Single Cycle CPU

中科院计算所
INSTITUTE OF COMPUTING TECHNOLOGY

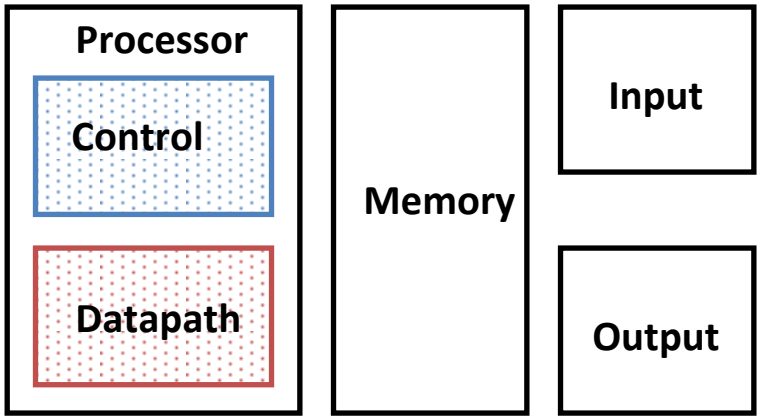
University of Chinese Academy of Sciences (UCAS)

The Processor

- **Processor (CPU)**: Implements the instructions of the Instruction Set Architecture (ISA)
  - *Datapath*: part of the processor that contains the hardware necessary to perform operations required by the processor ("the brawn")
    - 组合元件和存储元件通过总线或分散方式连接而成的进行数据存储、处理和传送的路径。
  - *Control*: part of the processor (also in hardware) which tells the datapath what needs to be done ("the brain")

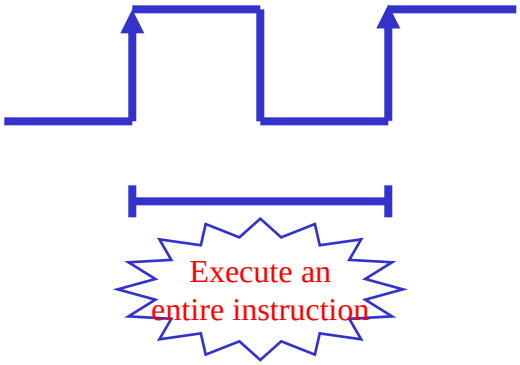
Processor Design Process

- Five steps to design a processor:
  - Datapath
    - 1. Analyze instruction set → datapath requirements
    - 2. Select set of datapath components & establish clock methodology
    - 3. Assemble datapath meeting the requirements
  - Control
    - 4. Analyze implementation of each instruction to determine setting of control points that effects the register transfer
    - 5. Assemble the control logic
      - Formulate Logic Equations
      - Design Circuits



The Big Picture: The Performance Perspective

- Processor design (datapath and control) will determine:
  - Clock cycle time
  - Clock cycles per instruction
- Starting today:
  - Single cycle processor:
    - Advantage: One clock cycle per instruction
    - Disadvantage: long cycle time



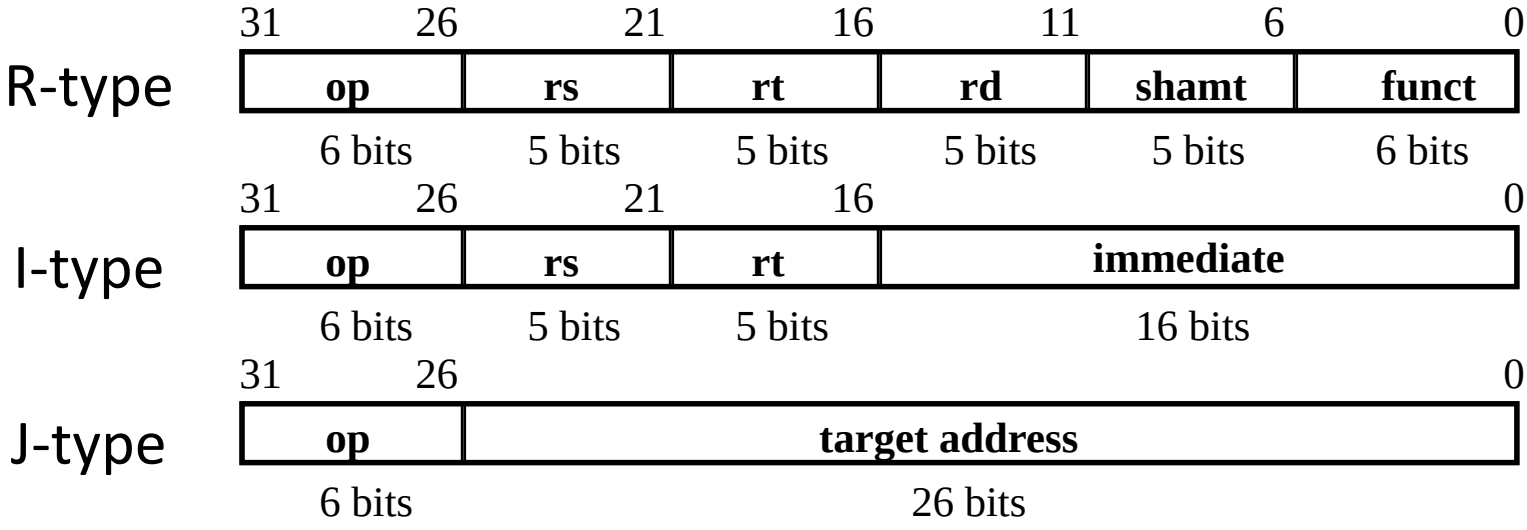
- ET = IC × CPI × Cycle Time

Processor Datapath and Control

- We're ready to look at an implementation of the MIPS simplified to contain only:
  - memory-reference instructions: lw, sw
  - arithmetic-logical instructions: add, sub, and, or, slt
  - control flow instructions: beq
- Generic Implementation:
  - use the program counter (PC) to supply instruction address
  - get the instruction from memory
  - read registers
  - use the instruction to decide exactly what to do
- All instructions use the ALU after reading the registers
  - memory-reference? arithmetic? control flow?

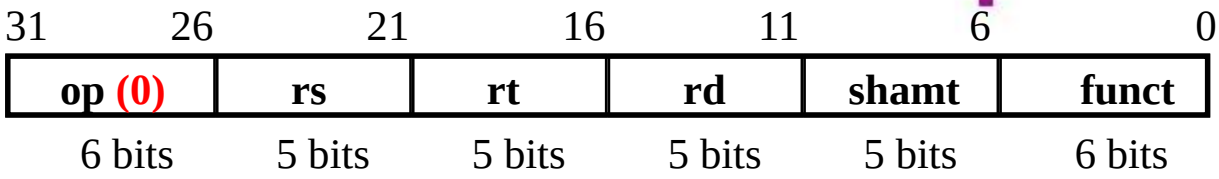
MIPS Instruction Formats

- All instructions 32-bits long
- 3 Formats:

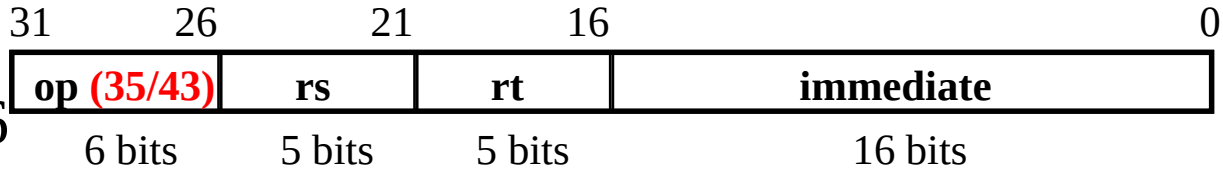


The MIPS Subset

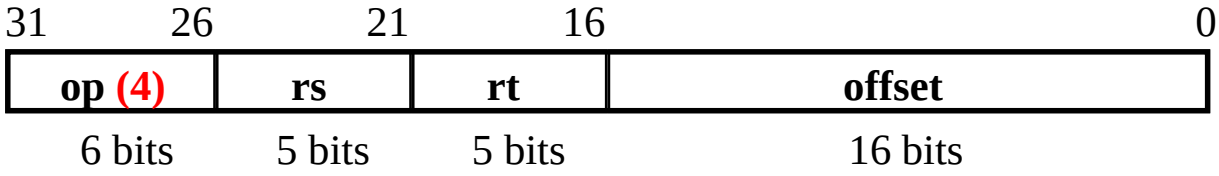
- R-Type
  - add rd, rs, rt
  - sub, and, or, slt



- LOAD and STORE
  - lw rt, rs, imm16
  - sw rt, rs, imm16



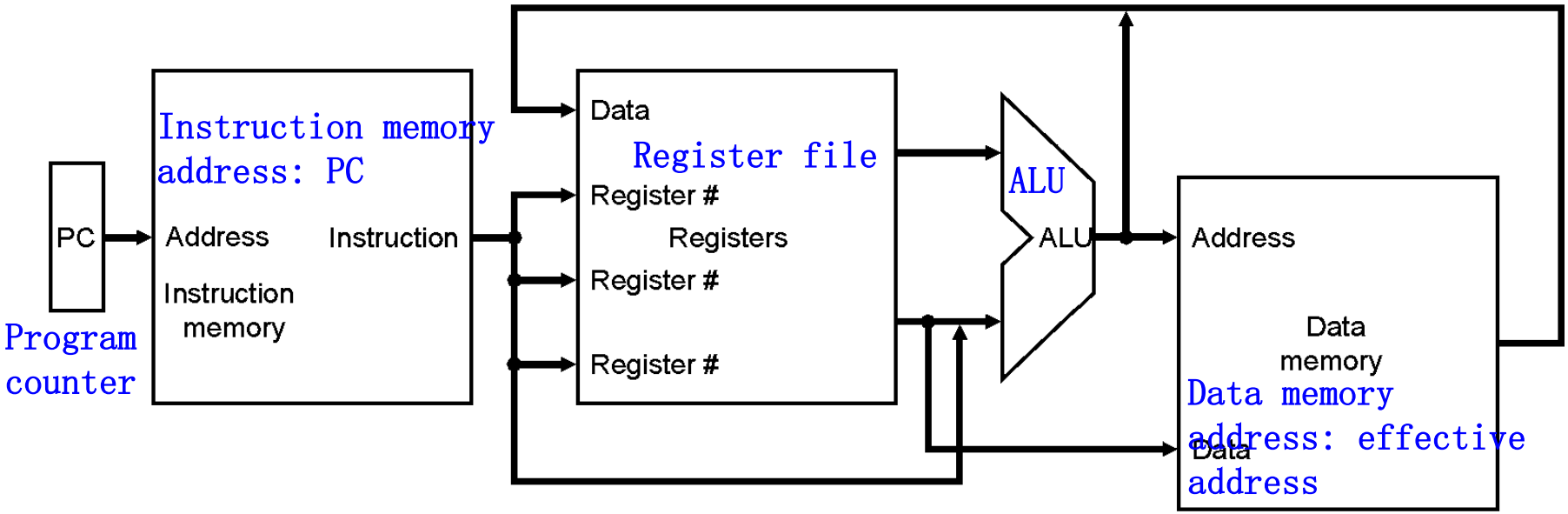
- BRANCH:
  - beq rs, rt, imm16



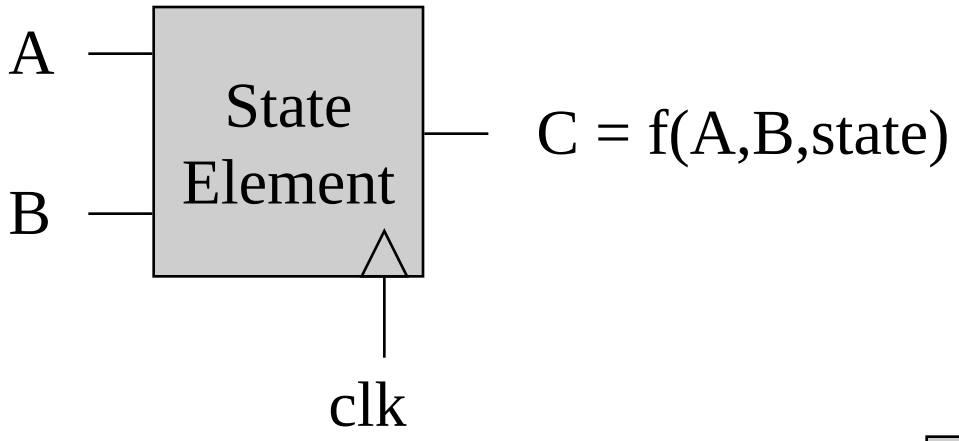
Basic Steps of Execution

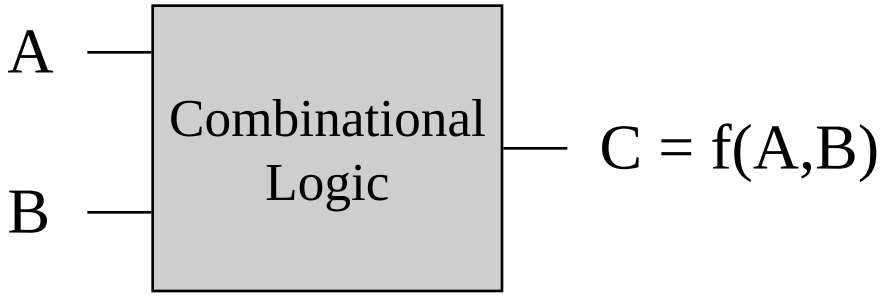
- Instruction Fetch — Instruction memory
  - Where is the instruction? — address: PC
- Decode
  - What's the incoming instruction? — Register file
  - Where are the operands in an instruction?
- Execution: ALU — ALU
  - What is the function that ALU should perform?
- Memory access — Data memory
  - Where is my data? — address: effective address
- Write back results to registers — Register file
  - Where to write?
- Determine the next PC — Program counter

