Computer Science 141
Computing Hardware

Fall 2009

Harvard University

Instructor: Prof. David Brooks

dbrooks@eecs.harvard.edu

Upcoming topics

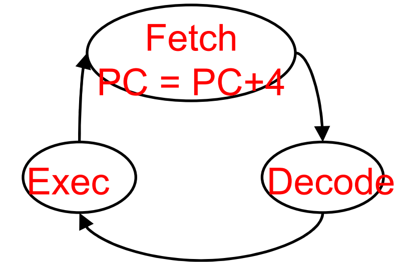
- Wed, Nov 4th – MIPS Basic Architecture (Part 1)
- Mon, Nov 9th – MIPS Basic Architecture (Part 2)
- Wed, Nov 11th – No Class (Holiday)
- Mon, Nov 16th – Computing Performance
- Wed, Nov 18th – MIPS Pipelining and Hazards (Part 1)
- Mon, Nov 23rd – MIPS Pipelining and Hazards (Part 2)
- Mon, Nov 30th – Memory and Caches (Part 1)
- Wed, Dec 2nd – Memory and Caches (Part 2)

Today's Topics

- Beginning to build a MIPS datapath
  - Single cycle datapath implementation

The Processor: Datapath & 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, j`**
- Generic implementation:
  - use the program counter (PC) to supply the instruction address and fetch the instruction from memory (and update the PC)
  - decode the instruction (and read registers)
  - execute the instruction

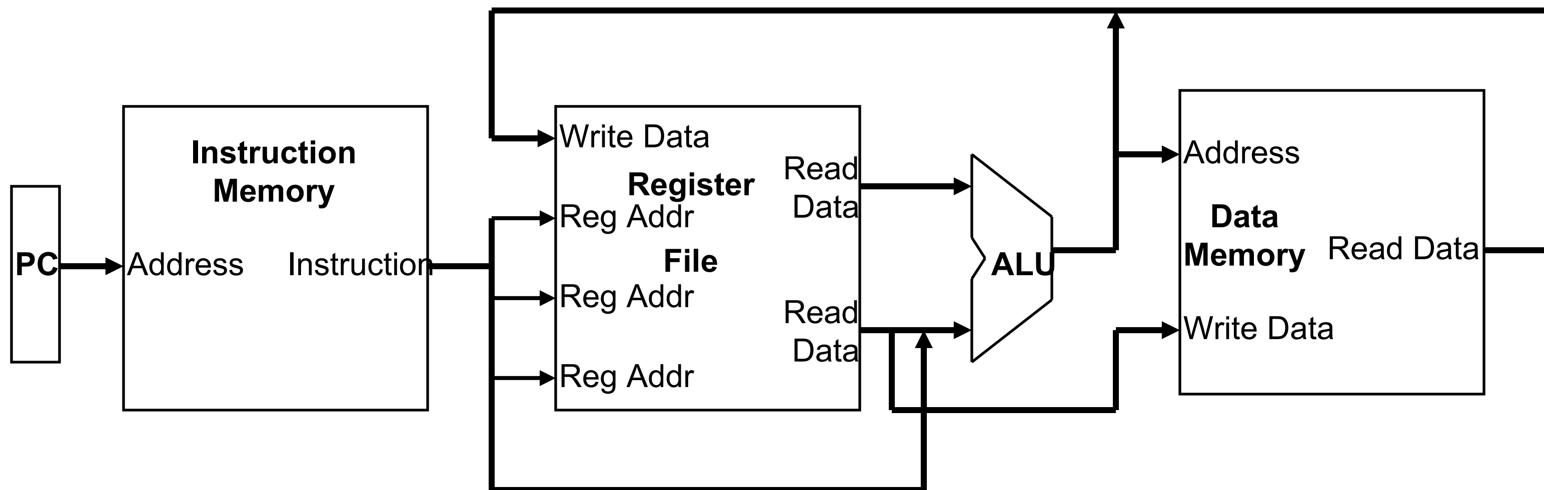


- All instructions (except **`j`**) use the ALU after reading the registers

Why? memory-reference? arithmetic? control flow?

Abstract Implementation View

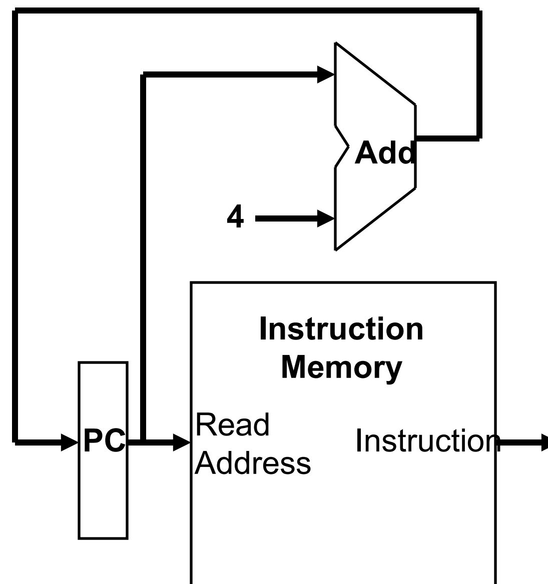
- Two types of functional units:
  - elements that operate on data values (combinational)
  - elements that contain state (sequential)



- Single cycle operation
- Split memory (Harvard) model - one memory for instructions and one for data

