C335
Computer Structures

Designing A MIPS Single Cycle Datapath (II)

Dr. Liqiang Zhang

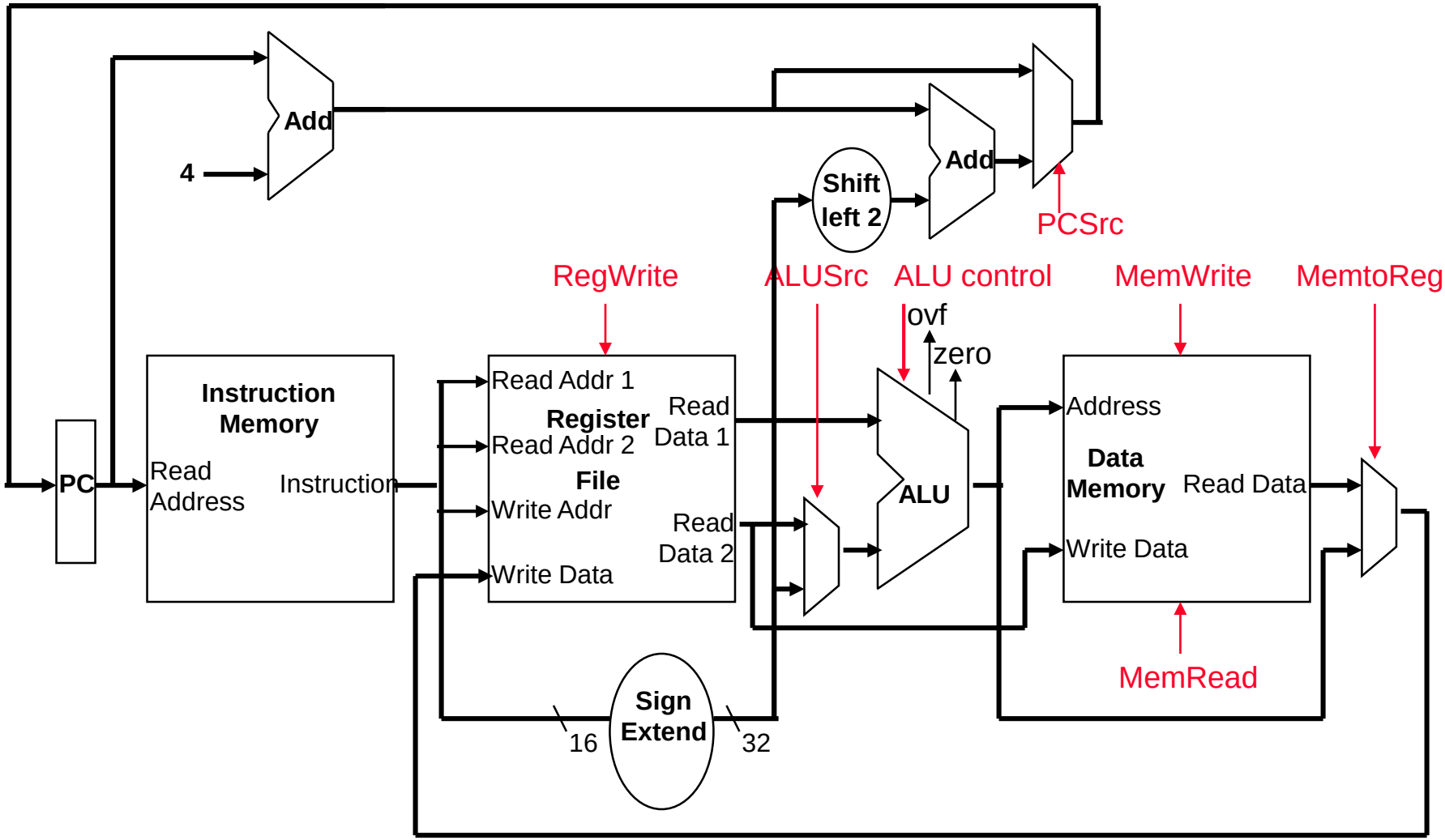
Department of Computer and Information Sciences

Adapted from Morgan Kaufmann and others

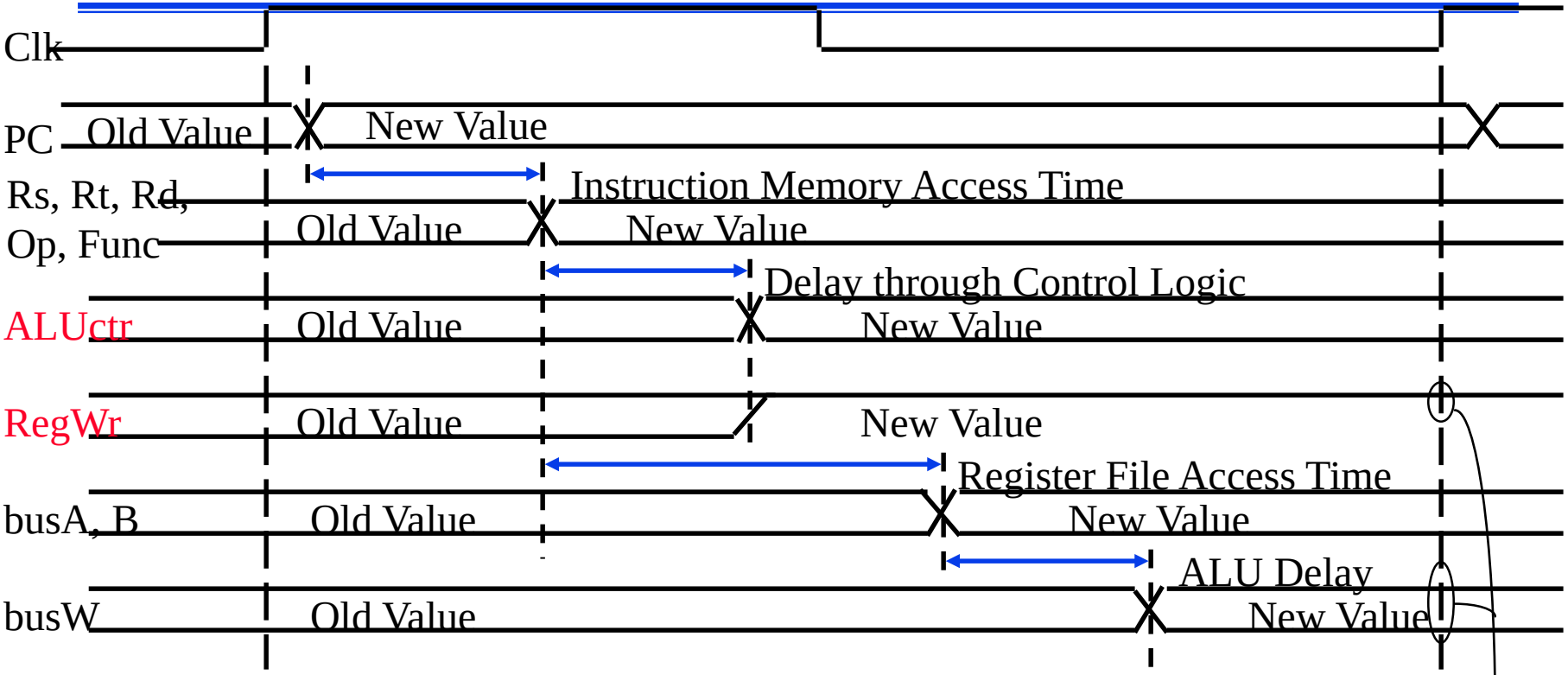
Review: Creating a Datapath from the Parts

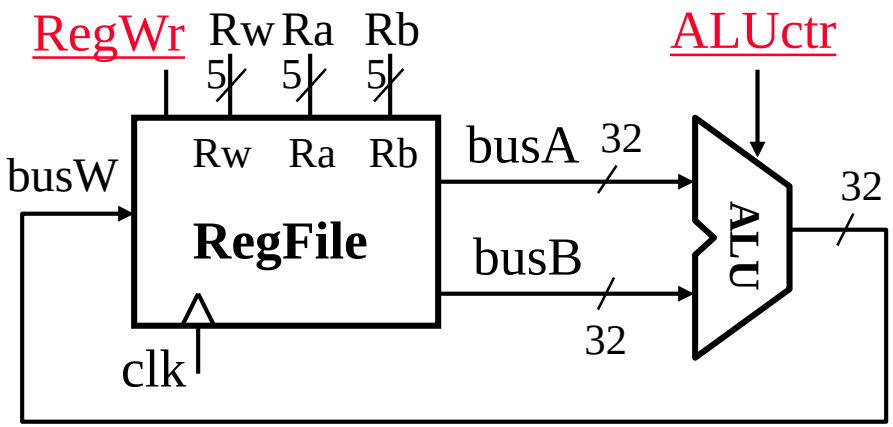
- Assemble the datapath elements, add control lines as needed, and design the control path
- Fetch, decode and execute each instruction in one clock cycle – single cycle design
  - no datapath resource can be used more than once per instruction, so some must be duplicated (e.g., why we have a separate Instruction Memory and Data Memory)
  - to share datapath elements between two different instruction classes need multiplexors at the input of the shared elements with control lines to do the selection
- Cycle time is determined by length of the longest path