Where We're Going: The High-level View



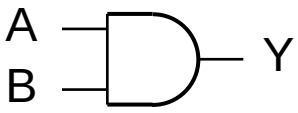
Review: Two Type of Logical Components



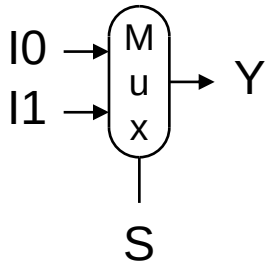


Combinational Elements

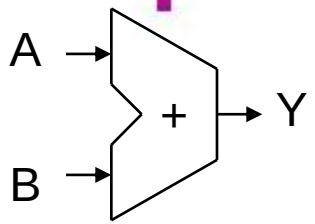
- AND-gate
  - Y = A & B



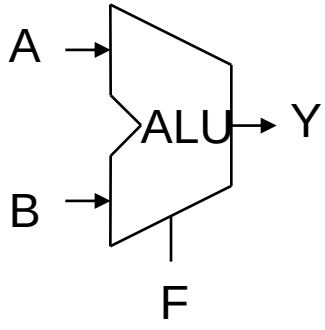
- Multiplexer
  - Y = S ? I1 : I0



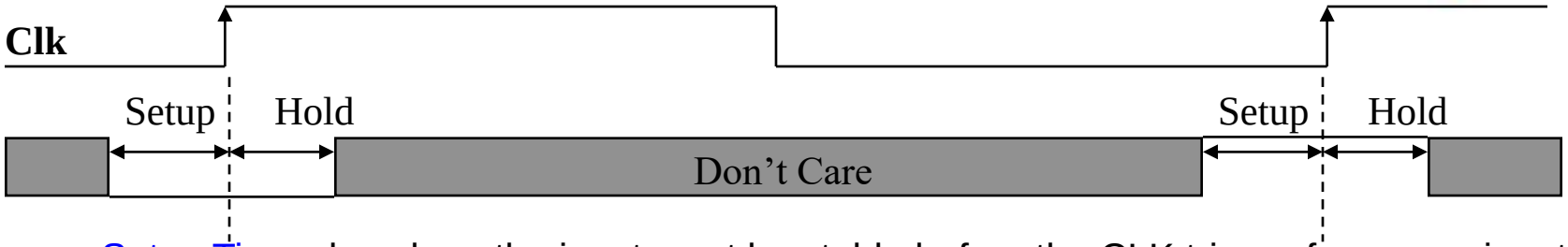
- Adder
  - Y = A + B



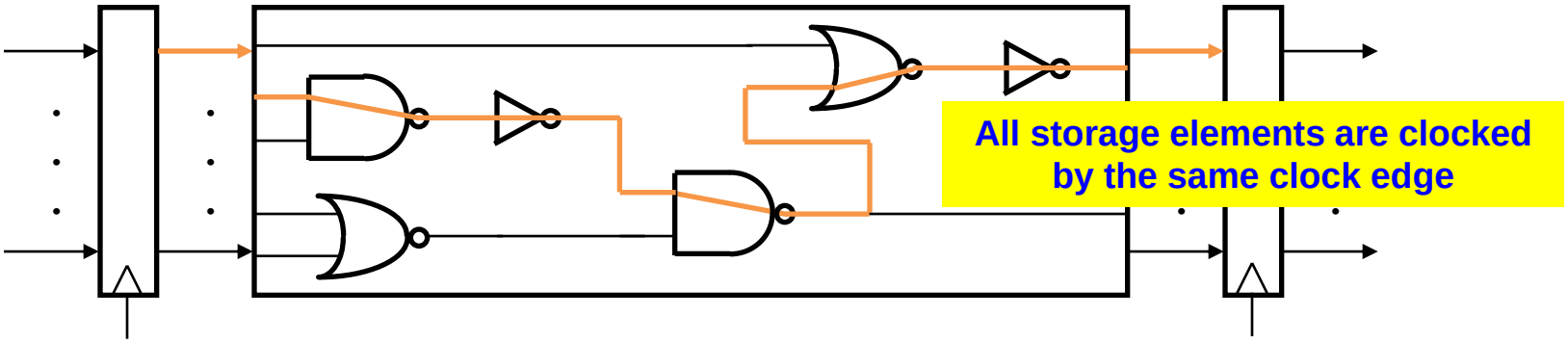
- Arithmetic/Logic Unit
  - Y = F(A, B)



Clocking Methodology (定时方法)



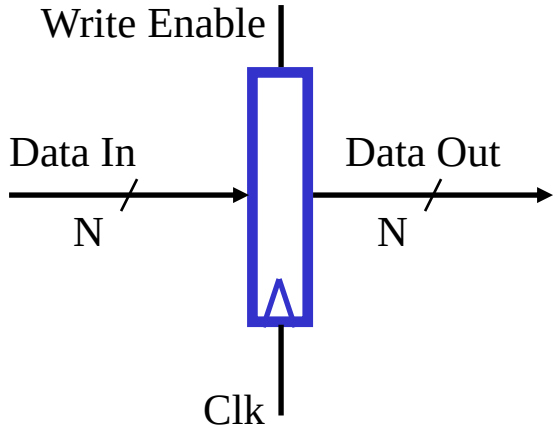
- Setup Time: how long the input must be stable before the CLK trigger for proper input read
- Hold Time: how long the input must be stable after the CLK trigger for proper input read
- CLK-to-Q Delay: how long it takes the output to change, measured from the CLK trigger

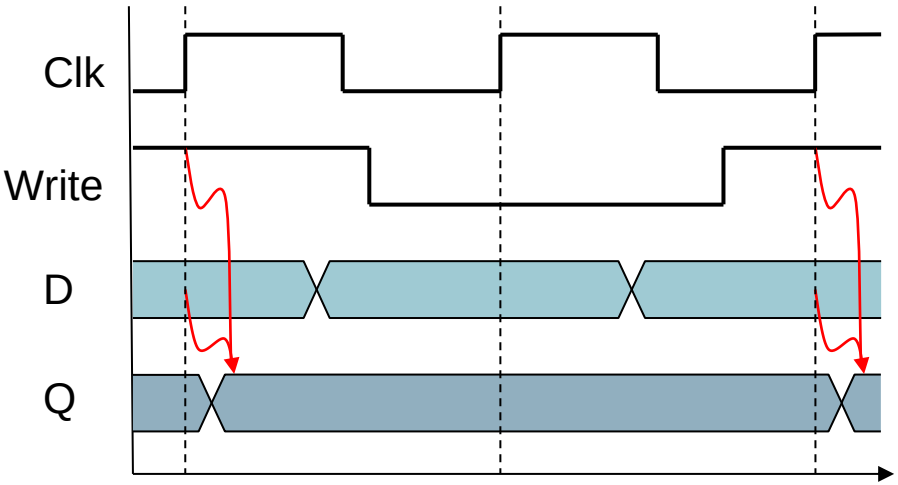


- The critical path is the longest delay between any two registers in a circuit
- Critical path determines length of clock period
  - The clock period must be longer than this critical path, or the signal will not propagate properly to that next register
  - This includes CLK-to-Q delay and setup delay

Storage Element: The Register

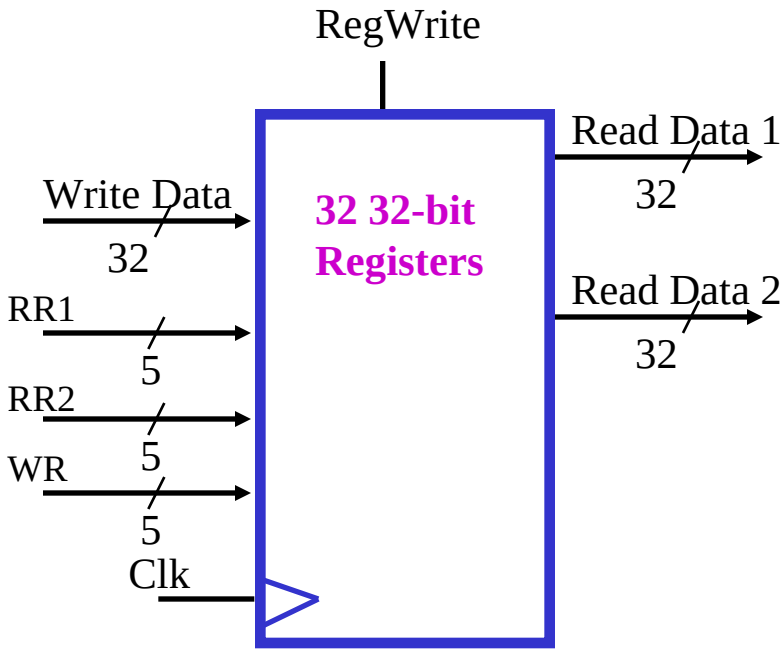
- Register
  - Similar to the D Flip Flop except
    - N-bit input and output
    - Write Enable input
- Write Enable:
  - 0: Data Out will not change
  - 1: Data Out will become Data In (on the clock edge)
    - Only updates on clock edge when write control input is 1
    - Used when stored value is required later





