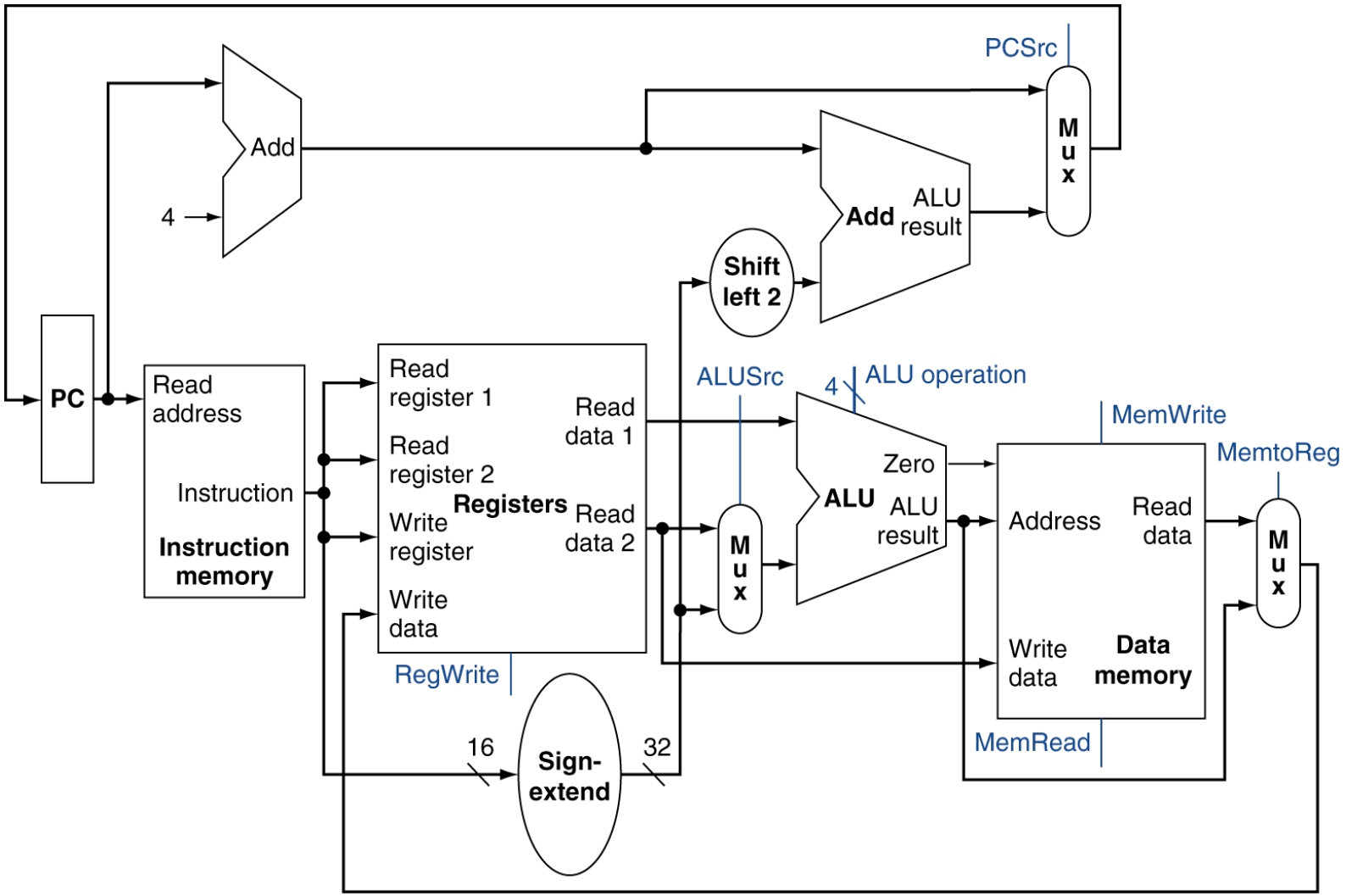
CCSYA

The Processor
Single-cycle Implementation
Part II

Departamento de Engenharia Informática
Instituto Superior de Engenharia do Porto

Luís Nogueira (lmn@isep.ipp.pt)