Review: A Simple MIPS Datapath Design



Review: Register-Register Timing: One complete cycle

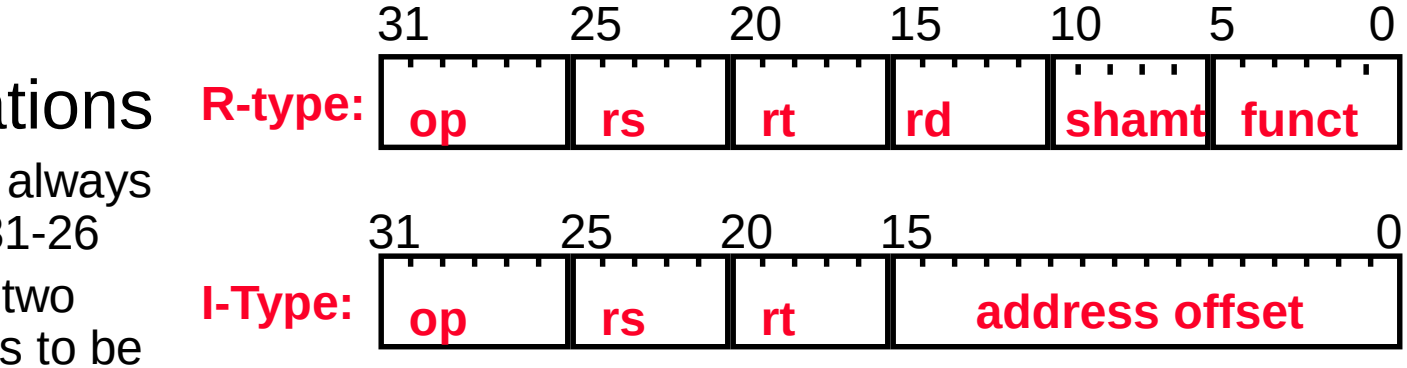




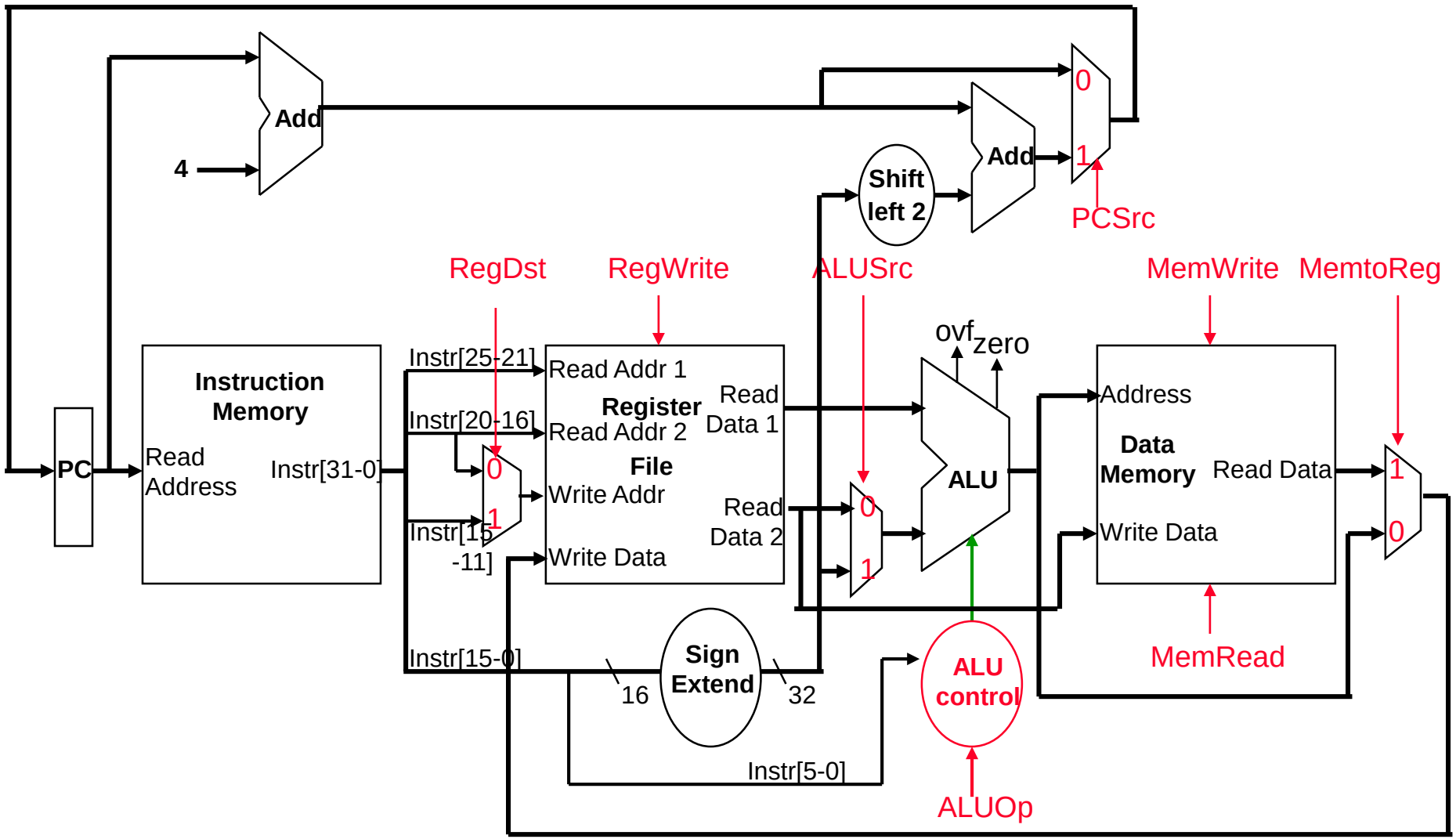
Register Write Occurs Here

Adding the Control

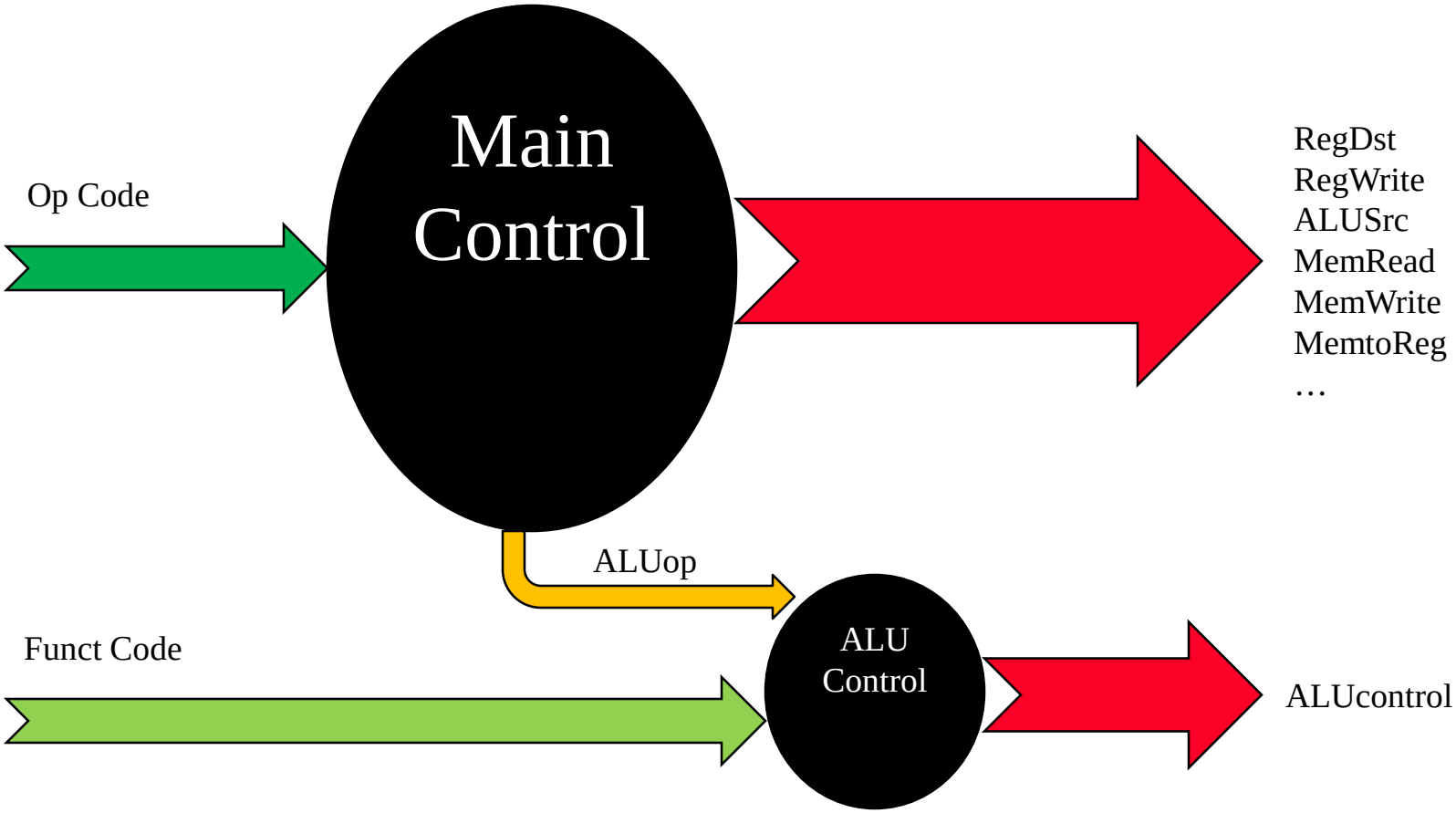
- Selecting the operations to perform (ALU, Register File and Memory read/write)
- Controlling the flow of data (multiplexor inputs)
- Information comes from the 32 bits of the instruction
- Observations
  - op field always in bits 31-26
  - addr of two registers to be read are always specified by the rs and rt fields (bits 25-21 and 20-16)
  - base register for lw and sw always in rs (bits 25-21)
  - addr. of register to be written is in one of two places – in rt (bits 20-16) for lw; in rd (bits 15-11) for R-type instructions
  - offset for beq, lw, and sw always in bits 15-0



(Almost) Complete Single Cycle Datapath



Design the Control



ALU Control

- ALU's operation based on instruction type and function code

ALU control signals	Function
0000	and
0001	or
0010	xor
0011	nor
0110	add
1110	subtract
1111	set on less than

- Notice that we are using different encodings than in the book

ALU Control (Con't)

- Controlling the ALU uses of multiple decoding levels
  - main control unit generates the ALUOp bits
  - ALU control unit generates ALUcontrol bits

Instr op	funct	ALUOp	action	ALUcontrol
lw	xxxxxx	00	add	0110
sw	xxxxxx	00	add	0110
beq	xxxxxx	01	subtract	1110
add	100000	10	add	0110
subt	100010	10	subtract	1110
and	100100	10	and	0000
or	100101	10	or	0001
xor	100110	10	xor	0010
nor	100111	10	nor	0011
slt	101010	10	slt	1111