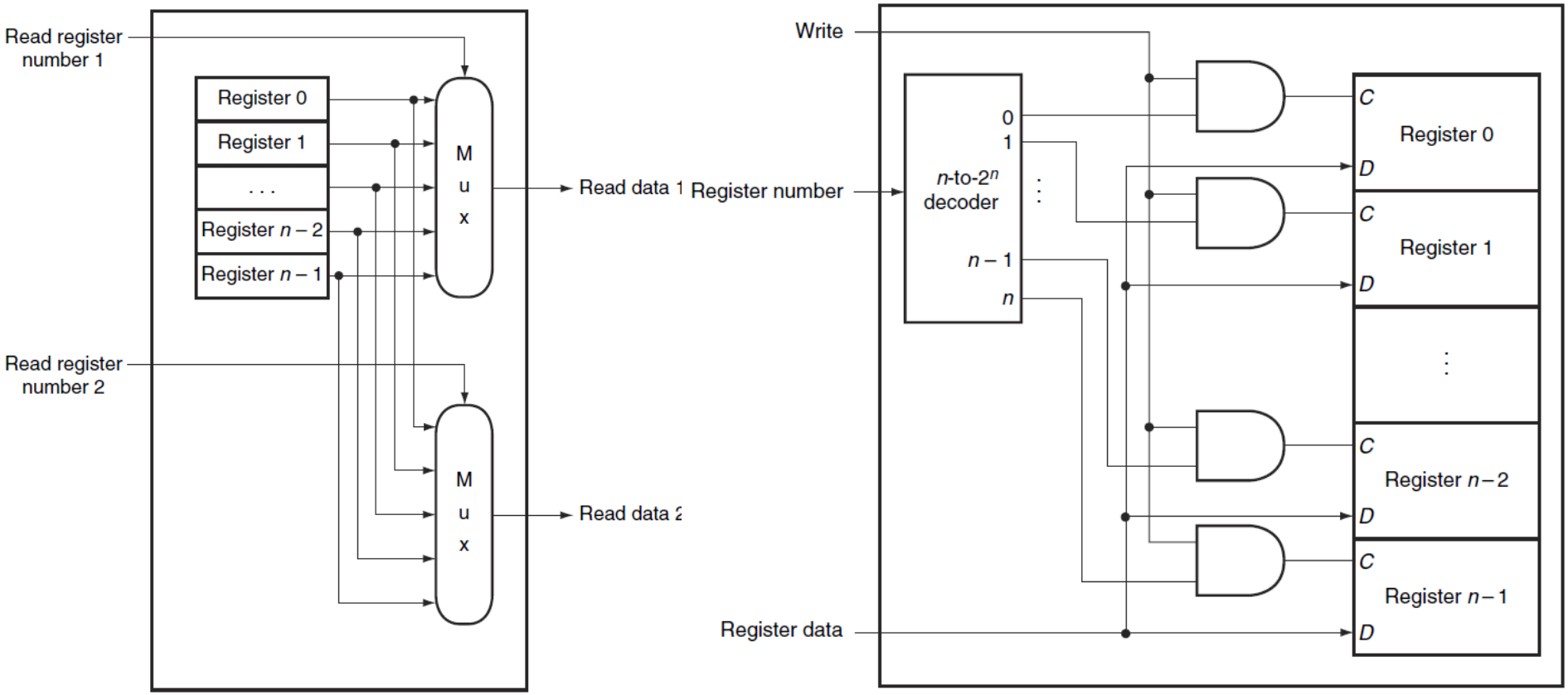
Storage Element: Register File

- Register File consists of (32) registers:
  - Two 32-bit output buses
  - One 32-bit input bus
- Register is selected by:
  - RR1 selects the register to put on bus "Read Data 1"
  - RR2 selects the register to put on bus "Read Data 2"
  - WR selects the register to be written
  - via WriteData when RegWrite is 1
- Clock input (CLK)



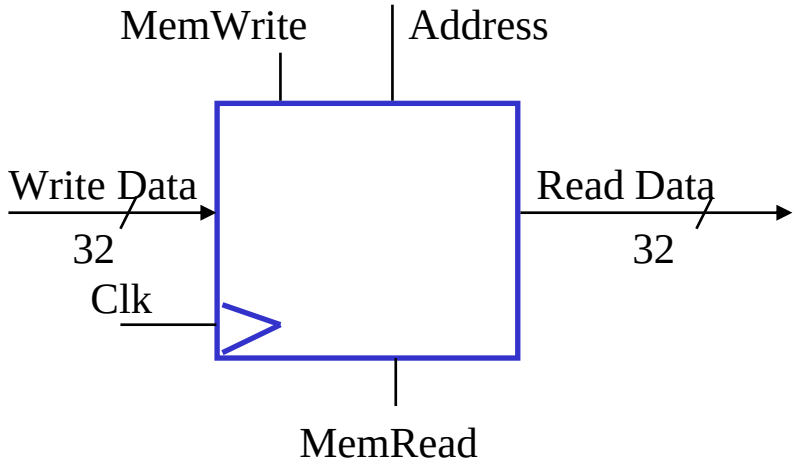
Inside the Register File

- The implementation of two read ports register file
  - n registers
  - done with a pair of n-to-1 multiplexors, each 32 bits wide.



Storage Element: Memory

- Memory
  - Two input buses: WriteData, Address
  - One output bus: ReadData
- Memory word is selected by:
  - Address selects the word to put on ReadData bus
  - If MemWrite = 1: address selects the memory word to be written via the WriteData bus
- Clock input (CLK)
  - The CLK input is a factor ONLY during write operation
  - During read operation, behaves as a combinational logic block:
    - Address valid => ReadData valid after "access time."

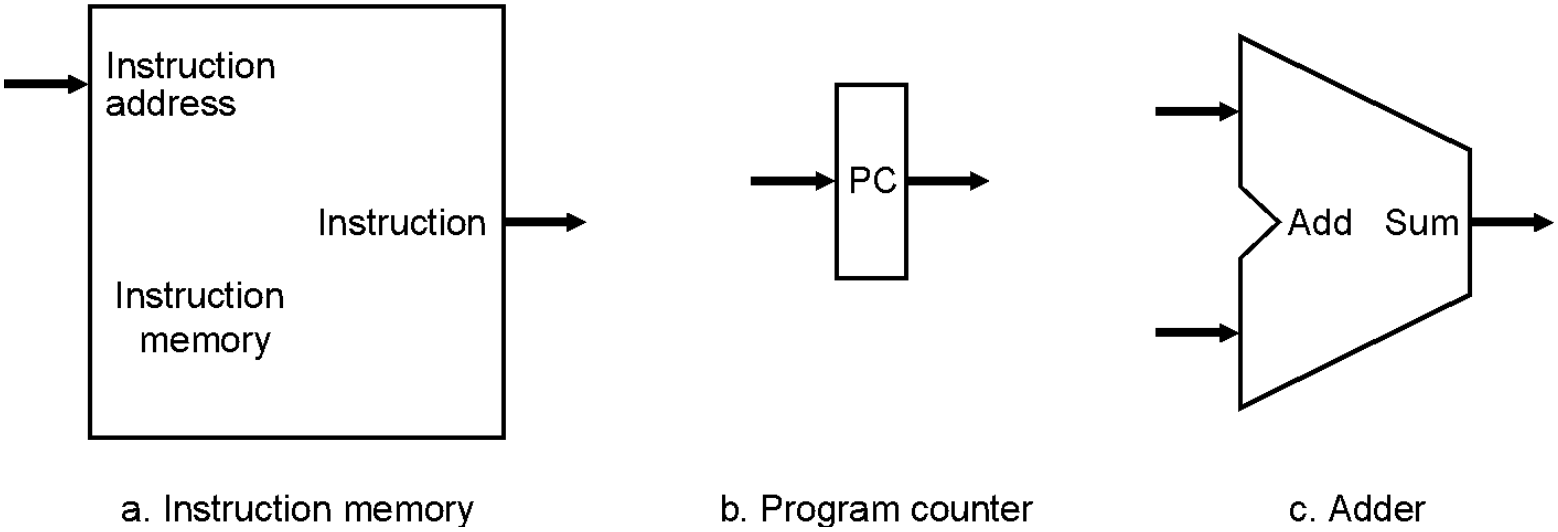


RTL: Register Transfer Language

- Describes the movement and manipulation of data between storage elements:

R[3] <- R[5] + R[7]
PC <- PC + 4 + R[5]
R[rd] <- R[rs] + R[rt]
R[rt] <- Mem[R[rs] + immed]

Instruction Fetch and Program Counter Management



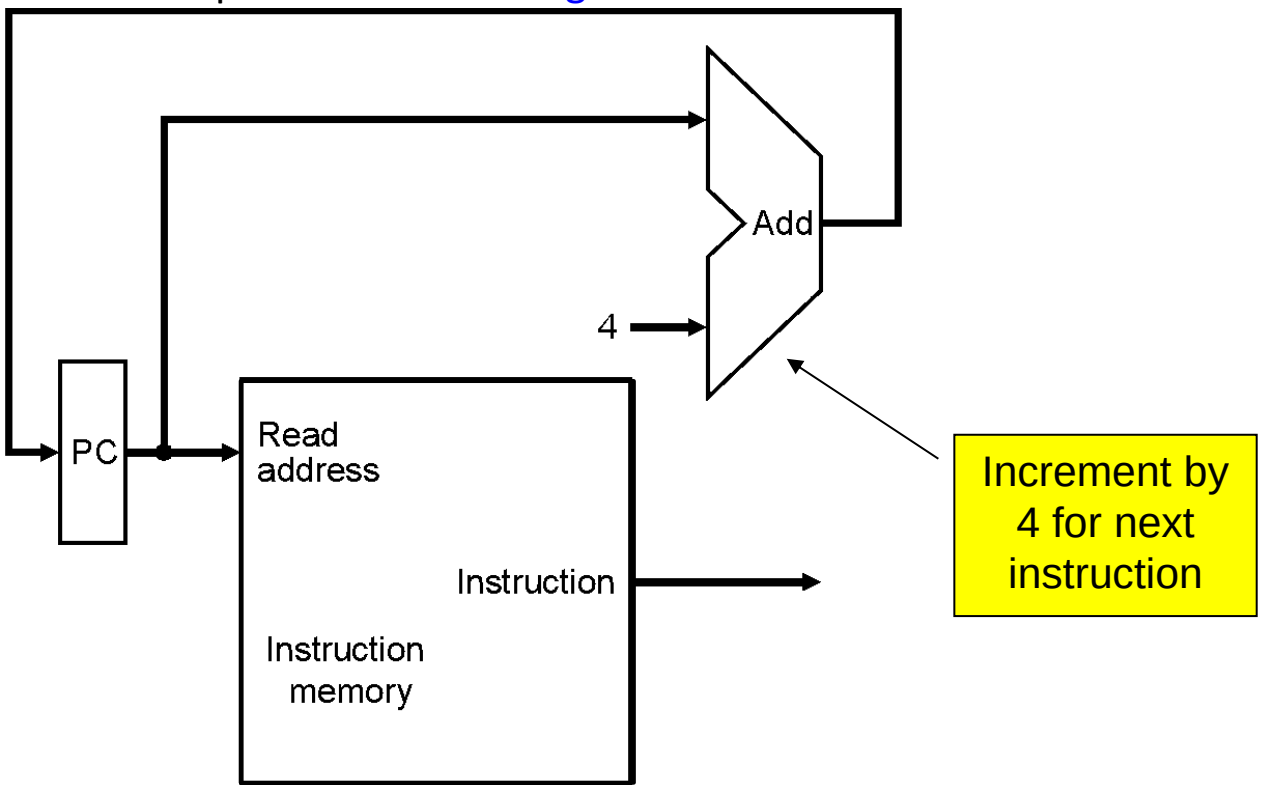
a. Instruction memory

b. Program counter

c. Adder

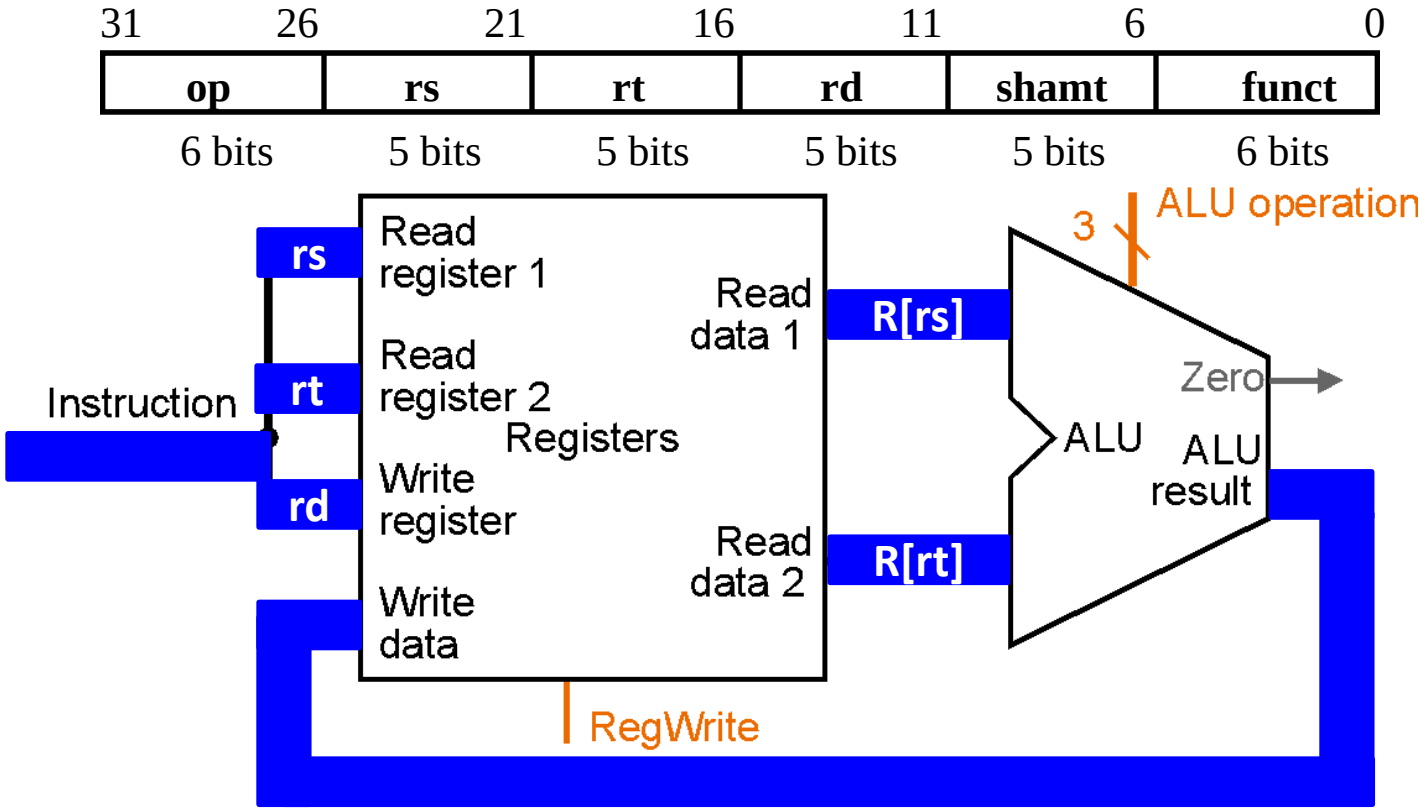
Overview of the Instruction Fetch Unit

- The common RTL operations
  - Fetch the Instruction: inst <- mem[PC]
  - Update the program counter:
    - Sequential Code: PC <- PC + 4
    - Branch and Jump PC <- “something else”



