



Topic 6

Single Cycle Processor

Introduction

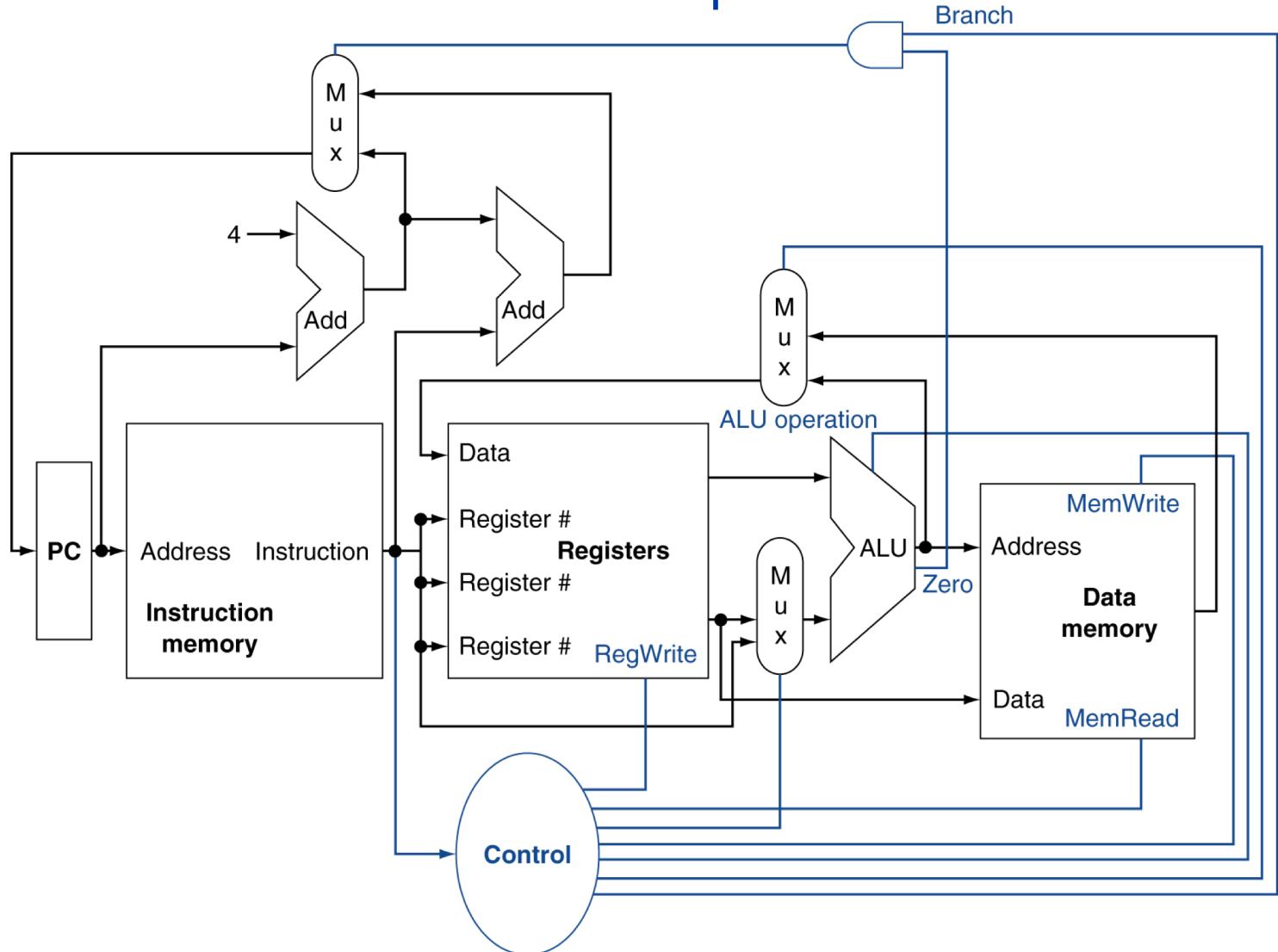
- MIPS implementations
 - A simplified single-cycle version
 - An FSM based multi-cycle version
 - A more realistic (simplified) pipelined version
- Supports simple subset, shows most aspects of MIPS architecture
 - Arithmetic/logical: add, sub, and, or, slt
 - Memory reference: lw, sw
 - Control transfer: beq, j

Instruction Execution

- PC → program memory, fetch instruction
- Register addresses to access register file, read registers
- Depending on instructions
 - Use ALU/Adder to calculate
 - Arithmetic/logic result
 - Memory address for load/store
 - Branch target address
- Access register or data memory for load/store
- Update PC \leftarrow target address or PC + 4