Datapath – Last Lecture

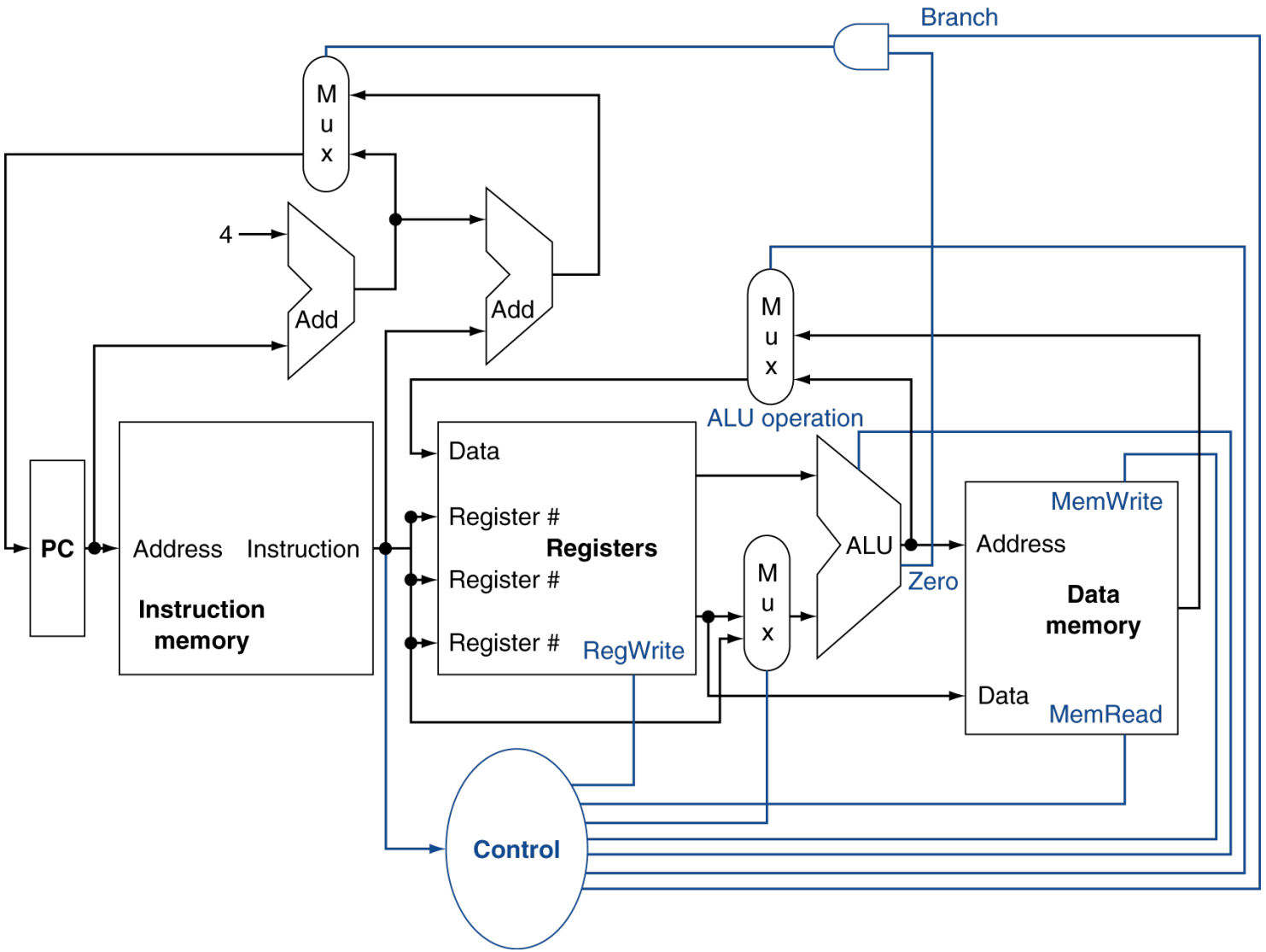


The Control Unit

- **Need to ensure that:**
  - Next instruction only begins when the current one is finished
  - ALU executes the correct operation
  - During a write in a register there is no read in that register
  - During a write in memory there is no read in memory
  - PC is set to the next instruction or jumps
  - In a `lw`/`sw` the ALU generates the memory address
  - After a arithmetic-logical instruction the result is written in a register

