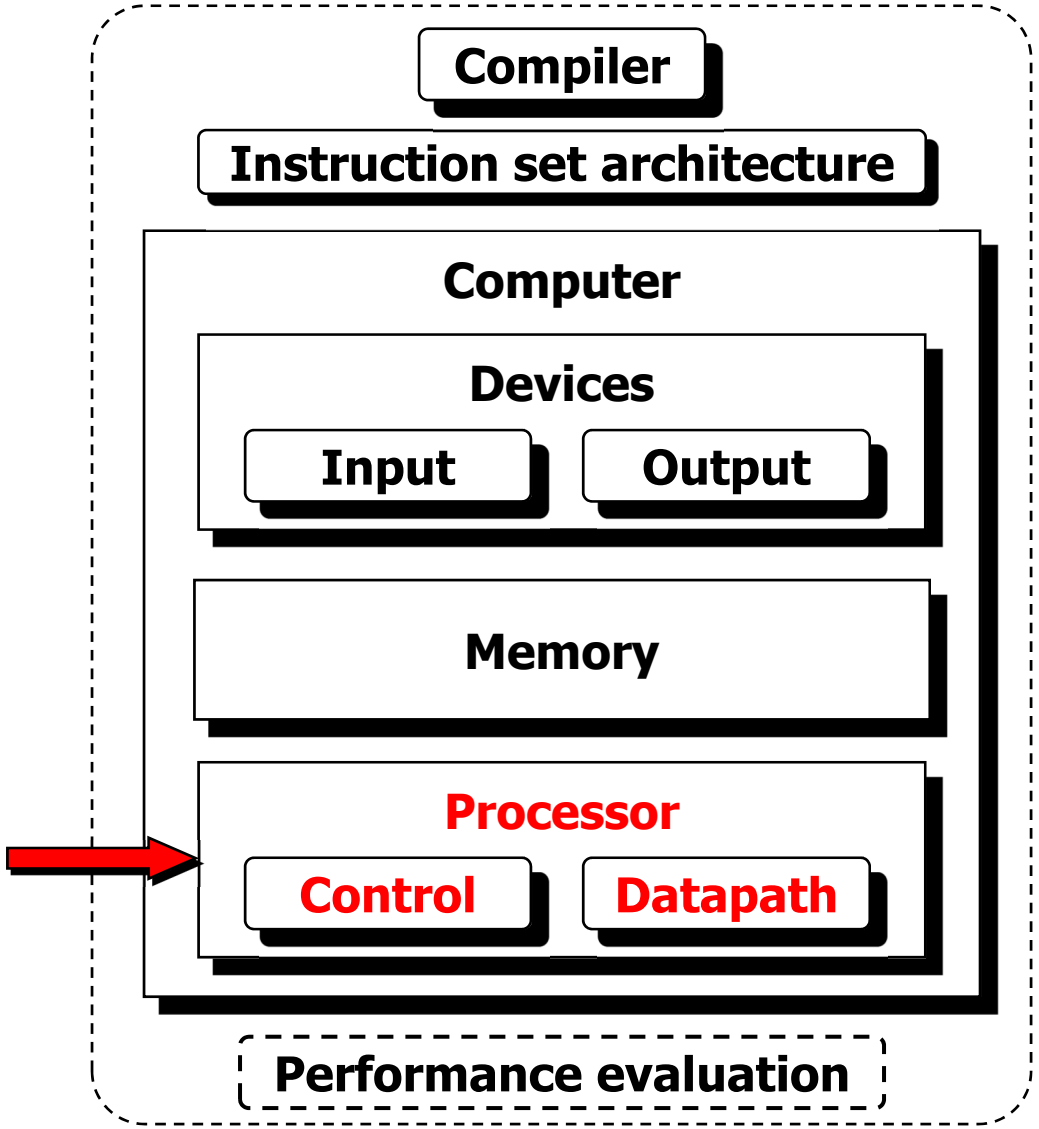
The Processor

Chapter 4

Part I



Agenda and Reading List

- Chapter goals
- Introduction (4.1, pp. 300-303)
- ISA summary
- The five classic components of a computer (1.4, pp. 16-17)
- The two classes of computer architectures
- Generic implementation (4.1, pp. 301-303)
- Clocking methodology (4.2, pp. 303-307)
- Datapath elements and building a datapath (4.3, pp. 307-313)
- Creating a single datapath (4.3, pp. 313-315)
- A simple implementation scheme (4.4, pp. 316-330)
- The ALU control (4.4, pp. 316-318)
- The main control unit (4.4, pp. 318-321)
- The operation of the datapath (4.4, pp. 321-327)
- Implementing unconditional jumps (4.4, p. 328)
- Inefficiencies in the single-cycle implementation (4.4, pp. 328-330)

Chapter Goals

- to explain the principles and techniques used in implementing a modern RISC processor, starting with a highly abstract and simplified overview
- to develop an understanding of combinational and clocked sequential circuits and the relationship between them
- to see how the ISA determines many aspects of the implementation
- to construct the datapath, ALU control unit, and main control unit for a single-cycle implementation of the MIPS instruction set
- to discuss how the choice of various implementation strategies affects the clock rate and CPI for the computer

Introduction

- The implementation of the processor determines both the clock cycle time and the number of clock cycles per instruction (CPI)
- We will be examining an implementation that includes a subset of the core MIPS instruction set:
  - The memory-reference instructions: `lw, sw`
  - The arithmetic-logical instructions: `add, sub, and, or, slt`
  - The control flow (branch and jump) instructions: `beq, j`
- A simple implementation that uses a **single long clock cycle** for every instruction will be shown
  - Every instruction begins execution on one clock edge and completes execution on the next clock edge
  - While easier to understand, this approach is not practical
    - since the clock cycle must be stretched to accommodate the longest instruction
    - it leads to a slow implementation
- We will concentrate on designing the control
  - Control is the most challenging aspect of the processor design
  - It is the hardest part to get right and to make fast

ISA Summary

Field bit positions	31-26	25-21	20-16	15-11	10-6	5-0
No. of bits	6	5	5	5	5	6
R-format	op	rs	rt	rd	shamt	funct
I-format	op	rs	rt	16- bit immediate/address		
J-format	op	26-bit address				

- The opcode field, op[5:0] 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
- The base register for `lw` and `sw` is always in bit positions 25:21 (rs)
- The 16-bit offset for `lw`, `sw`, and `beq` is always in bit positions 15:0
- The destination register is either
  - in bit positions 20:16 (rt) for `lw` or
  - in bit positions 15:11 (rd) for an R-format instruction

ISA Summary (cont.)

Instruction	Format	op	funct	Instruction	Format	op	funct
add	R	0_{10}	32_{10}	or	R	0	37
addi	I	8		ori	I	13	
and	R	0	36	sb	I	40	
andi	I	12		sh	I	41	
beq	I	4		sll	R	0	0
bne	I	5		slt	R	0	42
j	J	2		slti	I	10	
jal	J	3		sra	R	0	3
jr	R	0	8	srl	R	0	2
lb	I	32		sub	R	0	34
lh	I	33		sw	I	43	
lui	I	15		xor	R	0	38
lw	I	35		xori	I	14	
nor	R	0	39				

ISA Summary (cont.)

- **MIPS Registers**

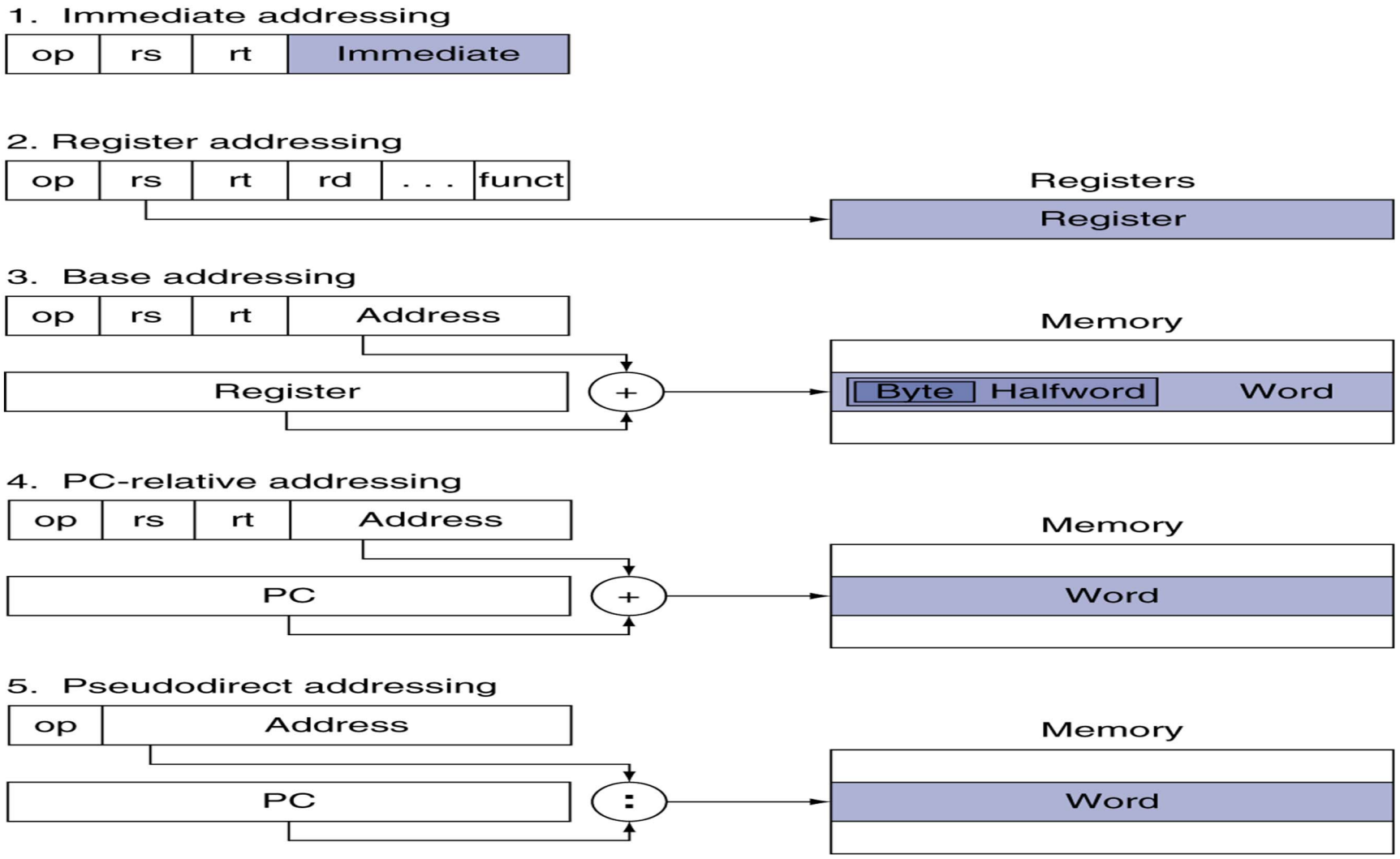
Name	Register number (decimal)	Usage	Preserve on call?
$zero	0	the constant value 0	n.a.
$at	1	reserved for the assembler	n.a.
$v0-$v1	2-3	procedure return values and expression evaluation	no
$a0-$a3	4-7	procedure arguments (parameters)	no
$t0-$t7	8-15	temporary registers	no
$s0-$s7	16-23	general purpose saved registers	yes
$t8-$t9	24-25	more temporary registers	no
$k0-$k1	26-27	reserved for the OS	n.a.
$gp	28	global pointer	yes
$sp	29	stack pointer	yes
$fp	30	frame pointer	yes
$ra	31	procedure return address	yes

ISA Summary (cont.)