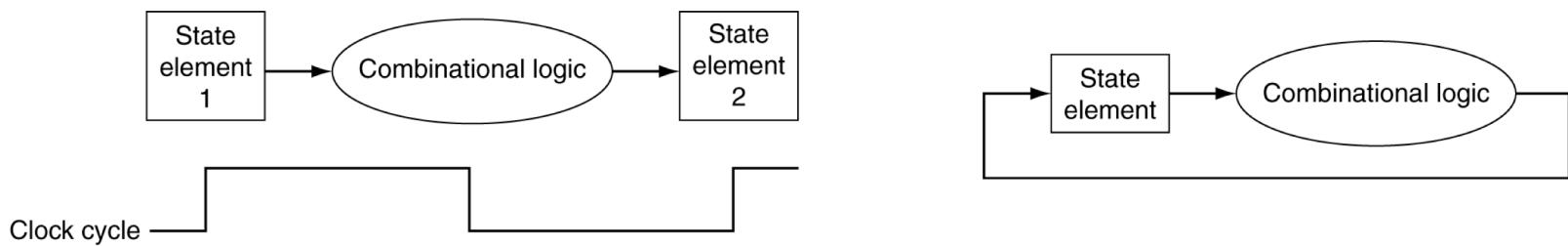
First View of CPU

Combination of Controller and Datapath



Clocking Methodology

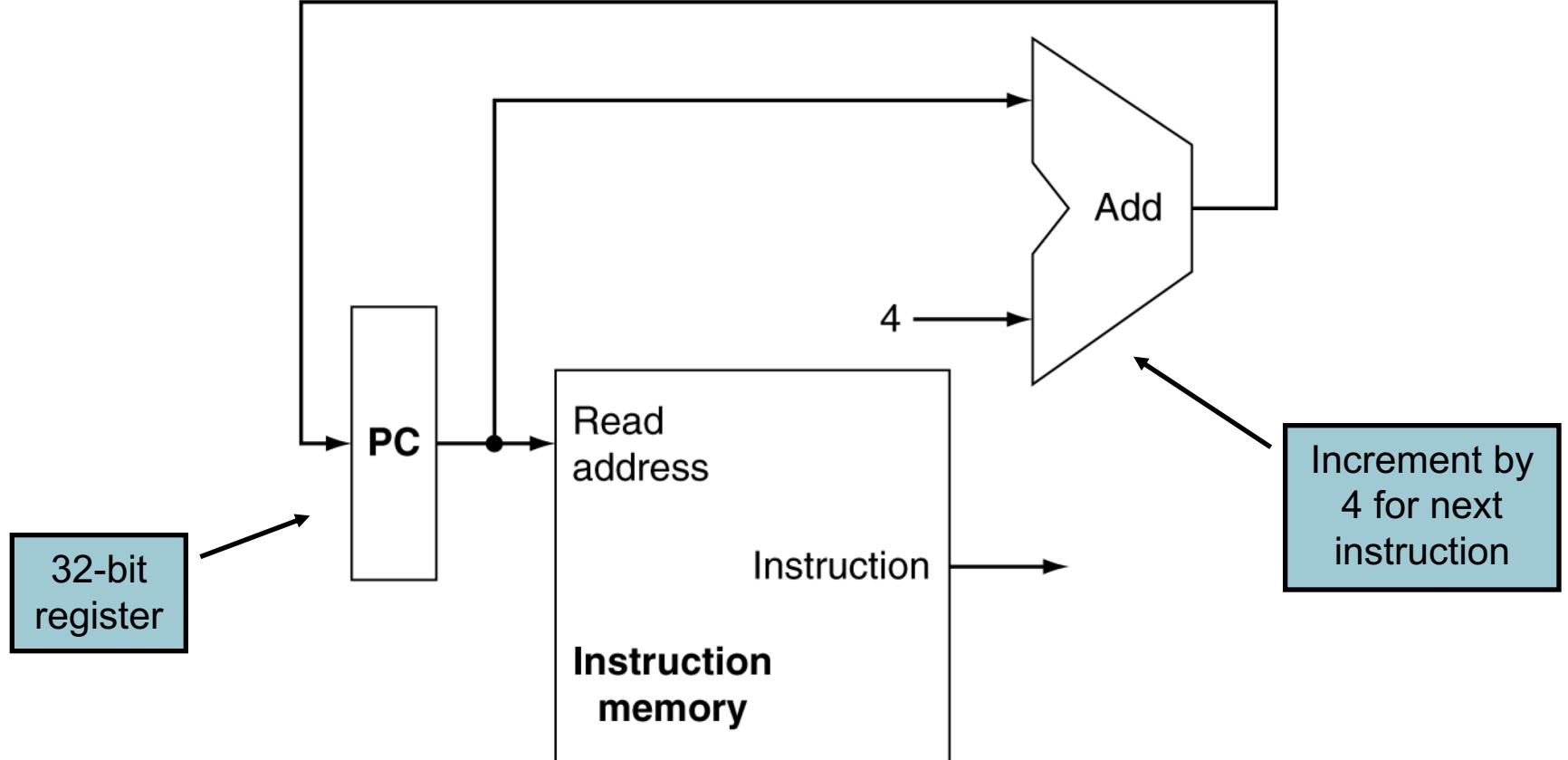
- Combinational logic does the computation during clock cycles
 - Between clock edges
 - Input (present state) from state elements, output (next state) to state element
 - Among all kinds of computations, longest delay determines clock period



Building a Datapath

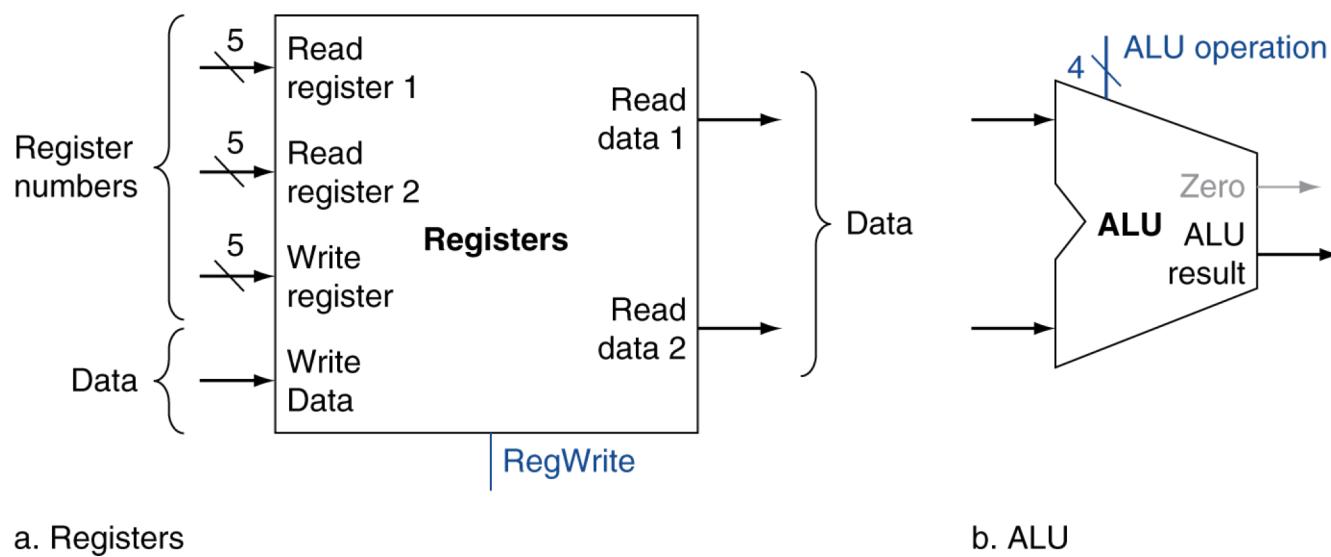
- Datapath
 - Elements that process data and addresses in the CPU
 - Registers, ALUs, mux's, memories, ...
- We will build a MIPS datapath incrementally