ALU Control Truth Table

Our ALU control signals

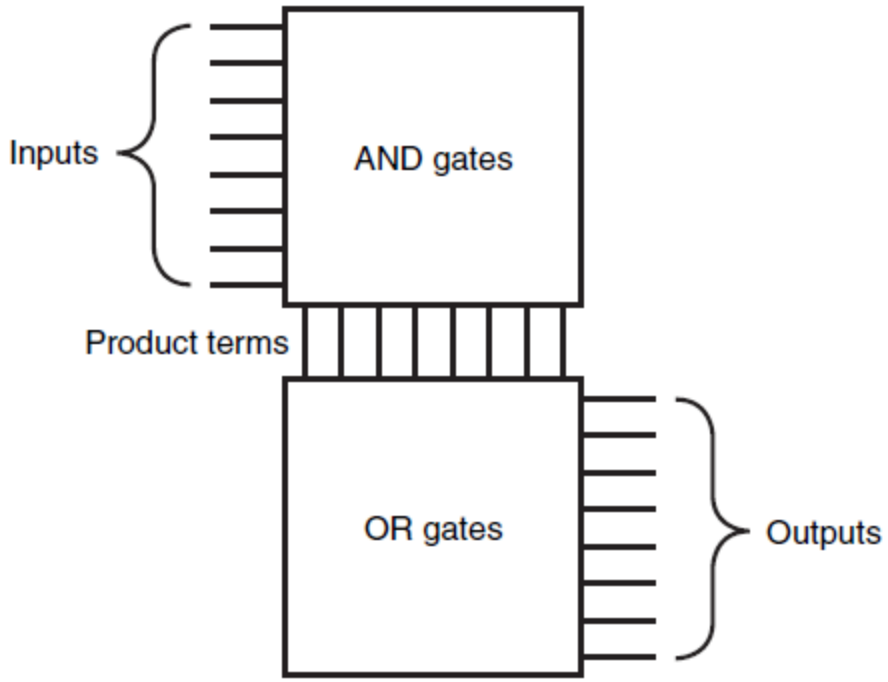
F5	F4	F3	F2	F1	F0	ALU Op_1	ALU Op_0	ALU $control_3$	ALU $control_2$	ALU $control_1$	ALU $control_0$
X	X	X	X	X	X	0	0	0	1	1	0
X	X	X	X	X	X	0	1	1	1	1	0
X	X	0	0	0	0	1	0	0	1	1	0
X	X	0	0	1	0	1	0	1	1	1	0
X	X	0	1	0	0	1	0	0	0	0	0
X	X	0	1	0	1	1	0	0	0	0	1
X	X	0	1	1	0	1	0	0	0	1	0
X	X	0	1	1	1	1	0	0	0	1	1
X	X	1	0	1	0	1	0	1	1	1	1

Add/subt

Mux control

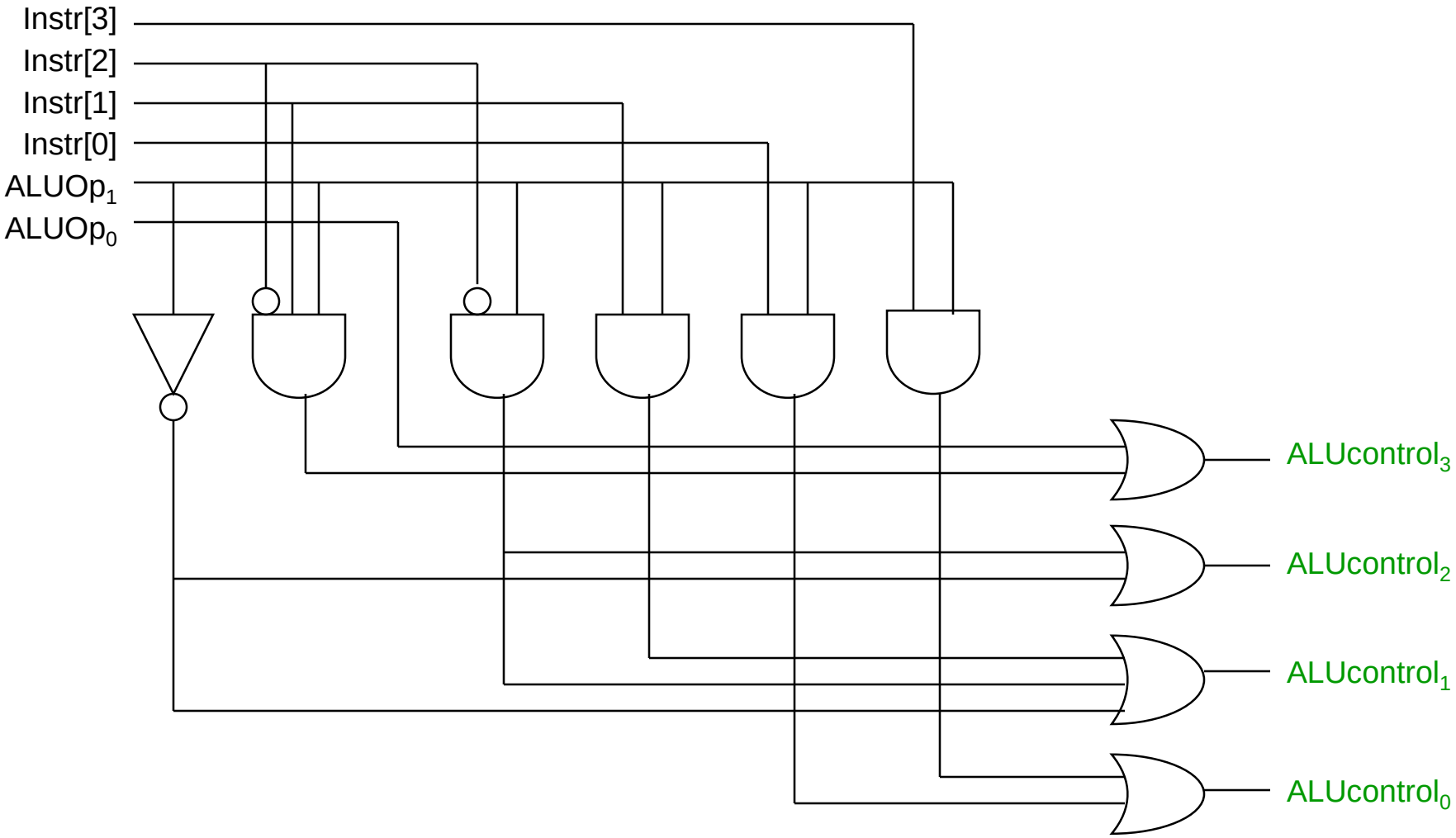
- Four, 6-input truth tables

PLA (Programmable Logic Array)

