## Logic Design Conventions

- To discuss the design of a computer, we must decide how the hardware logic implementing the computer will operate and how the computer is clocked.
- This section reviews a few key ideas in digital logic that we will use extensively.

## Logic Design Conventions

- The datapath elements in the MIPS implementation consist of two different types of logic elements: elements that operate on data values and elements that contain state.

- The elements that operate on data values are all combinational, which means that their outputs depend only on the current inputs.
- Given the same input, a combinational element always produces the same output.
- The ALU shown in Figure 1 is an example of a combinational element. Given a set of inputs, it always produces the same output because it has no internal storage.

## Logic Design Conventions

- Other elements in the design are not combinational, but instead contain state.
- An element contains state if it has some internal storage.
- We call these elements state elements because, if we pulled the power plug on the computer, we could restart it accurately by loading the state elements with the values they contained before we pulled the plug.
- Furthermore, if we saved and restored the state elements, it would be as if the computer had never lost power.
- Thus, these state elements completely characterize the computer.
- In Figure 1, the instruction and data memories, as well as the registers, are all examples of state elements.

## Logic Design Conventions

Combinational element: An operational element, such as an AND gate or an ALU.

State element: A memory element, such as a register or a memory.

## Logic Design Conventions

- A state element has at least two inputs(data and clock) and one output.
- The required inputs are the data value to be written into the element and the clock, which determines when the data value is written.
- The output from a state element provides the value that was written in an earlier clock cycle.
- For example, one of the logically simplest state elements is a D-type flip-flop, which has exactly these two inputs (a value and a clock) and one output.
- In addition to flip-flops, our MIPS implementation uses two other types of state elements: memories and registers.
- The clock is used to determine when the state element should be written; a state element can be read at any time.

## Logic Design Conventions

- Logic components that contain state are also called sequential, because their outputs depend on both their inputs and the contents of the internal state.
- For example, the output from the functional unit representing the registers depends both on the register numbers supplied and on what was written into the registers previously.
- The operation of both the combinational and sequential elements and their construction are already discussed in Module 1 and 2.

## Logic Design Conventions

Clocking Methodology

- A clocking methodology defines when signals can be read and when they can be written.
- It is important to specify the timing of reads and writes, because if a signal is written at the same time it is read, the value of the read could correspond to the old value, the newly written value, or even some mix of the two!
- Computer designs cannot tolerate such unpredictability.
- A clocking methodology is designed to make hardware predictable.

Clocking methodology: The approach used to determine when data is valid and stable relative to the clock.

## Logic Design Conventions

Clocking Methodology

- For simplicity, we will assume an edge-triggered clocking methodology.
- An edge-triggered clocking methodology means that any values stored in a sequential logic element are updated only on a clock edge, which is a quick transition from low to high or vice versa.

## Logic Design Conventions

### Clocking Methodology

- Because only state elements can store a data value, any collection of combinational logic must have its inputs come from a set of state elements and its outputs written into a set of state elements.
- The inputs are values that were written in a previous clock cycle, while the outputs are values that can be used in a following clock cycle.

Edge-triggered clocking: A clocking scheme in which all state changes occur on a clock edge.

## Logic Design Conventions

Clocking Methodology

- Figure 3 shows the two state elements surrounding a block of combinational logic, which operates in a single clock cycle: all signals must propagate from state element 1, through the combinational logic, and to state element 2 in the time of one clock cycle.
- The time necessary for the signals to reach state element 2 defines the length of the clock cycle.

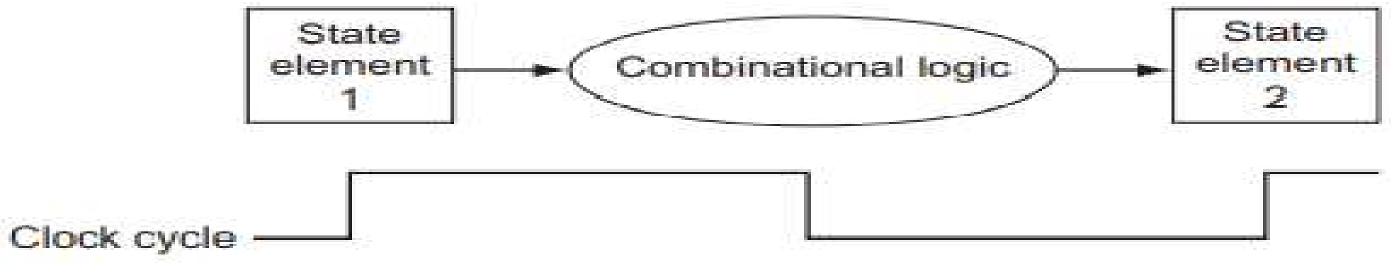

**FIGURE 3 Combinational logic, state elements, and the clock are closely related.** In a synchronous digital system, the clock determines when elements with state will write values into internal storage. Any inputs to a state element must reach a stable value (that is, have reached a value from which they will not change until after the clock edge) before the active clock edge causes the state to be updated. All state elements 'ncluding memory, are assumed to be positive edge-triggered; that is, they change on the rising clock edge.

[J. Hennessy and D. Patterson, "Computer Organization and Design: The Hardware/Software Interface", 5th Edition.]

## Logic Design Conventions

### Clocking Methodology

- For simplicity, we do not show a write control signal when a state element is written on every active clock edge.
- In contrast, if a state element is not updated on every clock, then an explicit write control signal is required.
- Both the clock signal and the write control signal are inputs, and the state element is changed only when the write control signal is asserted and a clock edge occurs.