Instruction Fetch



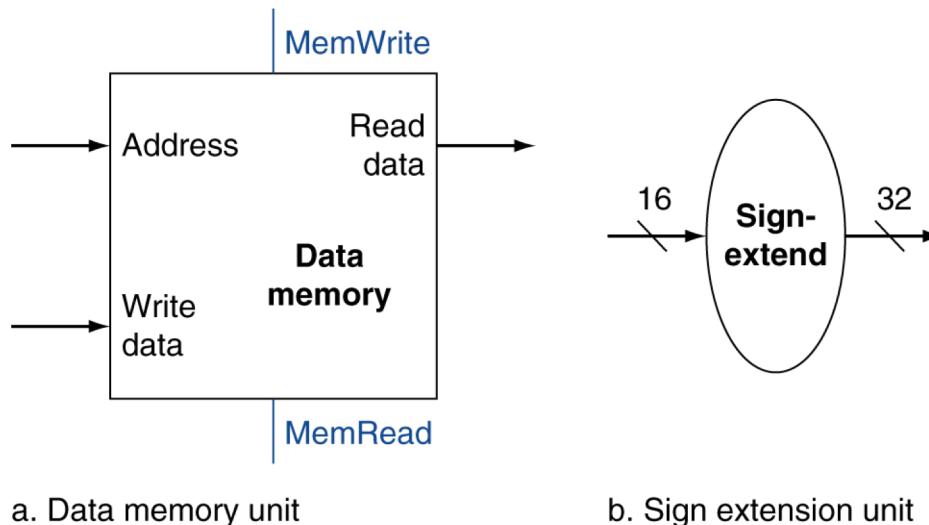
R-Format Instructions

- Read two register operands
- Perform arithmetic/logical operation
- Write register result



Load/Store Instructions

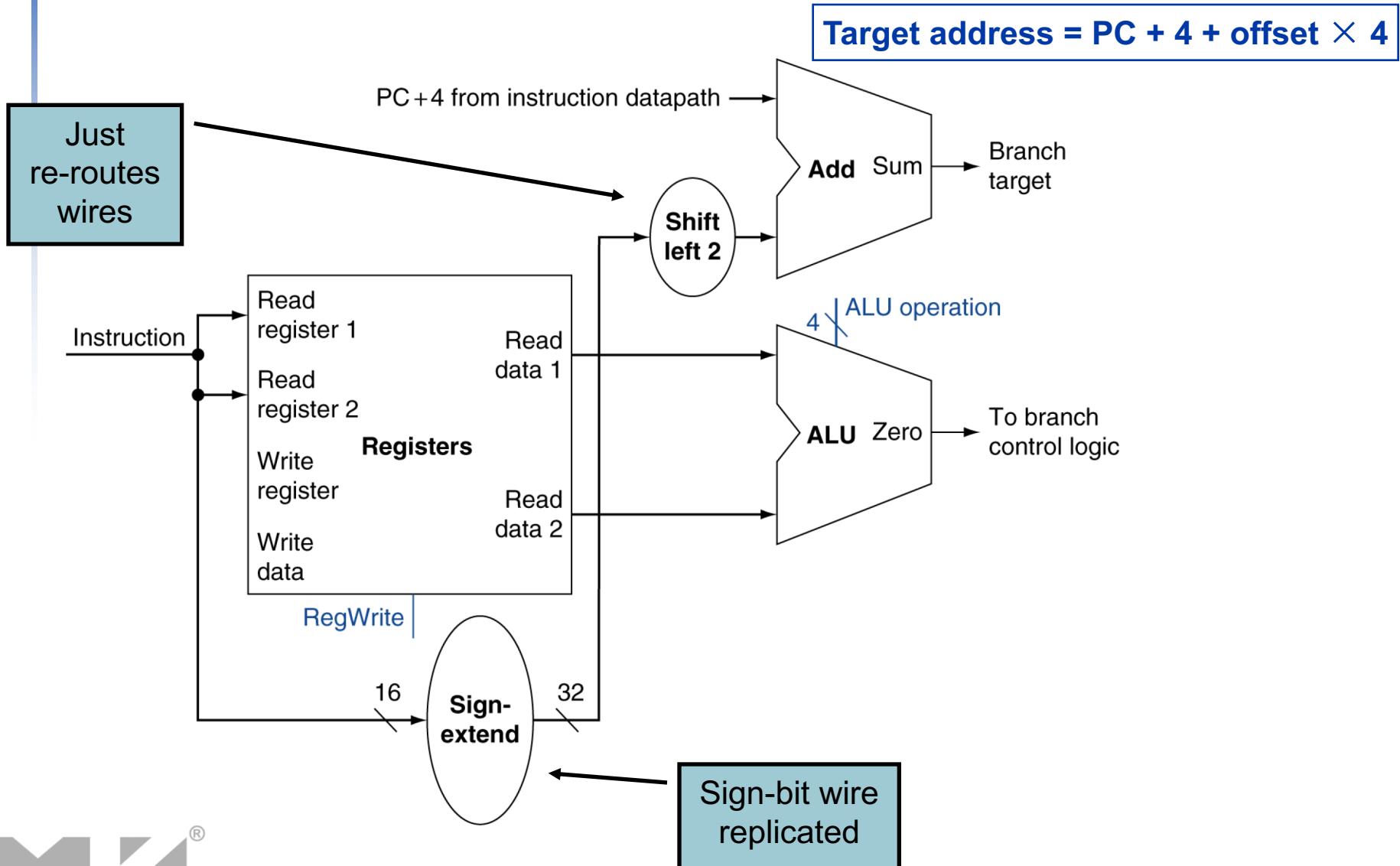
- Read register operands
- Calculate address using 16-bit offset
 - Use ALU, but sign-extend offset
- Load: Read memory and update register
- Store: Write register value to memory



Branch Instructions

- Read register operands
- Compare operands
 - Use ALU, subtract and check Zero output
- Calculate target address
 - Sign-extend the immediate relative address
 - Shift left 2 places (word displacement)
 - Add to PC + 4
 - PC+4 already calculated by instruction fetch