- **MIPS Addressing Modes**
  - **Immediate addressing:** the operand is a constant within the instruction
    ```
    addi $s1, $s2, 100
    ```
  - **Register addressing:** the operand is a register
    ```
    add  $s1, $s2, $s3
    ```
  - **Base or displacement addressing:** the operand is at the memory location whose address is the sum of a register and a constant in the instruction
    ```
    lw   $s1, 100 ($s2)
    ```
  - **PC-relative addressing:** the address is the sum of the current PC and a constant (multiplied by 4) in the instruction
    ```
    beq  $s1, $s2, 100
    ```
  - **Pseudodirect addressing:** the jump address is a constant (26 bits) in the instruction (multiplied by 4) concatenated with the upper 4 bits of the PC
    ```
    j    2500
    ```

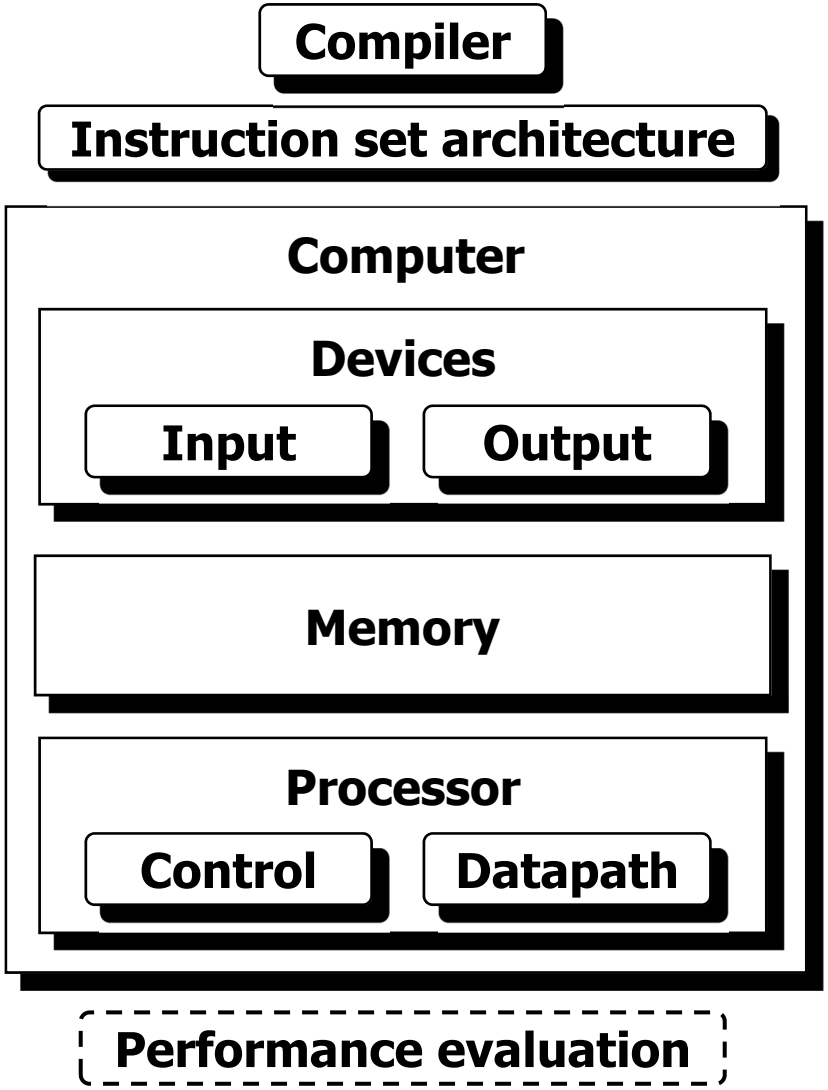
ISA Summary (cont.)

- **MIPS Addressing Modes (cont.)**



The Five Classic Components of a Computer

- **Five classic components:**
  - **Memory:** where instructions and data are kept
  - **Input:** writes data to memory
  - **Output:** reads data from memory
  - **Datapath:** performs arithmetic operations
  - **Control:** sends the signals that determine the operations of the datapath, memory, input, and output according to instructions



The Five Classic Components of a Computer (cont.)

- **Processor:** manipulates instructions and data
- A processor consists of datapath and control
- This computer organization is independent of hardware technology; you can place every piece of every computer into one of these five categories
- Underlying hardware in any computer performs the same basic functions:
  - inputting data,
  - outputting data,
  - processing data, and
  - storing data

The Two Classes of Computer Architectures

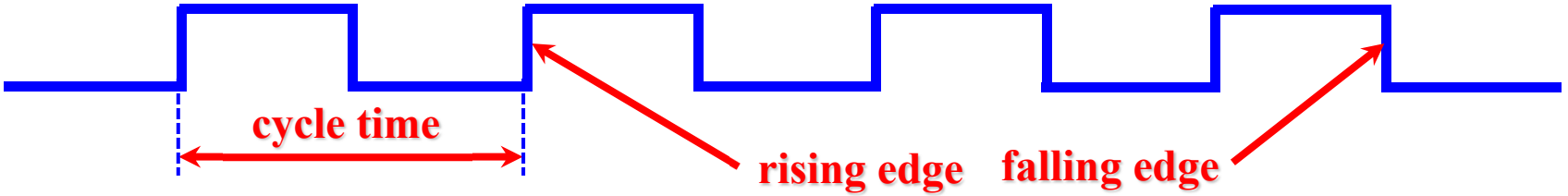
- Different computer architectures are classified into one of the following two types:

- **von Neumann (Princeton) architecture**
  - Describes computers based on the following three key concepts:
    - Data and instructions are stored in a single read-write memory
    - The contents of this memory are addressable by location, without regard to the type of data contained there
    - Execution occurs in a sequential fashion (unless explicitly modified) from one instruction to the next

- **Harvard architecture**
  - **traditionally:** describes computers with separate memories; one for data and the other for instructions
  - **today:** describes computers with a single main memory, but with separate caches for instructions and data

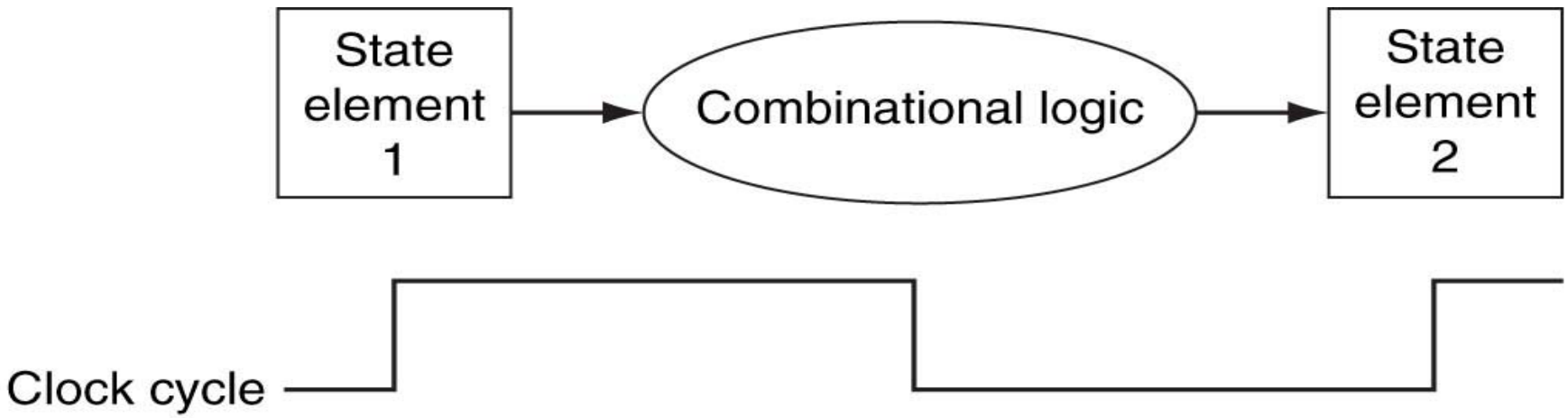
Clocking Methodology

- **There are two different types of logic elements:**
  1. elements that operate on data values (combinational), e.g., ALU
  2. elements that contain state (sequential), e.g., memory and registers



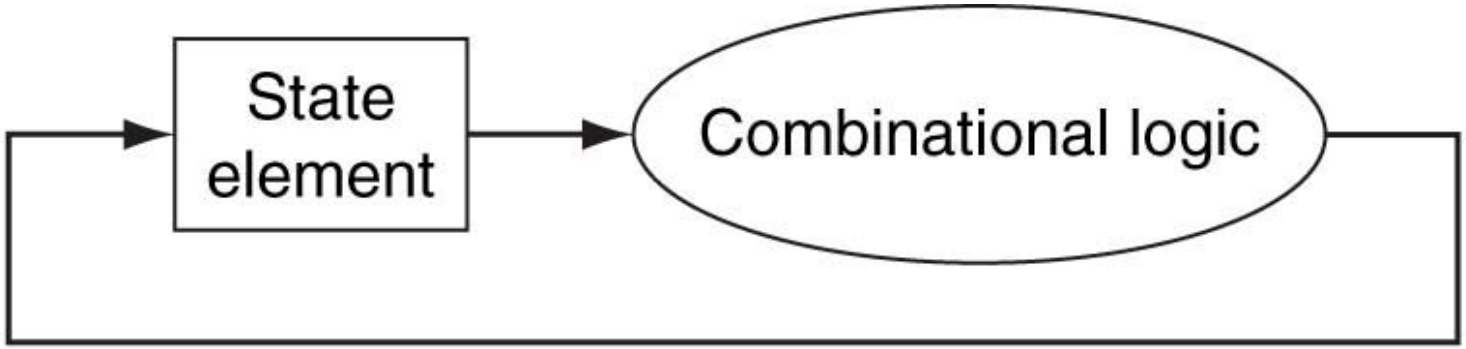
- A clocking methodology is used in synchronous systems to define when signals can be read and when they can be written
- **An edge-triggered clocking methodology will be used:**
  - Values stored in a state element are updated only on a clock edge
  - State elements update their internal storage on clock edges
  - As 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 written in a previous clock cycle, while the outputs are values that can be used in the following clock cycle

Clocking Methodology (cont.)



- All signals must propagate from state element 1, through the combinational logic, and to state element 2 in the time of one cycle
- The time necessary for the signals to reach state element 2 defines the length of the clock cycle
- The value stored in a state element is changed only when the write control signal is asserted and a clock edge occurs
- Clock signals are not shown
- For simplicity, write control signals will not be shown when a state element is written on every active clock edge

Clocking Methodology (cont.)