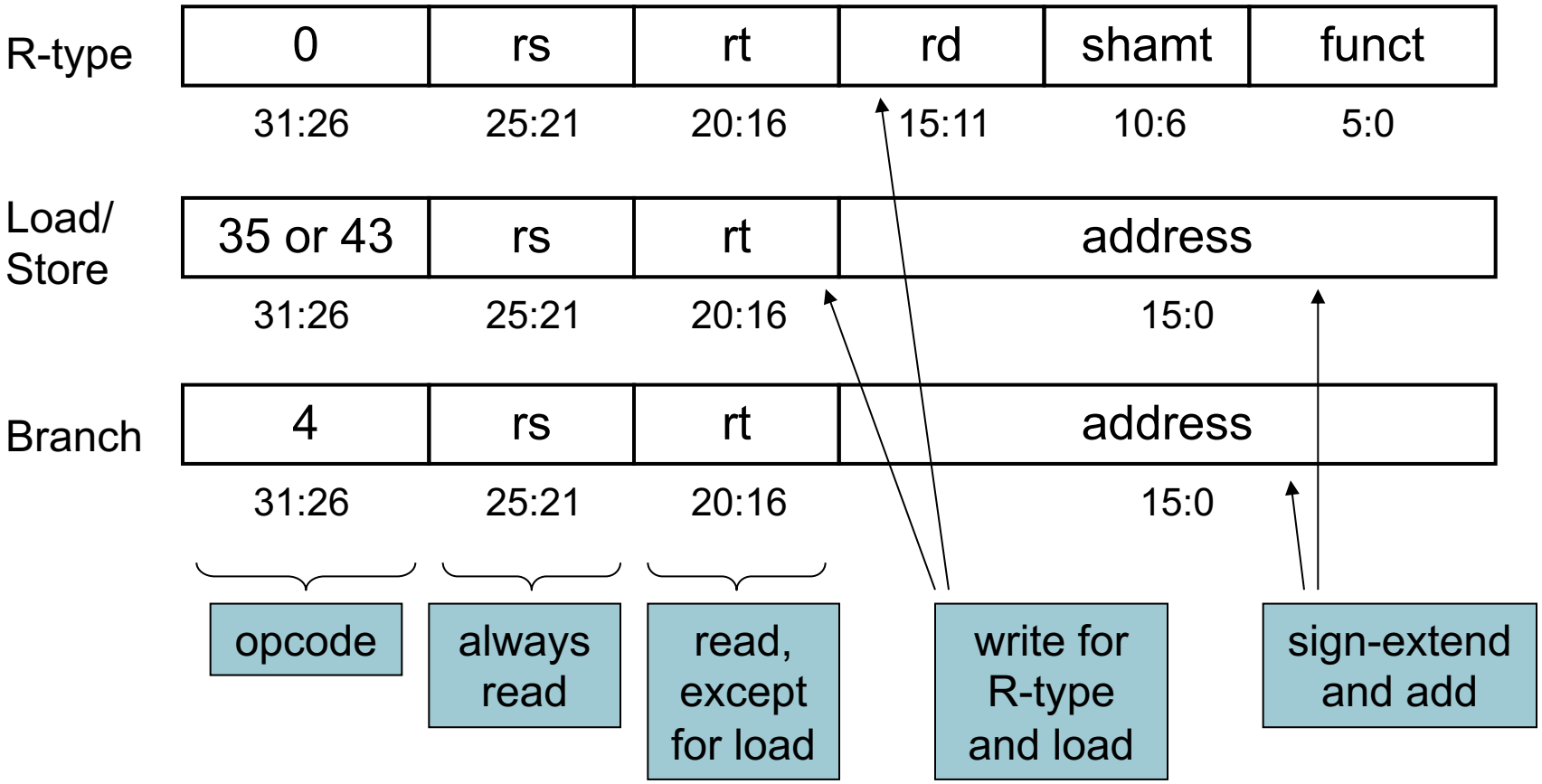
- **The control unit must be able to:**
  - Take inputs from the instruction to be executed
  - Generate a write signal for **each state element**, the selector control for **each multiplexor**, and the **ALU control**

The Control Unit



The Control Unit

- **The input to the control unit is the 6-bit opcode field from the instruction**



The Control Unit

- **The outputs of the control unit consist of:**
  - Three 1-bit signals that are used to control multiplexors (**RegDst**, **ALUSrc**, and **MemtoReg**)
  - Three 1-bit signals for controlling reads and writes in the register file and data memory (**RegWrite**, **MemRead**, and **MemWrite**)
  - A 1-bit signal used in determining whether to possibly branch (**Branch**)
  - A 2-bit control signal for the ALU (**ALUOp**)

- **An AND gate is used to control the selection of the next PC**
  - Combining the branch control signal and the Zero output from the ALU

1-bit 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 ALU Control

- **Depending on the instruction class, the ALU will need to perform**
  - For `lw`/`sw` instructions, compute the memory address by addition
  - For the R-type instructions, one of the five actions (`and`, `or`, `sub`, `add`, `slt`), depending on the value of the 6-bit *funct* field in the low-order bits of the instruction
  - For a `beq` instruction, a subtraction

- **4-bit ALU control that has as inputs the *funct* field of the instruction and a 2-bit control field (ALUOp)**
  - ALUOp indicates whether the operation to be performed should be add (00) for `lw`/`sw`, subtract (01) for `beq`, or determined by the operation encoded in the *funct* field (10)

The ALU Control

- **How the ALU control bits are set depends on the ALUOp control bits and the different function codes for the R-type instruction**
  - 2-bit ALUOp derived from opcode