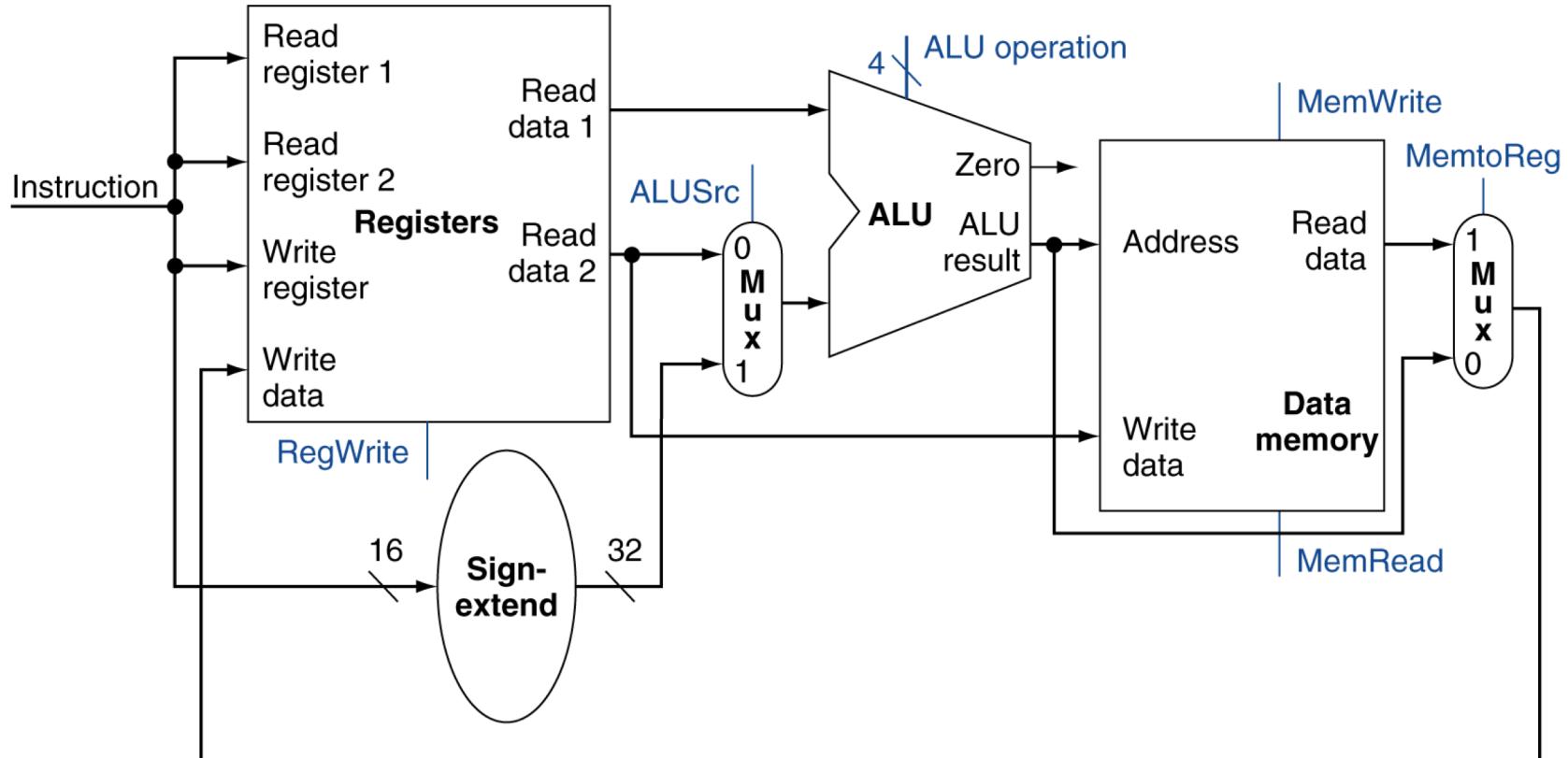
Branch Instructions



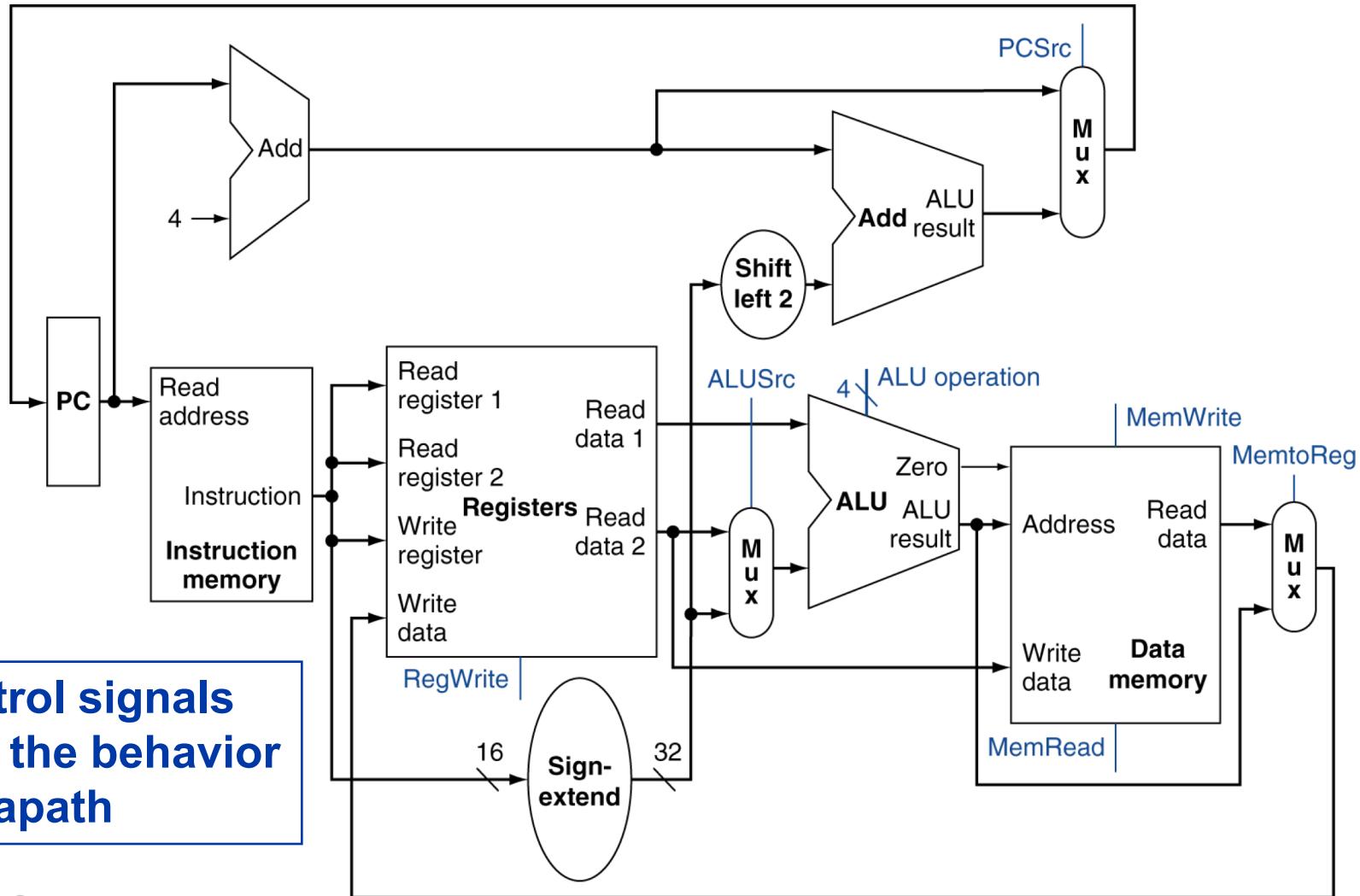
Composing the Elements

- First version of data path does an instruction in one clock cycle
 - Each datapath element can only do one function at a time
 - Hence, we need separate instruction memory and data memory
- Use multiplexers where alternate data sources are used for different instructions

