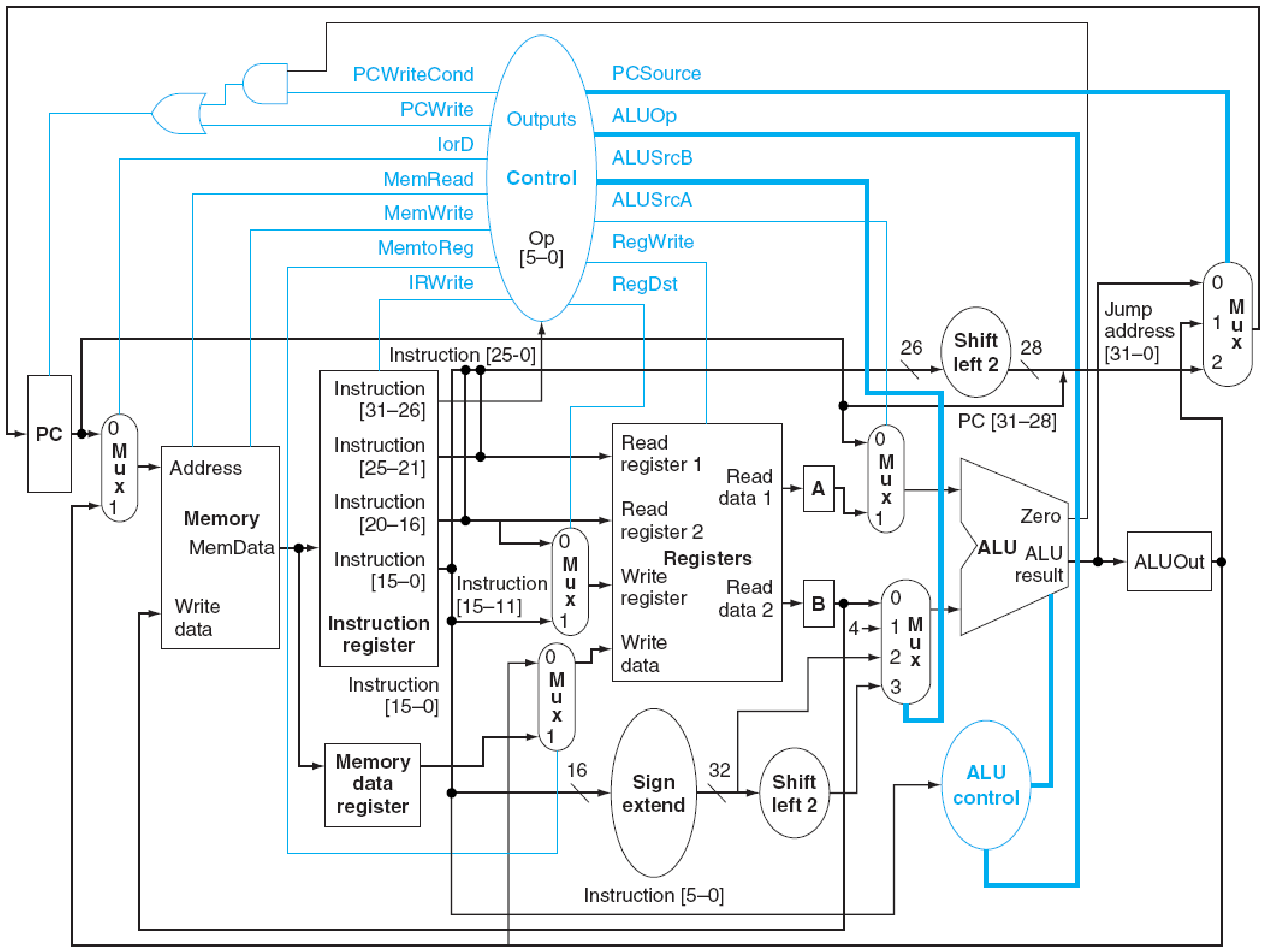
Corso di Laurea in Informatica
Architettura degli elaboratori
a.a. 2025-26