- An edge-triggered methodology allows doing all of the following in the same clock cycle:
  - reading the contents of a register
  - sending the value read through some combinational logic
  - writing back to that register

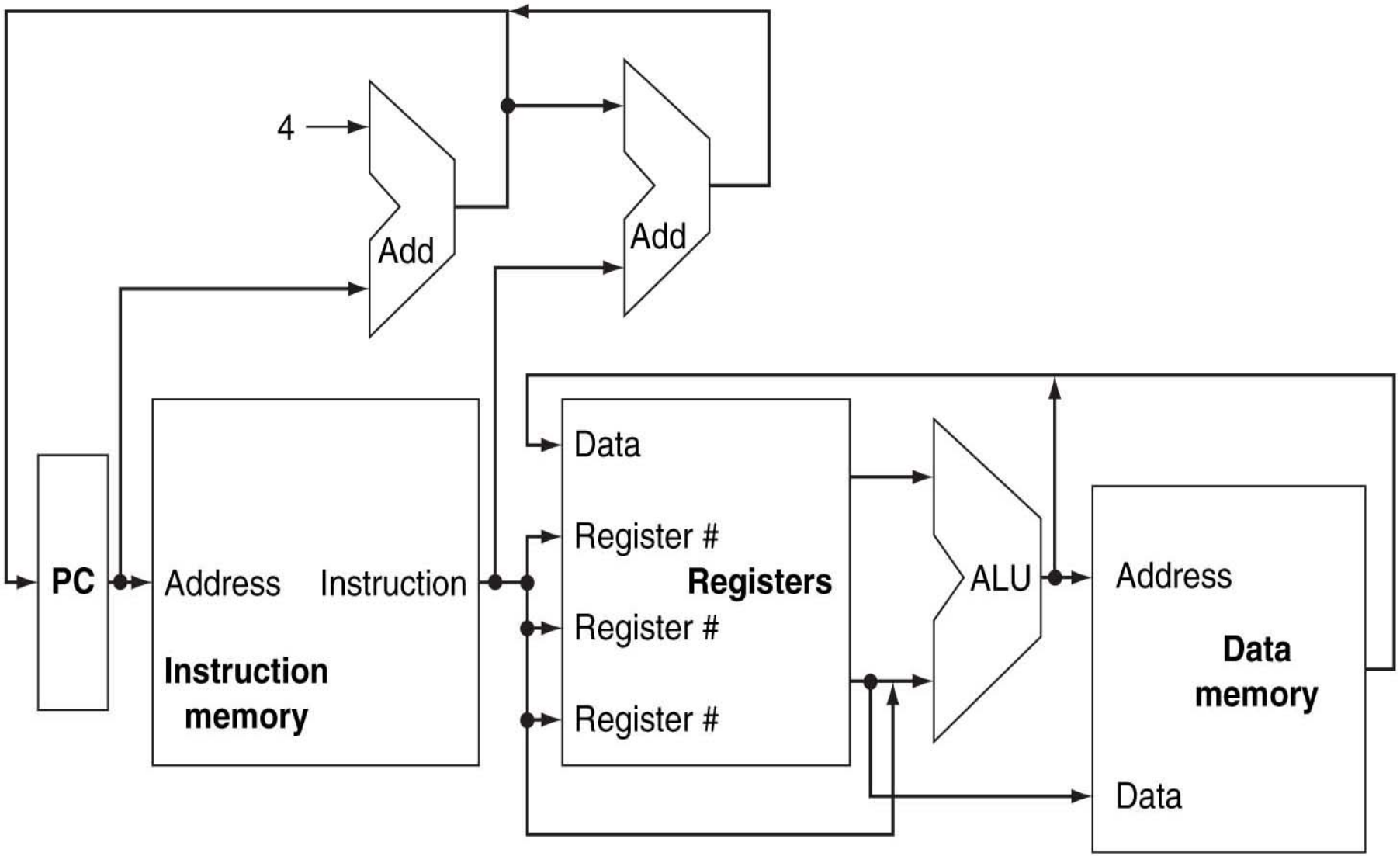
- Feedback cannot occur within one clock cycle because of the edge-triggered update of the state element

Generic Implementation

- Much of what needs to be done to implement the instructions is the same, independent of the exact class of instruction
- For every instruction, the first two implementation steps are identical:
  - use PC to fetch the instruction from the instruction memory
  - use the instruction fields to select one or two registers to read:
    - `lw` requires reading only one register
    - `j` does not require accessing any register
    - All other instructions require reading two registers
- After these two steps, the actions required to complete the instruction depend on the instruction class

- **ISA simplicity and regularity:**
  - simplifies the implementation by making the execution of many of the instruction classes similar
    - Within a class, actions are largely independent of exact opcode

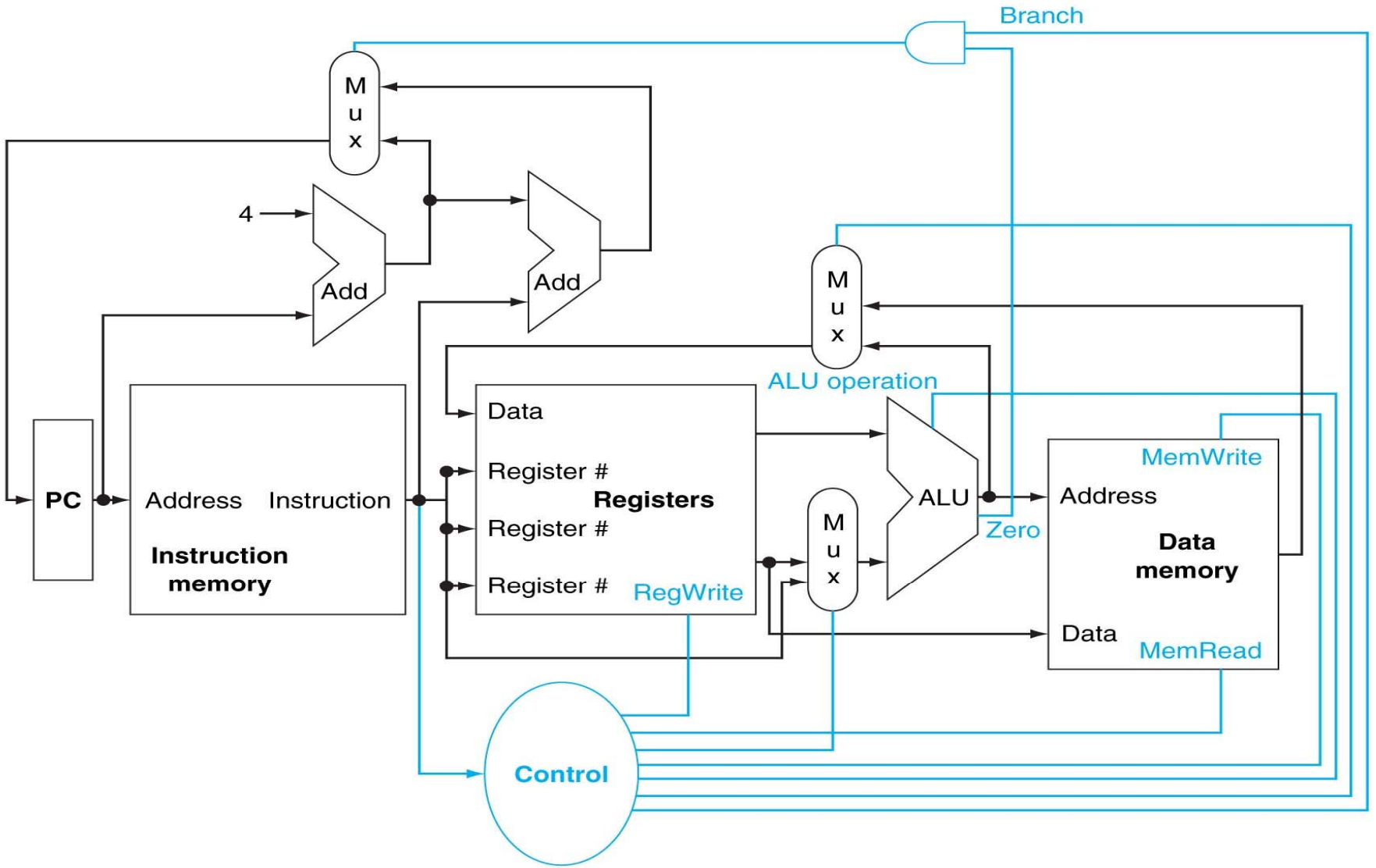
Generic Implementation (cont.)



Generic Implementation (cont.)

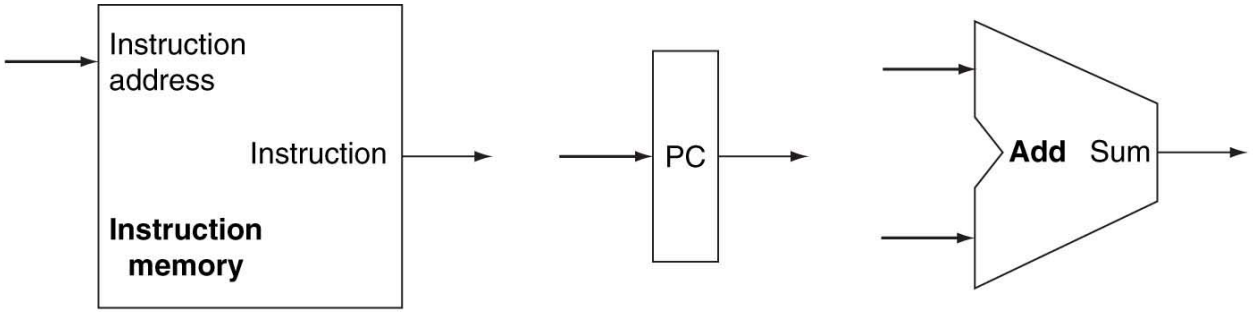
- There are some similarities across different instruction classes, e.g.,
  - All instruction classes, except `j`, use the ALU after reading the registers:
    - Memory-reference instructions use the ALU for an address calculation
    - Arithmetic-logical instructions use it for the operation execution
    - The branch instruction (`beq`) uses it for comparison
  - After using the ALU, the actions required to complete various instruction classes differ:
    - A memory-reference instruction accesses the data memory either to write data for a store or read data for a load
    - An arithmetic-logical instruction writes the data from the ALU back into a register
    - A load instruction writes the data from the memory back into a register
    - A branch instruction may need to change the next instruction address based on the comparison; otherwise the PC should be incremented by 4 to get the address of the next instruction

Generic Implementation (cont.)



Datapath Elements (1)

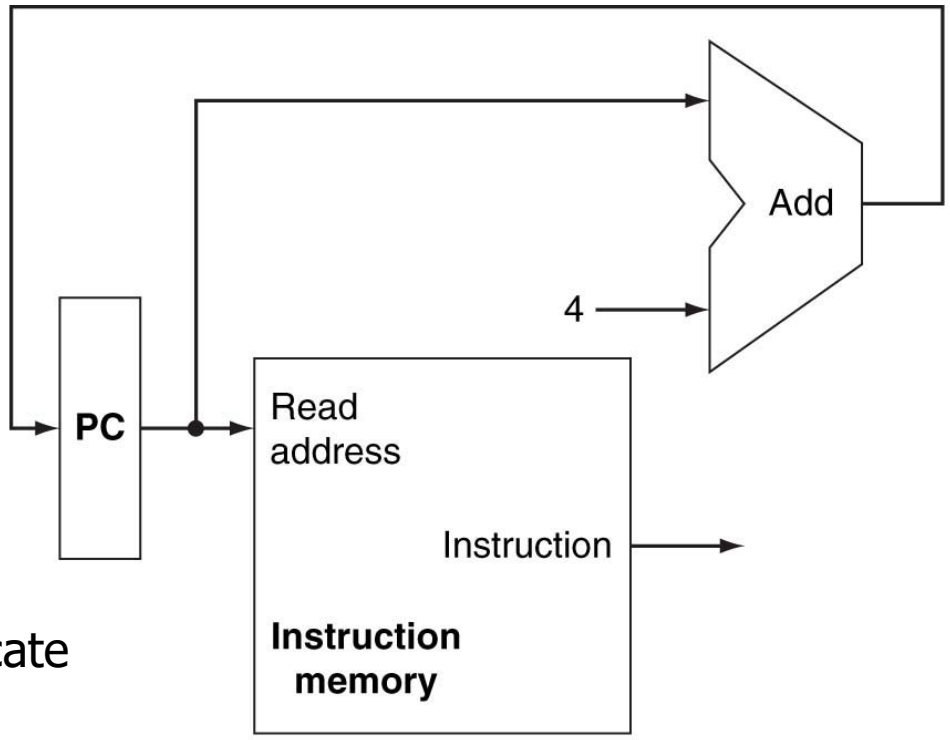
- **Datapath element:** A functional unit used to operate on or hold data within a processor
- **Instruction memory:**
  - Stores and supplies the instructions of a program given an address
  - No read signal is needed as it is read every clock cycle
  - This memory is not written
- **Program counter (PC):**
  - Holds the address of the current instruction (to be fetched)
  - No write signal is needed as it is written every clock cycle
- **PC adder:**
  - Used to increment the PC to the address of the next instruction
  - A combinational circuit that can be built from the ALU by writing the control line so that the control always specifies an add operation



a. Instruction memory b. Program counter c. Adder

Building a Datapath (1)