Control signal: A signal used for multiplexor selection or for directing the operation of a functional unit; contrasts with a data signal, which contains information that is operated on by a functional unit.

## Logic Design Conventions

Clocking Methodology

- We will use the word asserted to indicate a signal that is logically high and assert to specify that a signal should be driven logically high, and deassert or deasserted to represent logically low.
- We use the terms assert and deassert because when we implement hardware, at times 1 represents logically high and at times it can represent logically low.

Asserted: The signal is logically high or true.
Deasserted: The signal is logically low or false.

## Logic Design Conventions

### Clocking Methodology

- An edge-triggered methodology allows us to read the contents of a register, send the value through some combinational logic, and write that register in the same clock cycle.
- Figure 4 gives a generic example.

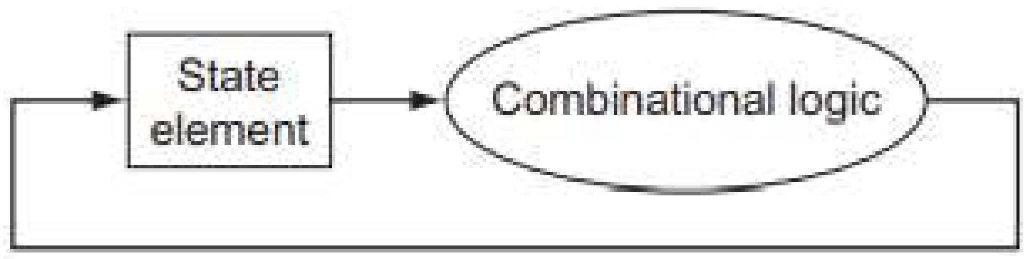

**FIGURE 4. An edge-triggered methodology allows a state element to be read and written in the same clock cycle without creating a race that could lead to indeterminate data values.**

[J. Hennessy and D. Patterson, "Computer Organization and Design: The Hardware/Software Interface", 5th Edition.]

## Logic Design Conventions

Clocking Methodology

- It doesn't matter whether we assume that all writes take place on the rising clock edge (from low to high) or on the falling clock edge (from high to low), since the inputs to the combinational logic block cannot change except on the chosen clock edge.

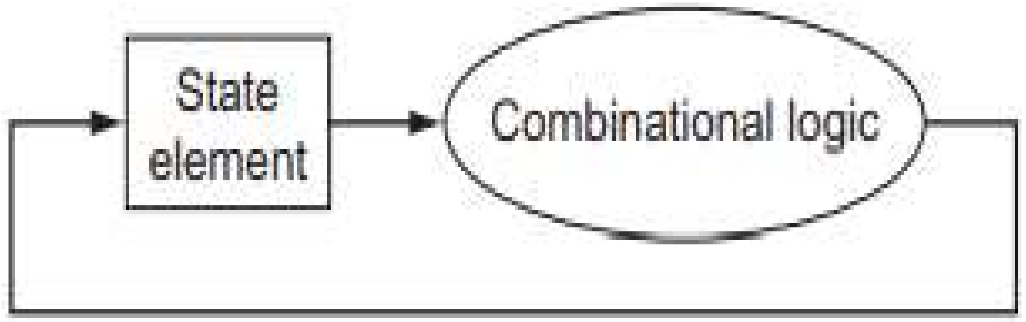

**FIGURE 4.** An edge-triggered methodology allows a state element to be read and written in the same clock cycle without creating a race that could lead to indeterminate data values.

[J. Hennessy and D. Patterson, "Computer Organization and Design: The Hardware/Software Interface", 5th Edition.]

## Logic Design Conventions

Clocking Methodology

- For the 32-bit MIPS architecture, nearly all of these state and logic elements will have inputs and outputs that are 32 bits wide, since that is the width of most of the data handled by the processor.
- The figures will indicate buses, which are signals wider than 1 bit, with thicker lines.
- At times, we will want to combine several buses to form a wider bus; for example, we may want to obtain a 32-bit bus by combining two 16-bit buses.
- In such cases, labels on the bus lines will make it clear that we are concatenating buses to form a wider bus.

## Logic Design Conventions

Clocking Methodology

- Arrows are also added to help clarify the direction of the flow of data between elements.
- Color indicates a control signal as opposed to a signal that carries data.

## Building a Datapath

- A reasonable way to start a datapath design is to examine the major components required to execute each class of MIPS instructions.
- Let’s start at the top by looking at which datapath elements each instruction needs, and then work our way down through the levels of abstraction.
- When we show the datapath elements, we will also show their control signals.
- We use abstraction, starting from the bottom up.

Datapath element: A unit used to operate on or hold data within a processor. In the MIPS implementation, the datapath elements include the instruction and data memories, the register file, the ALU, and adders.

## Building a Datapath

- Figure 5a shows the first element we need: a memory unit to store the instructions of a program and supply instructions given an address.
- Figure 5b shows the program counter (PC), which is a register that holds the address of the current instruction..