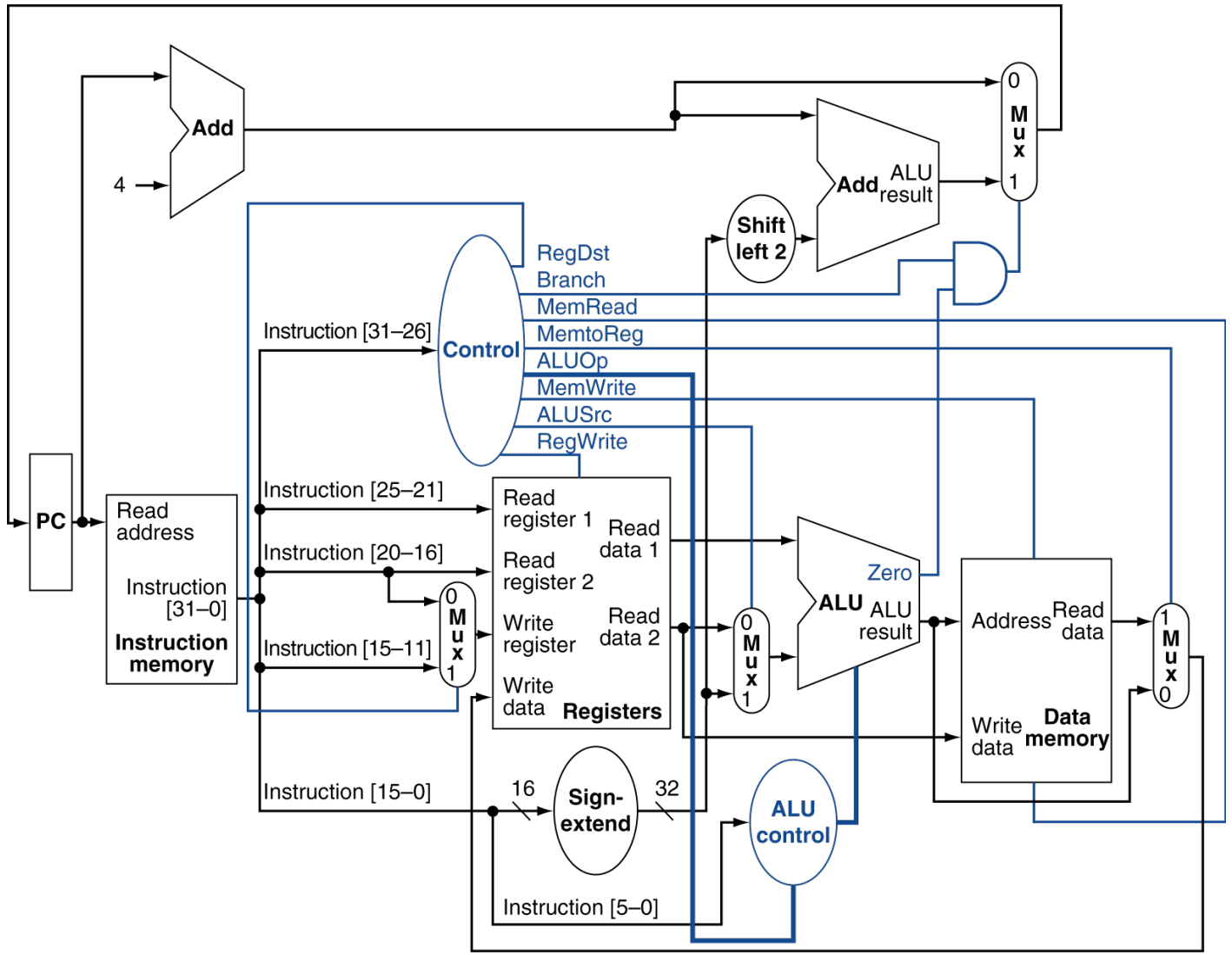
Instruction opcode	ALUOp	Instruction operation	Funct field	Desired ALU action	ALU control input
LW	00	load word	XXXXXX	add	0010
SW	00	store word	XXXXXX	add	0010
Branch equal	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
R-type	10	subtract	100010	subtract	0110
R-type	10	AND	100100	AND	0000
R-type	10	OR	100101	OR	0001
R-type	10	set on less than	101010	set on less than	0111

Control Lines Setting

- **The setting of the control lines is completely determined by the opcode fields of the instruction**
  - Each control signal can be 0, 1, or don't care (X) for each of the opcode values