# Datapath: concetti base

Claudia Raibulet
claudia.raibulet@unimib.it

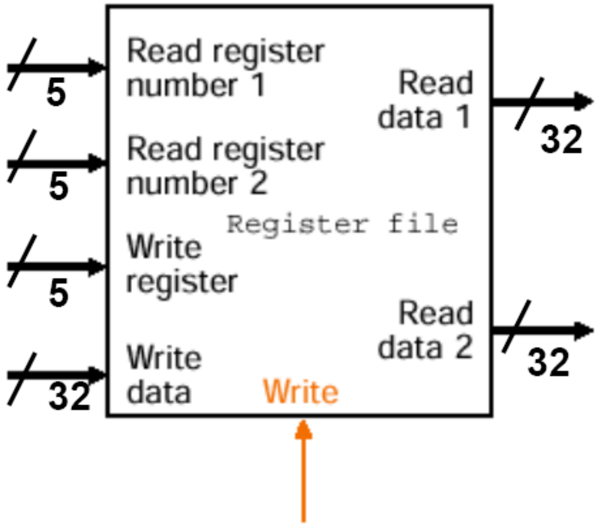

PCWriteCond
PCWrite
IorD
MemRead
MemWrite
MemtoReg
IRWrite
Outputs
Control
Op [5–0]
PCSource
ALUOp
ALUSrcB
ALUSrcA
RegWrite
RegDst
PC
0 Mux 1
Address
Memory
MemData
Write data
Instruction [25-0]
Instruction [31–26]
Instruction [25–21]
Instruction [20–16]
Instruction [15–0]
Instruction register
Instruction [15–11]
Instruction [15–0]
Memory data register
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
A
B
4
0 1 2 3 Mux
16
Sign extend
32
Shift left 2
26
Shift left 2
28
PC [31–28]
Jump address [31–0]
0 1 2 Mux
Zero
ALU
ALU result
ALUOut
ALU control
Instruction [5–0]

## Datapath

- Consideriamo un processore MIPS che esegue le seguenti istruzioni:
  - Aritmetico-logiche (arithmetic-logical instruction): add, sub, and, or, slt
  - Interazione con la memoria (memory-reference instruction): lw, sw
  - Salto (control-flow instruction): beq, j

## Datapath

- **Abbiamo gia' visto il Register File & ALU**



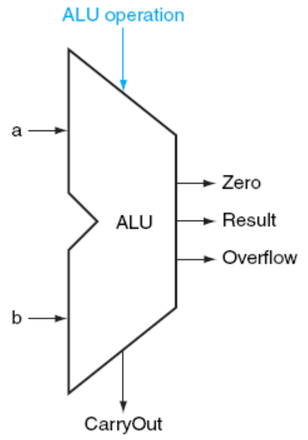
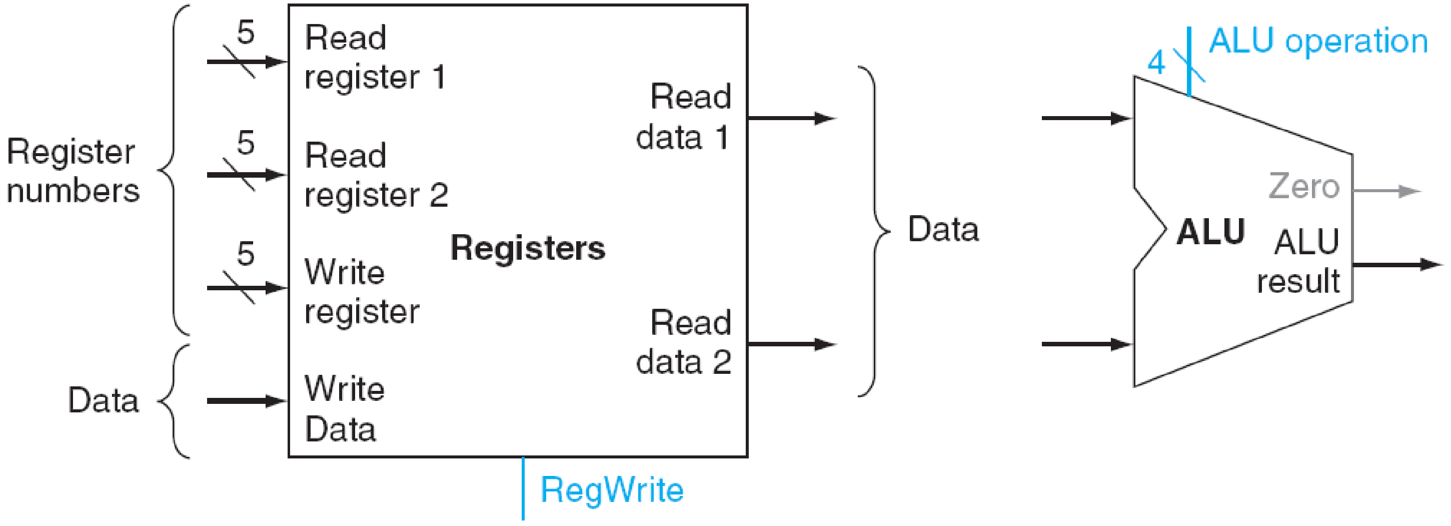


**FIGURE C.5.14 The symbol commonly used to represent an ALU, as shown in Figure C.5.12.** This symbol is also used to represent an adder, so it is normally labeled either with ALU or Adder.



a. Registers

b. ALU

## Realizzare un datapath

- Si stabilisce il set di istruzioni da implementare
- Si identificano i componenti del datapath (ALU, register file, ecc)
- Si stabilisce la metodologia di clocking
- Si assembla il datapath e si identifcano i segnali di controllo
- Si analizza l'implementazione di ogni istruzione per determinare il setting dei segnali di controllo
- Si assembla la logica di controllo