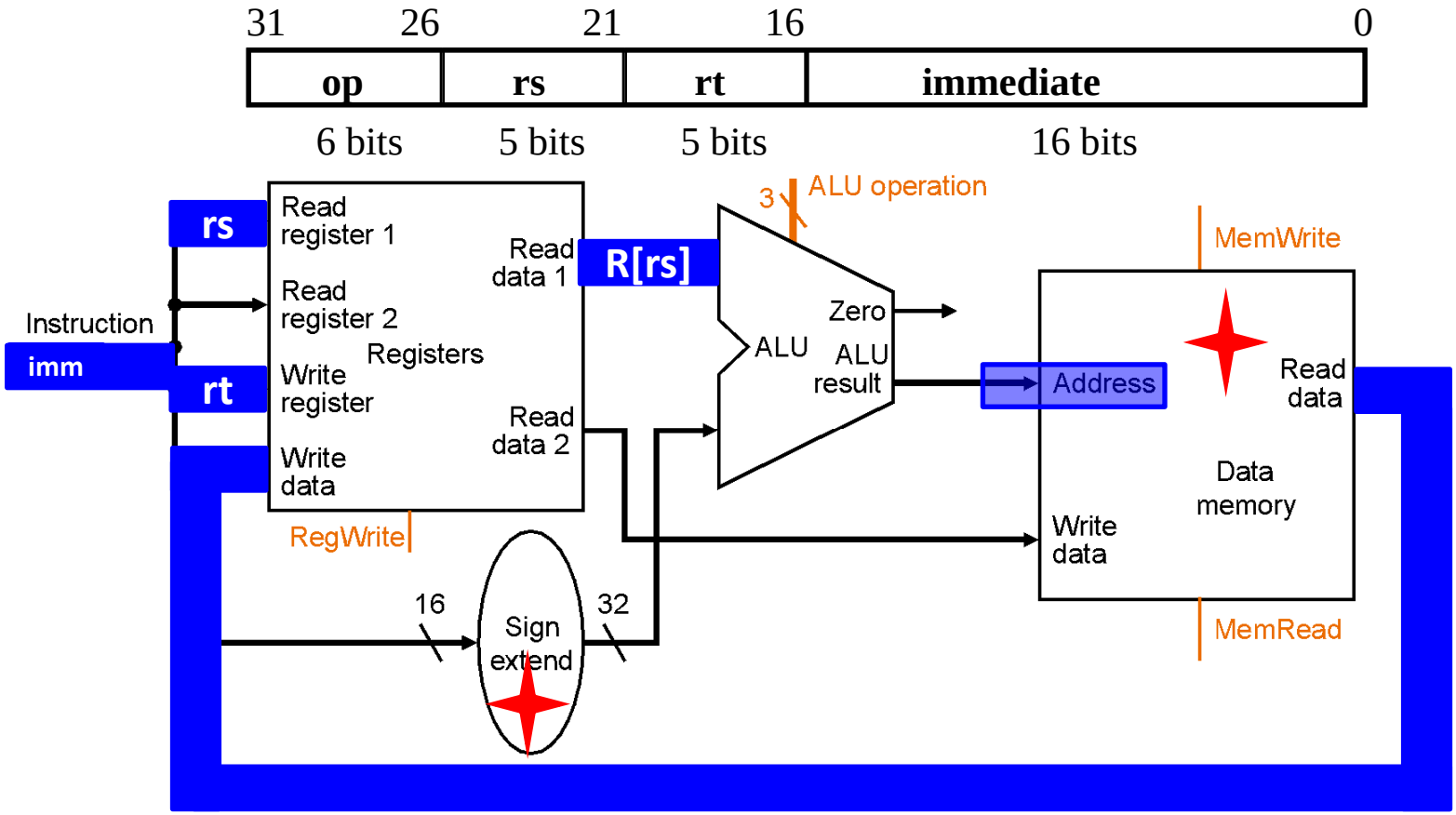
Datapath for Register-Register Operations

- R[rd] <- R[rs] op R[rt] Example: *add rd, rs, rt*
  - RR1, RR2, and WR comes from instruction's rs, rt, and rd fields
  - ALUoperation and RegWrite: control logic after decoding instruction