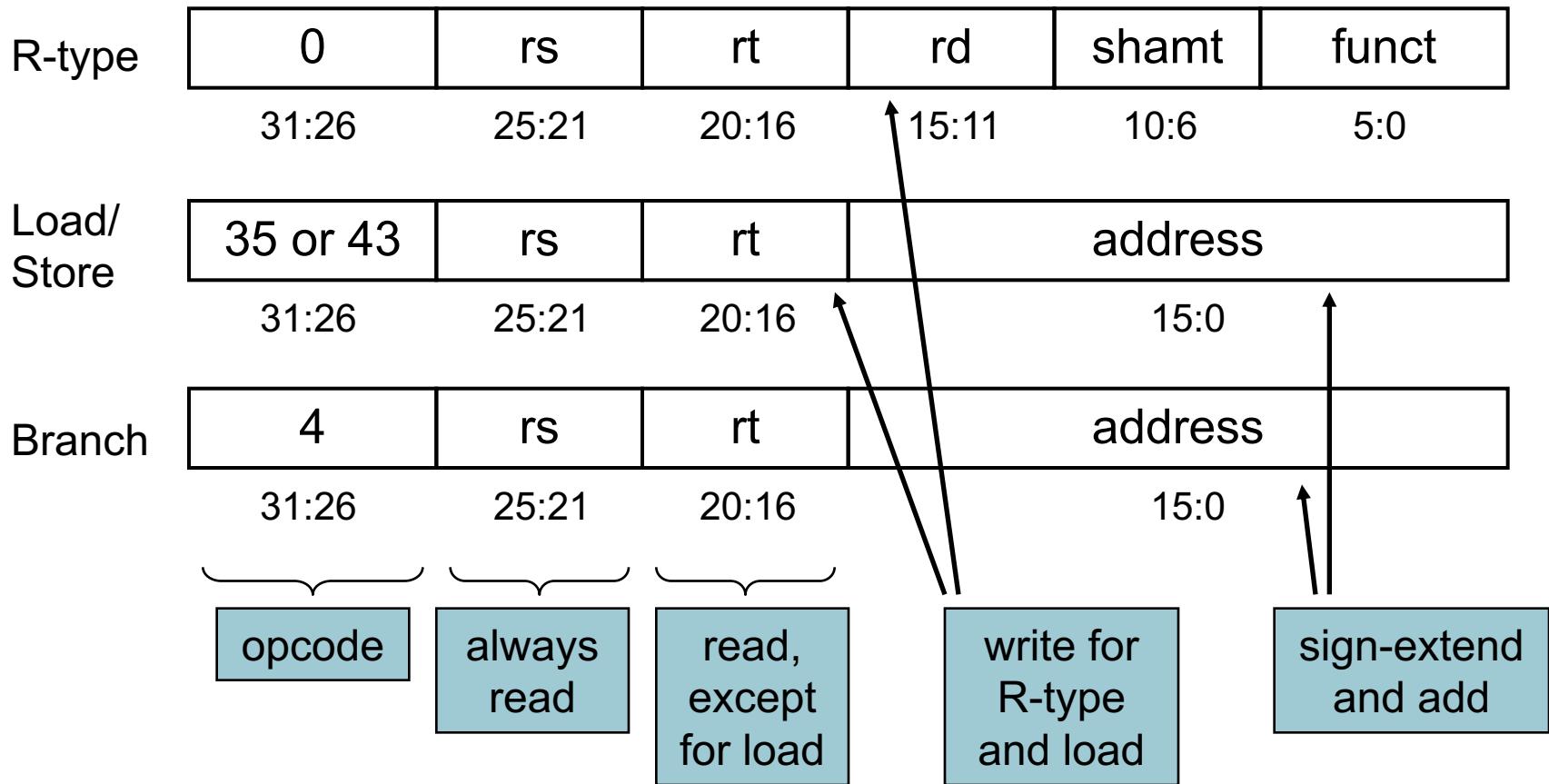
R-Type/Load/Store Datapath



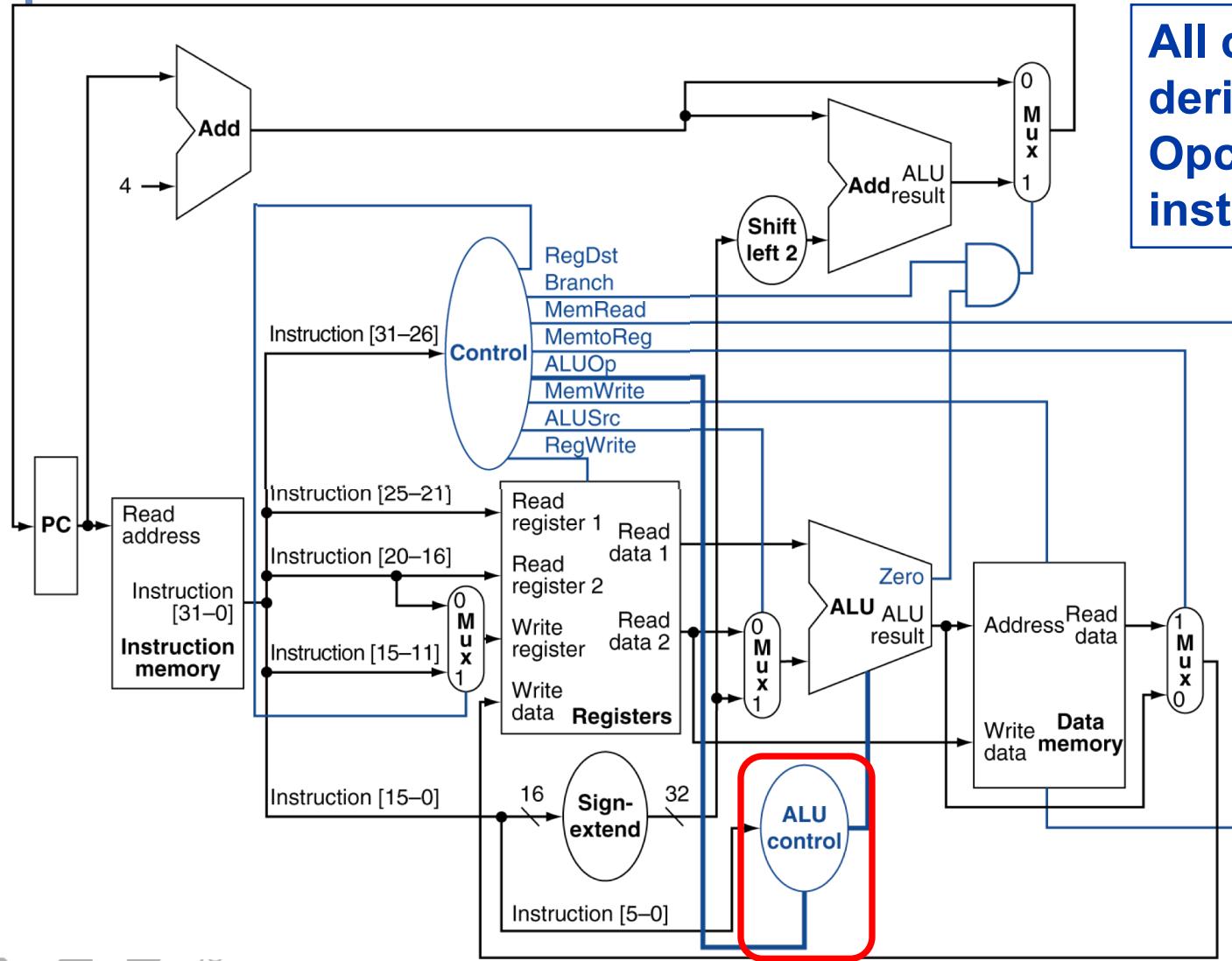
First Version Datapath



How Operands are Used?