## Passi per l'esecuzione di una istruzione

- **Fetch**:
  - Legge l'istruzione dalla memoria e la salva in un registro dedicato (Instruction Register)
  - L'indirizzo di memoria che indica l'istruzione da leggere si trova nel registro Program Counter (PC)
  - Dopo la lettura dell'istruzione in PC viene incrementato di 4 (usando l'ALU) per indicare la prossima istruzione da leggere
- **Decode**:
  - Decodifica i vari campi dell'istruzione per decidere quali sono i passi necessari per la sua esecuzione
- **Execute**:
  - Esegue i passi necessari per eseguire l'istruzione

## Fetch (I)

- **Cosa serve per implementare il fetch?**

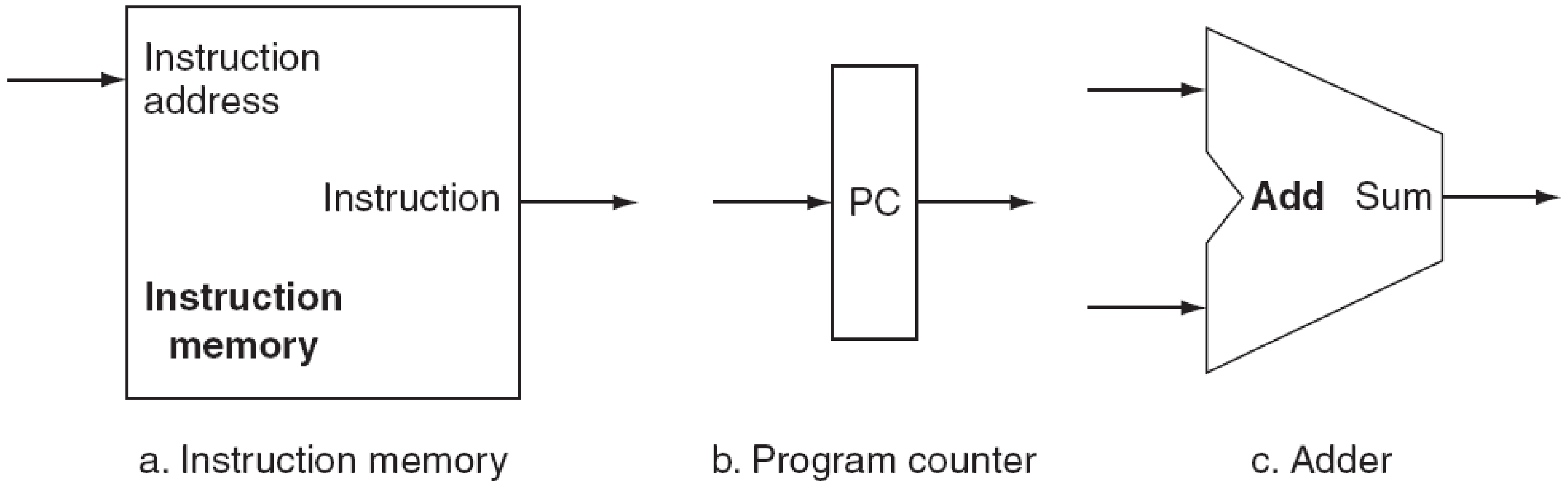


**FIGURE 5.5 Two state elements are needed to store and access instructions, and an adder is needed to compute the next instruction address.** The state elements are the instruction memory and the program counter. The instruction memory need only provide read access because the datapath does not write instructions. Since the instruction memory only reads, we treat it as combinational logic: the output at any time reflects the contents of the location specified by the address input, and no read control signal is needed. (We will need to write the instruction memory when we load the program; this is not hard to add, and we ignore it for simplicity.) The program counter is a 32-bit register that will be written at the end of every clock cycle and thus does not need a write control signal. The adder is an ALU wired to always perform an add of its two 32-bit inputs and place the result on its output.