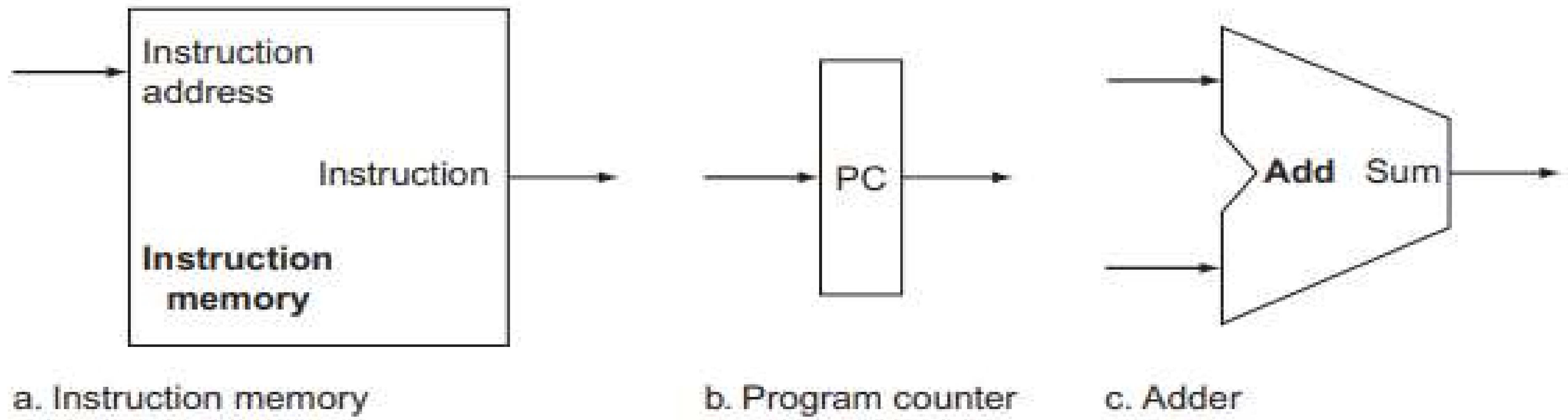

**FIGURE 5 Two state elements are needed to store and access instructions, and an adder is needed to compute the next instruction address.**

[J. Hennessy and D. Patterson, “Computer Organization and Design: The Hardware/Software Interface”, 5th Edition.]

## Building a Datapath

- Lastly, we will need an adder to increment the PC to the address of the next instruction.
- This adder, which is combinational, can be built from the ALU simply by wiring the control lines so that the control always specifies an add operation.
- We will draw such an ALU with the label Add, as in Figure 5c, to indicate that it has been permanently made an adder and cannot perform the other ALU functions.

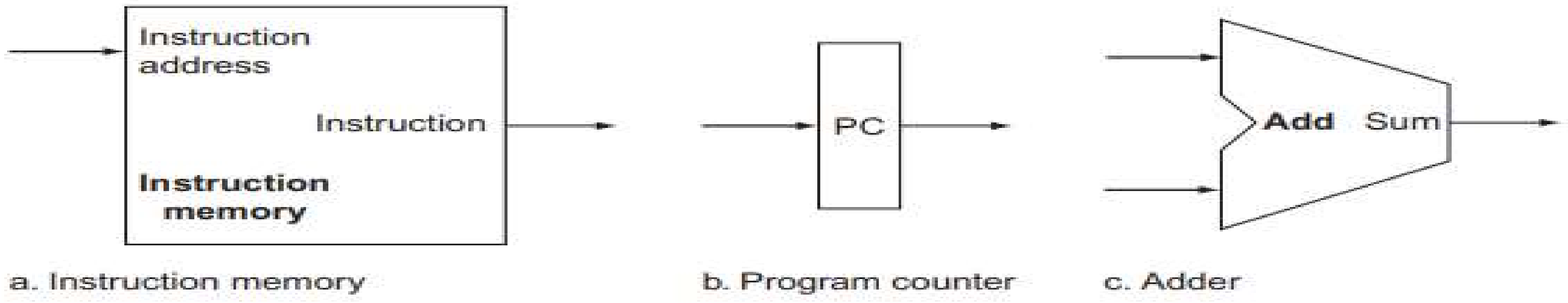

**FIGURE 5** Two state elements are needed to store and access instructions, and an adder is needed to compute the next instruction address.

[J. Hennessy and D. Patterson, "Computer Organization and Design: The Hardware/Software Interface", 5th Edition.]

## Building a Datapath – Fetching instructions

- To execute any instruction, we must start by fetching the instruction from memory.
- To prepare for executing the next instruction, we must also increment the program counter so that it points at the next instruction, 4 bytes later.
- Figure 6 shows how to combine the three elements from Figure 5 to form a datapath that fetches instructions and increments the PC to obtain the address of the next sequential instruction.

Back

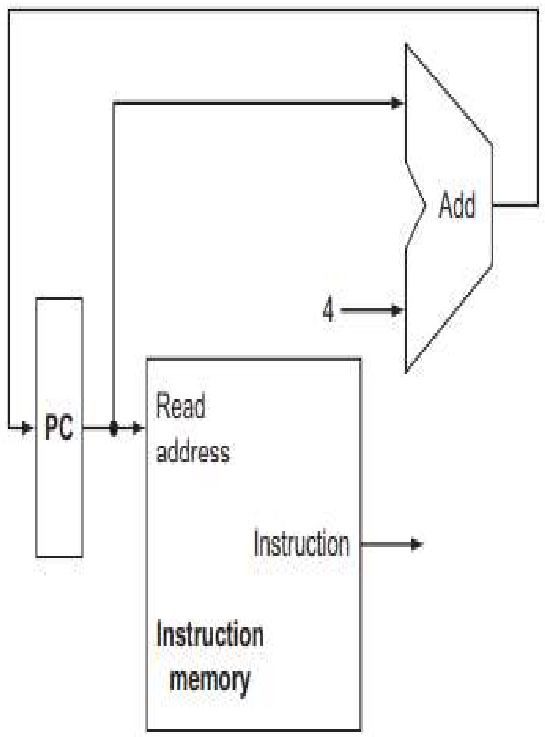

**FIGURE 6 A portion of the datapath used for fetching instructions and incrementing the program counter.** The fetched instruction is used by other parts of the datapath.