- **Instruction fetch and PC increment:**
  - To execute any instruction, we must start by fetching the instruction from the instruction memory
  - To prepare for executing the next instruction, we must also increment the PC so that it points at the next instruction, 4 bytes later



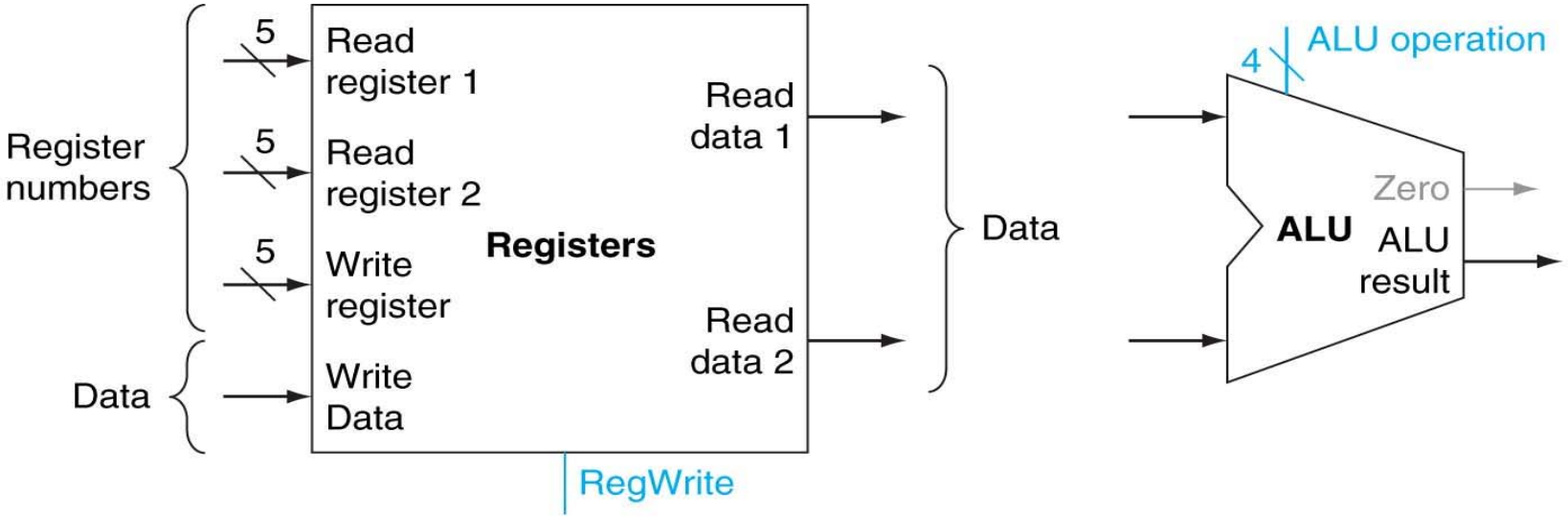
Thick lines (with no width indication) indicate buses of 32 bits

Datapath Elements (2)

- **Register file:**
  - A collection of the processor 32 general-purpose registers in which any register can be read or written by specifying its number in the file
  - The register file contains the register state of the computer
  - No read control signal is needed
    - The register file always outputs the contents of whatever register numbers are on the read register inputs
  - Writes are controlled by the write control signal (**RegWrite**)
    - must be asserted for a write to occur at the clock edge
  - Since writes to the register file are edge triggered, the design can legally read and write the same register within a clock cycle:
    - The read will get the value written in an earlier clock cycle
    - The value written will be available to read in a subsequent clock cycle

Datapath Elements (2) (cont.)

- **Arithmetic and logic unit:**
  - The ALU is used to operate on the values read from the registers
  - It takes 2 32-bit inputs and produces a 32-bit result, as well as a 1-bit signal if the result is zero
  - It is controlled by a 4-bit control signal (**ALU operation**)

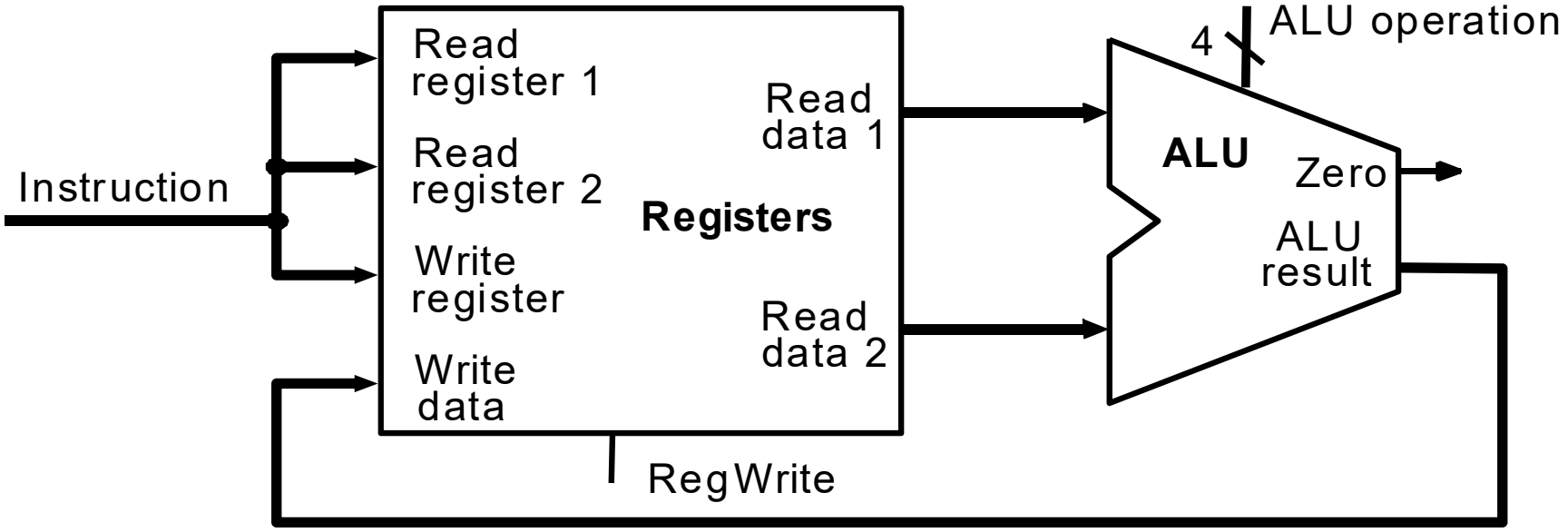


a. Registers

b. ALU

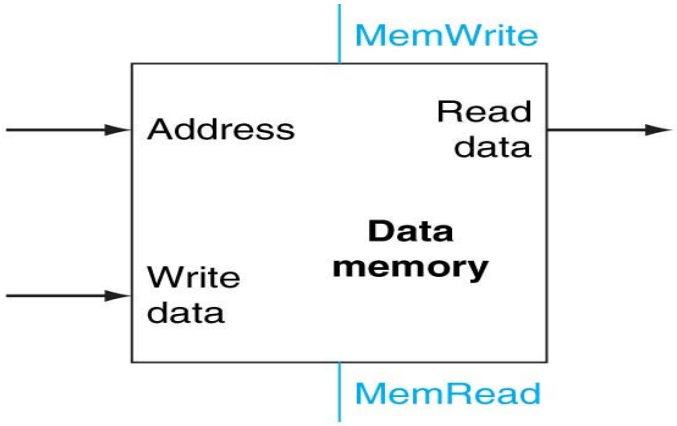
Building a Datapath (2)

- **Executing R-format instructions: (`add $t1, $t2, $t3`):**
  - Read 2 data words from the register file, for each we need to input the register number (5 bits) to the register file and there will be an output (32 bits) that will carry the value that has been read from the registers
  - Perform an ALU operation on the contents of the registers
  - Write one data word result into the register file, for which we need two inputs: one (5 bits) to specify the register number to be written and another to supply the data (32 bits) to be written into the register

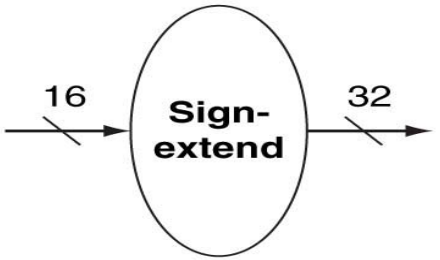


Datapath Elements (3)

- **Sign-extension unit:**
  - Sign-extends the 16-bit offset field in the instruction to a 32-bit signed value
  - Replicates the high-order sign bit of the original data item in the high-order bits of the larger, destination data item
- **Data memory:**
  - To be read from or written to
  - It must be written on store instructions; hence, it has both read and write control signals, an address input, and an input for the data to be written into memory