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

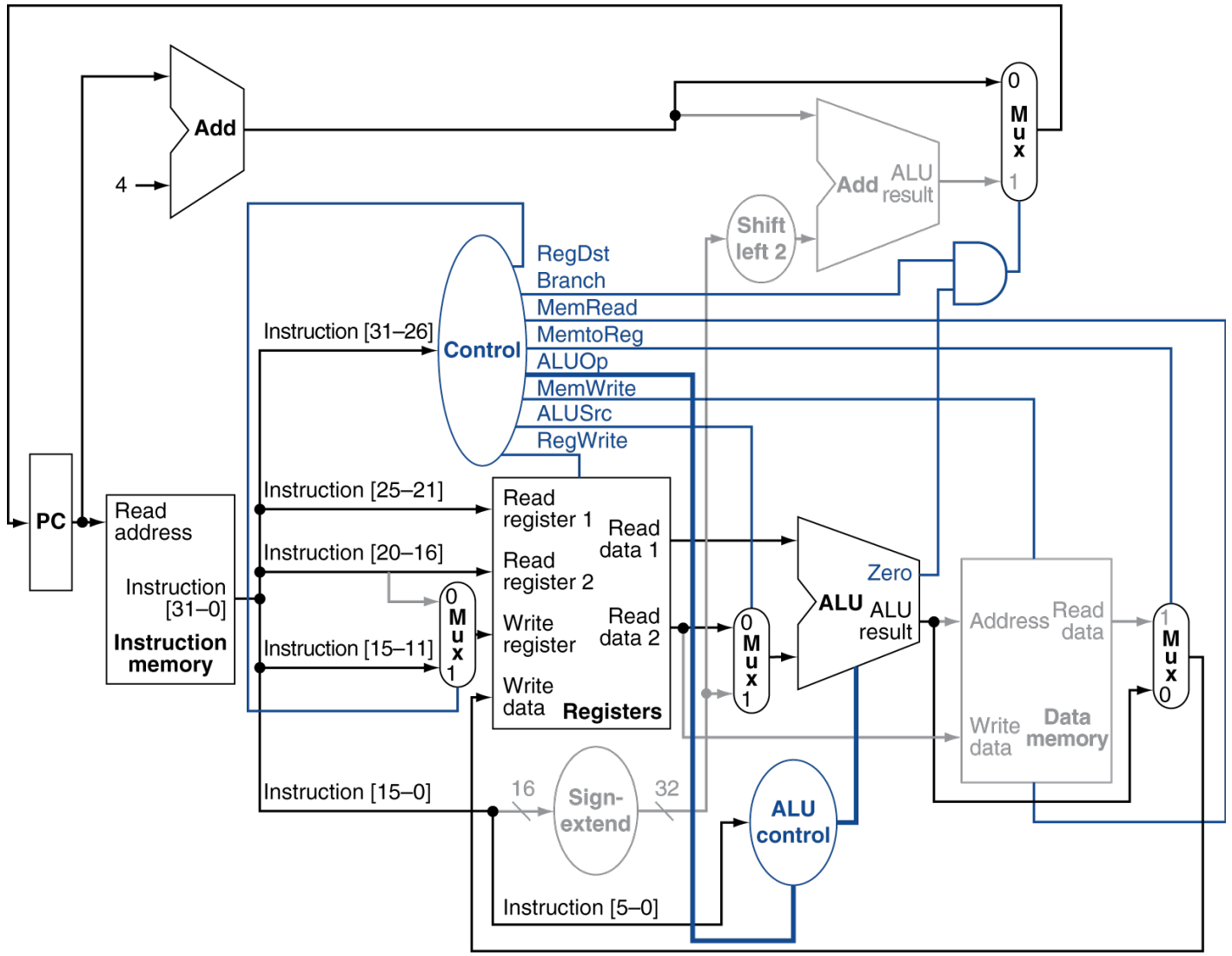
Datapath With Control



Datapath – R-Type Instructions

- **We can think of a R-type instruction (e.g., `add $t1, $t2, $t3`) as operating in four steps:**
  1. The **instruction is fetched** and the **PC is incremented**
  2. The **main control unit computes the setting** of the control lines and **two registers are read** (`$t2` and `$t3`) from the register file
  3. The **ALU operates on the data** read from the register file using the function code (bits 5:0 of the instruction)
  4. The **result from the ALU is written into the register file** using bits 15:11 of the instruction to select the destination register (`$t1`)