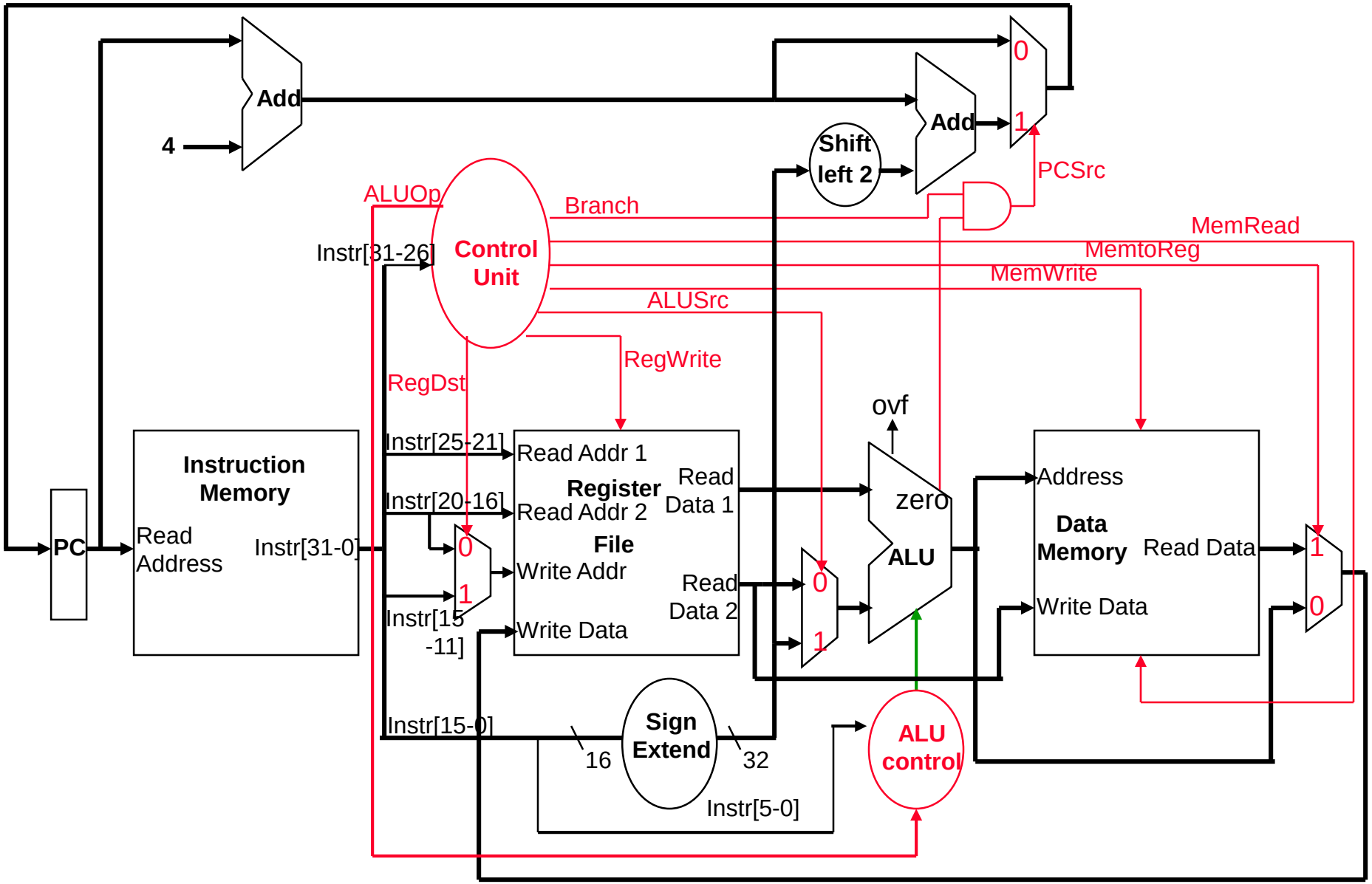


ALU Control Logic

- From the truth table can design the ALU Control logic



(Almost) Complete Datapath with Control Unit

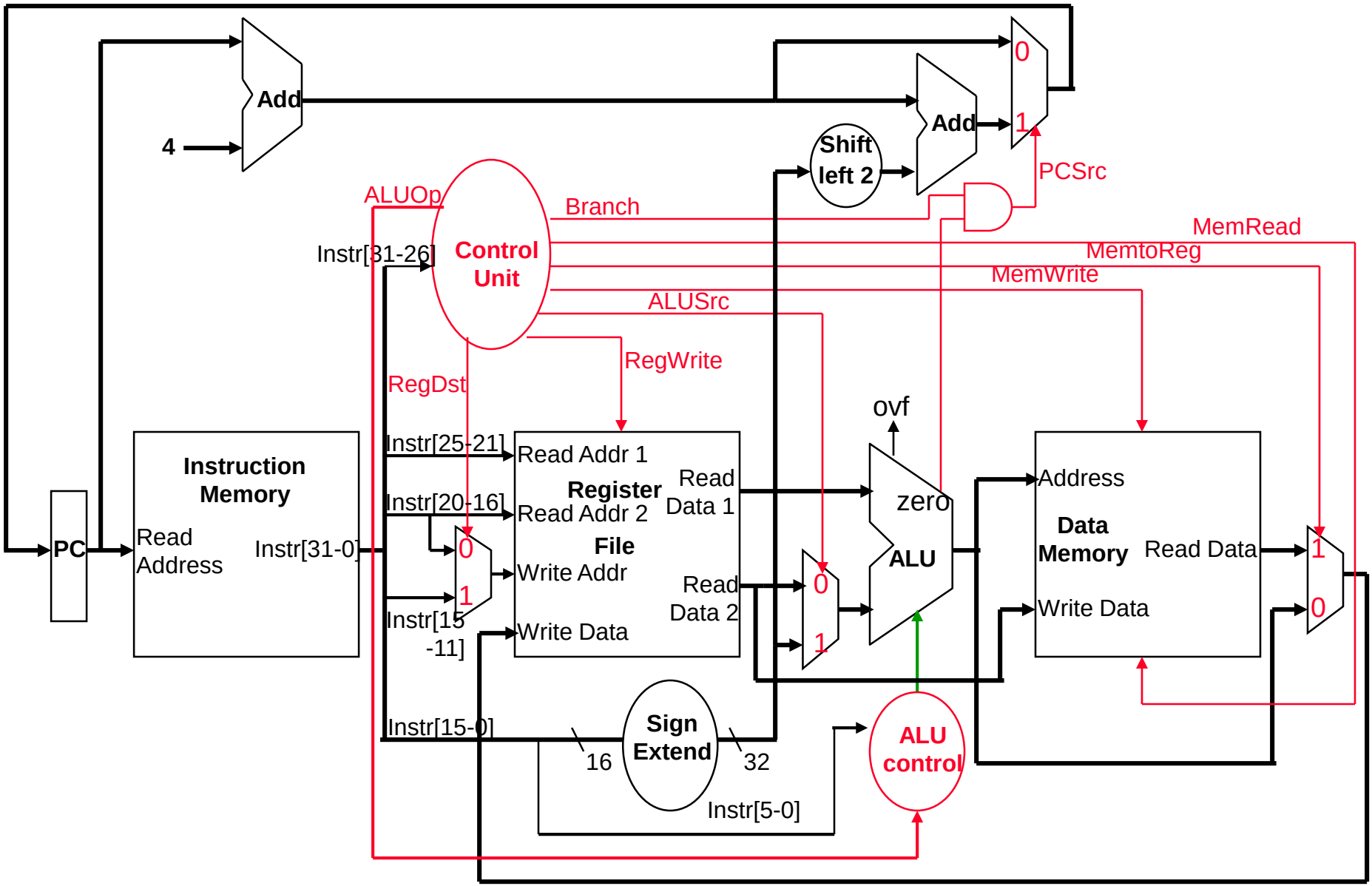


Main Control Unit

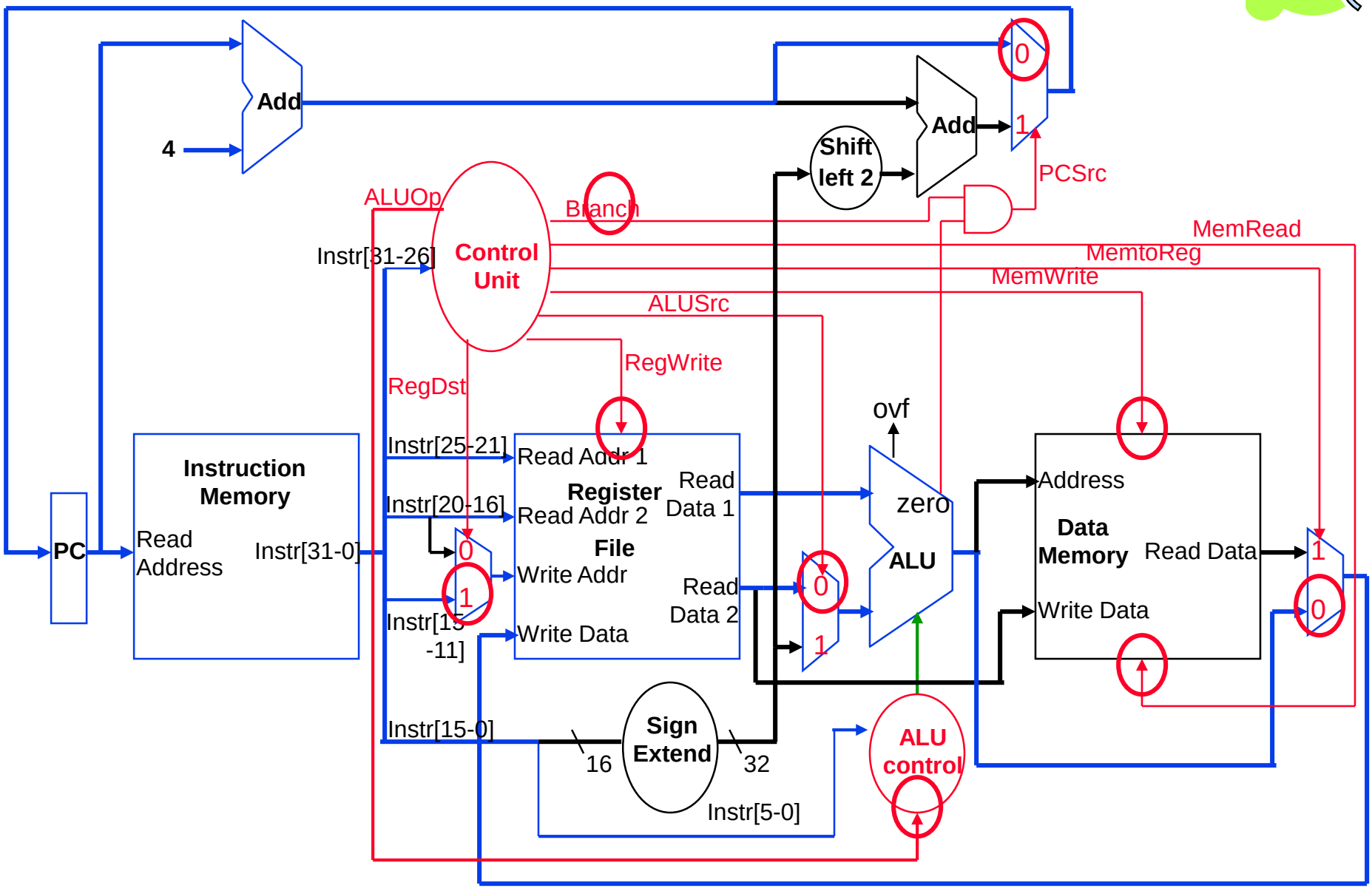
Instr	RegDst	ALUSrc	MemReg	RegWr	MemRd	MemWr	Branch	ALUOp
R-type 000000								
lw 100011								
sw 101011								
beq 000100								

- Completely determined by the instruction opcode field
  - Note that a multiplexor whose control input is 0 has a definite action, even if it is not used in performing the operation