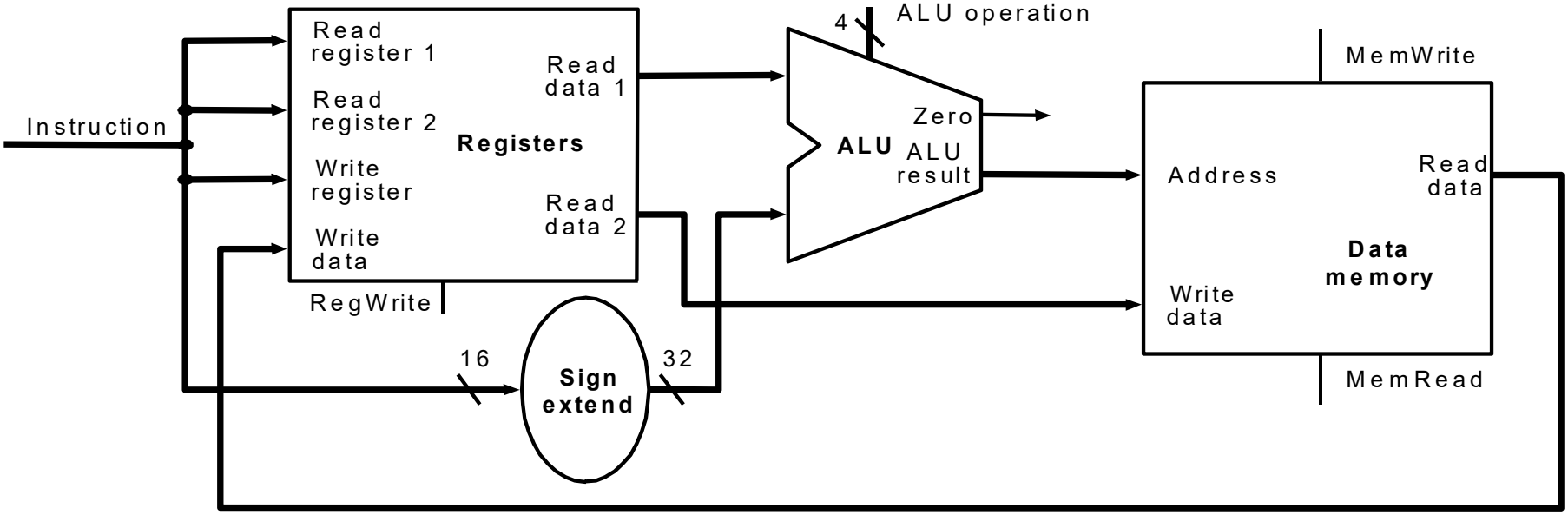
a. Data memory unit



b. Sign extension unit

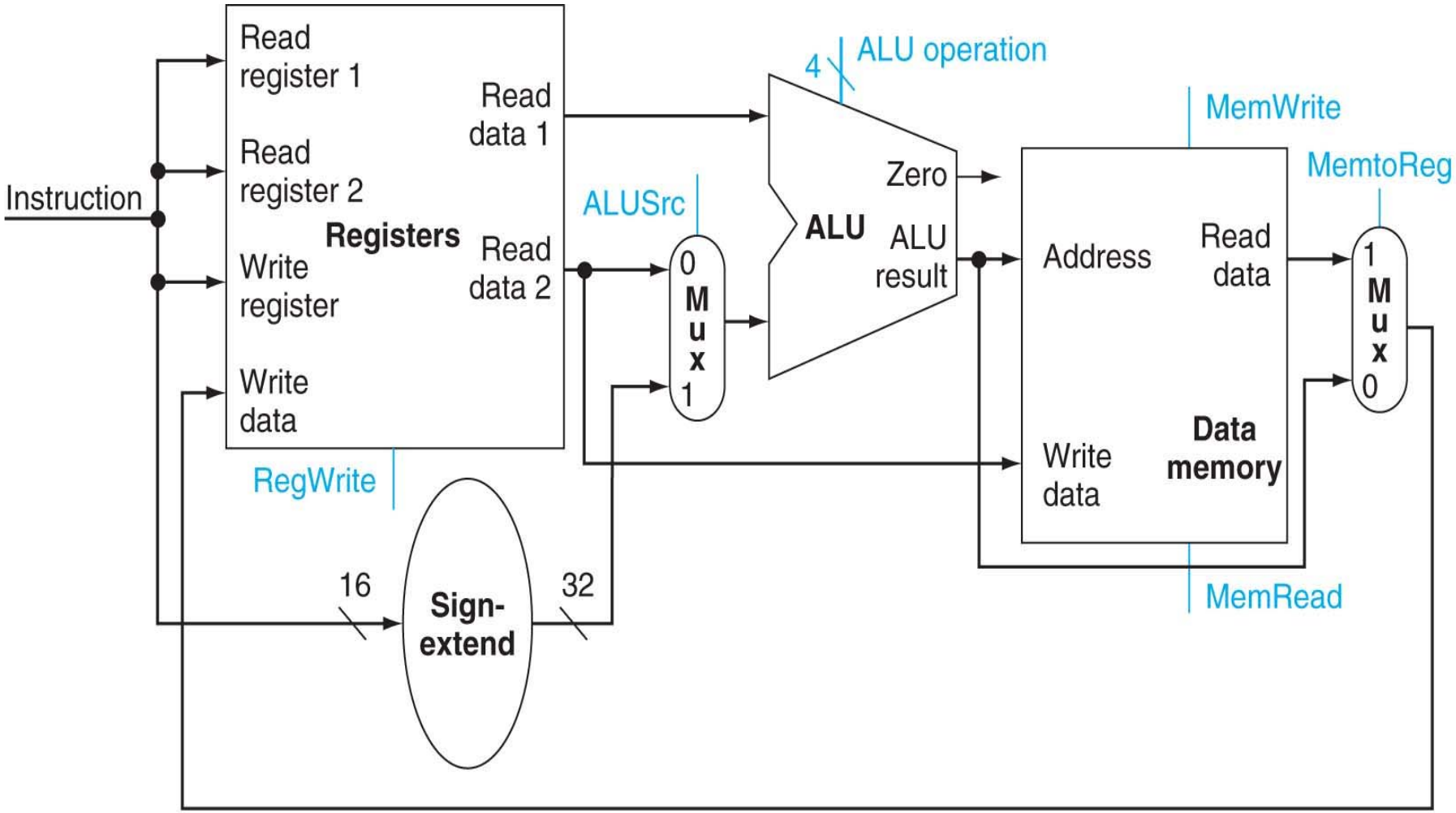
Building a Datapath (3)

- **Executing loads/stores (`lw $t1, 4($t2)`/`sw $t1, 4($t2)`):**
  - Compute a memory address by adding the base register 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
  - If the instruction is a load, the value read from memory must be written into the register file in the specified register



Building a Datapath (3) (cont.)

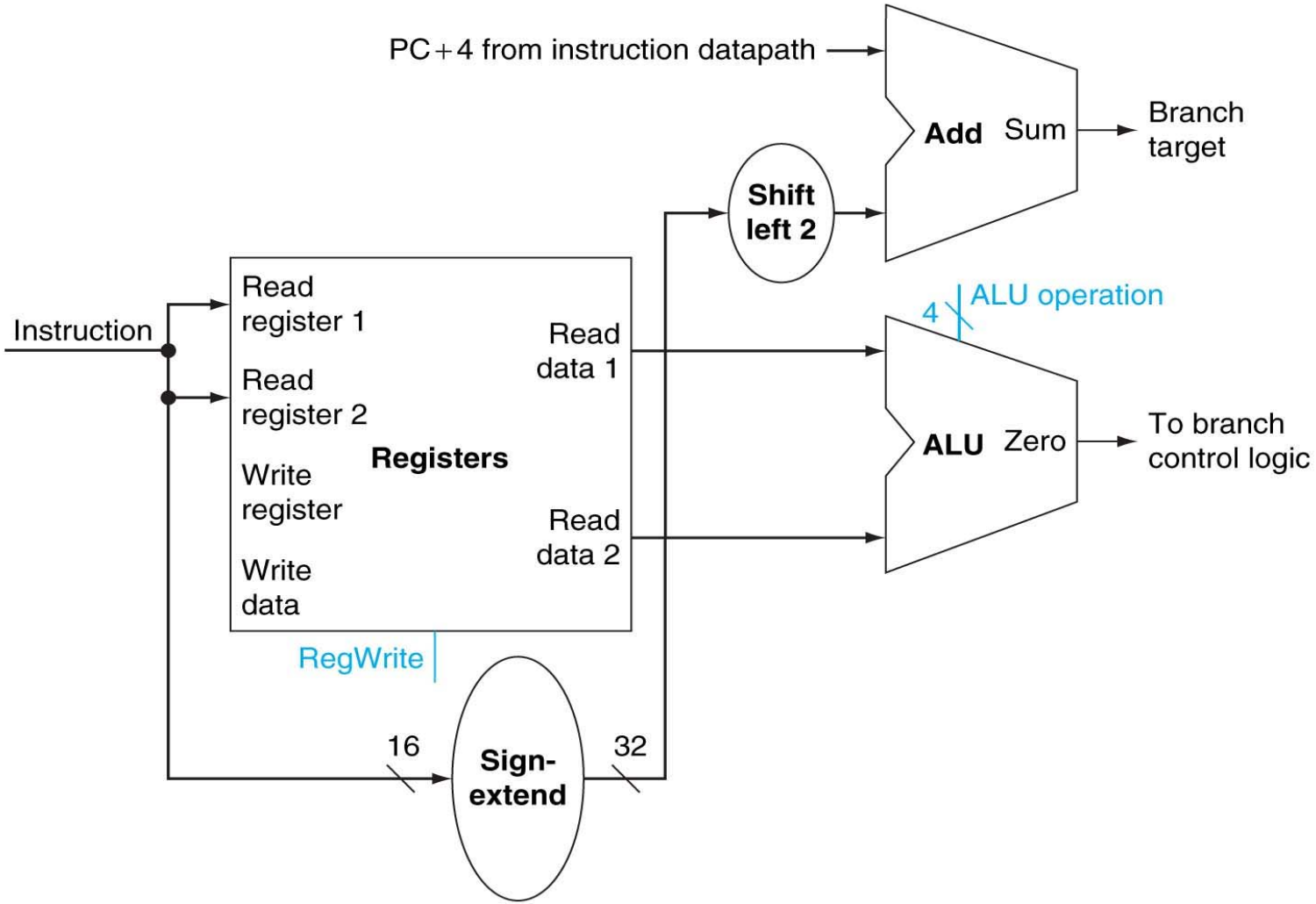
- **Executing R-format instructions and memory instructions:**



Building a Datapath (4)

- **Executing `beq` instructions (`beq $t1, $t2, 5`):**
  - Compute the **branch target address** by adding the sign-extended offset field of the instruction to the PC that was incremented by 4
    - Shift left the sign-extended offset by 2 bits as it is a word offset
  - Compare the register contents to determine whether the next instruction is the one that follows sequentially or the instruction at the branch target address
    - Send the two register operands to the ALU to subtract them
    - If the two operands are equal, the **Zero** signal out of the ALU will be asserted
    - If the condition is true (operands are equal), the branch target address becomes the new PC (**the branch is taken**)
    - If the condition is false (operands are not equal), the incremented PC replaces the current PC (**the branch is not taken**)
- **Executing `j` instructions (`j 95`):**
  - Replace the lower 28 bits of the PC with the lower 26 bits of the instruction shifted left by 2 bits (concatenate 00 to the jump offset)

Building a Datapath (4) (cont.)



- J instructions are not supported in this diagram

Creating a Single Datapath

- The simplest datapath executes all instructions in one clock cycle
- This means that no datapath resource can be used more than once per instruction
  - Any element needed more than once, must be duplicated
- We therefore need a memory for instructions separate from the one for data
- Many of the datapath elements can be shared by different instruction flows
- A multiplexor (data selector) is used to share a datapath element between two different instruction classes
  - It allows multiple connections to the input of an element
  - It has a control signal to select among the multiple inputs

Creating a Single Datapath (cont.)

- **The key difference between the three datapaths are:**
  - The second input to the ALU unit is one of two things, it is either
    - a register, if it is an R-format instruction or
    - the sign-extended lower half of the instruction, if it is a lw or sw instruction
    - **So, place a multiplexor at the second input to the ALU unit**
  - The value stored into a destination register comes from
    - the ALU, if it is an R-format instruction or
    - memory, if it is a memory load instruction
    - **So, place a multiplexor at the data input to the register file**

- A multiplexor (with a selector control signal **RegDst**) is needed to select the field of instruction used to indicate the register number to be written
  - bit positions 20:16 (rt) for `lw` or
  - bit positions 15:11 (rd) for an R-format instruction

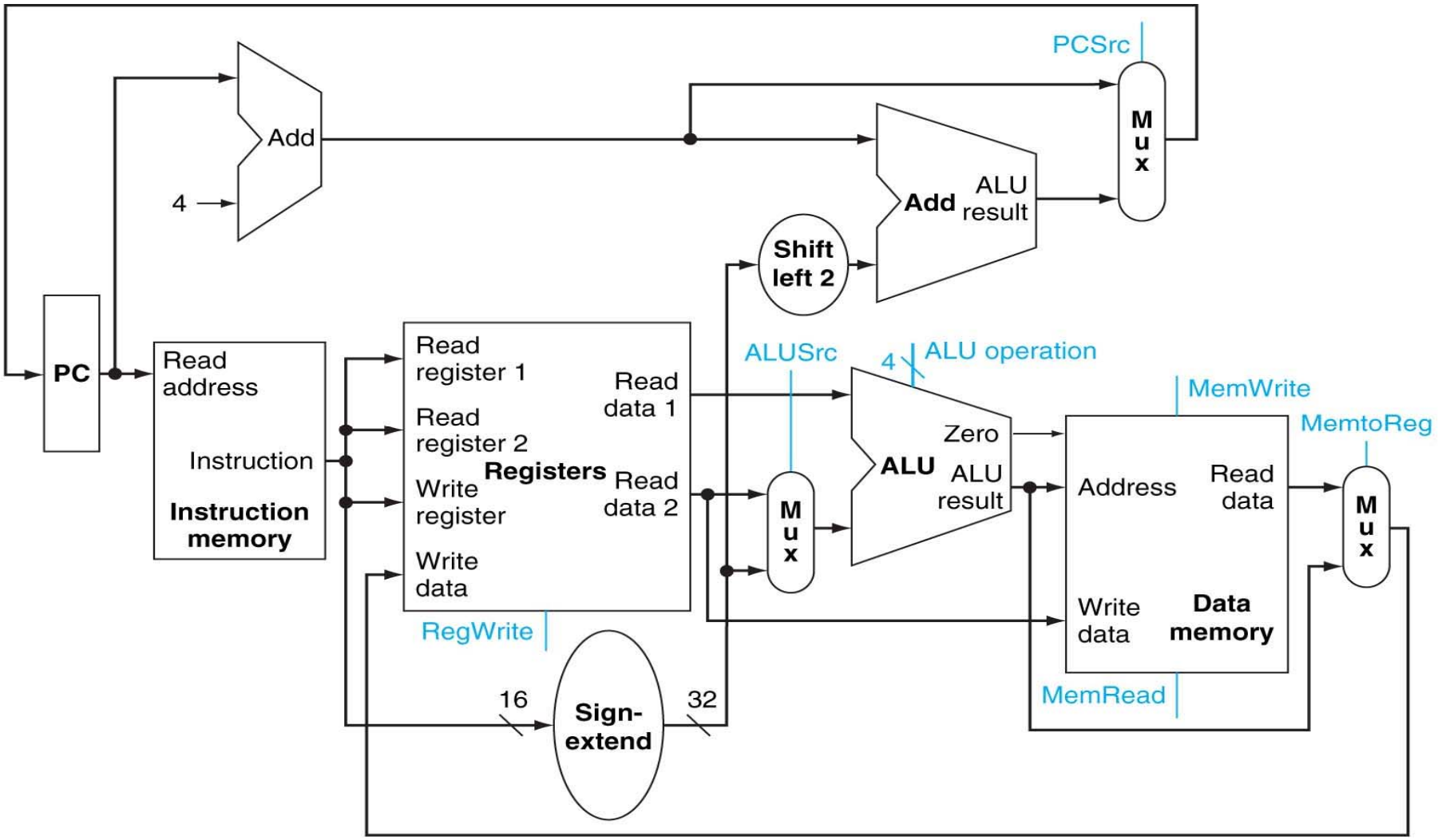
- Add the instruction fetch portion of the datapath

Creating a Single Datapath (cont.)