[J. Hennessy and D. Patterson, “Computer Organization and Design: The Hardware/Software Interface”, 5th Edition.]

## Building a Datapath R-format instructions

- The R-format instructions read two registers, perform an ALU operation on the contents of the registers, and write the result to a register.
- We call these instructions either R-type instructions or arithmetic-logical instructions (since they perform arithmetic or logical operations).
- This instruction class includes add, sub, AND, OR, and slt.
- A typical instance of such an instruction is add $t1,$t2,$t3, which reads $t2 and $t3 and writes $t1.

## Building a Datapath R-format instructions

- The processor's 32 general-purpose registers are stored in a structure called a register file.
- A register file is a collection of registers in which any register can be read or written by specifying the number of the register in the file.
- The register file contains the register state of the computer.
- In addition, we will need an ALU to operate on the values read from the registers.

Register file: A state element that consists of a set of registers that can be read and written by supplying a register number to be accessed

## Building a Datapath R-format instructions

- R-format instructions have three register operands, so we will need to read two data words from the register file and write one data word into the register file for each instruction.
- For each data word to be read from the registers, we need an input to the register file that specifies the 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, we will need two inputs: one to specify the register number to be written and one to supply the data to be written into the register.
- The register file always outputs the contents of whatever register numbers are on the Read register inputs.

# Building a Datapath R-format instructions

- Writes are controlled by the write control signal, which must be asserted for a write to occur at the clock edge.
- Figure 7a shows the register; we need a total of four inputs (three for register numbers and one for data) and two outputs (both for data). The register number inputs are 5 bits wide to specify one of 32 registers ($32 = 2^5$ ),, whereas the data input and two data output buses are each 32 bits wide.

[J. Hennessy and D. Patterson, “Computer Organization and Design: The Hardware/Software Interface”, 5th Edition.]

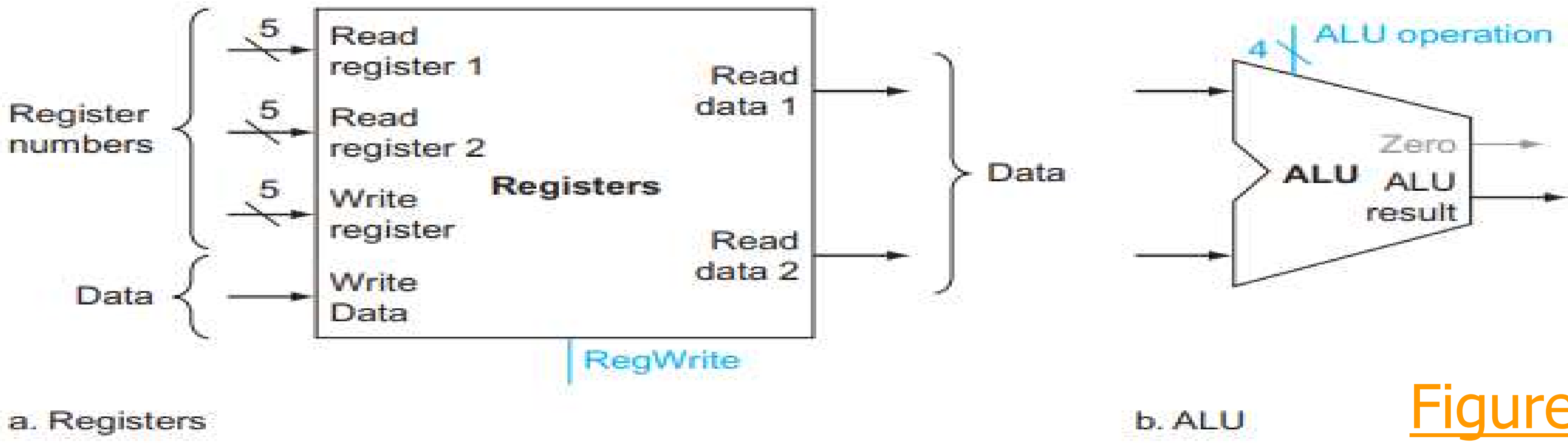

Figure 2

FIGURE 7 The two elements needed to implement R-format ALU operations are the register file and the ALU.

## Building a Datapath R-format instructions

- Figure 7b shows the ALU, which takes two 32-bit inputs and produces a 32-bit result, as well as a 1-bit signal(flag) if the result is 0 (to implement branch). Figure 2
- There is a 4-bit control signal for ALU.

