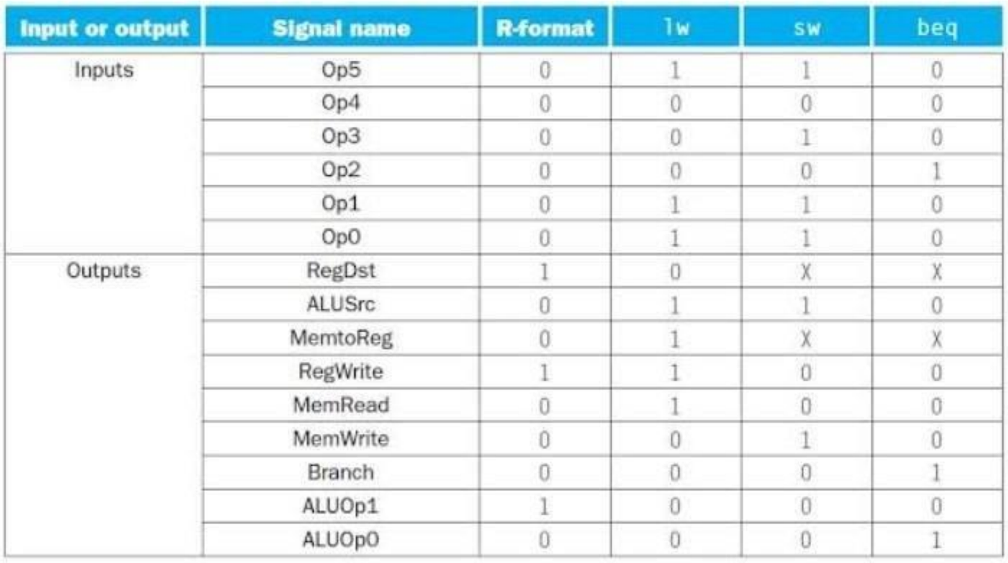
Signal name	Effect when deasserted	Effect when asserted
RegDst	The register destination number for the Write register comes from the rt field (bits 20:16).	The register destination number for the Write register comes from the rd field (bits 15:11).
RegWrite	None.	The register on the Write register input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

Table 3.4: The effect of each of the seven control signals.

These nine control signals (seven from Figure 4.16 and two for ALU Op) can now be set on the basis of six input signals to the control unit, which are the opcode bits 31 to 26. Figure 3.10 shows the data path with the control unit and the control signals.

Finalizing Control

Table 3.6 shows the logic in the control unit as one large truth table that combines all the outputs and that uses the opcode bits as inputs. It completely specifies the control function,

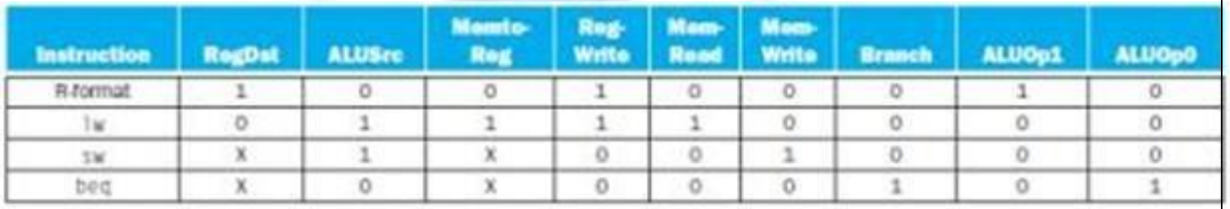


Input or output	Signal name	R-format	lw	sw	beq
Inputs	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Outputs	RegDst	1	0	X	X
	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1

The control function for the simple single-cycle implementation is completely specified by this truth table.

<u>OPERATION OF THE DATAPATH</u>

The flow of three different instruction classes through the data path is shown in table 3.5. The asserted control signals and active data path elements are highlighted in each of these. Note that a multiplexor whose control is 0 has a definite action, even if its control line is not highlighted. Multiple-bit control signals are highlighted if any constituent signal is asserted.