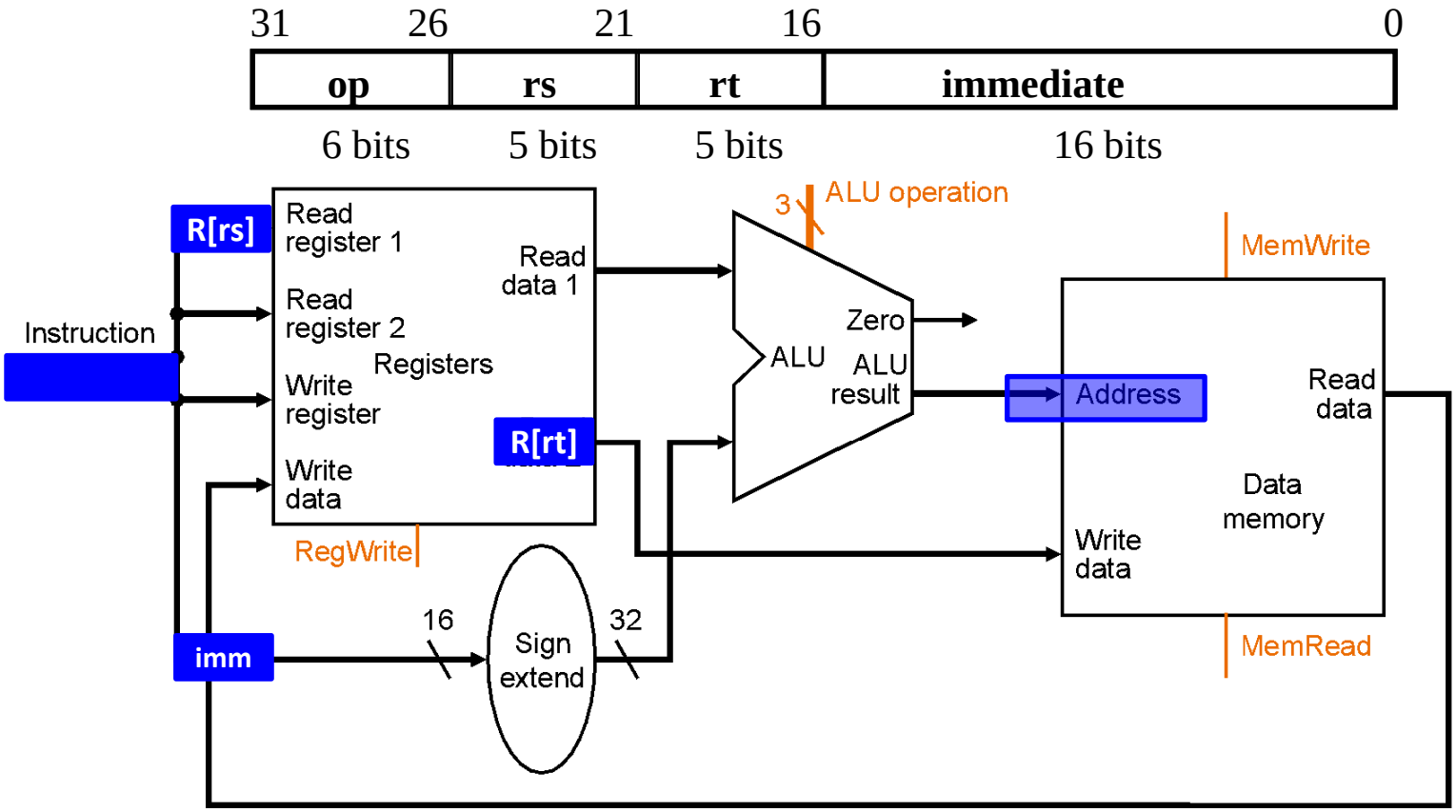
Datapath for Load Operations

- R[rt] <- Mem[R[rs] + SignExt[imm16]]
  - Example: *lw rt, rs, imm16*



Datapath for Store Operations

- Mem[R[rs] + SignExt[imm16]] <- R[rt]
  - Example: *sw rt, rs, imm16*

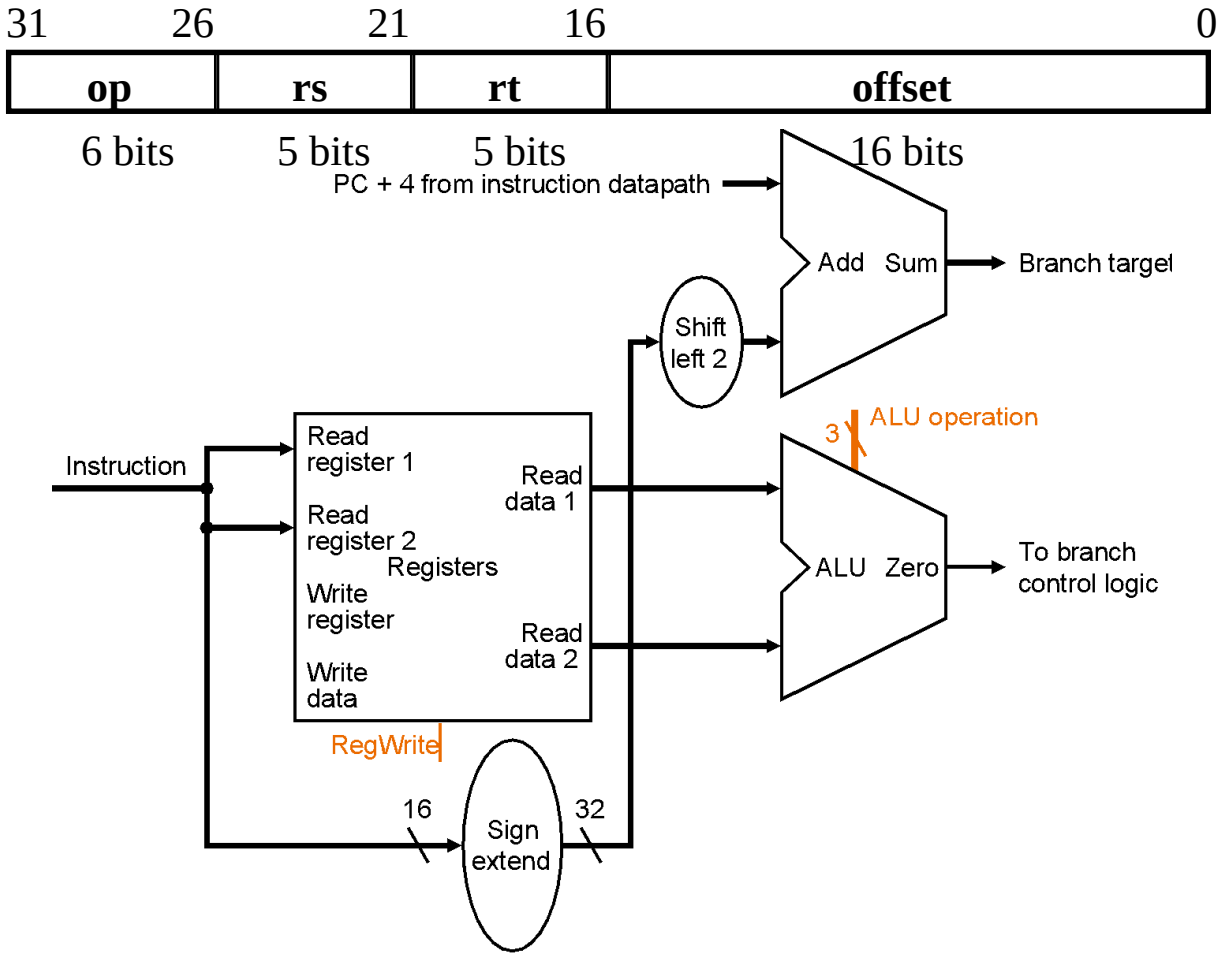


Datapath for Branch Operations

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

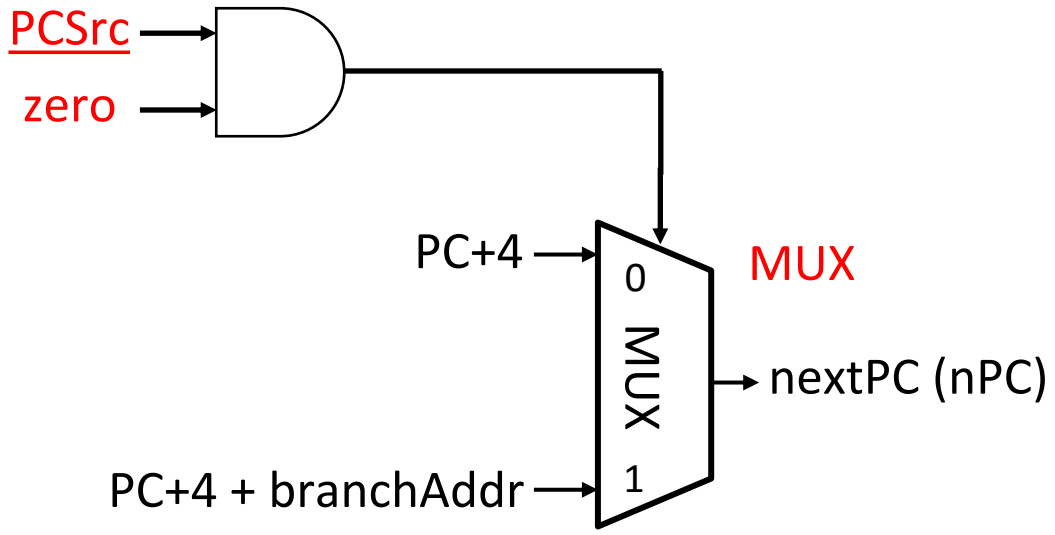
Datapath for Branch Operations

- Z <- (rs == rt); if Z, PC = PC+4+imm16; else PC = PC+4
  - Example: *beq rs, rt, imm16*



Datapath for Branch Operations

- Revisit "next address logic":
  - PCSrc should be 1 if branch, 0 otherwise



PCSrc	zero	MUX
0	0	0
0	1	0
1	0	0
1	1	1

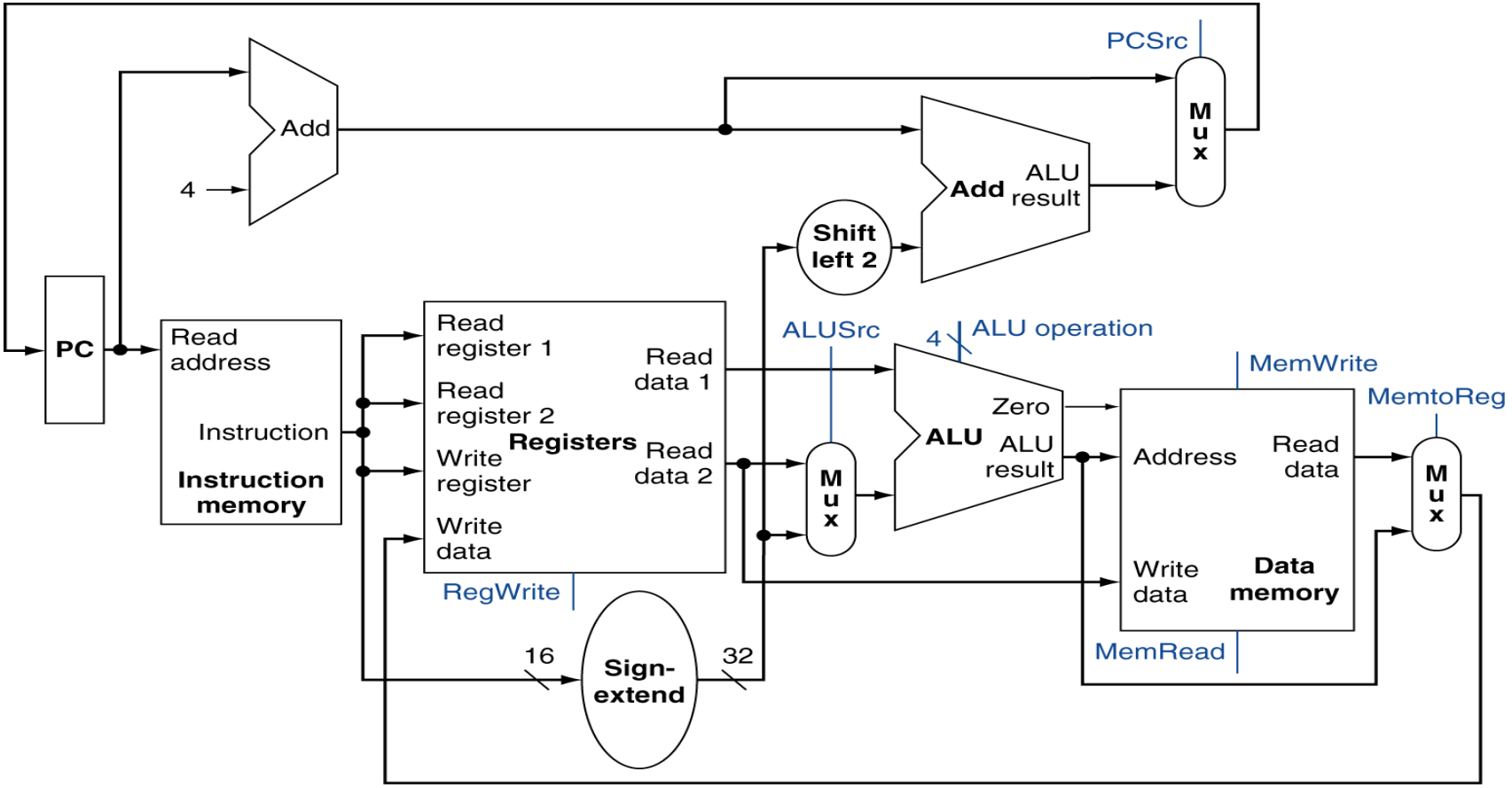
How does this change if we add `bne`*?*

Binary Arithmetic for the Next Address

- In theory, the PC is a 32-bit byte address into the instruction memory:
  - Sequential operation: PC<31:0> = PC<31:0> + 4
  - Branch operation: PC<31:0> = PC<31:0> + 4 + SignExt[Imm16] * 4
- The magic number "4" always comes up because:
  - The 32-bit PC is a byte address
  - And all our instructions are 4 bytes (32 bits) long
  - The 2 LSBs (Least Significant Bit) of the 32-bit PC are always zeros
  - There is no reason to have hardware to keep the 2 LSBs
- In practice, we can simplify the hardware by using a 30-bit PC<31:2>:
  - Sequential operation: PC<31:2> = PC<31:2> + 1
  - Branch operation: PC<31:2> = PC<31:2> + 1 + SignExt[Imm16]
  - In either case: Instruction Memory Address = PC<31:2> concat "00"

Putting it All Together: A Single Cycle Datapath

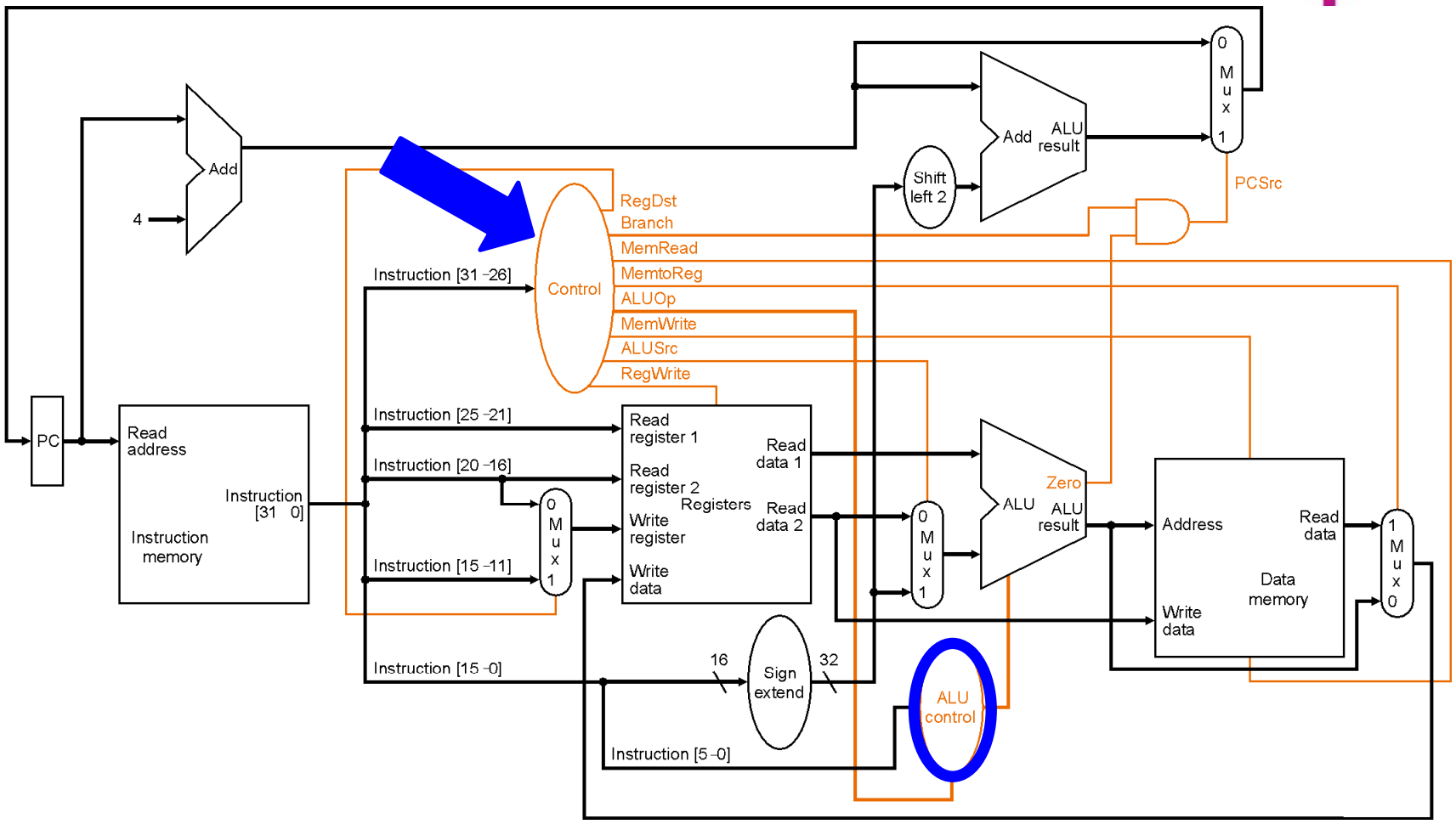
- We have everything except control signals



Key Points

- CPU is just a collection of state and combinational logic
- We just designed a very rich processor, at least in terms of functionality
- ET = IC × CPI × Cycle Time
  - where does the single-cycle machine fit in?