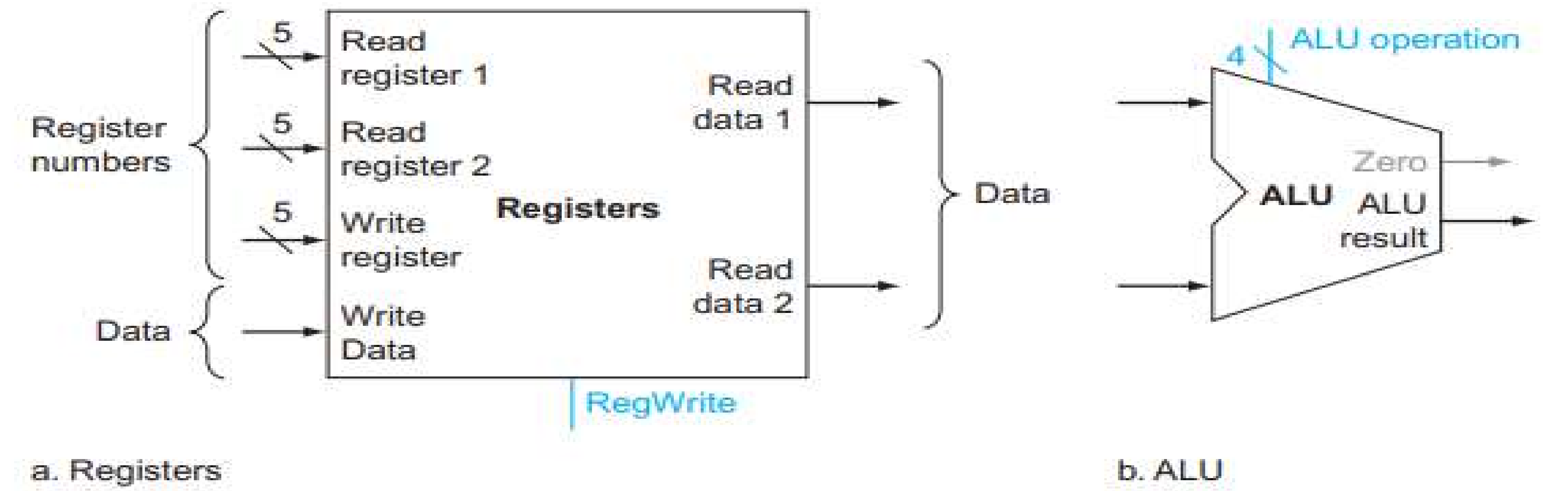

**FIGURE 7 The two elements needed to implement R-format ALU operations are the register file and the ALU.**

[J. Hennessy and D. Patterson, “Computer Organization and Design: The Hardware/Software Interface”, 5th Edition.]

## Building a Datapath Memory reference instructions

- Consider the MIPS load word and store word instructions, which have the general form *lw $t1,offset_value($t2)* or *sw $t1,offset_value ($t2).*
- These instructions compute a memory address by adding the base register, which is $t2, to the 16-bit signed offset field contained in the instruction.
- If the instruction is a store, the value to be stored must also be read from the register file where it resides in $t1.
- If the instruction is a load, the value read from memory must be written into the register file in the specified register, which is $t1.
- Thus, we will need both the register file and the ALU.

## Building a Datapath Memory reference instructions

- In addition, we will need a unit to sign-extend the 16-bit offset field in the instruction to a 32-bit signed value, and a data memory unit to read from or write to.
- The data memory must be written on store instructions; hence, data memory has read and write control signals, an address input, and an input for the data to be written into memory. Figure 8 shows these two elements.

[J. Hennessy and D. Patterson, "Computer Organization and Design: The Hardware/Software Interface", 5th Edition.]

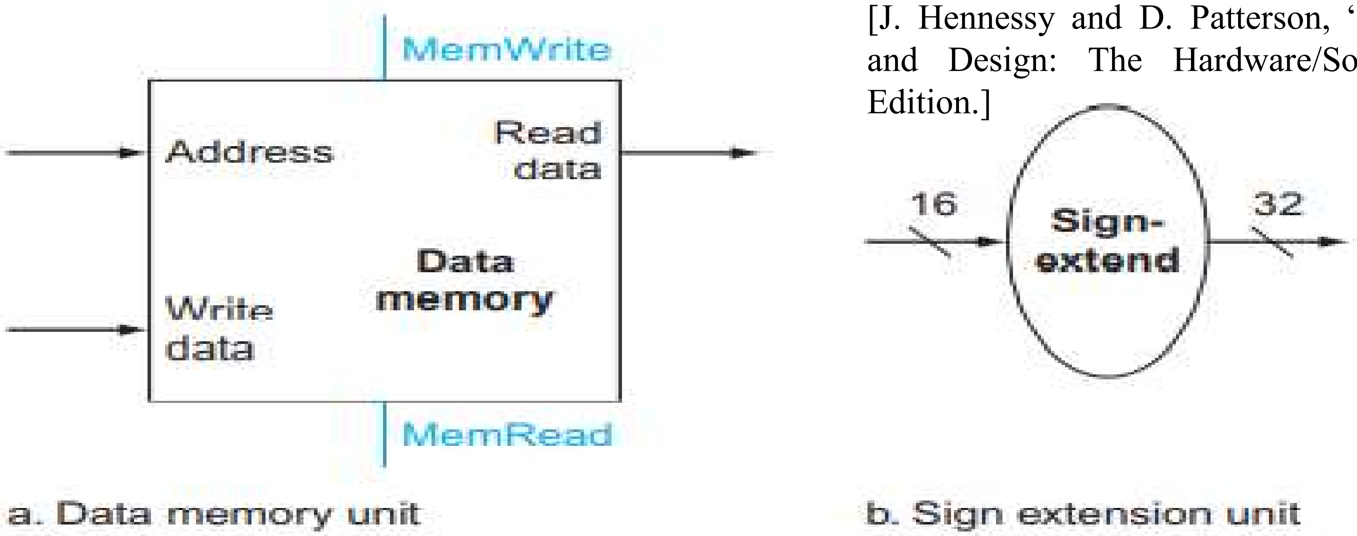

FIGURE 8 The two units needed to implement loads and stores, in addition to the register file and ALU are the data memory unit and the sign extension unit.