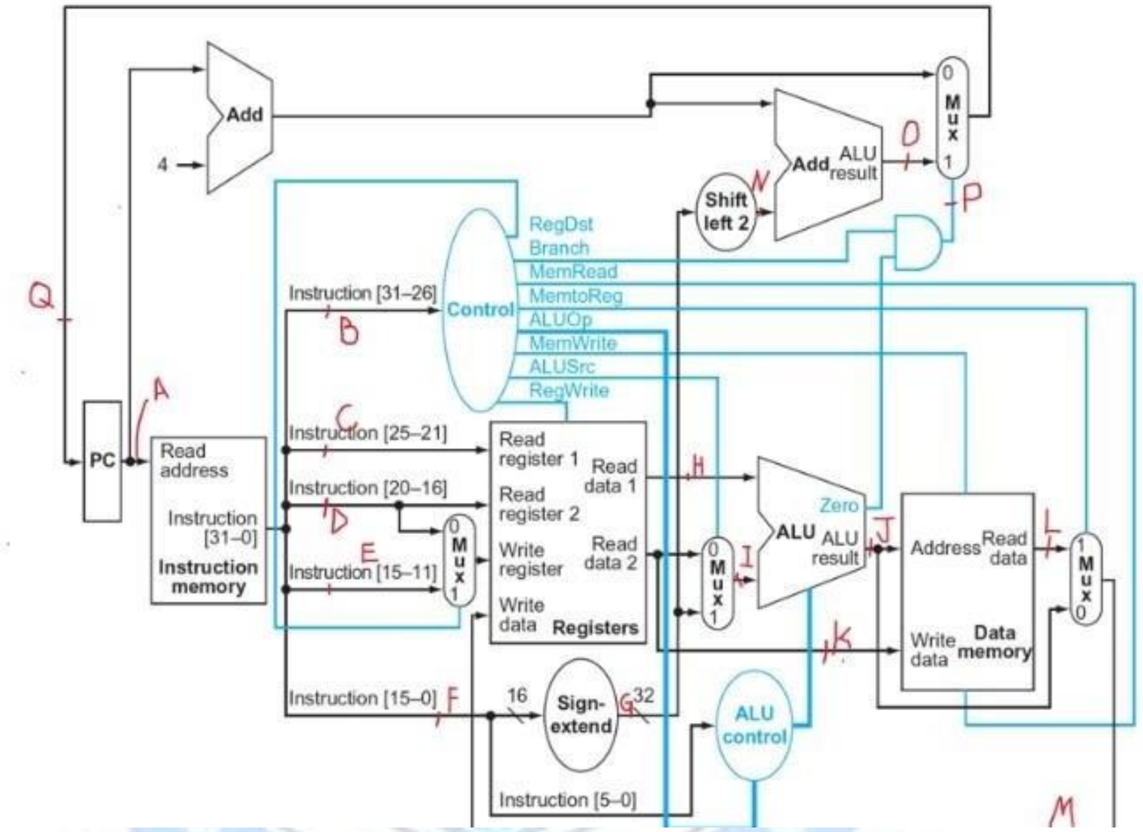


Instruction	RegDst	ALUSrc	Memto-Reg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
R-format	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

Table 3.5: The setting of the control lines is completely determined by the opcode fields of the instruction.

Fig 3.10: shows the data path with the control unit and the control signals. The setting of the control lines depends only on the opcode, we define whether each control signal should be 0,

1, or don't care (X) for each of the opcode values.



The simple data path with the control unit.

<u>DATA PATH FOR THE OPERATION OF A*R-TYPE* INSTRUCTION</u>

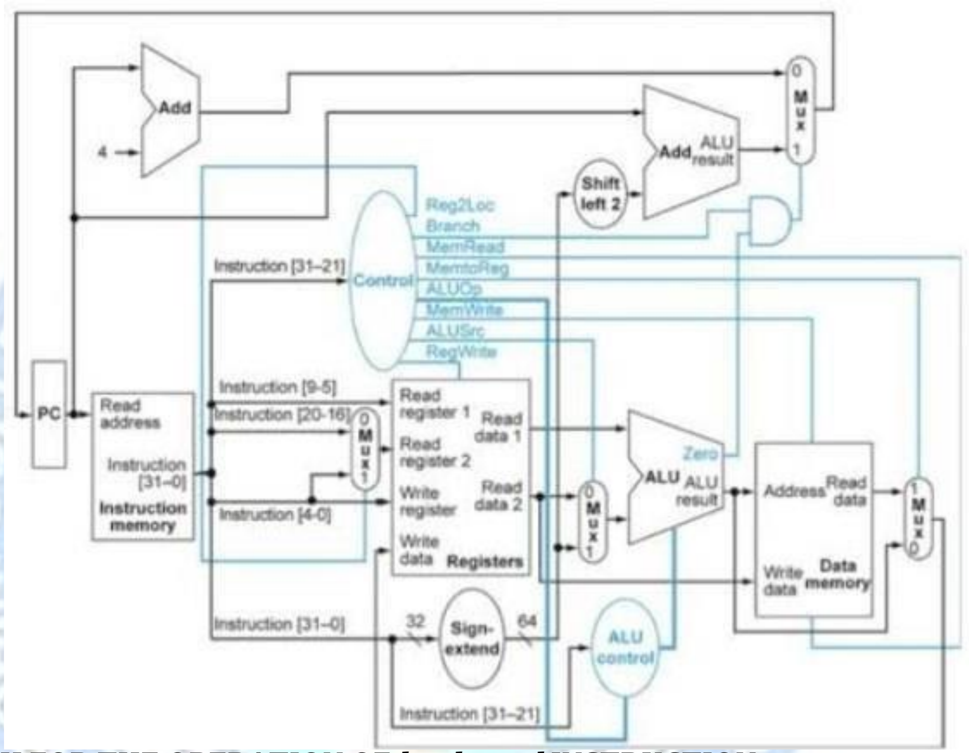
Figure 3.11 shows the operation of the data path for an R-type instruction,such as add $t1,$t2,$t3. Although everything occurs in one clock cycle, we can think of four steps to execute the instruction; these steps are ordered by the flowof information:

1. The instruction is fetched, and the PC is incremented.

2. Two registers, $t2 and $t3, are read from the register file; also, the main control unit computes the setting of the control lines during this step.

3. The ALU operates on the data read from the register file, using the function code (bits 5:0, which is the funct field, of the instruction) to generate the ALU function.

3.The result from the ALU is written into the register file using bits 15:11 of the instruction to select the destination register ($t1).



<u>DATAPATH FOR THE OPERATION OF *load word* INSTRUCTION</u>

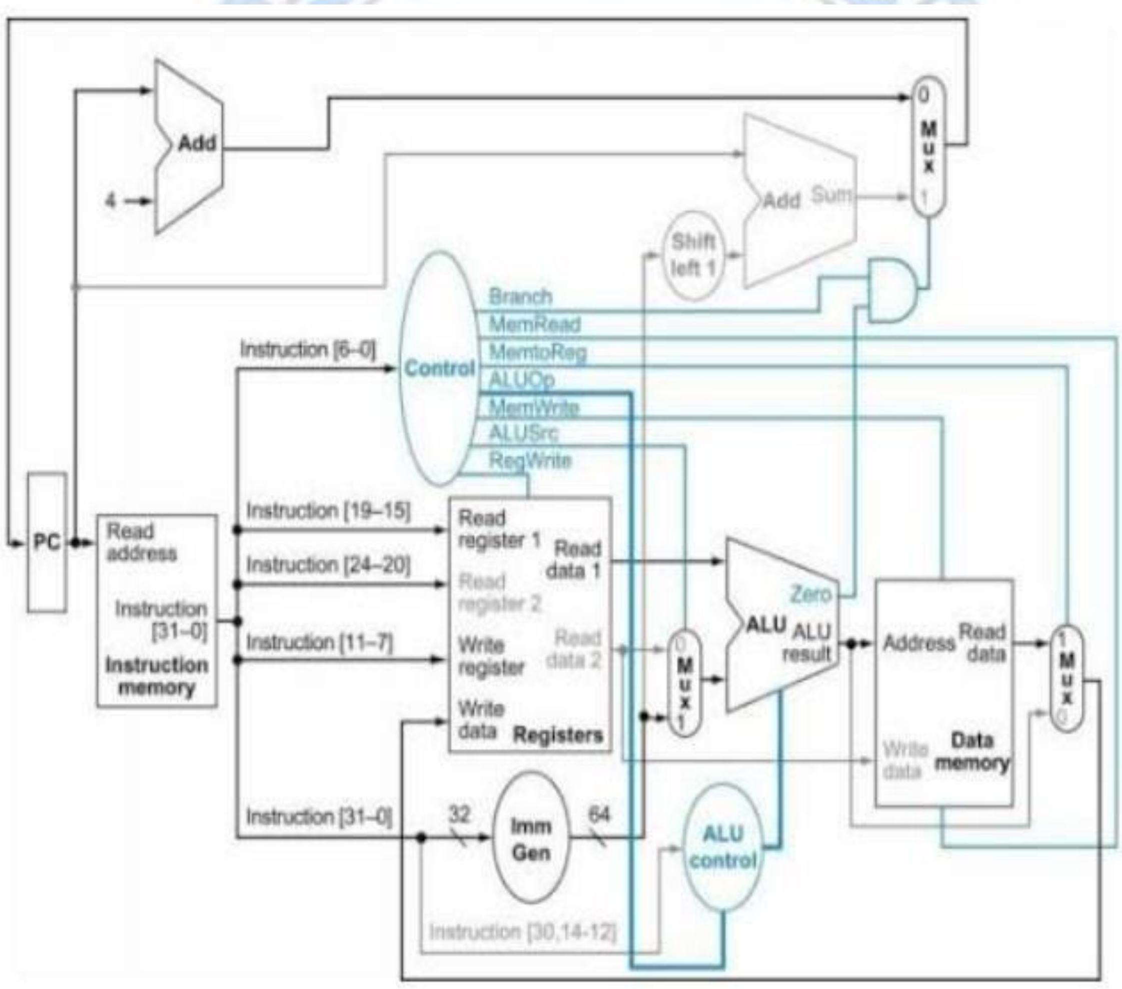
The given figure 3.12 shows the active functional units and asserted control lines for a load. We can think of a load instruction as operating in five steps (similar to how the R-type executed in four):

1. An instruction is fetched from the instruction memory, and the PC isincremented.
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 thesign-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).