Okay, then, what about those Control Signals?

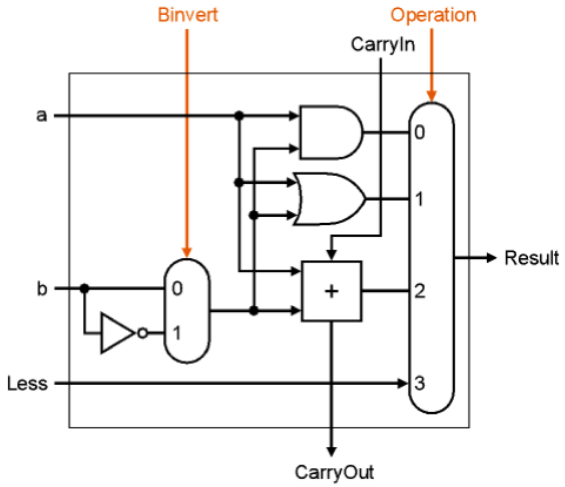


ALU control bits

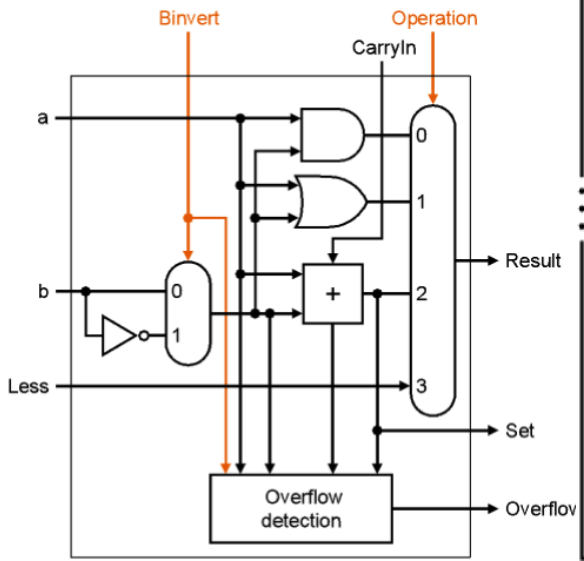
- 5-Function ALU

ALU control input	Function	Operations
000	And	and
001	Or	or
010	Add	add, lw, sw
110	Subtract	sub, beq
111	Slt	slt

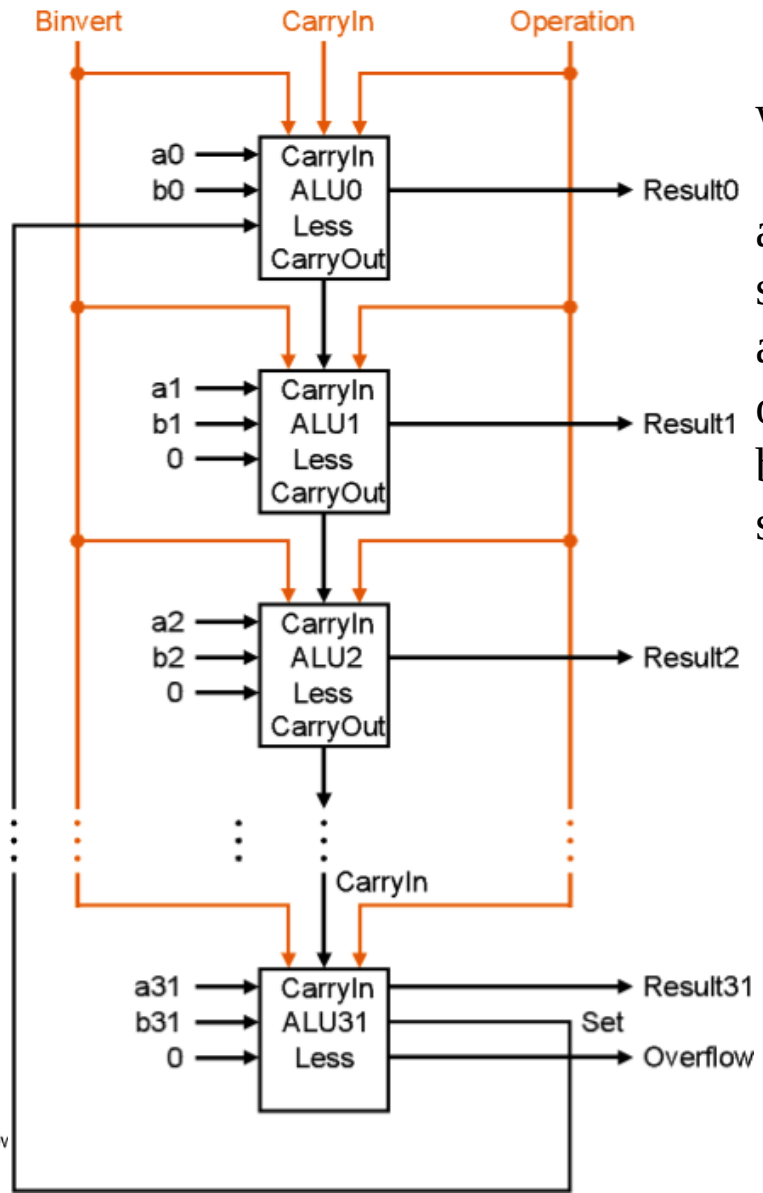
Full ALU



a.

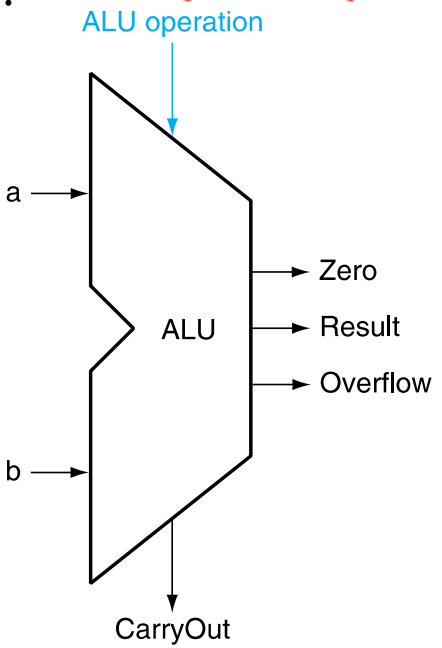


b.



what signals accomplish:

	Binvert	CIn	Oper
add?	0	0	10
sub?	1	1	10
and?	0	0	00
or?	0	0	01
beq?	1	1	10
slt?	1	1	11

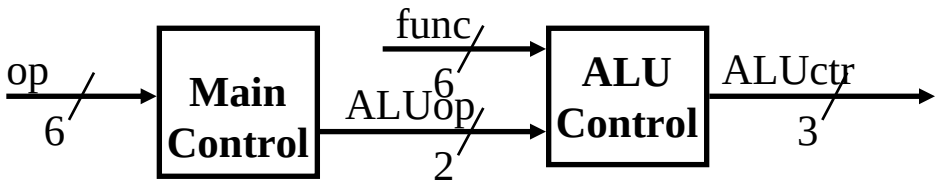


ALU control bits

- 5-Function ALU

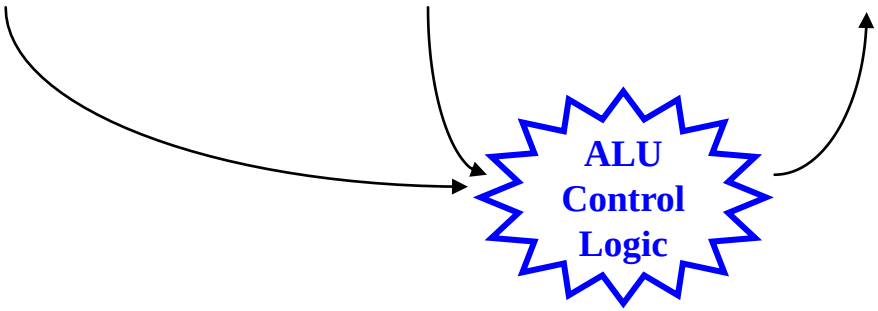
ALU control input	Function	Operations
000	And	and
001	Or	or
010	Add	add, lw, sw
110	Subtract	sub, beq
111	Slt	slt

- based on opcode (bits 31-26) and function code (bits 5-0) from instruction
- ALU doesn't need to know all opcodes--we will summarize opcode with ALUOp (2 bits):
  - **00 - lw,sw** **01 – beq** **10 - R-format**

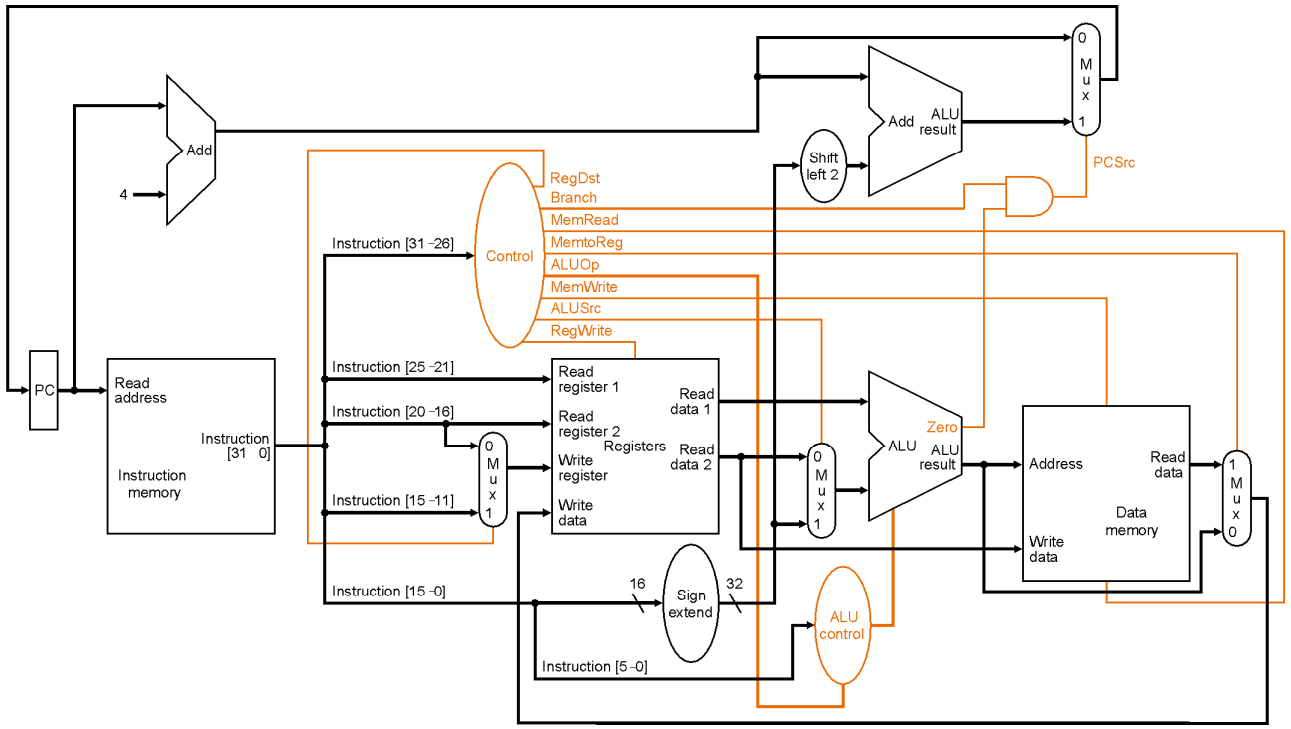


Generating ALU Control

Instruction opcode	ALUOp	Instruction operation	Function code	Desired ALU action	ALU control input
lw	00	load word	xxxxxx	add	010
sw	00	store word	xxxxxx	add	010
beq	01	branch eq	xxxxxx	subtract	110
R-type	10	add	100000	add	010
R-type	10	subtract	100010	subtract	110
R-type	10	AND	100100	and	000
R-type	10	OR	100101	or	001
R-type	10	slt	101010	slt	111