Datapath With Control



All control signals derived from 6-bit Opcode of an instruction

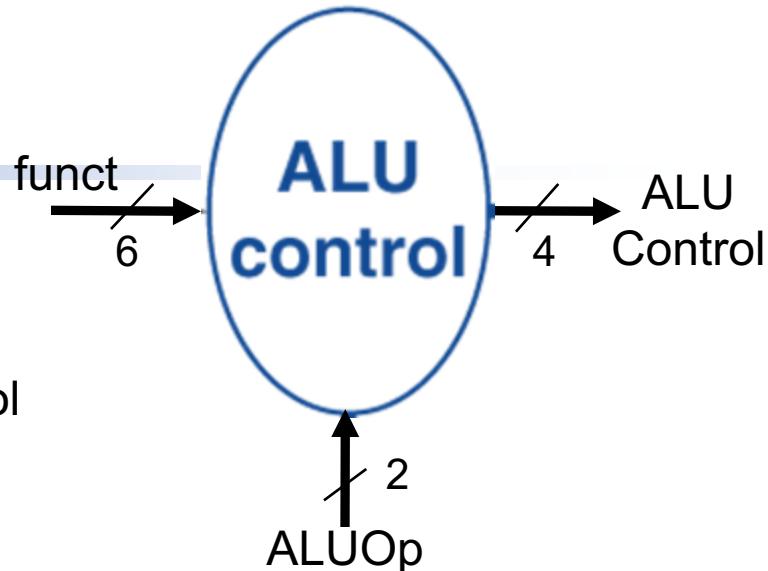
ALU Control Unit

- ALU used for
 - Load/Store: Function = add
 - Branch: Function = subtract
 - R-type: Function = add/subtract/logical, depends on funct field in instructions

ALU Control (output)	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set-on-less-than
1100	NOR

ALU Control Unit

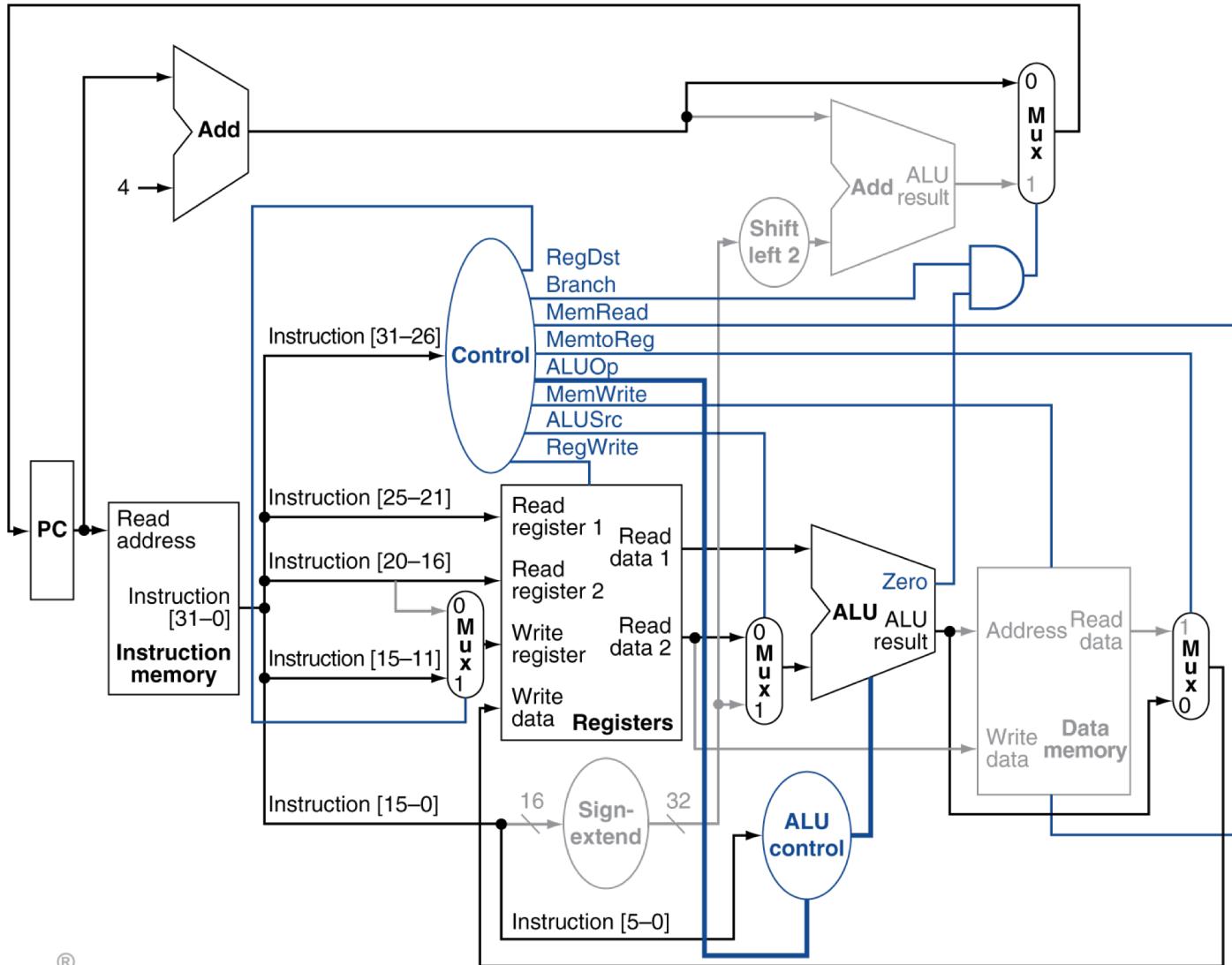
- Assume 2-bit ALUOp generated by CPU controller