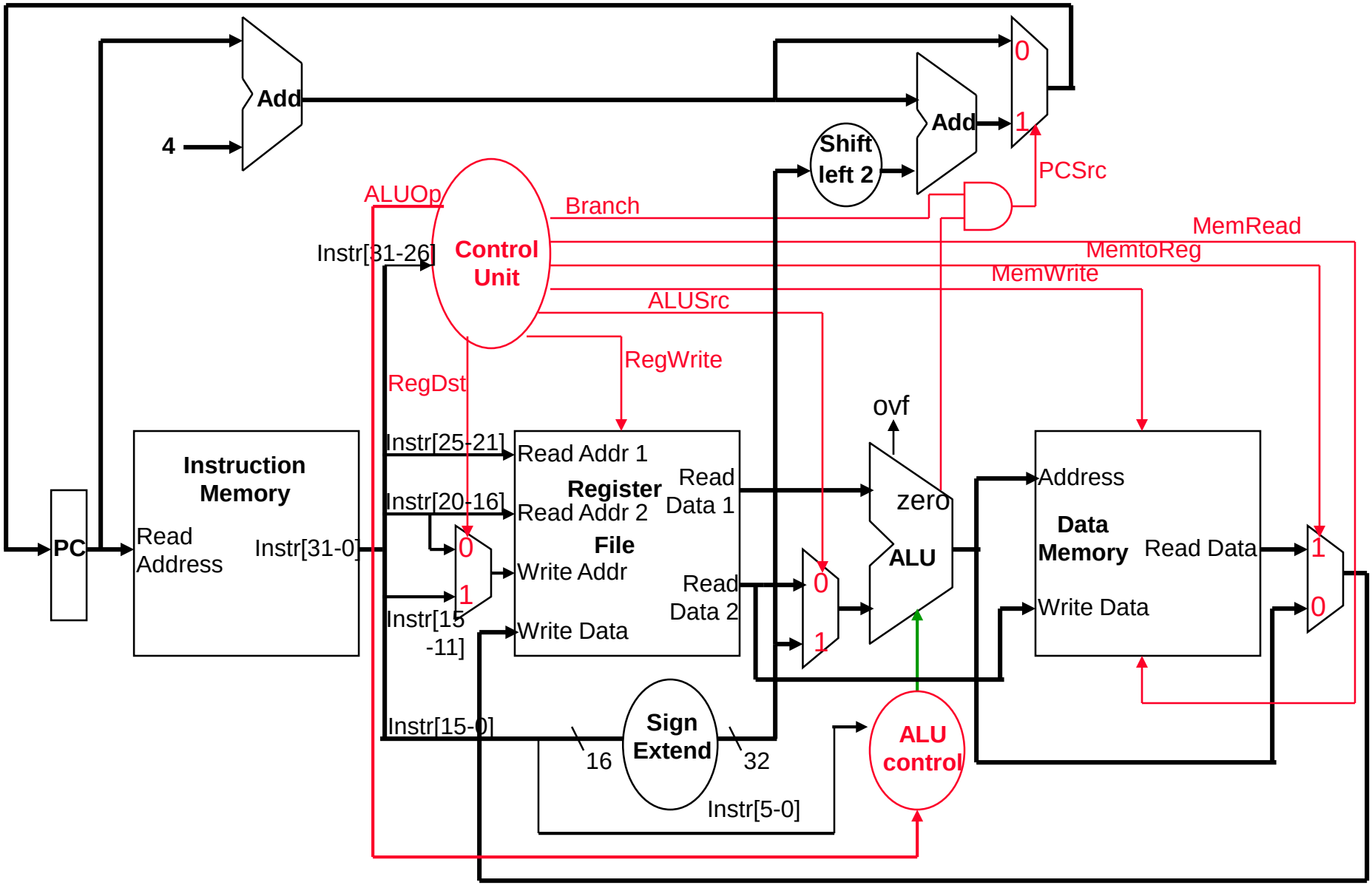
R-type Instruction Data/Control Flow



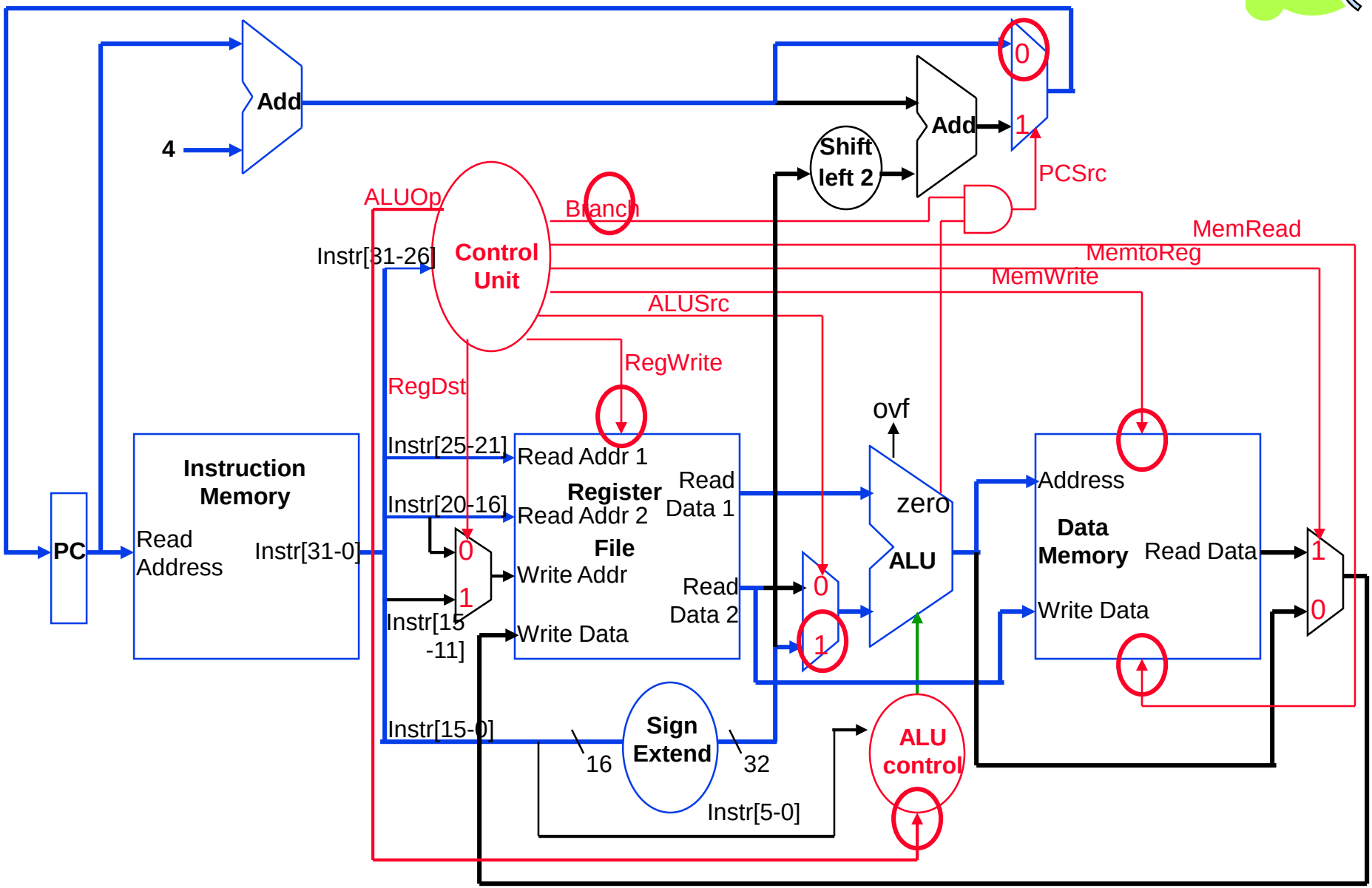
R-type Instruction Data/Control Flow



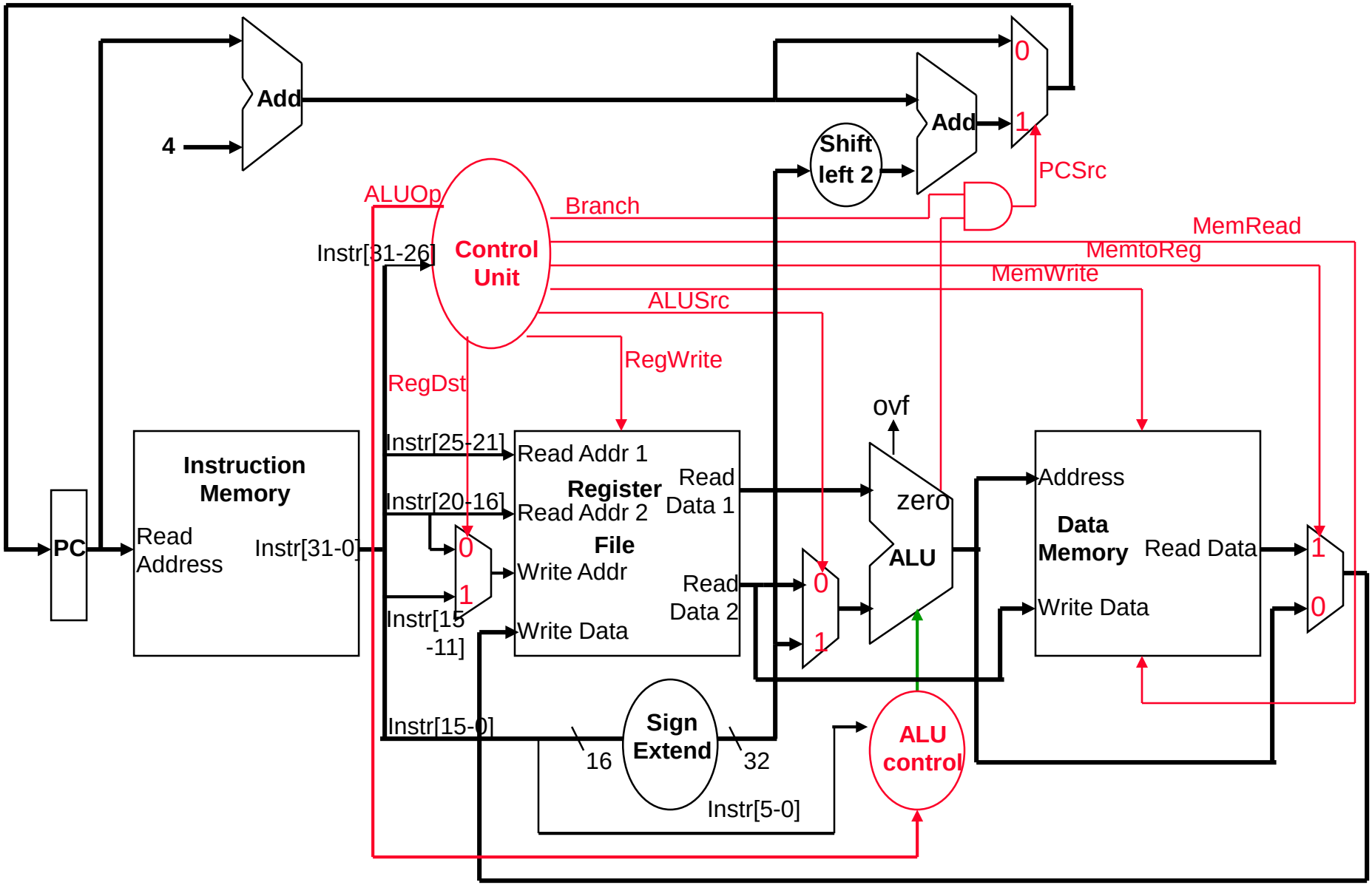
Store Word Instruction Data/Control Flow