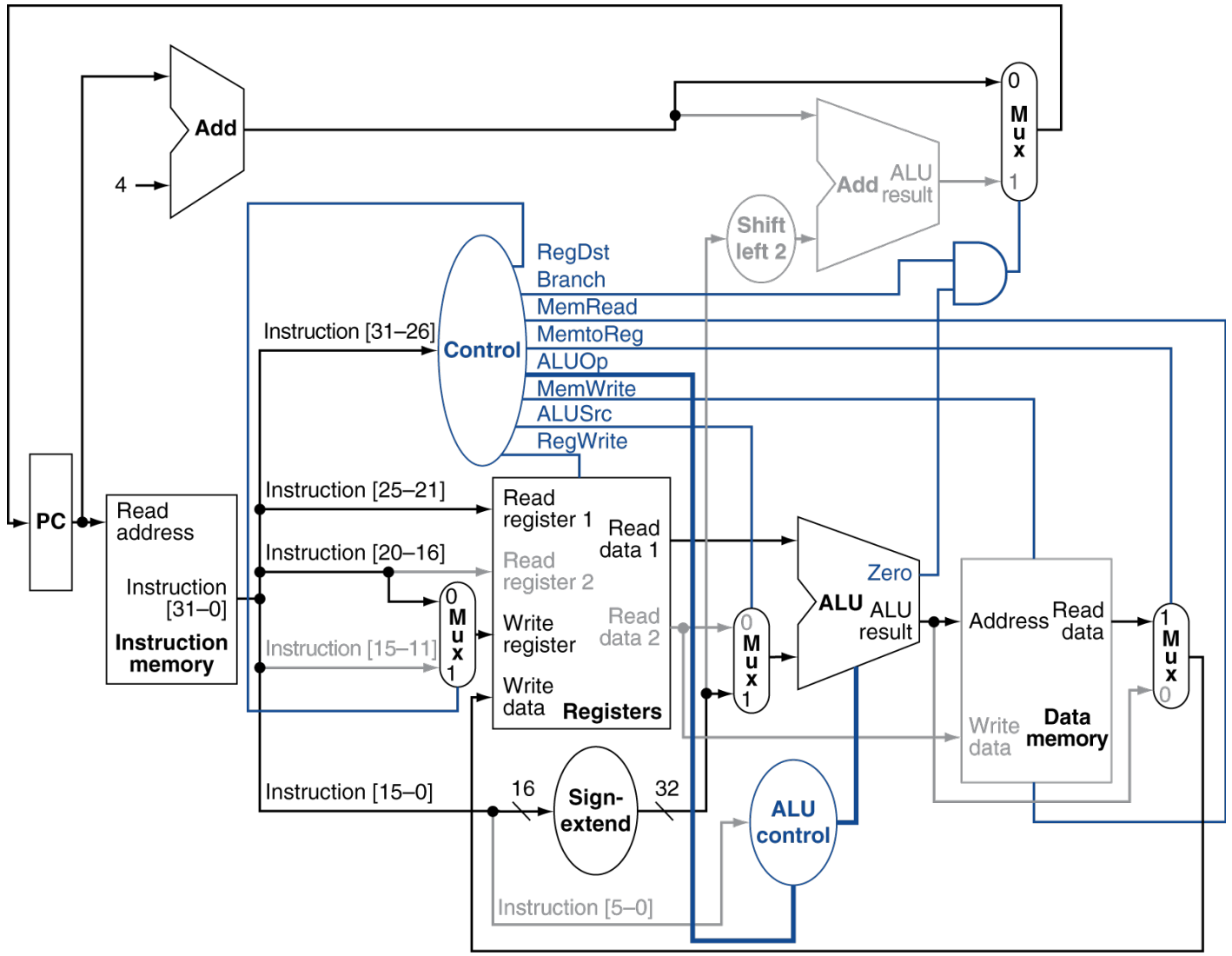
Datapath – R-Type Instructions



Datapath – Load Instruction

- **We can think of a load instruction (e.g., `lw $t1, offset($t2)`) as operating in five steps:**
  1. The **instruction is fetched** and the **PC is incremented**
  2. The **main control unit computes the setting** of the control lines and **one register is read** (`$t2`) from the register file
  3. The **ALU computes the sum** of the value read from the register file and the sign-extended lower 16 bits of the instruction (offset)
  4. The **result from the ALU is used as the address** for the data memory
  5. The **data from the memory unit is written into the register file** using bits 20:16 of the instruction to select the destination register (`$t1`)