## Fetch (II)

- **Calcolo: PC+4**

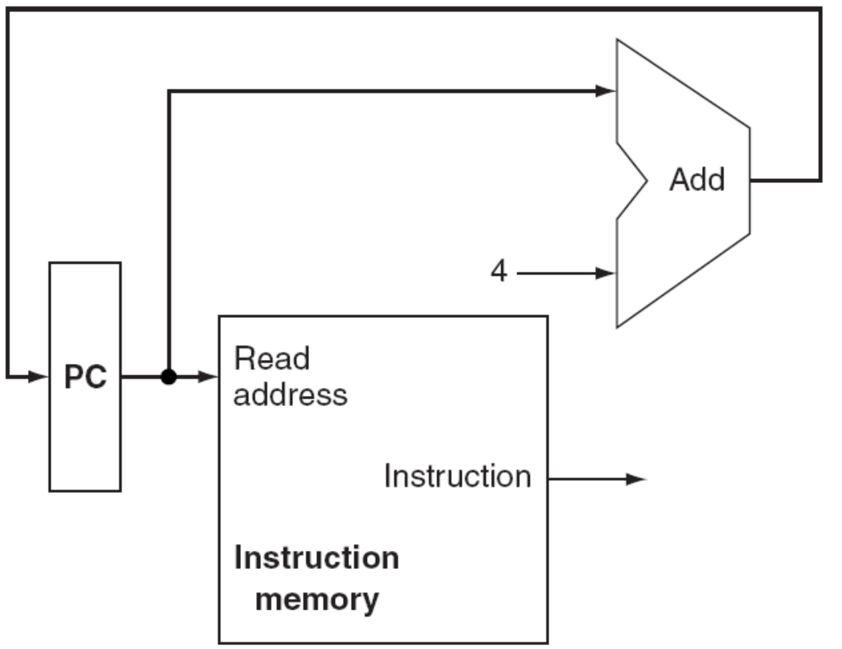


**FIGURE 5.6 A portion of the datapath used for fetching instructions and incrementing the program counter.** The fetched instruction is used by other parts of the datapath.

## Decode

- **Il processore MIPS legge i vari campi dell’istruzione**
- **Il processore identifica il tipo di istruzione da eseguire (usando OPCODE; se necessario usa anche FUNC CODE)**

# Execute: R-Type

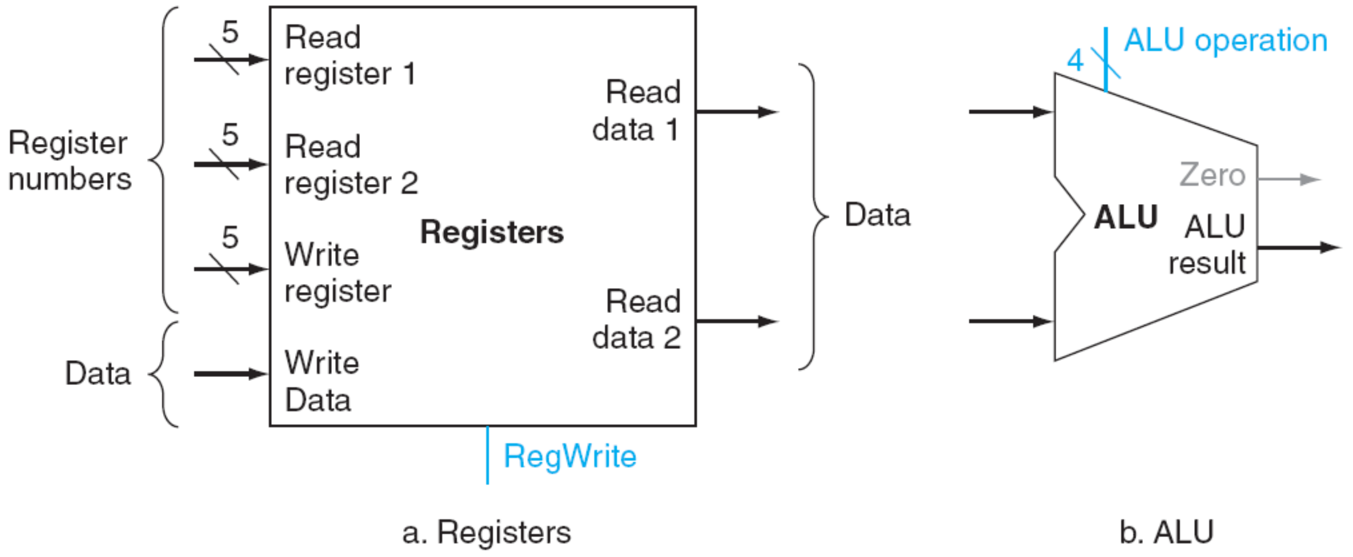


**FIGURE 5.7 The two elements needed to implement R-format ALU operations are the register file and the ALU.** The register file contains all the registers and has two read ports and one write port. The design of multiported register files is discussed in Section B.8 of Appendix B. The register file always outputs the contents of the registers corresponding to the Read register inputs on the outputs; no other control inputs are needed. In contrast, a register write must be explicitly indicated by asserting the write control signal. Remember that writes are edge-triggered, so that all the write inputs (i.e., the value to be written, the register number, and the write control signal) must be valid at the clock edge. Since writes to the register file are edge-triggered, our 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, while the value written will be available to a read in a subsequent clock cycle. The inputs carrying the register number to the register file are all 5 bits wide, whereas the lines carrying data values are 32 bits wide. The operation to be performed by the ALU is controlled with the ALU operation signal, which will be 4 bits wide, using the ALU designed in Appendix B. We will use the Zero detection output of the ALU shortly to implement branches. The overflow output will not be needed until Section 5.6, when we discuss exceptions; we omit it until then.