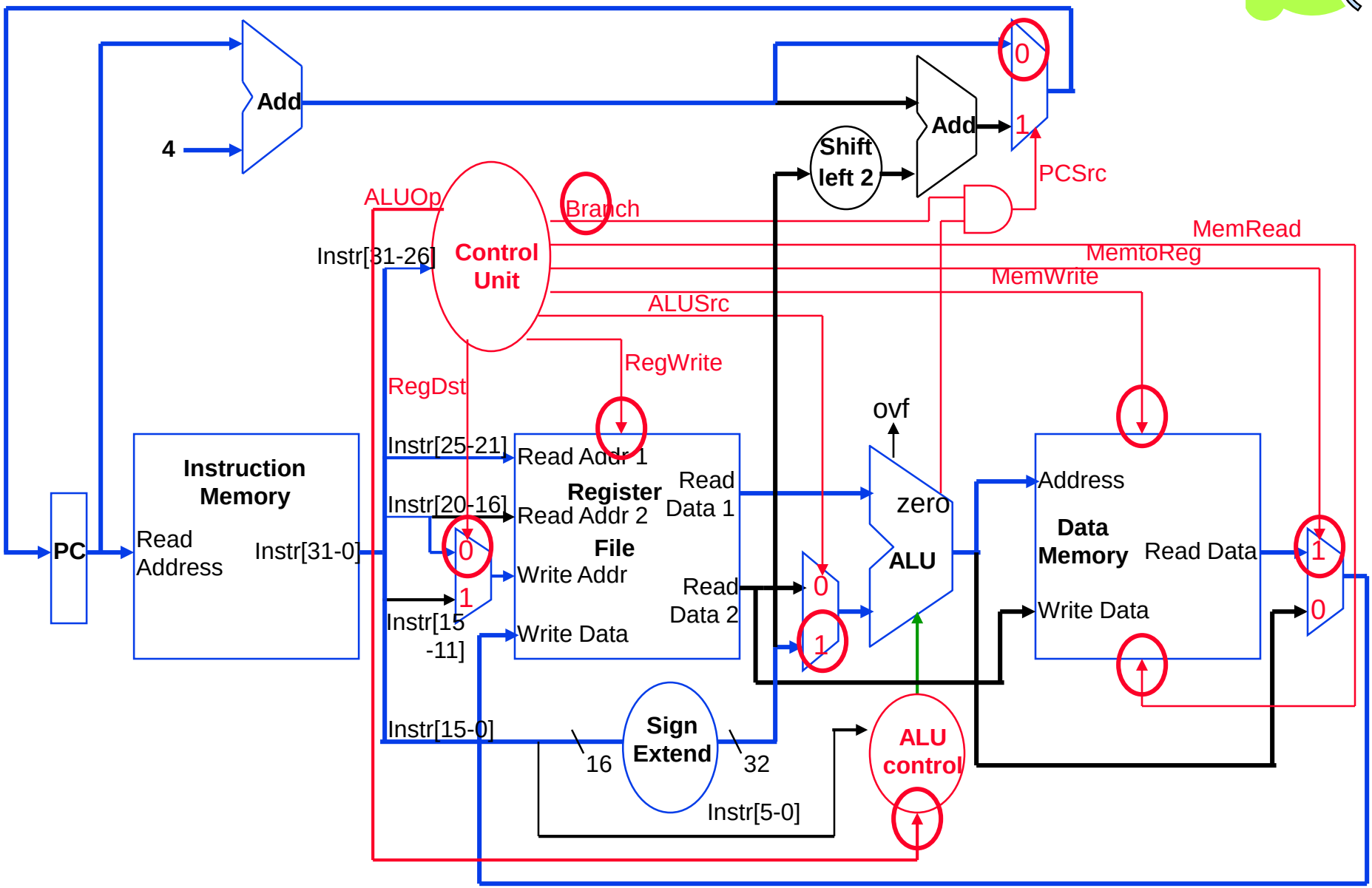
Store Word Instruction Data/Control Flow