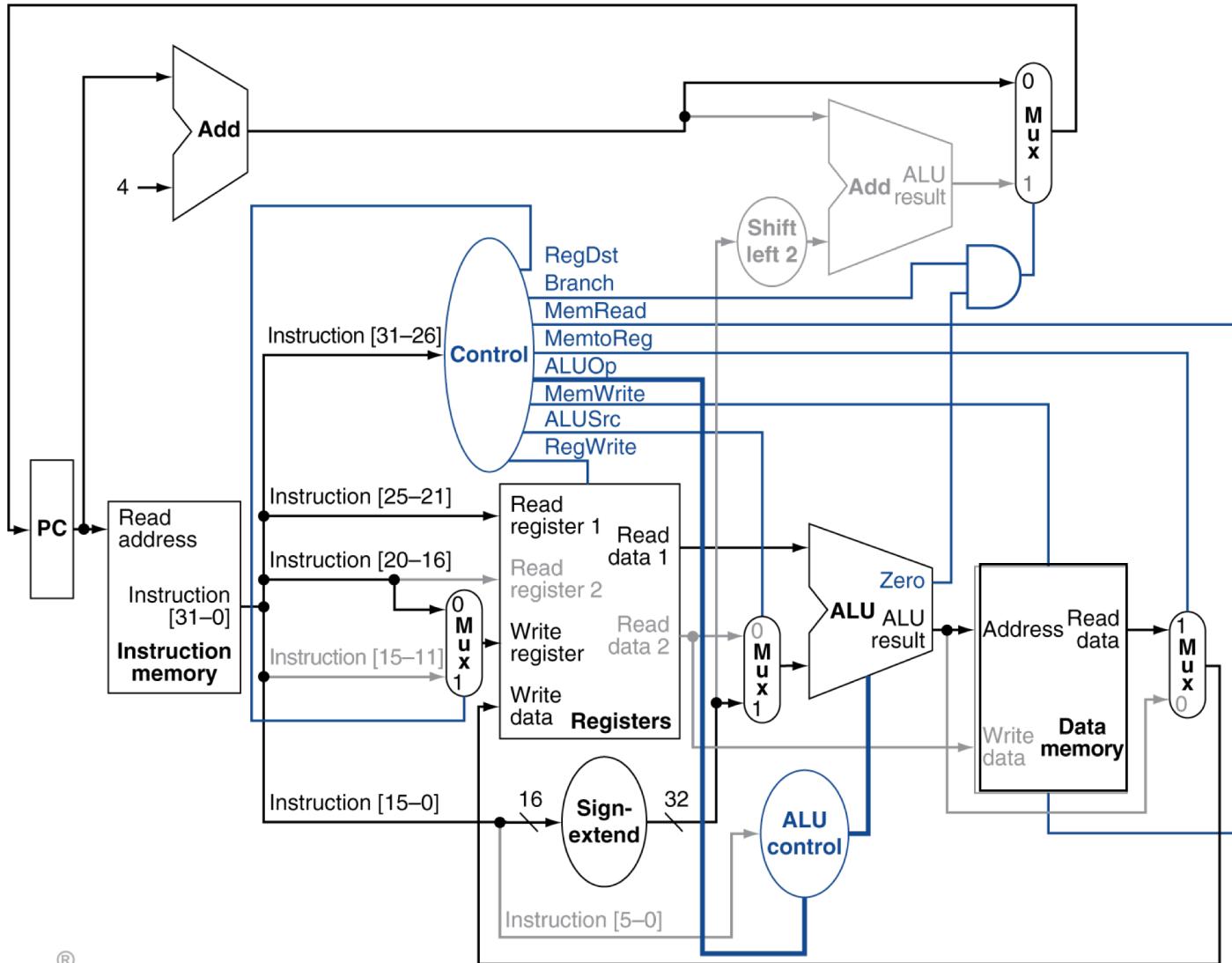
- Combinational logic derives ALU control
- With inputs ALUOp and funct – 8 bits

opcode	Operation	ALUOp	funct	ALU Control	ALU function
lw	load word	00	XXXXXX	0010	add
sw	store word				
beq	branch equal	01	XXXXXX	0110	subtract
R-type	add	10	100000	0010	add
	subtract		100010	0110	subtract
	AND		100100	0000	AND
	OR		100101	0001	OR
	set-on-less-than		101010	0111	set-on-less-than