## Execute: load & store

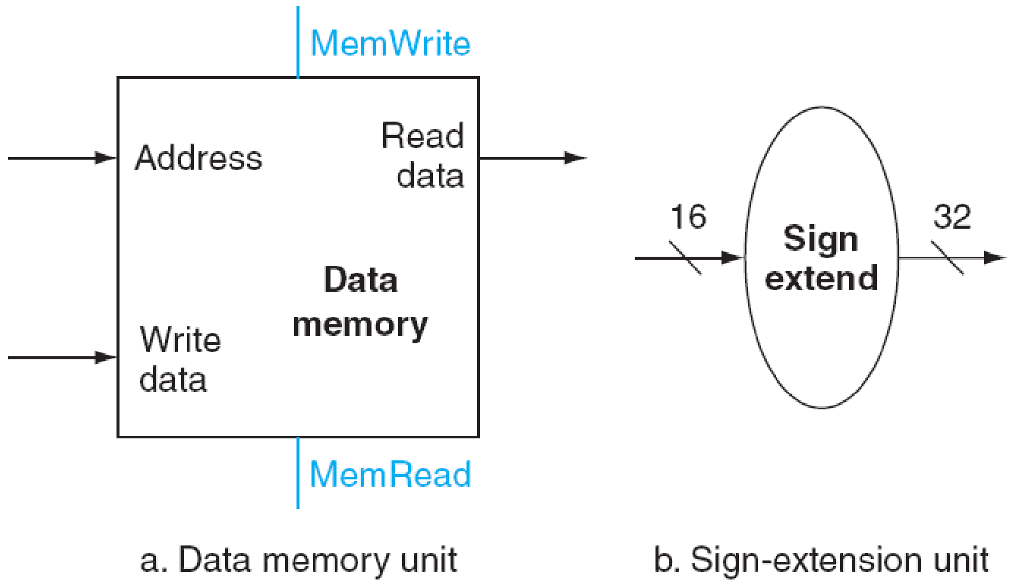


**FIGURE 5.8 The two units needed to implement loads and stores, in addition to the register file and ALU of Figure 5.7, are the data memory unit and the sign extension unit.** The memory unit is a state element with inputs for the address and the write data, and a single output for the read result. 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, as we will see in Chapter 7. The sign extension unit has a 16-bit input that is sign-extended into a 32-bit result appearing on the output (see Chapter 3). We assume the data memory is edge-triggered for writes. Standard memory chips actually have a write enable signal that is used for writes. Although the write enable is not edge-triggered, our edge-triggered design could easily be adapted to work with real memory chips. See Section B.8 of Appendix B for a further discussion of how real memory chips work.