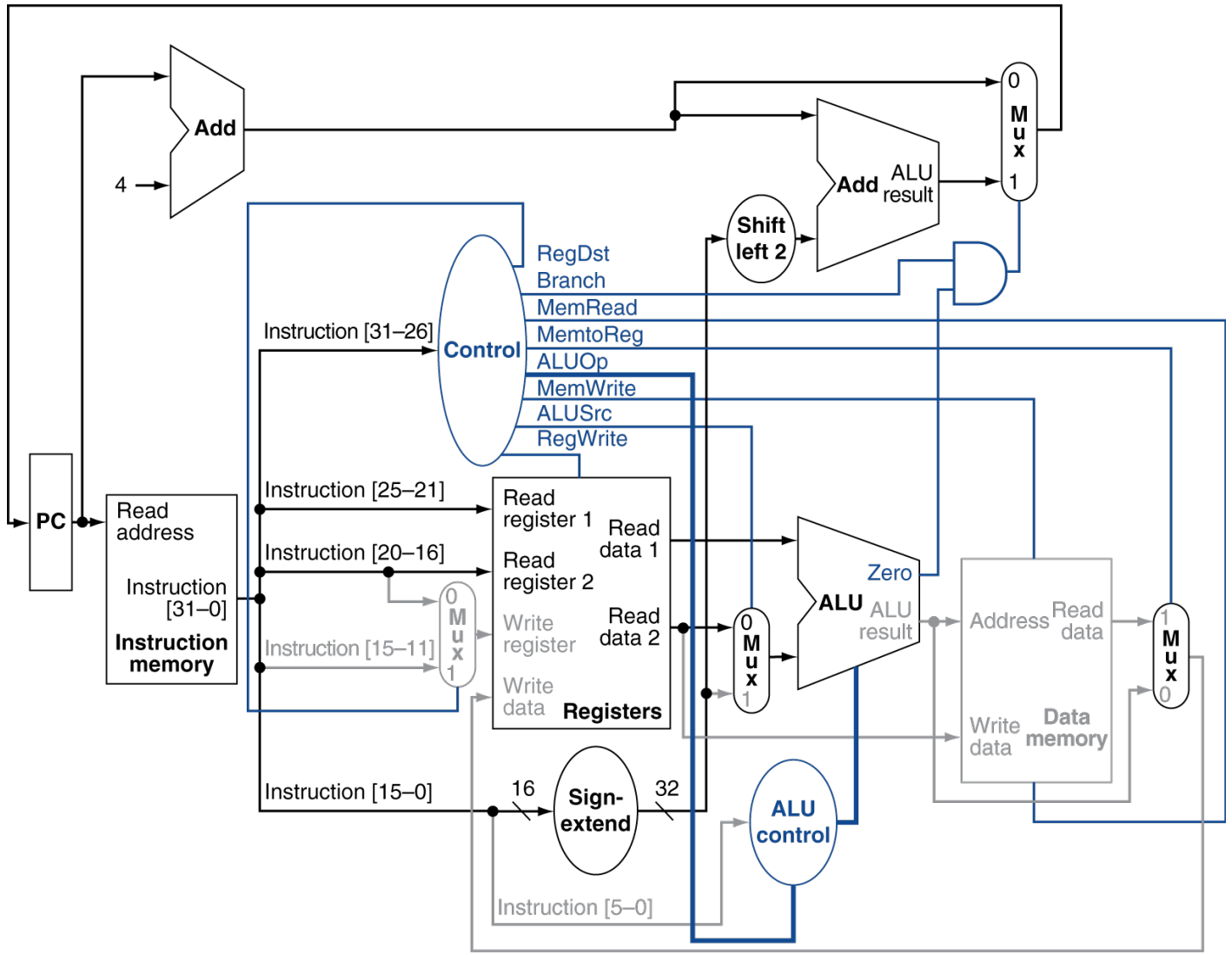
Datapath – Load Instruction



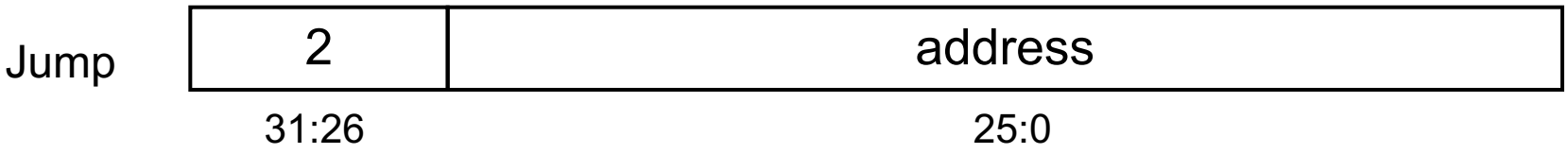
Datapath – Branch Instructions

- **We can think of a branch instruction (e.g., `beq $t1, $t2, offset`) as operating in four steps:**
  1. The **instruction is fetched** and the **PC is incremented**
  2. The **main control unit computes the setting** of the control lines and **two registers are read** (`$t2` and `$t3`) from the register file
  3. The **ALU performs a subtract** on the values read from the register file and **the adder computes the sum** (branch target address) of the PC+4 and the sign extended lower 16 bits of the instruction (offset) shifted left by two
  4. The **Zero result from the ALU is used to decide** which result (PC+4 or branch target address) to store into the PC