## Building a Datapath Memory reference instructions

- The memory unit is a state element with inputs for the address and the write data, and a single output for the read result.
- There are separate read and write controls, although only one of these may be asserted on any given clock.
- The memory unit needs a read signal, since, unlike the register file, reading the value of an invalid address can cause problems.
- The sign extension unit has a 16-bit input that is sign-extended into a 32-bit result appearing on the output.
- We assume the data memory is edge-triggered for writes. Standard memory chips actually have a write enable signal that is used for writes.

[J. Hennessy and D. Patterson, "Computer Organization and Design: The Hardware/Software Interface", 5th Edition.]

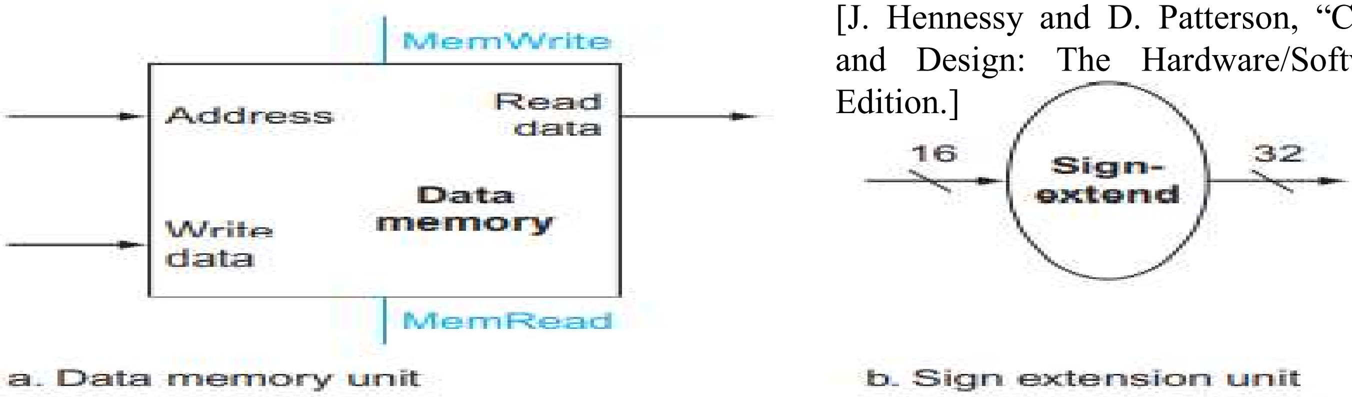

FIGURE 8 The two units needed to implement loads and stores, in addition to the register file and ALU are the data memory unit and the sign extension unit.

## Building a Datapath Branch instructions

- The *beq* instruction has three operands, two registers that are compared for equality, and a 16-bit off set used to compute the branch target address relative to the branch instruction address.
- Its form is *beq $t1,$t2,offset*.
- To implement this instruction, we must compute the branch target address by adding the sign-extended offset field of the instruction to the PC.

Branch target address: The address specified in a branch, which becomes the new program counter (PC) if the branch is taken. In the MIPS architecture the branch target is given by the sum of the offset field of the instruction and the address of the instruction following the branch.

## Building a Datapath Branch instructions

- There are two details in the definition of branch instructions to which we must pay attention:
  - ■ The instruction set architecture specifies that the base for the branch address calculation is the address of the instruction following the branch. Since we compute PC + 4 (the address of the next instruction) in the instruction fetch datapath, it is easy to use this value as the base for computing the branch target address.

■ The architecture also states that the offset field is shifted left 2 bits so that it is a word offset; this shift increases the effective range of the offset field by a factor of 4.

(The unit labeled Shift left 2 is simply a routing of the signals between input and output that adds $00_{two}$ to the low-order end of the sign-extended off set field; no actual shift hardware is needed, since the amount of the "shift " is constant. Since we know that the off set was sign-extended from 16 bits, the shift will throw away only "sign bits.")

## Building a Datapath Branch instructions

- While computing the branch target address, we must also determine whether the next instruction is the instruction that follows sequentially or the instruction at the branch target address.
- When the condition is true (i.e., the operands are equal), the branch target address becomes the new PC, and we say that the branch is taken.
- If the operands are not equal, the incremented PC should replace the current PC (just as for any other normal instruction); in this case, we say that the branch is not taken.

# Building a Datapath Branch instructions

- The branch datapath must do two operations: compute the branch target address and compare the register contents.
- Branches also affect the instruction fetch portion of the datapath.
- Figure 9 shows the structure of the datapath segment that handles branches.
- To compute the branch target address, the branch datapath includes a sign extension unit and an adder.