# Execute: beq

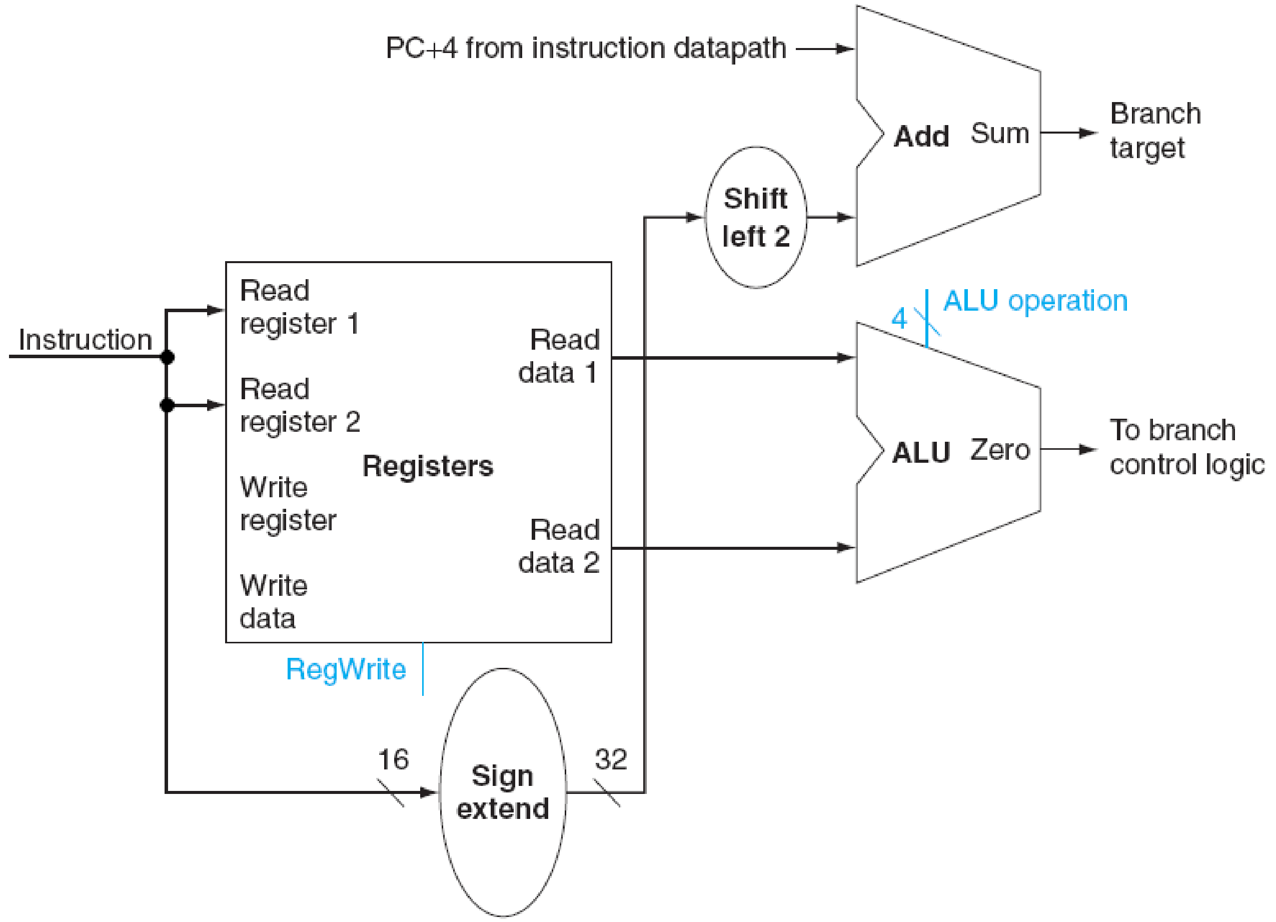


**FIGURE 5.9 The datapath for a branch uses the ALU to evaluate the branch condition and a separate adder to compute the branch target as the sum of the incremented PC and the sign-extended, lower 16 bits of the instruction (the branch displacement), shifted left 2 bits.** The unit labeled *Shift left 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 offset field; no actual shift hardware is needed, since the amount of the "shift" is constant. Since we know that the offset was sign-extended from 16 bits, the shift will throw away only "sign bits." Control logic is used to decide whether the incremented PC or branch target should replace the PC, based on the Zero output of the ALU.