Controlling the CPU



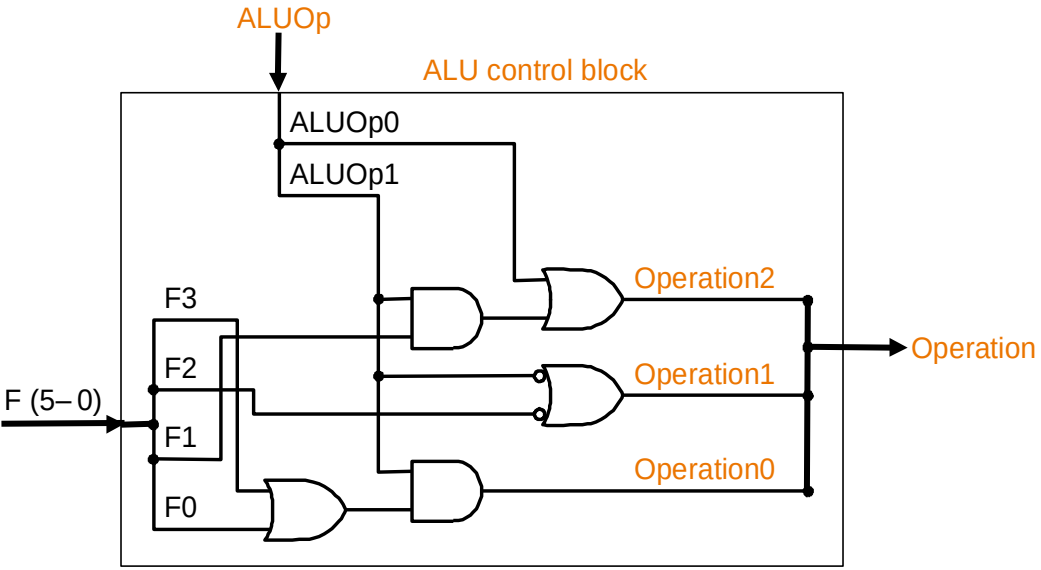
Instruction	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUp0
R-format	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

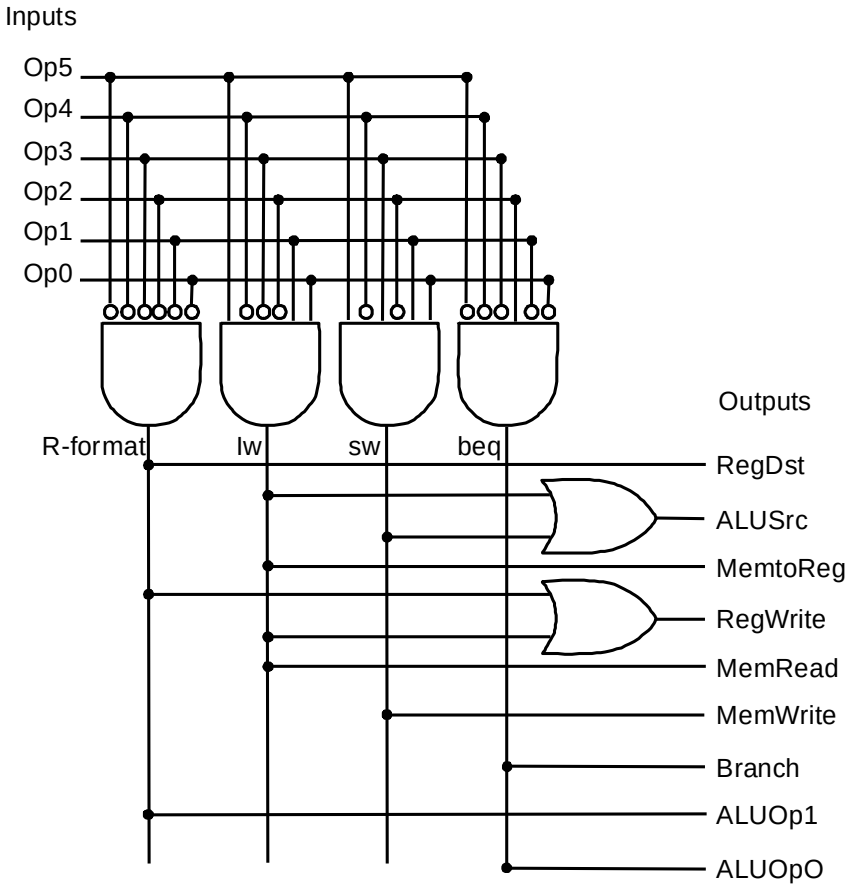
Control Truth Table

		R-format	lw	sw	beq
Opcode		000000	100011	101011	000100
Outputs	RegDst	1	0	x	x
	ALUSrc	0	1	1	0
	MemtoReg	0	1	x	x
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1

Control

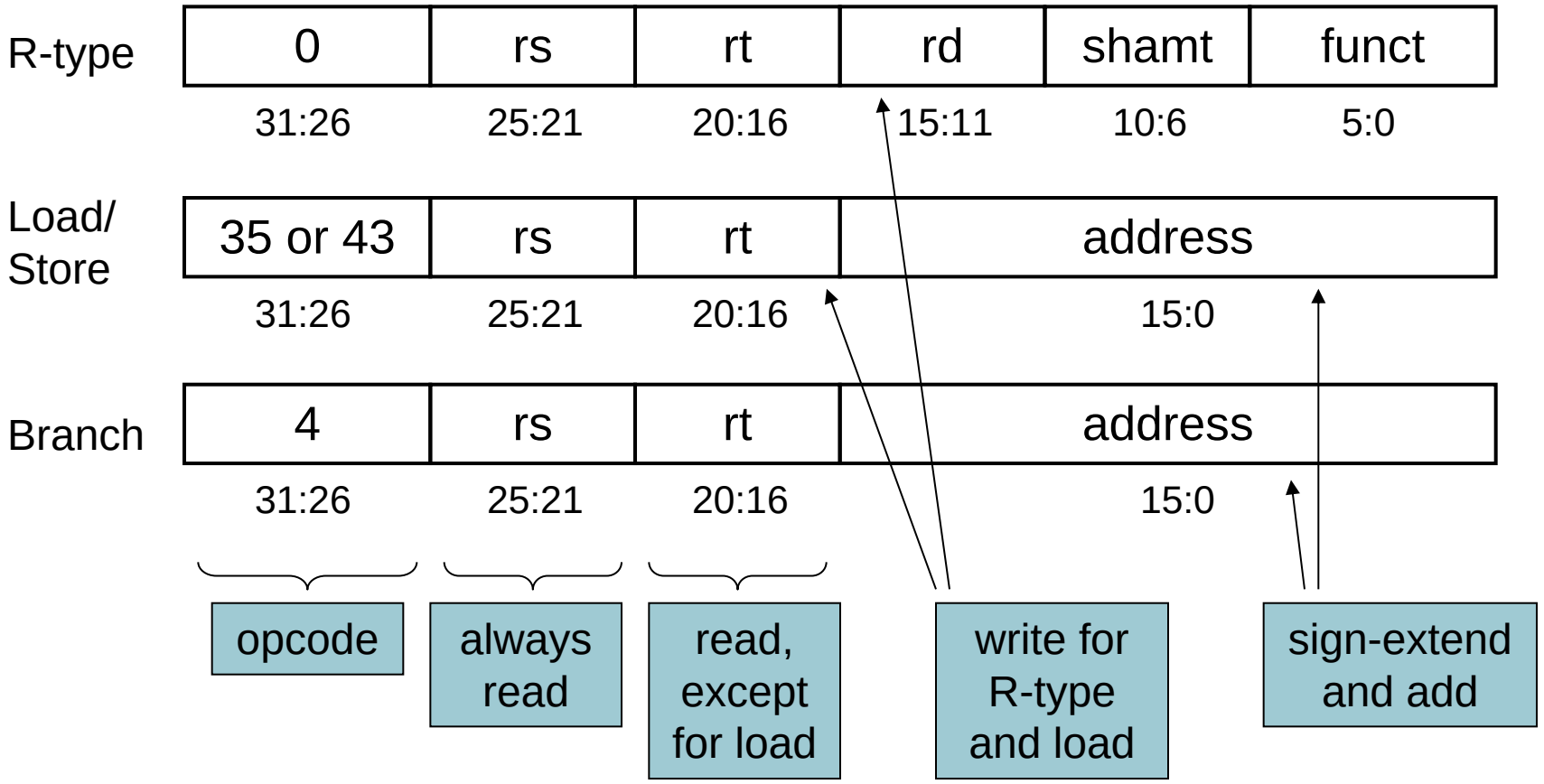
- Simple combinational logic (truth tables)