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

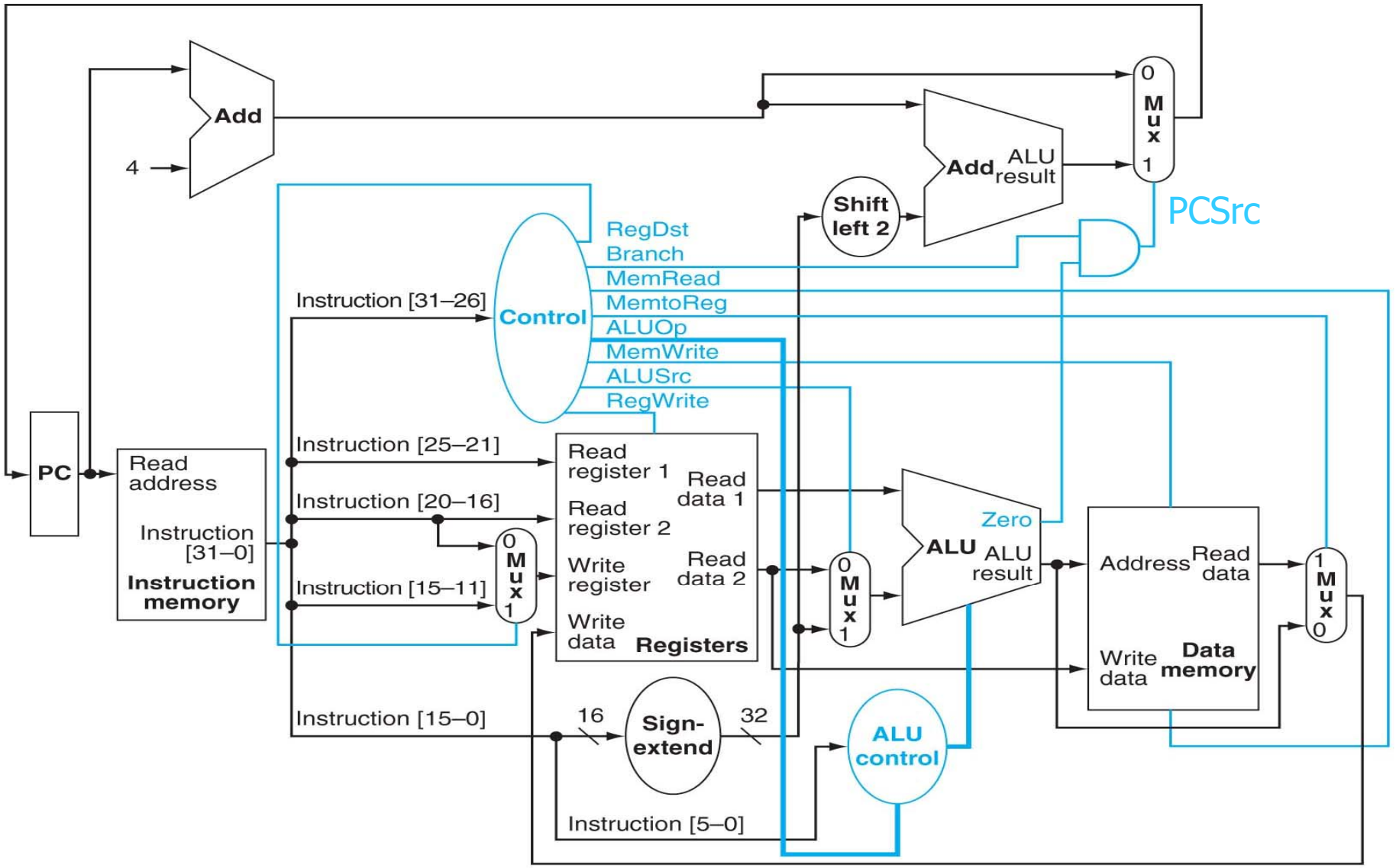
- **PCSrc** is set if the instruction is `beq` and the ALU **Zero** output is true
  - **PCSrc** is generated by ANDing together the **Branch** signal from the control unit and the **Zero** signal out of the ALU

Creating a Single Datapath (cont.)



- `J` instructions are not supported in this diagram

Creating a Single Datapath (cont.)



The ALU Control

- ALU has four control inputs

ALU control lines	Function
0000	and
0001	or
0010	add
0110	sub
0111	slt
1100	nor

- Depending on the instruction class, the ALU will need to perform one of the first five functions:
  - `lw` **and** `sw`: the ALU computes the memory address by addition
  - **R-format**: the ALU performs one of the actions (`and, or, sub, add, slt`), depending on the value of the 6-bit funct field
  - `beq`: the ALU must perform a subtraction
- **nor** is needed for other parts of the MIPS ISA

The ALU Control (cont.)

- A small ALU control unit (combinational circuit) generates the 4-bit ALU control input
  - This unit has as inputs the instruction funct field of the instruction and a 2-bit control field (ALUOp) from the main control unit
  - ALUOp indicates whether the operation to be performed should be:

add (00)	**for loads and stores**
sub (01)	**for beq**
determined by funct (10)	**for R-format**

- **Multiple levels of decoding style:** e.g., the main control unit generates the ALUOp bits, which then are used as input to the ALU control that generates the actual signals to control the ALU unit:
  - can reduce the size of the main control unit
  - may potentially increase the speed of the control unit by using several smaller control units
  - These optimizations are important, since the control unit is often performance-critical (clock cycle time)

The Main Control Unit

- The control unit takes inputs and generates a write signal for state elements, the selector control for multiplexors, and the ALU control
- The control unit uses the 6-bit opcode field, Op [5:0], to generate these control signals
- MIPS ISA regularity and simplicity means that a simple decoding process can be used to determine how to set the control lines
- Control unit implementation is combinational
- The datapath operates in a single clock cycle and the signals within the datapath can vary unpredictably during the clock cycle
- Signals stabilize in the order of the flow of information

The Main Control Unit (cont.)

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.

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

Individual Exercise (1)

- Determine whether any of the control signals in the single-cycle implementation can be eliminated and replaced by another existing control signal, or its inverse

Individual Exercise (1): Answer

- Determine whether any of the control signals in the single-cycle implementation can be eliminated and replaced by another existing control signal, or its inverse

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

  - RegDst can be replaced by ALUSrc', MemtoReg', MemRead', ALUop1
  - MemtoReg can be replaced by RegDst', ALUSrc, MemRead, ALUOp1'
  - Branch and ALUOp0 can replace each other

Individual Exercise (2)

- A friend is proposing to modify the single-cycle datapath by eliminating the control signal MemtoReg
- The multiplexor that has MemtoReg as an input will instead use either the ALUSrc or the MemRead control signal

- Will your friend's modification work?
- Can one of the two signals (ALUSrc and MemRead) substitute for the other? Explain

Individual Exercise (2): Answer

- A friend is proposing to modify the single-cycle datapath by eliminating the control signal MemtoReg. The multiplexor that has MemtoReg as an input will instead use either the ALUSrc or the MemRead control signal. Will your friend's modification work?
  - MemtoReg looks identical to both ALUSrc and MemRead signals, except for the don't care entries, which have different settings for the other signals
  - A don't care can be replaced by any signal; hence both signals can substitute for the MemtoReg signal

- Can one of the two signals (ALUSrc and MemRead) substitute for the other? Explain
  - Signals ALUSrc and MemRead differ in that sw sets ALUSrc (for address calculation) and resets MemRead (writes memory: cannot have a read and a write in the same cycle), so they cannot replace each other

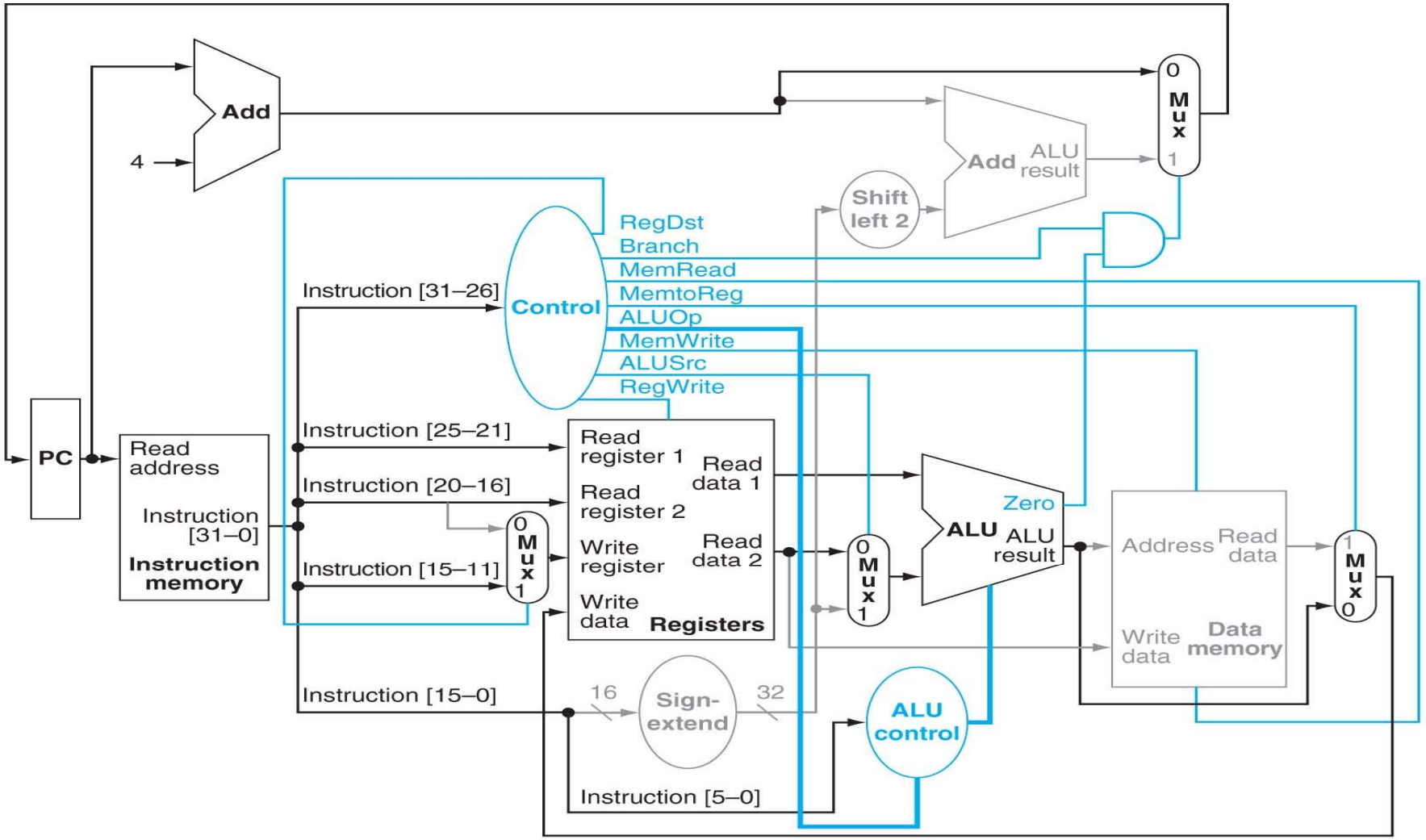
The Operation of the Datapath

- Although everything occurs in one clock cycle, we can think of four steps to execute the instructions
- These steps are ordered by the flow of information

- **Execution steps for R-format instructions:**
  1. The instruction is fetched from the instruction memory, and the PC is incremented
  2. Two register values are read from the register file using bits 25:21 and 20:16 of the instruction to select the source registers
     a) also, the main control unit computes the setting of the control lines during this step
  3. The ALU operates on the data values 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
  4. The result from the ALU is written into the register file using bits 15:11 of the instruction to select the destination register

The Operation of the Datapath (cont.)