# Una vista astratta del datapath

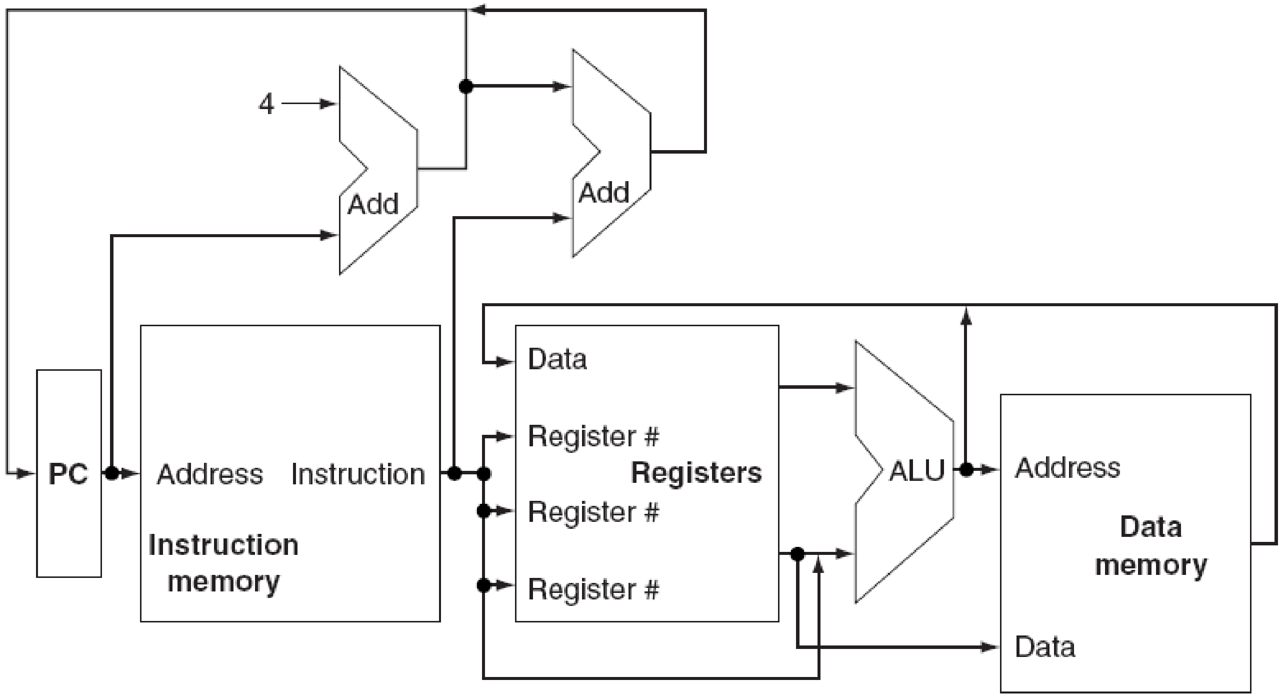


**FIGURE 5.1 An abstract view of the implementation of the MIPS subset showing the major functional units and the major connections between them.** All instructions start by using the program counter to supply the instruction address to the instruction memory. After the instruction is fetched, the register operands used by an instruction are specified by fields of that instruction. Once the register operands have been fetched, they can be operated on to compute a memory address (for a load or store), to compute an arithmetic result (for an integer arithmetic-logical instruction), or a compare (for a branch). If the instruction is an arithmetic-logical instruction, the result from the ALU must be written to a register. If the operation is a load or store, the ALU result is used as an address to either store a value from the registers or load a value from memory into the registers. The result from the ALU or memory is written back into the register file. Branches require the use of the ALU output to determine the next instruction address, which comes from either the ALU (where the PC and branch offset are summed) or from an adder that increments the current PC by 4. The thick lines interconnecting the functional units represent buses, which consist of multiple signals. The arrows are used to guide the reader in knowing how information flows. Since signal lines may cross, we explicitly show when crossing lines are connected by the presence of a dot where the lines cross.

# Multiplexer: per integrare i vari componenti (I)

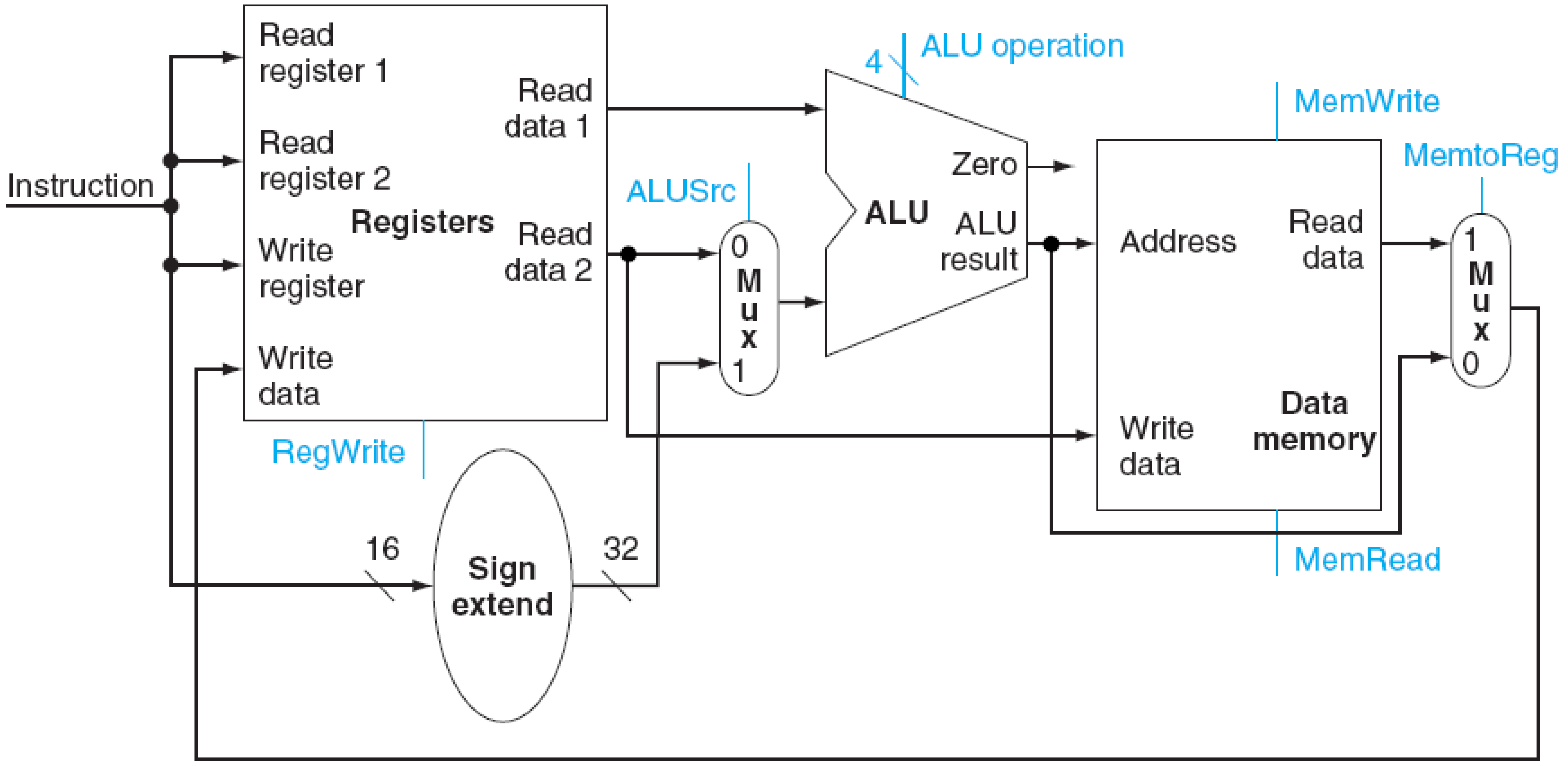


**FIGURE 5.10 The datapath for the memory instructions and the R-type instructions.** This example shows how a single datapath can be assembled from the pieces in Figures 5.7 and 5.8 by adding multiplexors. Two multiplexors are needed, as described as in the example.