Datapath – Branch Instructions



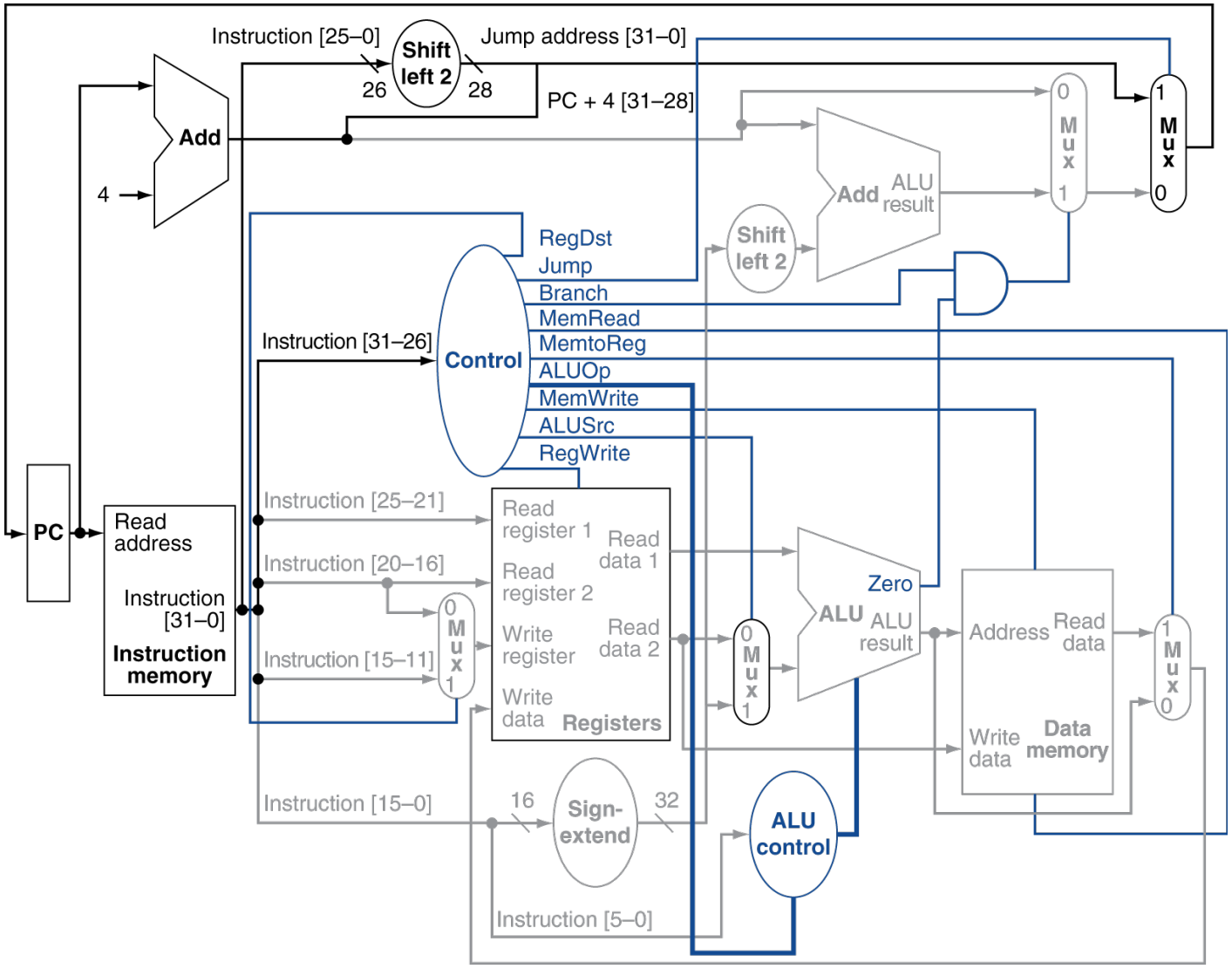
Implementing Jumps



- **Jump instructions (e.g., `j label`)**
  - Operate by **replacing the lower 28 bits of the PC** with the lower 26 bits of label field contained in the instruction shifted left by 2 bits
  - This shift increases the label range by a factor of 4

- **Need an extra control signal decoded from opcode**

Datapath With Jumps Added



Performance Issues

- **This implementation is easy to understand, but too inefficient to be practical**
  - In a single-cycle implementation, all instructions are executed in one clock cycle and take the same time to execute

- **Longest delay determines clock period**
  - Critical path: `lw` instructions
    - Instruction memory → register file → ALU → data memory → register file
  - Although all the other instructions (`sw`, `add`, `sub`, `and`, `or`, `slt`, `beq`) take less time, they still execute in the needed time for `lw`

Performance Issues

- **Not feasible to vary period for different instructions**
  - Overhead can easily surpass benefits due to difficulty in implementation
- **Violates the "making the common case fast" design principle**
- **We will improve performance by pipelining**
  - Uses a datapath very similar to the single-cycle datapath but is much more efficient by having a much **higher throughput**
  - Pipelining improves efficiency by executing multiple instructions simultaneously

Concluding Remarks

- **ISA influences design of datapath and control**
  - The datapath and control for a processor can be designed starting with the instruction set architecture
  - The simplicity and regularity of the MIPS instruction set simplifies the implementation by making the execution of many of the instruction classes similar

- **Datapath and control influence design of ISA**
  - The underlying technology also affects many design decisions by dictating what components can be used in the datapath, as well as whether a single-cycle implementation even makes sense