- **Execution steps for R-format instructions (cont.)**

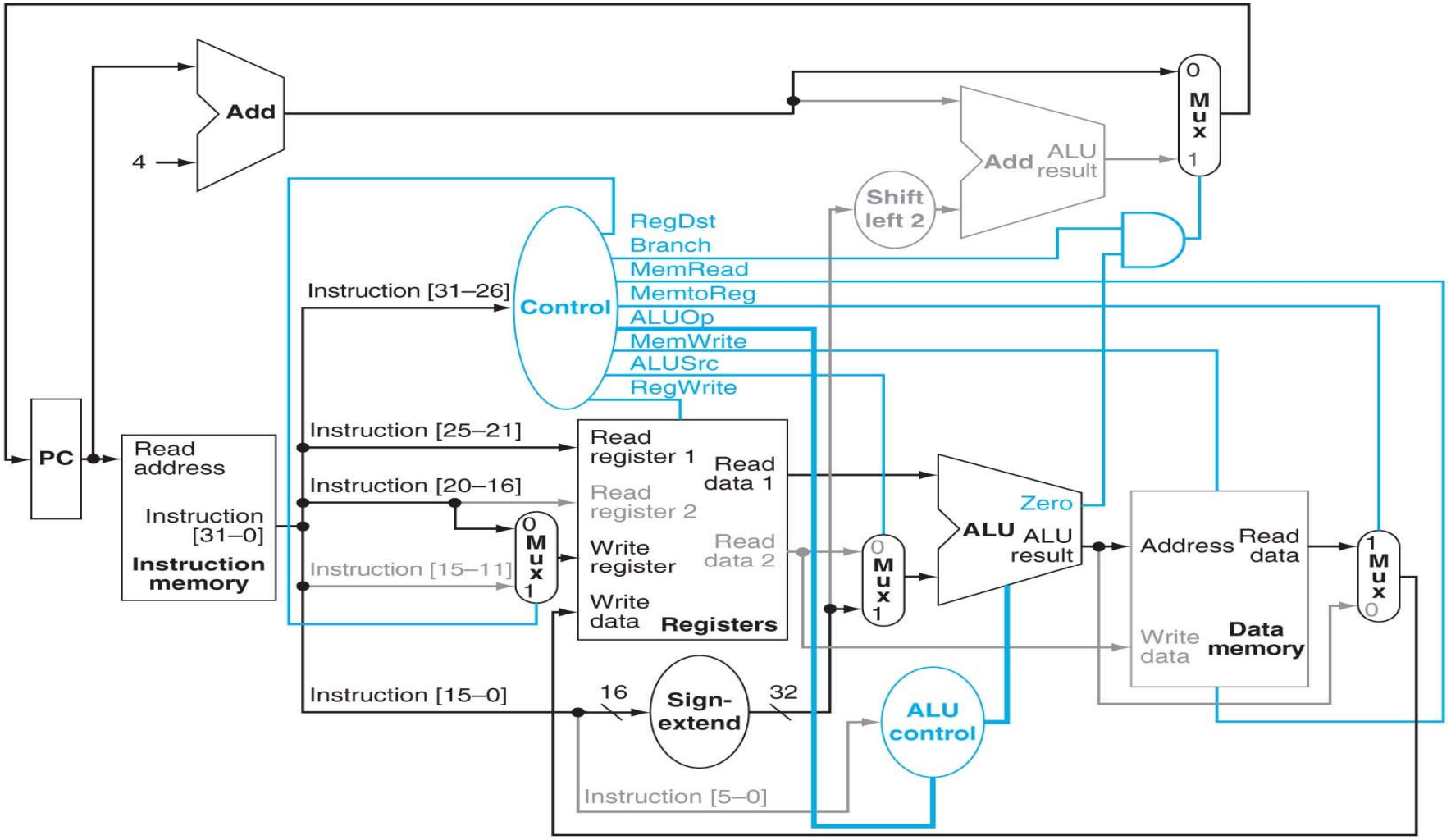


The Operation of the Datapath (cont.)

- **Execution steps for a `lw` instruction:**
  1. The instruction is fetched from the instruction memory, and the PC is incremented
  2. A register value is read from the register file using bits 25:21 of the instruction to select the source register
     a) also, the main control unit computes the setting of the control lines during this step
  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 the register file using bits 20:16 of the instruction to select the destination register

The Operation of the Datapath (cont.)

- **Execution steps for a `lw` instruction (cont.):**

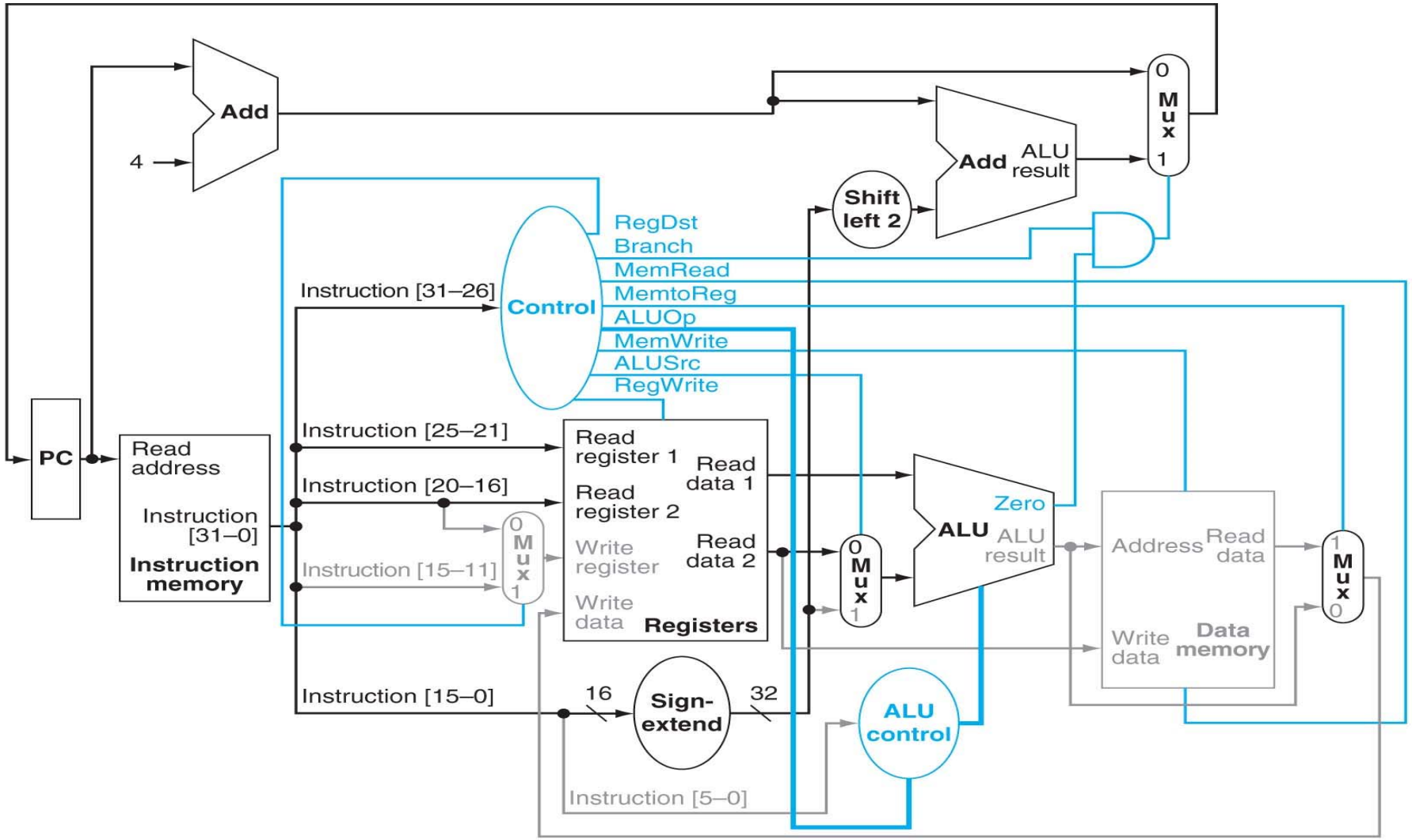


The Operation of the Datapath (cont.)

- **Execution steps for a `beq` instruction:**
  1. The instruction is fetched from the instruction memory, and the PC is incremented
  2. Two register values are read from the register file using bits 25:21 and 20:16 of the instruction to select the source registers
     a) also, the main control unit computes the setting of the control lines during this step
  3. The ALU performs a subtract on the data values read from the register file
     a) The value of PC+4 is added to the sign-extended, lower 16 bits of the instruction 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

The Operation of the Datapath (cont.)