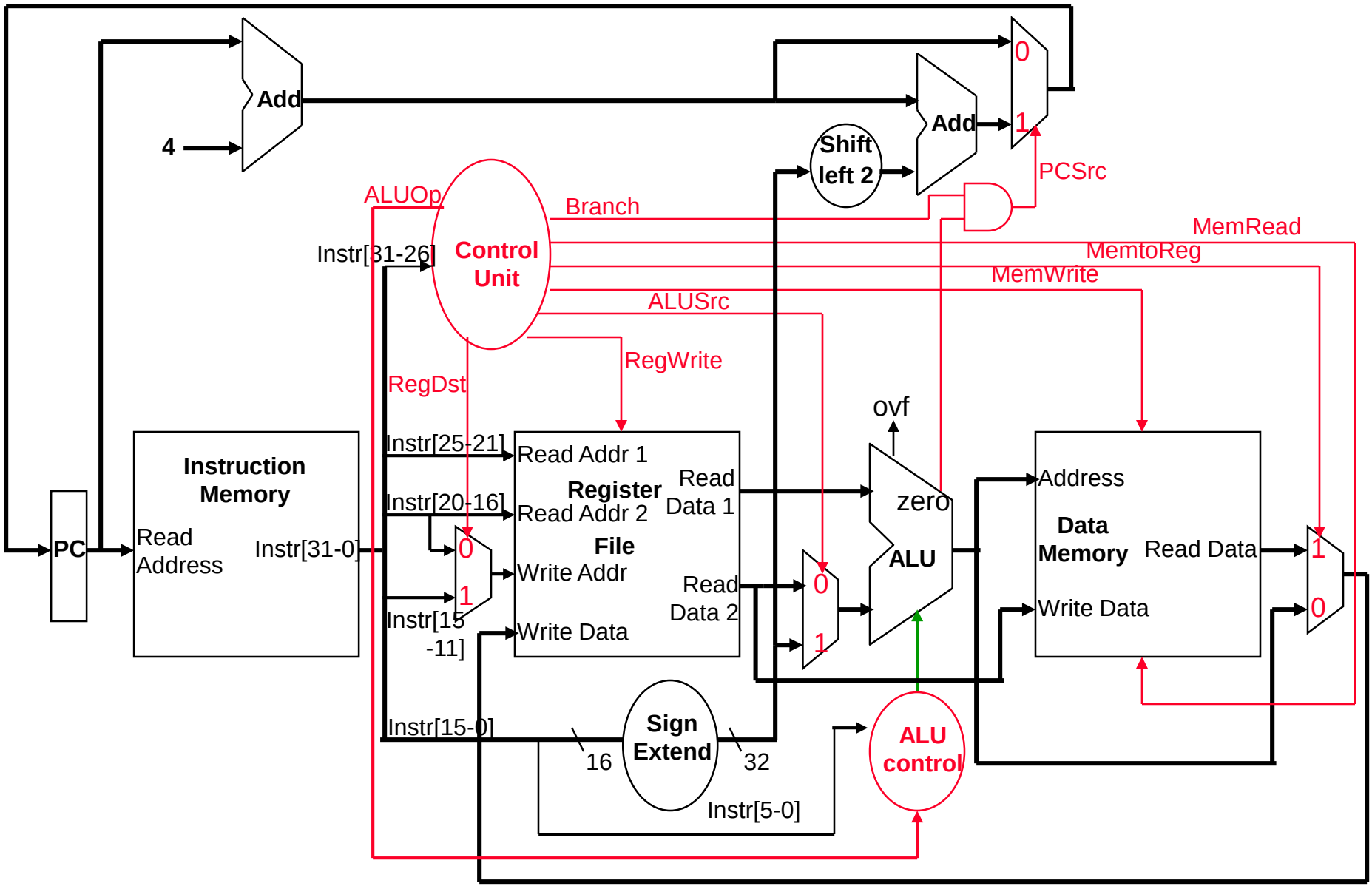
Load Word Instruction Data/Control Flow



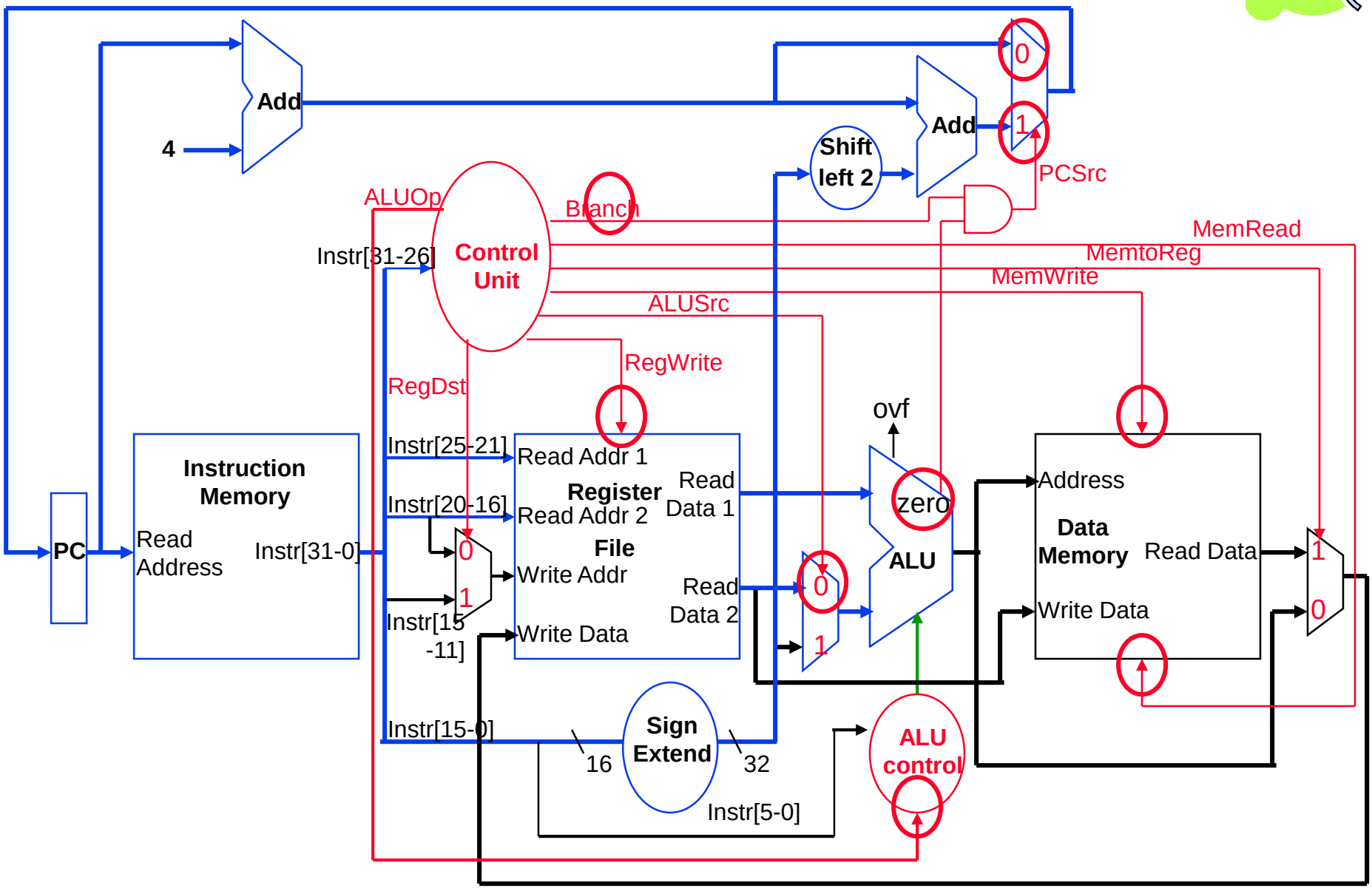
Load Word Instruction Data/Control Flow



Branch Instruction Data/Control Flow



Branch Instruction Data/Control Flow



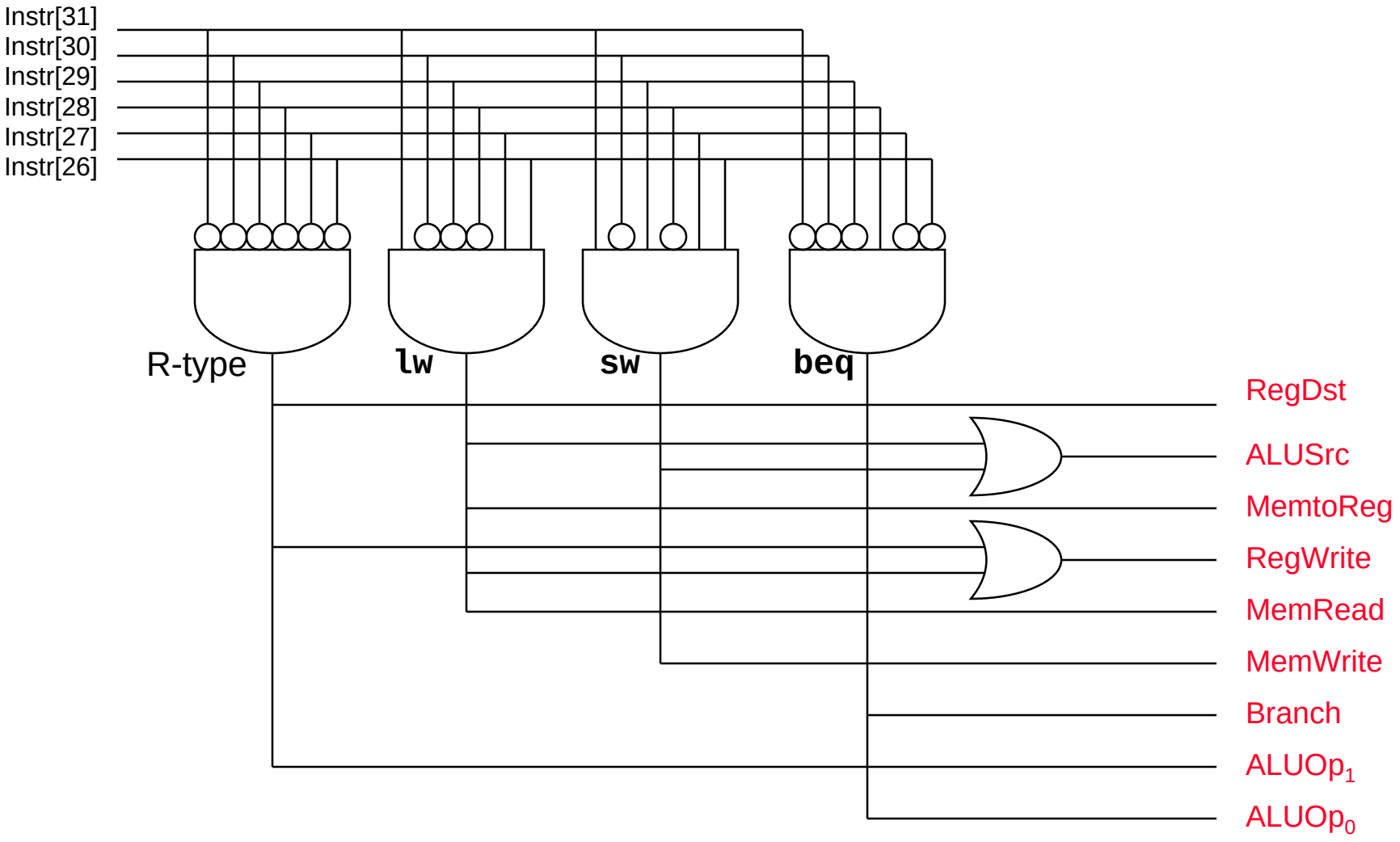
Main Control Unit

Instr	RegDst	ALUSrc	MemReg	RegWr	MemRd	MemWr	Branch	ALUOp
R-type 000000	1	0	0	1	0	0	0	10
lw 100011	0	1	1	1	1	0	0	00
sw 101011	X	1	X	0	0	1	0	00
beq 000100	X	0	X	0	0	0	1	01

- Setting of the MemRd signal (for R-type, sw, beq) depends on the memory design

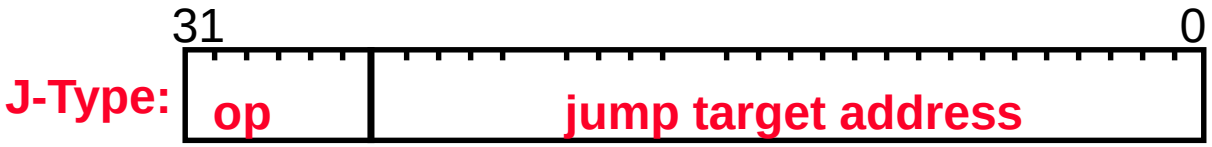
Control Unit Logic

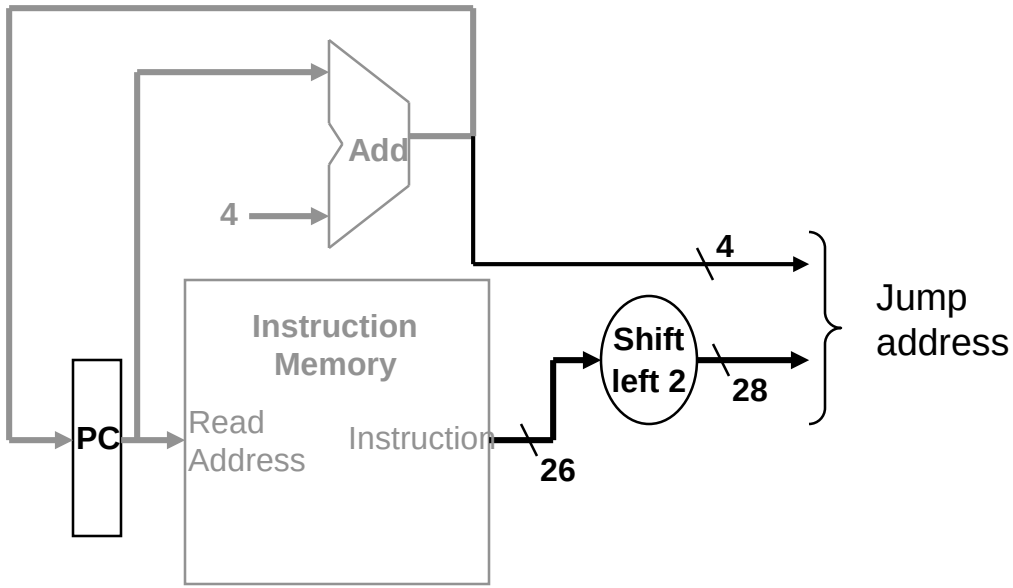
- From the truth table can design the Main Control logic