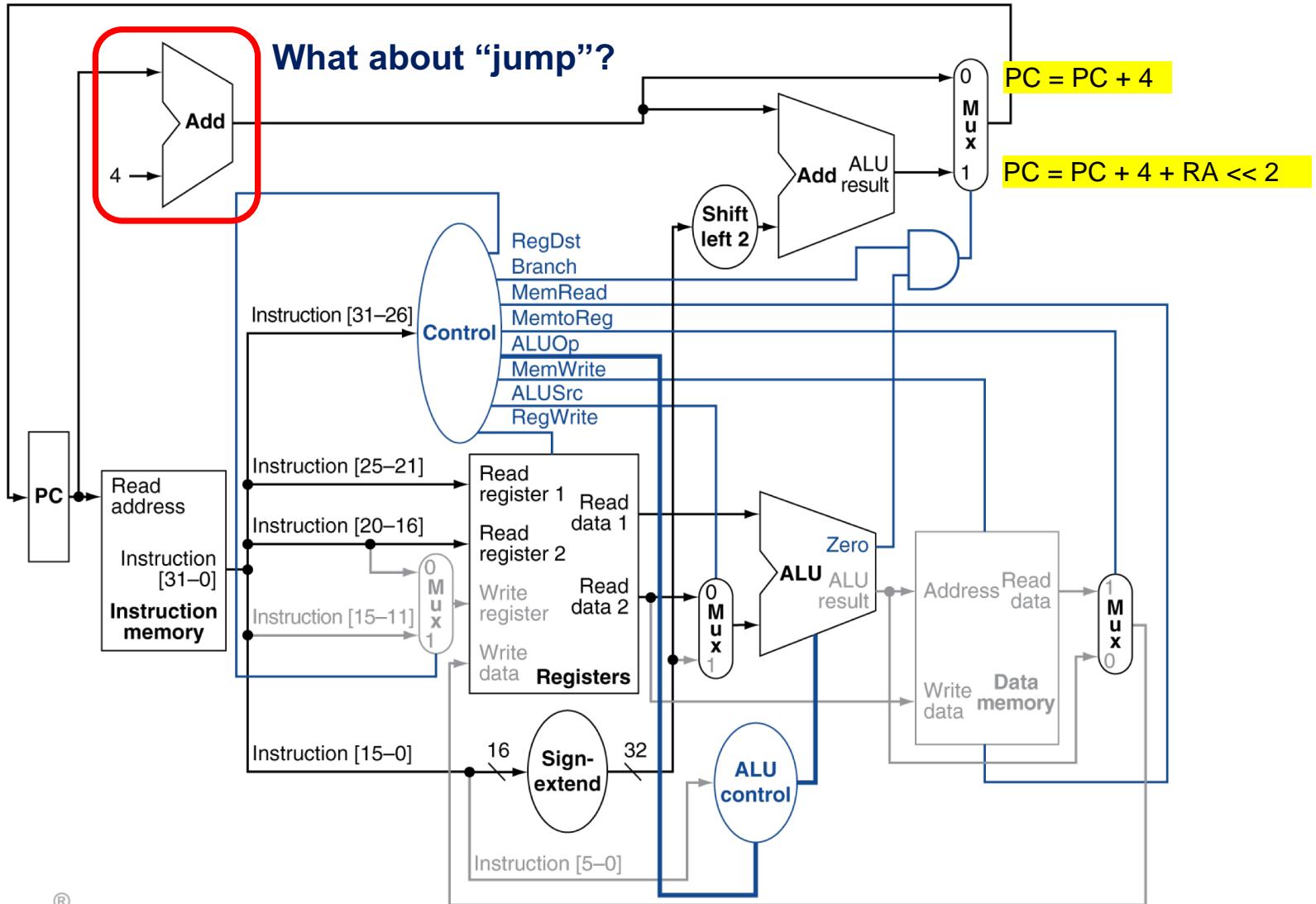
R-Type Instruction



Load Instruction

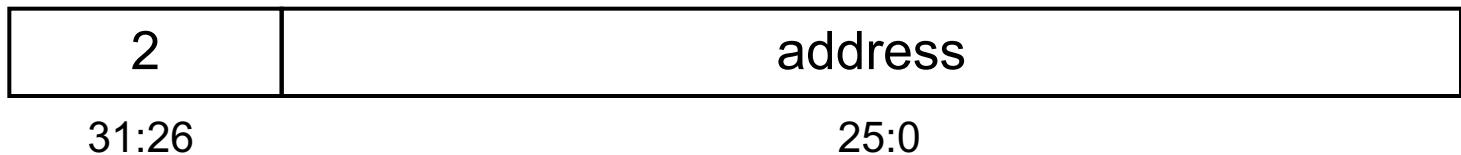


Branch-on-Equal Instruction



Implementing Jumps

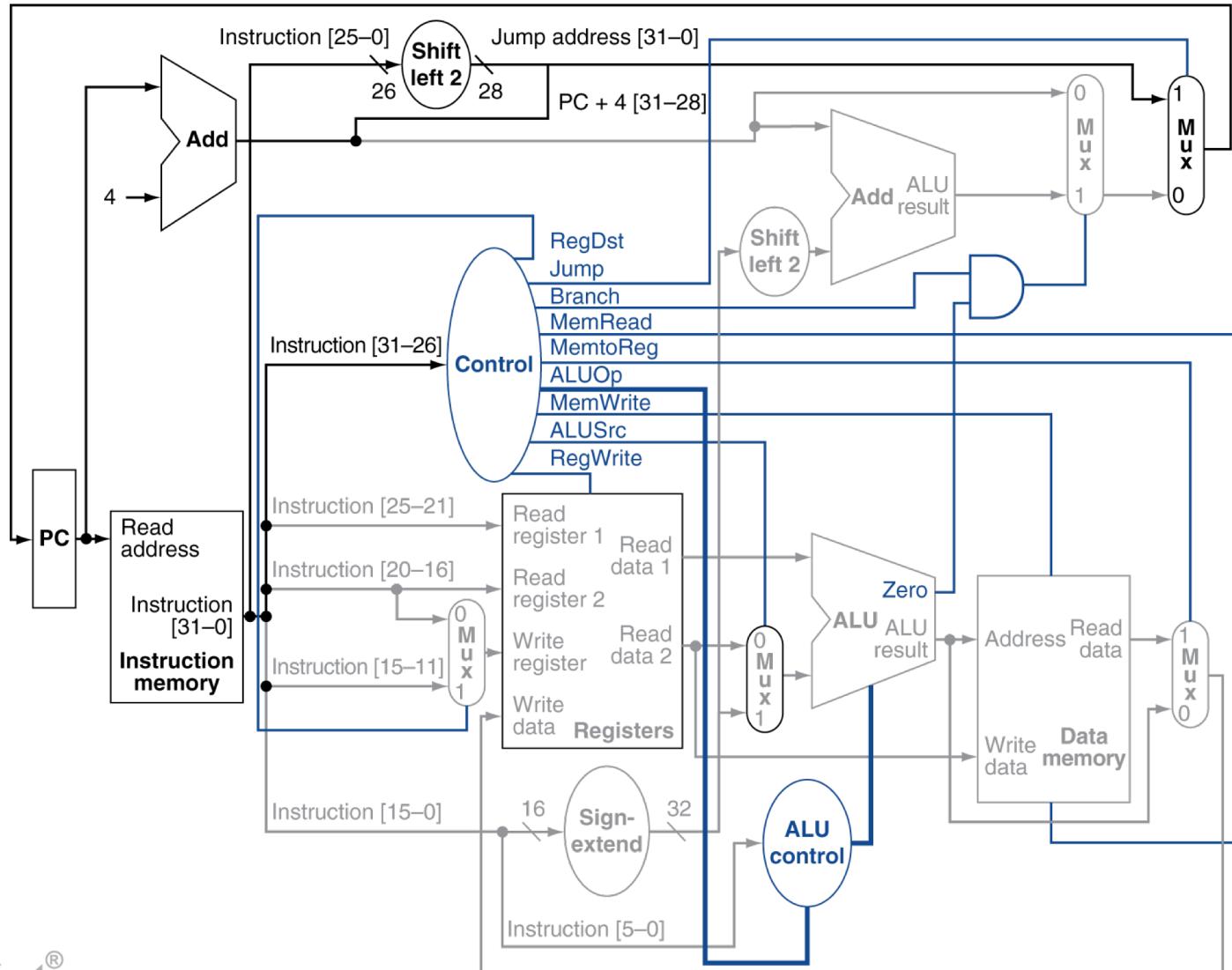
Jump



- Jump needs to calculate new address
- Update PC with new address:
 - Top 4 bits of old PC:26-bit jump address:00
- Need an extra control signal decoded from opcode by Control Unit to alternate the PC value

Datapath With Jump Supported

PC = PC[31:28]; Address;00



Performance Issues

- Longest delay determines clock period
 - Critical path: the path having longest delay
 - load instruction

Instruction memory → register file → ALU → data memory → register file (plus MUXes)
- Not feasible to vary period for different instructions
 - Waste time on other instructions
- How to improve the performance (faster speed)?