- **Execution steps for a `beq` instruction (cont.):**



Home Exercise (3)

- Show how each of the following instructions uses the datapath

```
add     $t1, $t2, $t3
lw      $t1, 04 ($t2)
beq     $t1, $t2, 34
```

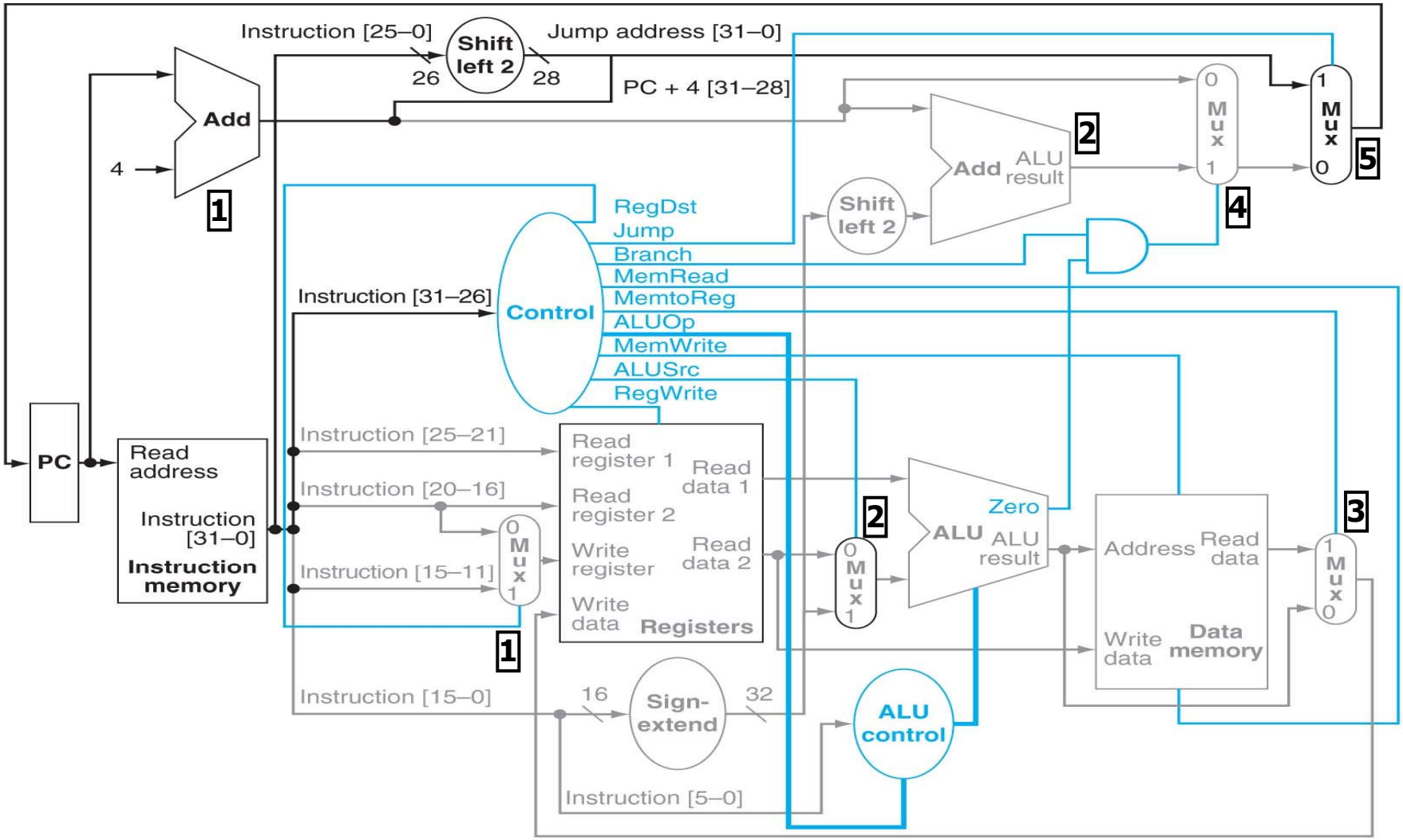
- Work out the R-format example on Page 266 and in Figure 4.19 step by step in the textbook!
- Work out the lw example on Page 267 and in Figure 4.20 step by step in the textbook!
- Work out the beq example on Page 268 and in Figure 4.21 step by step in the textbook!

Home Exercise (4)

- In the single-cycle datapath, each instruction uses a datapath element to carry out its execution
- Many of the datapath elements operate in series, using the output of another element as an input
- Some datapath elements operate in parallel
- **Show which elements operate in parallel**

Implementing Unconditional Jumps

- An additional multiplexor is used with a new control signal: **jump**



Group Exercise (5)

- We wish to add the instruction addi to the single-cycle datapath
- Add any necessary datapaths and control signals to the single-cycle datapath in Figure 4.24 (Slide 51)
- Show the necessary additions to the table in Figure 4.18 (Slide 38)

Group Exercise (5): Answer

- We wish to add the instruction addi to the single-cycle datapath
- Add any necessary datapaths and control signals to the single-cycle datapath in Figure 4.24 (Slide 51)
- Show the necessary additions to the table in Figure 4.18 (Slide 38)

- No additions to the datapath are required
- The new control is similar to
  - lw as we want to use the ALU to add the immediate to a register
  - an R-format instruction as we want to write the result of the ALU into a register and we are not branching or using memory

Instruction	RegDst	ALUSrc	Memto-Reg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0	Jump
R-format	1	0	0	1	0	0	0	1	0	0
lw	0	1	1	1	1	0	0	0	0	0
addi	0	1	0	1	0	0	0	0	0	0

Group Exercise (6)

- We wish to add the instruction sll to the single-cycle datapath
- Add any necessary datapaths and control signals to the single-cycle datapath in Figure 4.24 (Slide 51)
- Show the necessary additions to the table in Figure 4.18 (Slide 38)

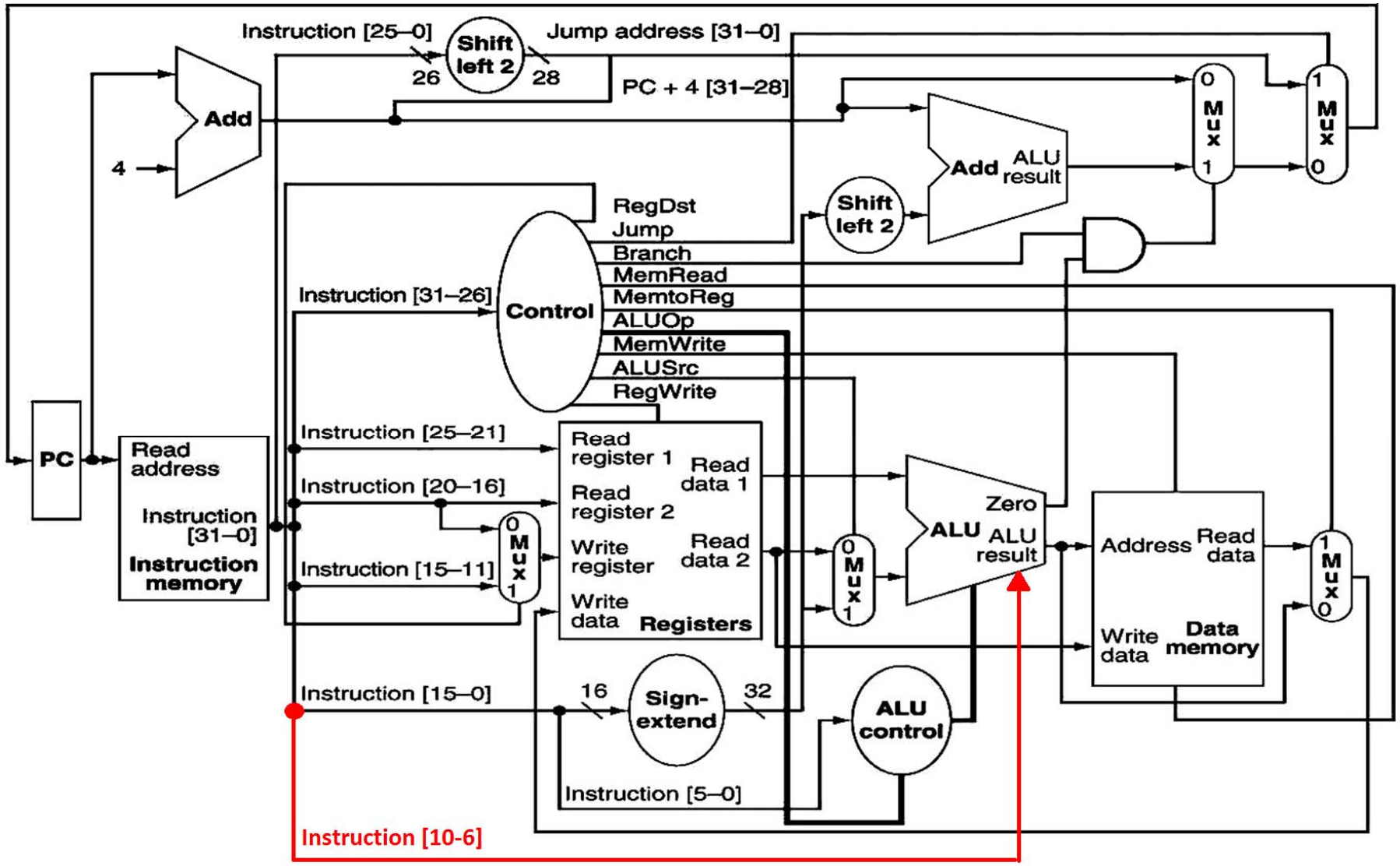
Group Exercise (6): Answer

- We wish to add the instruction sll to the single-cycle datapath
- Add any necessary datapaths and control signals to the single-cycle datapath in Figure 4.24 (Slide 51)
- Show the necessary additions to the table in Figure 4.18 (Slide 38)

- We need to add a new operation to the ALU
- We need to input the shamt field (Instruction [10:6]) to the ALU

ALUOp	Funct	ALU control	Control action
10	00 0000 (sll)	1110 (shift left logical)	Shift the second ALU operand ($rt) by the amount in the shamt field (Instruction [10-6]), input to the ALU

Group Exercise (6): Answer (cont.)



Inefficiencies in the Single-Cycle Implementation

- The clock cycle must have the same length for every instruction in the single-cycle design and the CPI will therefore be 1
- The clock cycle is determined by the **critical path**:
  - **Critical path: the longest possible path in the processor**
  - This path is almost certainly a load instruction, which uses 5 functional units in series:
    1. the instruction memory,
    2. the register file (read),
    3. the ALU,
    4. the data memory, and
    5. the register file (write)

- The clock cycle is assumed equal to the worst-case delay for all instructions
  - It is useless to try implementation techniques that reduce the delay of the common case but do not improve the worst-case cycle time
  - A single-cycle implementation thus violates the key principle of **making the common case fast**

Inefficiencies in the Single-Cycle Implementation (cont.)

- Although the CPI is 1, the overall performance of a single-cycle implementation is likely to be poor, since the clock cycle is too long and several of the instruction classes could fit in a shorter clock cycle

- If we had a processor with more powerful and complicated operations (like floating-point instructions) and addressing modes, instruction delays could vary a lot

- With this single-cycle implementation, each functional unit can be used only once per clock; therefore, some functional units must be duplicated, raising the cost of the implementation and wasting area

- So, a single-cycle implementation is inefficient in both its performance and its hardware cost!

Inefficiencies in the Single-Cycle Implementation (cont.)

- The penalty for using the single-cycle design with a fixed clock cycle is significant

- **Solutions:**
  - **Using variable length clocks**, which is extremely difficult, and the overhead of it could be larger than any advantage gained!
  - **Using implementation techniques that have a shorter clock cycle** and that require **multiple clock cycles** for each instruction. These techniques do less work every cycle and then vary the number of clock cycles for different instruction classes
  - **Use pipelining** that uses a datapath very similar to the single-cycle datapath, but is much more efficient
    - It gains efficiency by overlapping the execution of multiple instructions, increasing hardware utilization and throughput
    - It improves performance

Group Exercise (7)

- Assume that the operation times for the major functional units in the single-cycle implementation are the following:
  - 200 ps for memory access,
  - 200 ps for ALU and adders operation,
  - 100 ps for register file read or write,
  - multiplexors, control unit, PC accesses, sign extension unit, and wires have no delay

- What is the minimum clock period for an implementation in which every instruction operates in 1 clock cycle of a fixed length?

Group Exercise (7): Answer

- The critical path for the different instruction classes is as follows:

Instruction class	Functional units used by the instruction class				
Load word (lw)	Instruction fetch	Register access	ALU	Memory access	Register access
Store word (sw)	Instruction fetch	Register access	ALU	Memory access	
R-format (ALU)	Instruction fetch	Register access	ALU		Register access
Branch (beq)	Instruction fetch	Register access	ALU		
Jump (j)	Instruction fetch				

- Using these critical paths, the required clock cycle for each instruction is:

Instruction class	Instruction fetch	Register read	ALU operation	Data access	Register write	Total time
Load word (lw)	200 ps	100 ps	200 ps	200 ps	100 ps	800 ps
Store word (sw)	200 ps	100 ps	200 ps	200 ps		700 ps
R-format (ALU)	200 ps	100 ps	200 ps		100 ps	600 ps
Branch (beq)	200 ps	100 ps	200 ps			500 ps
Jump (j)	200 ps					200 ps

- The clock cycle for a processor with a single clock for all instructions will be determined by the longest instruction, which is 800 ps