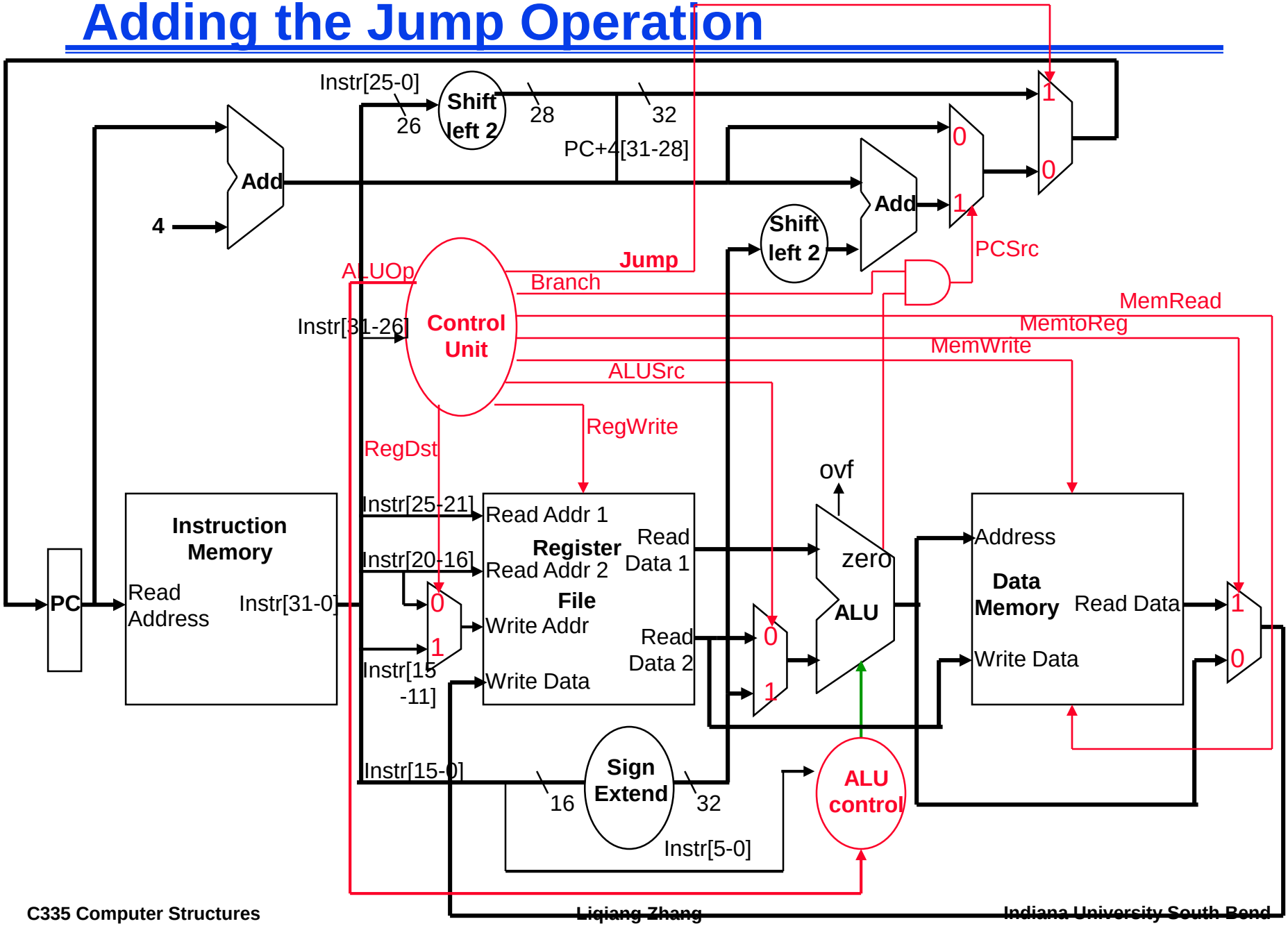
Review: Handling Jump Operations

- Jump operation have to
  - replace the lower 28 bits of the PC with the lower 26 bits of the fetched instruction shifted left by 2 bits

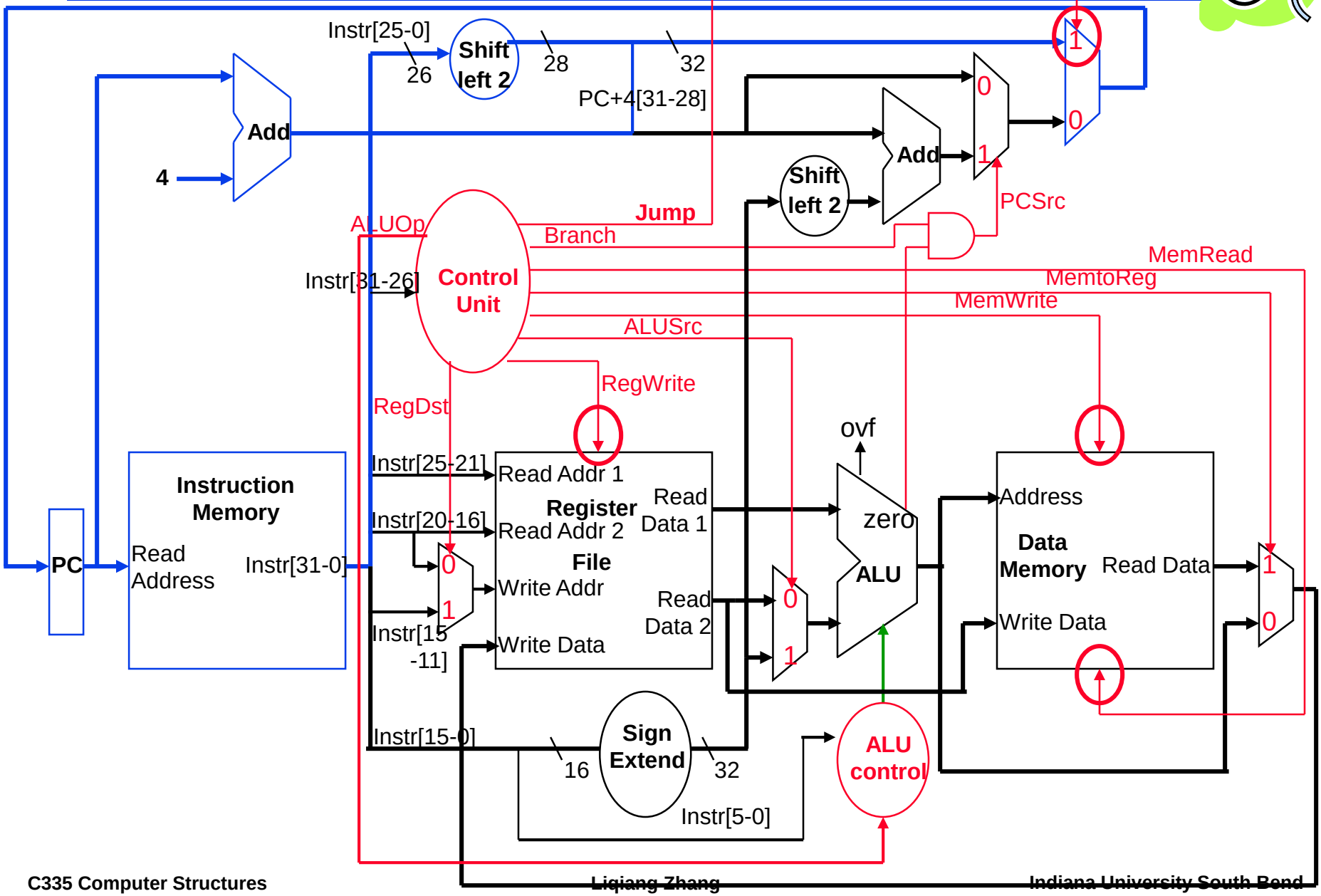




Adding the Jump Operation



Adding the Jump Operation



Main Control Unit

Instr	RegDst	ALUSrc	MemReg	RegWr	MemRd	MemWr	Branch	ALUOp	Jump
R-type 000000	1	0	0	1	0	0	0	10	0
lw 100011	0	1	1	1	1	0	0	00	0
sw 101011	X	1	X	0	0	1	0	00	0
beq 000100	X	0	X	0	0	0	1	01	0
j 000010	X	X	X	0	0	0	X	XX	1

- Setting of the MemRd signal (for R-type, sw, beq) depends on the memory design

Single Cycle Implementation Cycle Time

- Unfortunately, though simple, the single cycle approach is not used because it is very slow
- Clock cycle must have the same length for every instruction

- What is the longest path (slowest instruction)?

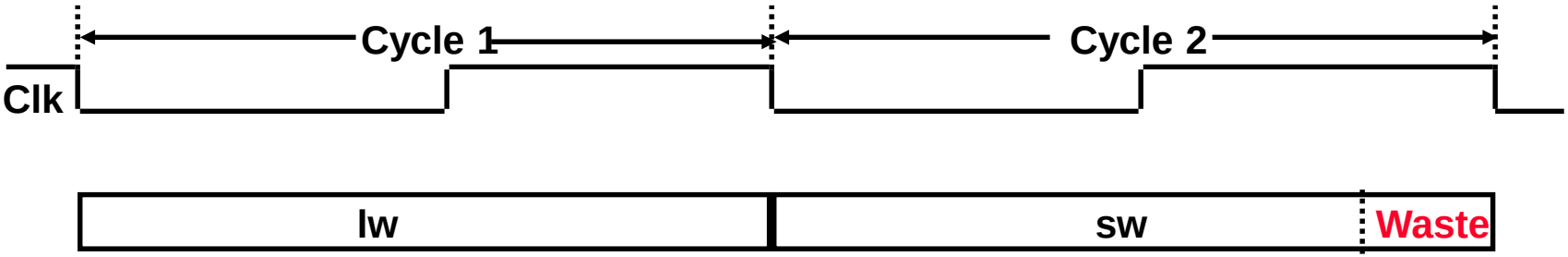
Instruction Critical Paths

- Calculate cycle time assuming negligible delays (for muxes, control unit, sign extend, PC access, shift left 2, wires) except:
  - Instruction and Data Memory (4 ns)
  - ALU and adders (2 ns)
  - Register File access (reads or writes) (1 ns)

Instr.	I Mem	Reg Rd	ALU Op	D Mem	Reg Wr	Total
R-type	4	1	2		1	8
load	4	1	2	4	1	12
store	4	1	2	4		11
beq	4	1	2			7
jump	4					4

Single Cycle Disadvantages & Advantages

- Uses the clock cycle inefficiently – the clock cycle must be timed to accommodate the slowest instr
  - especially problematic for more complex instructions like floating point multiply



- May be wasteful of area since some functional units (e.g., adders) must be duplicated since they can not be shared during a clock cycle

but

- It is simple and easy to understand

Multi-cycle Datapath

- Another approach
  - use a “smaller” cycle time
  - have different instructions take different numbers of cycles
  - a “multicycle” datapath