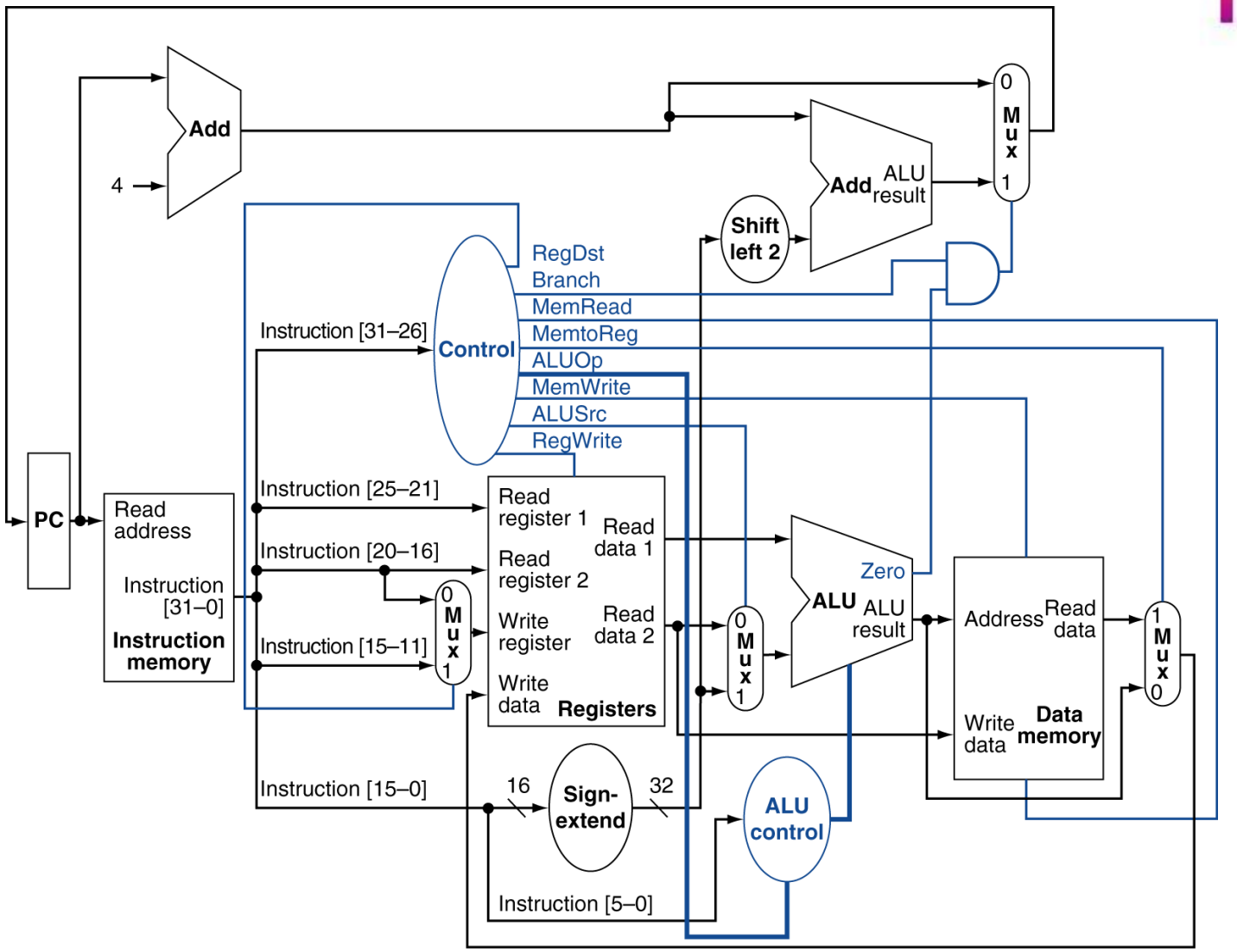


The Main Control Unit

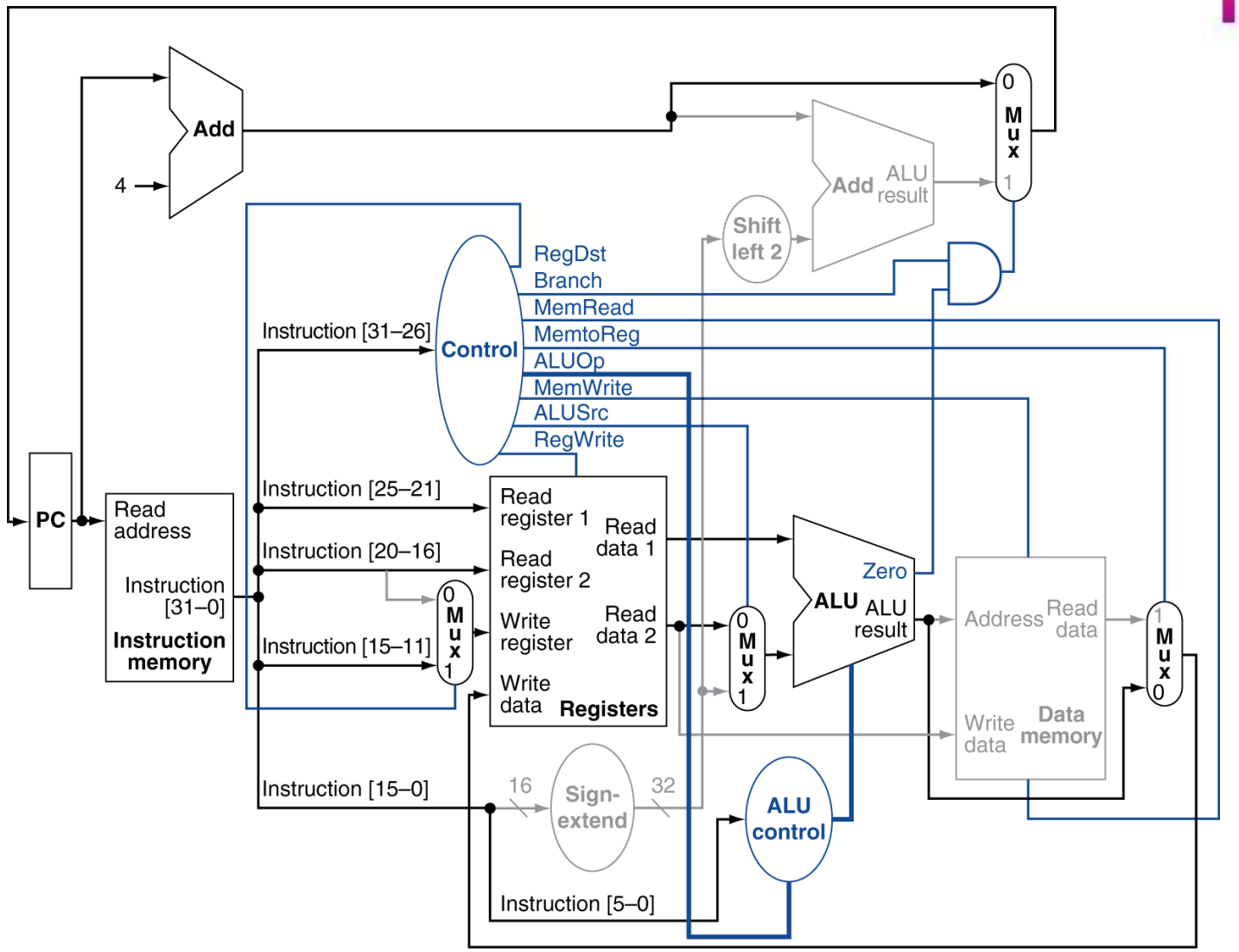
- Control signals derived from instruction



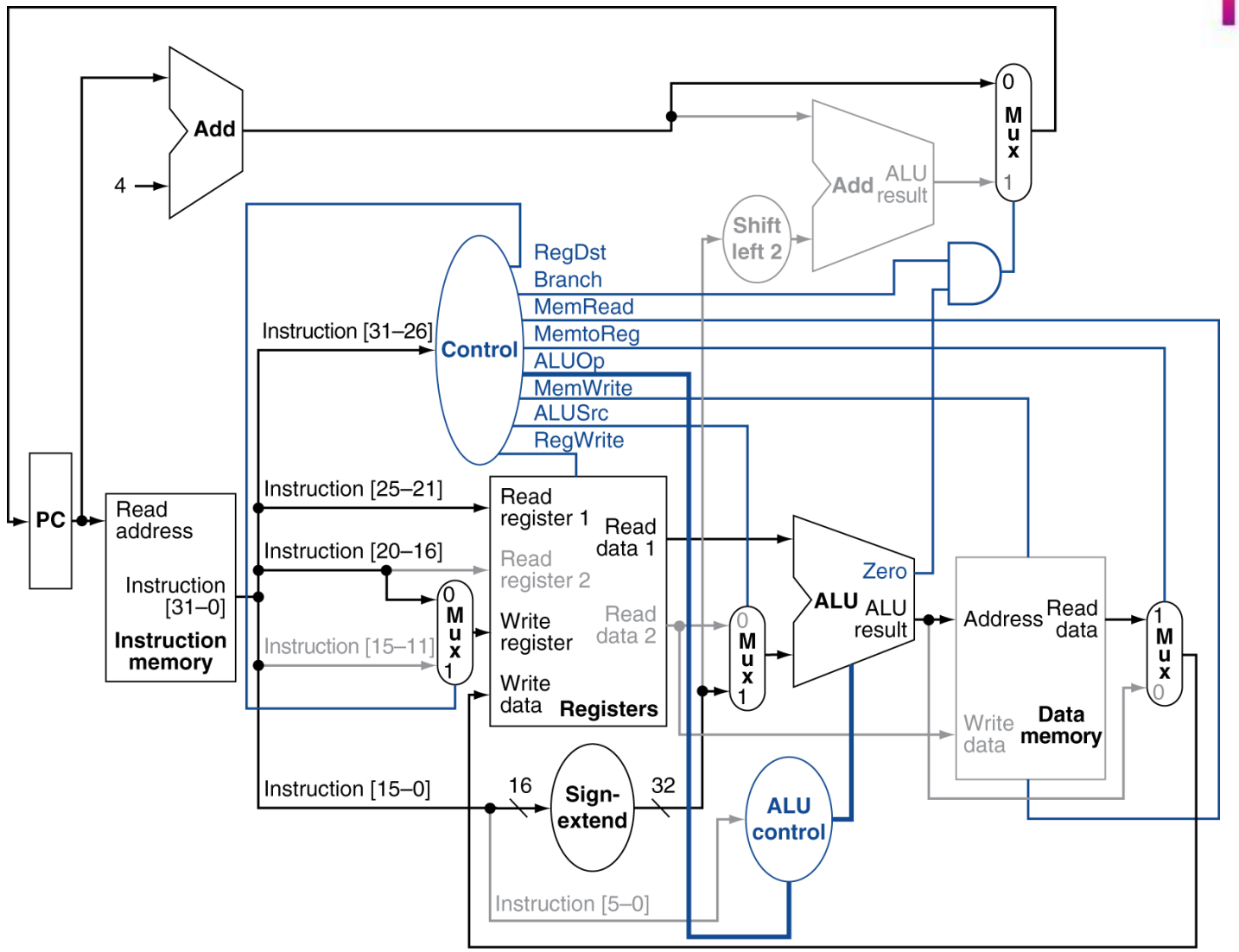
Datapath With Control



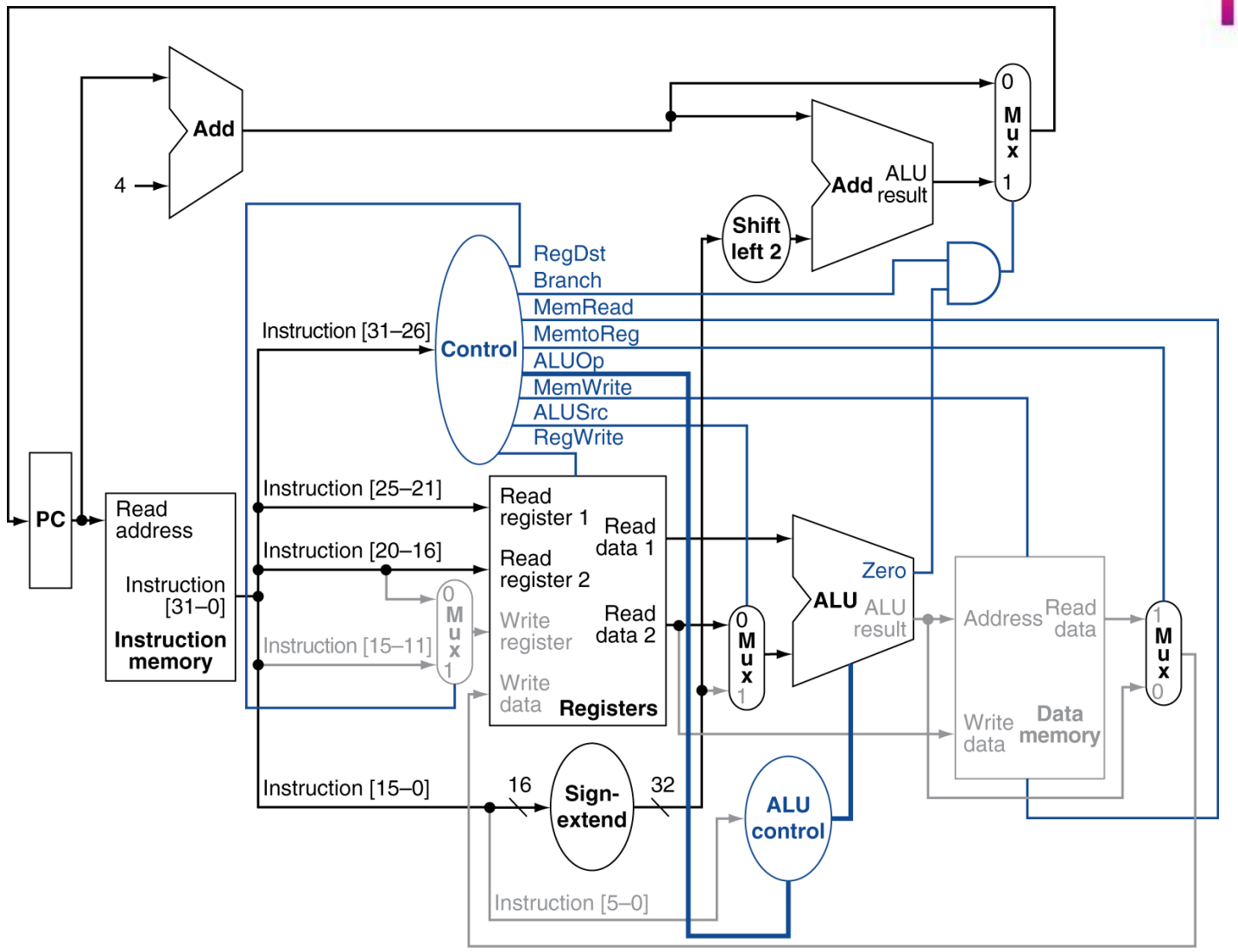
R-Type Instruction



Load Instruction

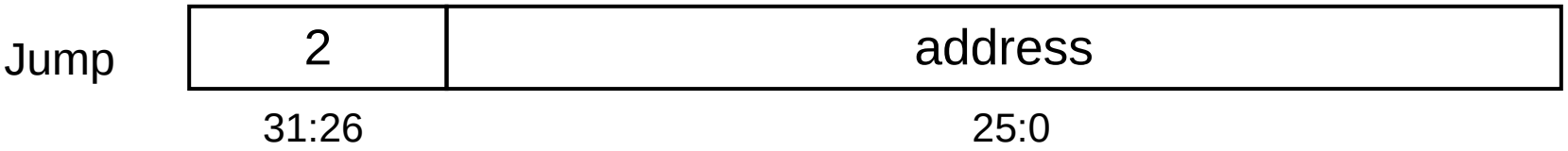


Branch-on-Equal Instruction



Implementing Jumps

- Jump uses word address
- Update PC with concatenation of
  - Top 4 bits of old PC
  - 26-bit jump address
  - 00
- Need an extra control signal decoded from opcode



Datapath With Jumps Added