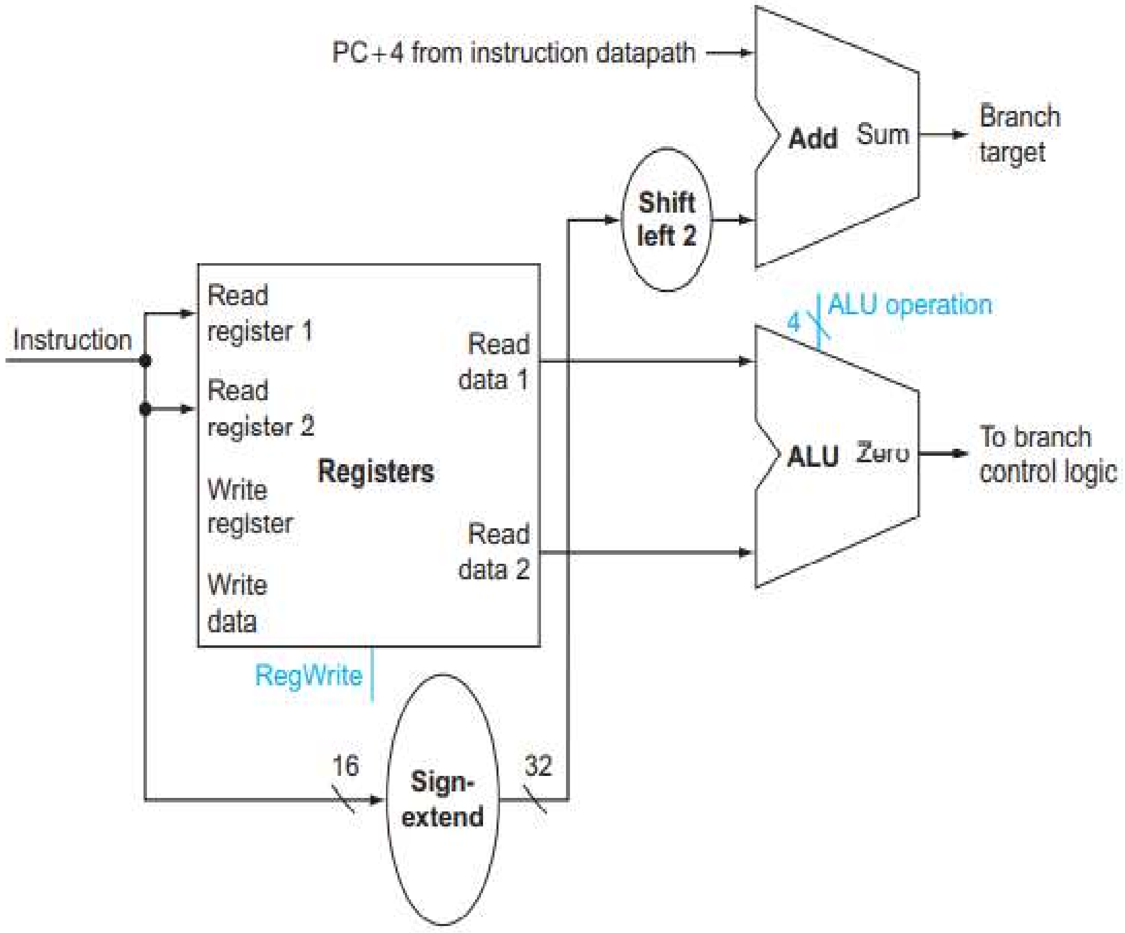

**FIGURE 9** The datapath for a branch uses the ALU to evaluate the branch condition and a separate adder to compute the branch target as the sum of the incremented PC and the sign-extended, lower 16 bits of the instruction (the branch displacement), shifted left 2 bits.

[J. Hennessy and D. Patterson, "Computer Organization and Design: The Hardware/Software Interface", 5th Edition.]

## Building a Datapath Branch instructions

- To perform the compare, we need to use the register file to supply the two register operands (although we will not need to write into the register file).
- In addition, the comparison can be done using the ALU.