## Multiplexer: per integrare i vari componenti (II)

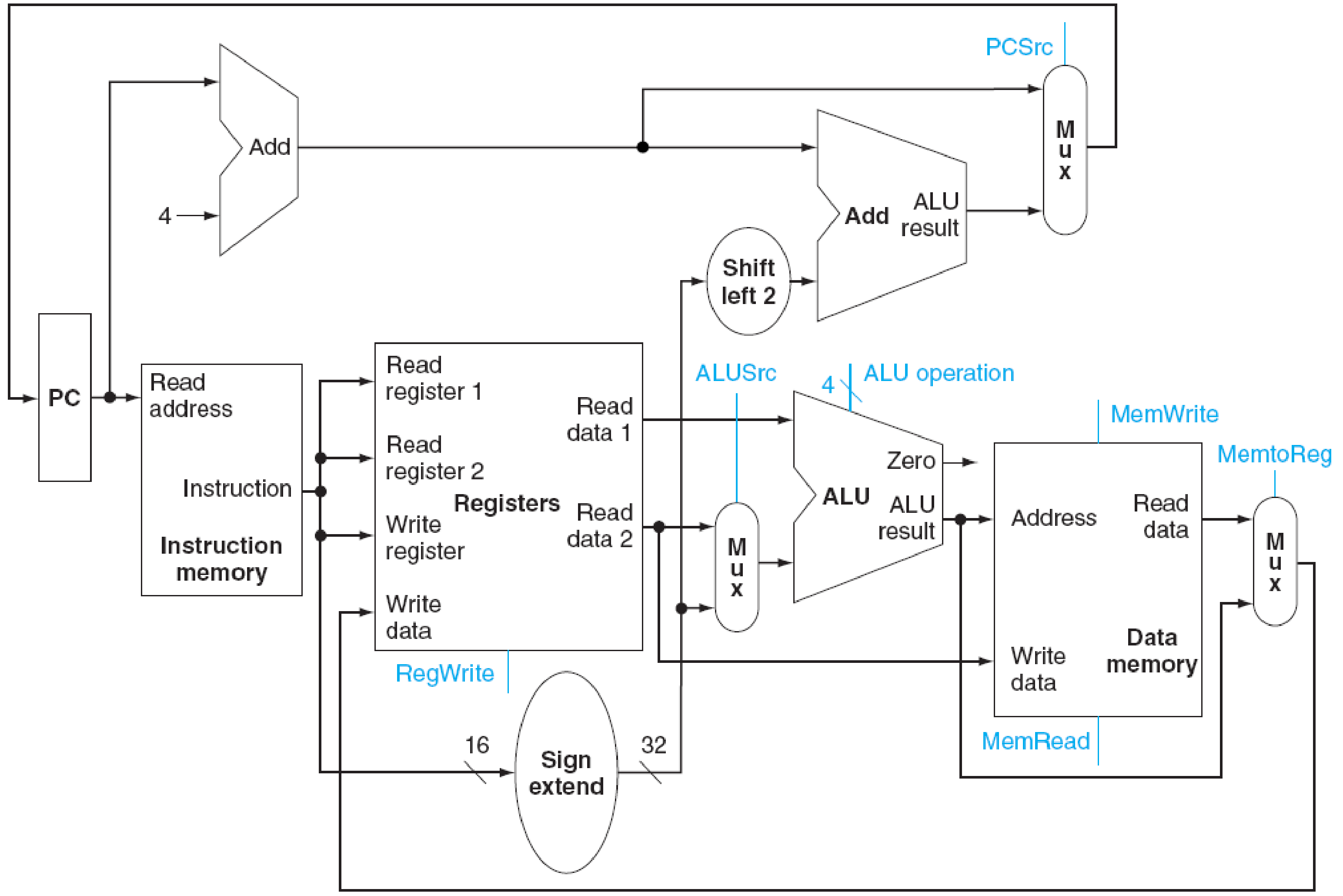


**FIGURE 5.11 The simple datapath for the MIPS architecture combines the elements required by different instruction classes.** This datapath can execute the basic instructions (load/store word, ALU operations, and branches) in a single clock cycle. An additional multiplexor is needed to integrate branches. The support for jumps will be added later.

## Metodologia di clocking

- **Singolo ciclo**
  - Ciclo singolo di lunghezza fissa uguale al tempo necessario per eseguire l'istruzione piu' lunga
  - Ogni istruzione viene eseguita in un ciclo di clock
  - Svantaggi: spreco di tempo

- **Multi-ciclo**
  - Ciclo di lunghezza fissa piu' corto
  - Ogni istruzione viene eseguita in piu' cicli di clock
  - Istruzioni di tipo diverso – eseguite in un numero di cicli di clock diverso

## Da leggere

- Chapter 5: The processor: Datapath and Control – disponibile sul sito elearning del corso