Finally, we can show the operation of the branch-on-equal instruction, such as beq $t1, $t2, offset, in the same fashion. It operates much like an R- format instruction, but the ALU output is used to determine whether the PC is written with PC + 4 or the branch target address.

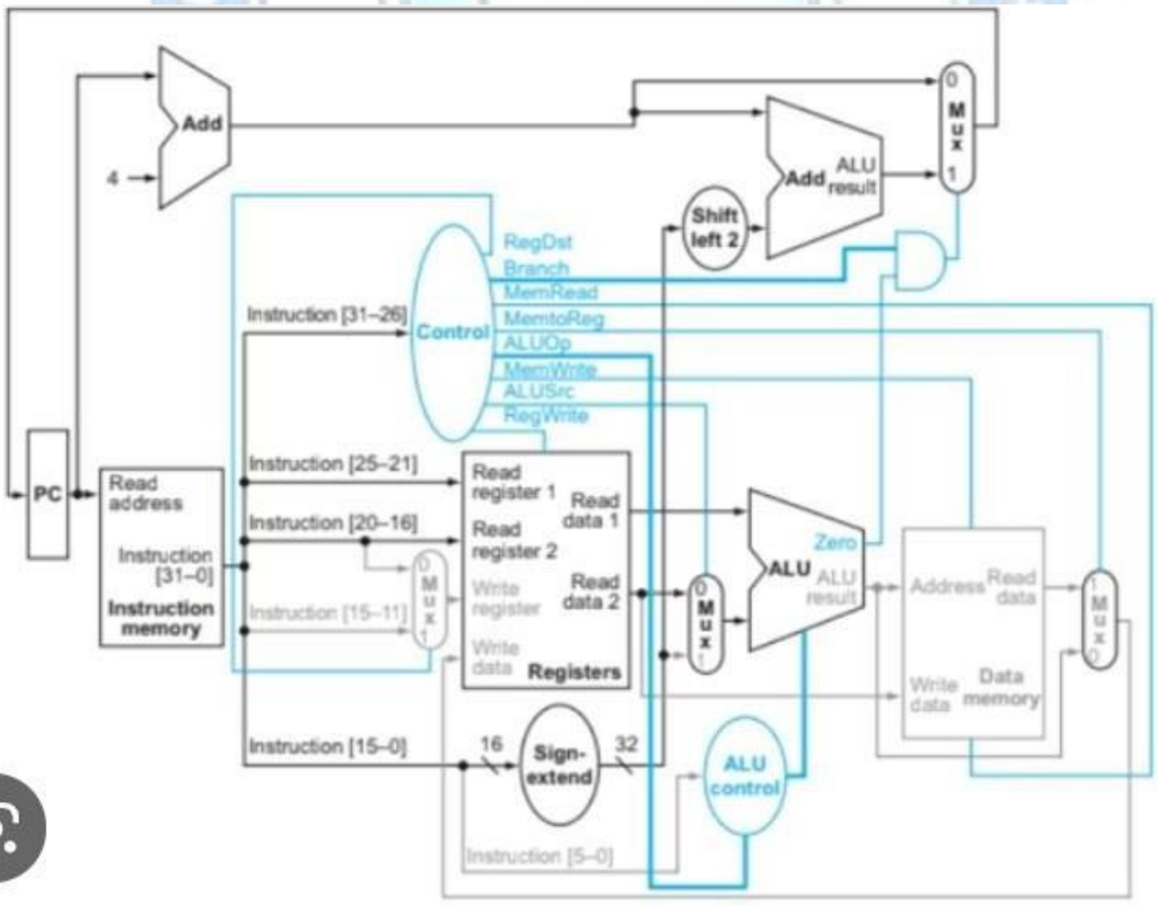
The data path in operation for a load instruction.



DATAPATH FOR THE OPERATION OF *BRANCH-ON-EQUAL* INSTRUCTION

The given figure 3.13 shows the four steps in execution:

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. 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.



Add
4
PC
Read address
Instruction [31–0]
Instruction memory
Instruction [31–26]
Control
RegDst
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Instruction [15–0]
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
16
Sign-extend
32
Instruction [5–0]
ALU control
Shift left 2
Add
ALU result
Mux
0
1
ALU
Zero
ALU result
Address
Read data
Write data
Data memory