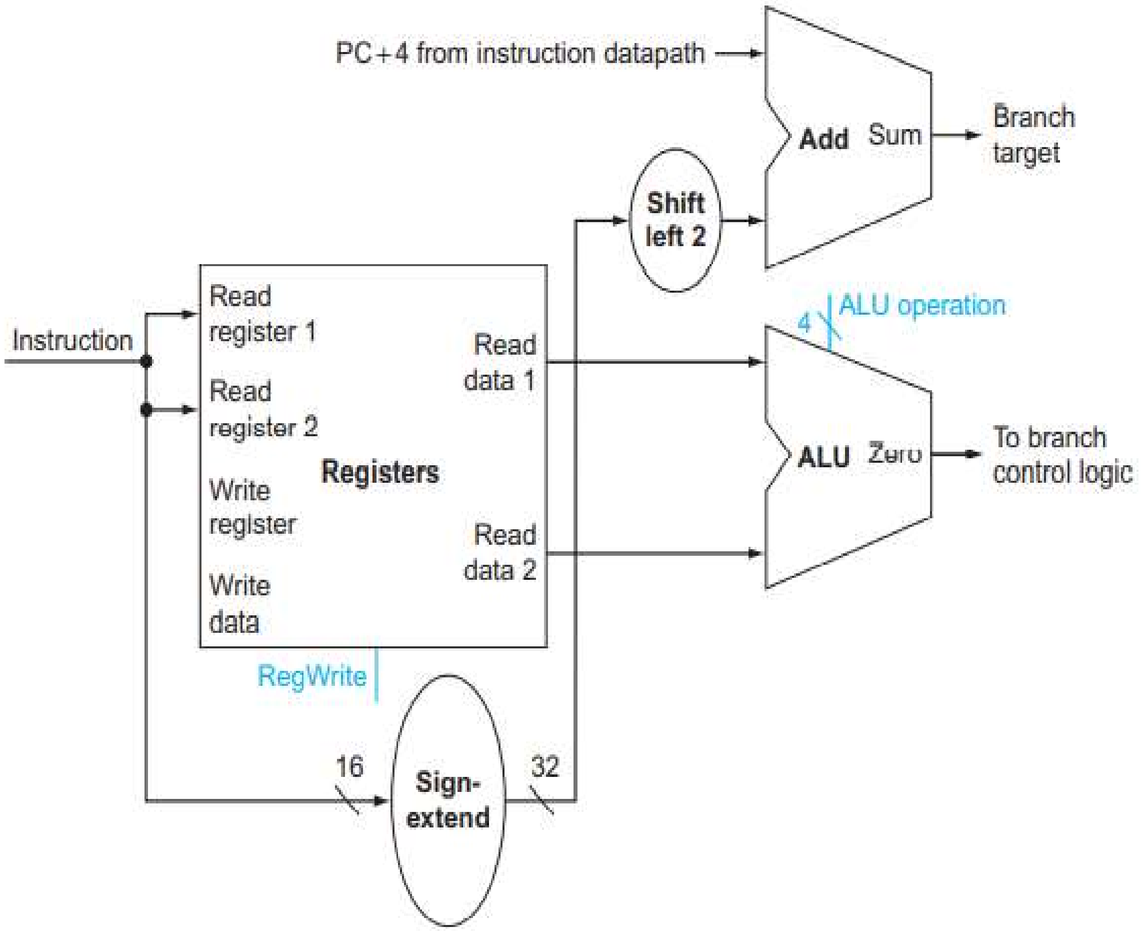

**FIGURE 9 The datapath for a branch uses the ALU to evaluate the branch condition and a separate adder to compute the branch target as the sum of the incremented PC and the sign-extended, lower 16 bits of the instruction (the branch displacement), shifted left 2 bits.**

[J. Hennessy and D. Patterson, "Computer Organization and Design: The Hardware/Software Interface", 5th Edition.]

## Building a Datapath Branch instructions

- Since that ALU provides an output signal that indicates whether the result was 0, we can send the two register operands to the ALU with the control set to do a subtract.
- If the Zero signal out of the ALU unit is asserted, we know that the two values are equal.
- Although the Zero output always signals if the result is 0, we will be using it only to implement the equal test of branches.

Back

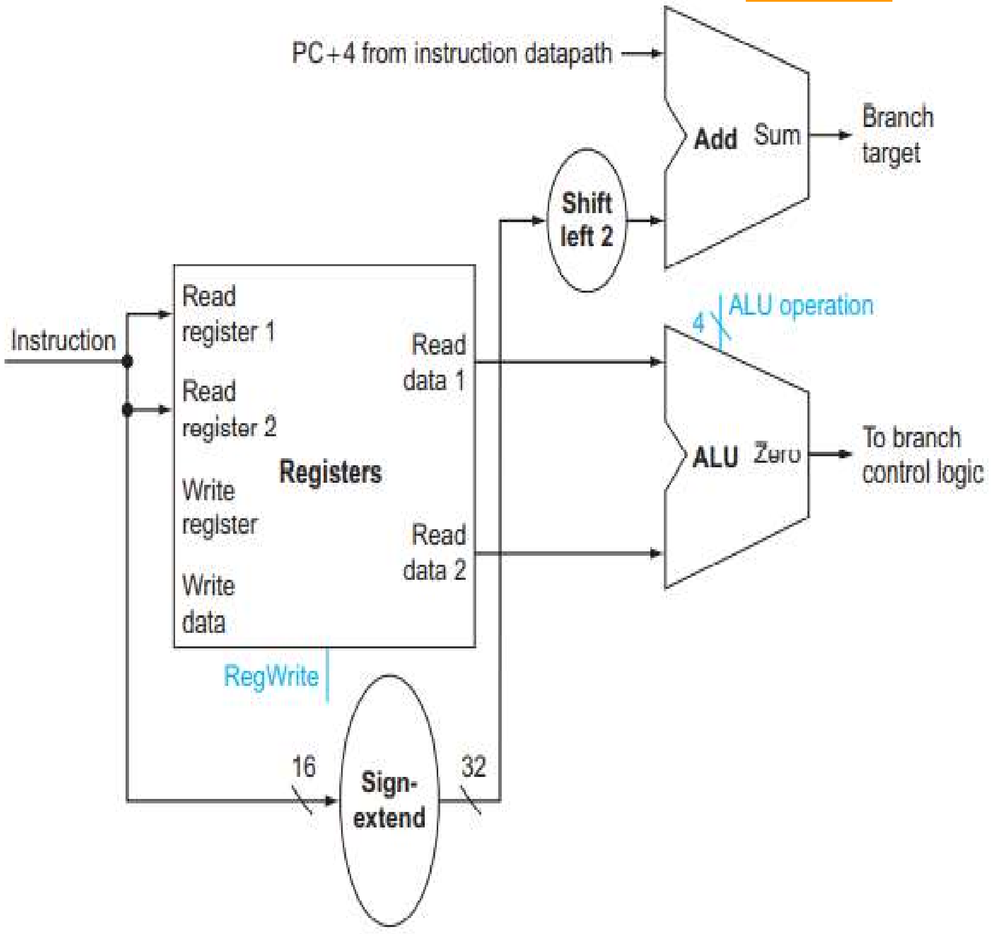

**FIGURE 9** The datapath for a branch uses the ALU to evaluate the branch condition and a separate adder to compute the branch target as the sum of the incremented PC and the sign-extended, lower 16 bits of the instruction (the branch displacement), shifted left 2 bits.

[J. Hennessy and D. Patterson, "Computer Organization and Design: The Hardware/Software Interface", 5th Edition.]