Fetching Instructions

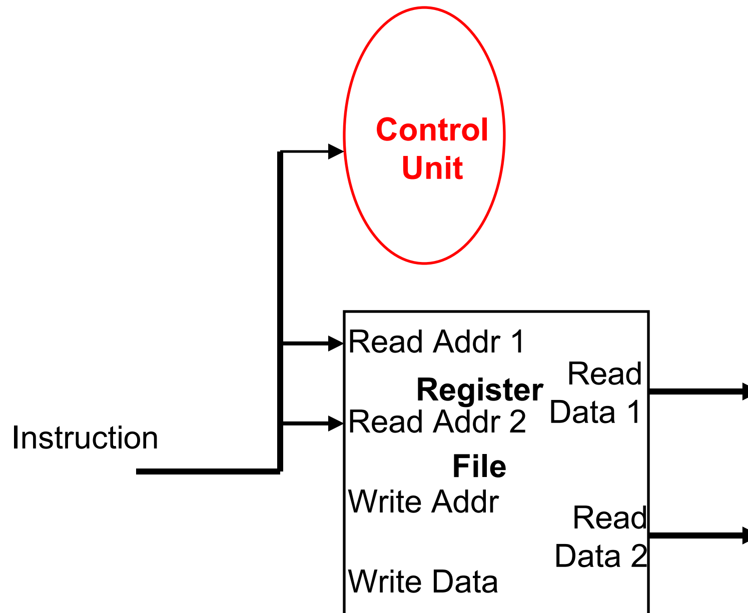
- Fetching instructions involves
  - reading the instruction from the Instruction Memory
  - updating the PC to hold the address of the next instruction



  - PC is updated every cycle, so it does not need an explicit write control signal
  - Instruction Memory is read every cycle, so it doesn't need an explicit read control signal

Decoding Instructions

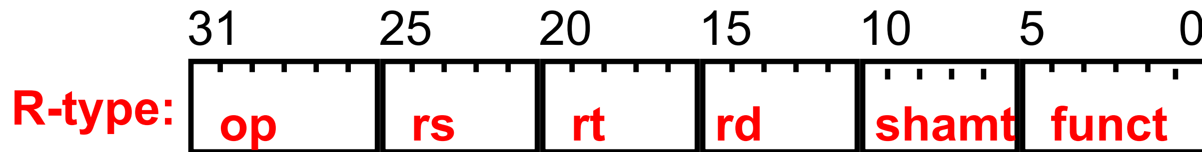
- Decoding instructions involves
  - sending the fetched instruction's opcode and function field bits to the control unit



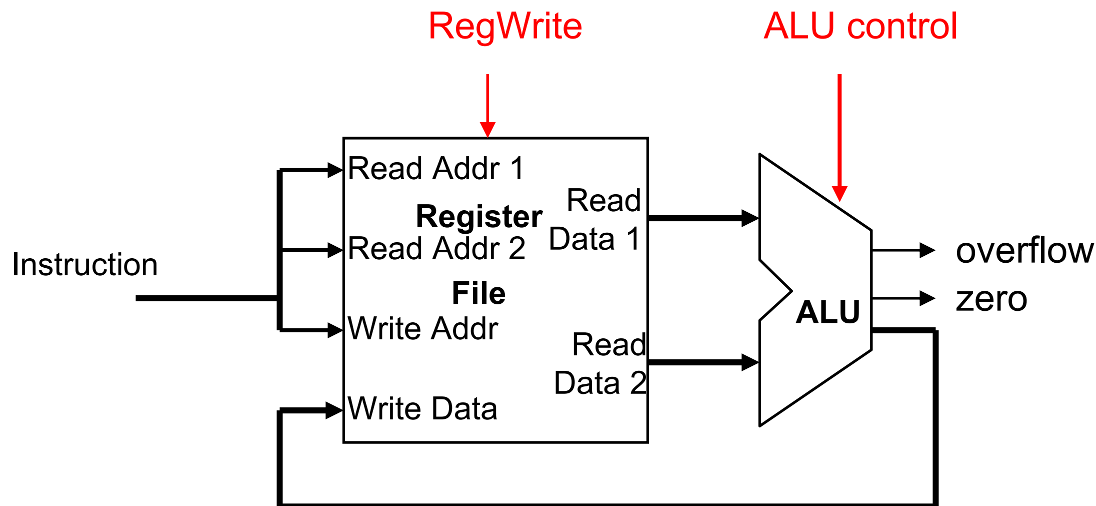
  - reading two values from the Register File
    - Register File addresses are contained in the instruction

Executing R Format Operations

- R format operations (**add, sub, slt, and, or**)



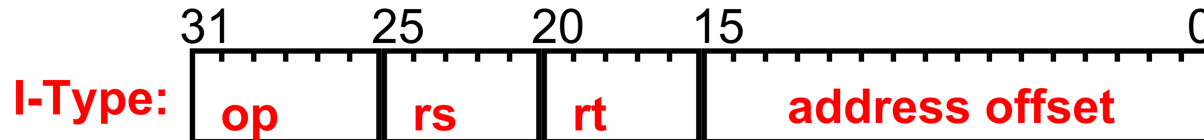
  - perform the indicated (by op and funct) operation on values in rs and rt
  - store the result back into the Register File (into location rd)



  - Note that Register File is not written every cycle (e.g. **sw**), so we need an explicit write control signal for the Register File

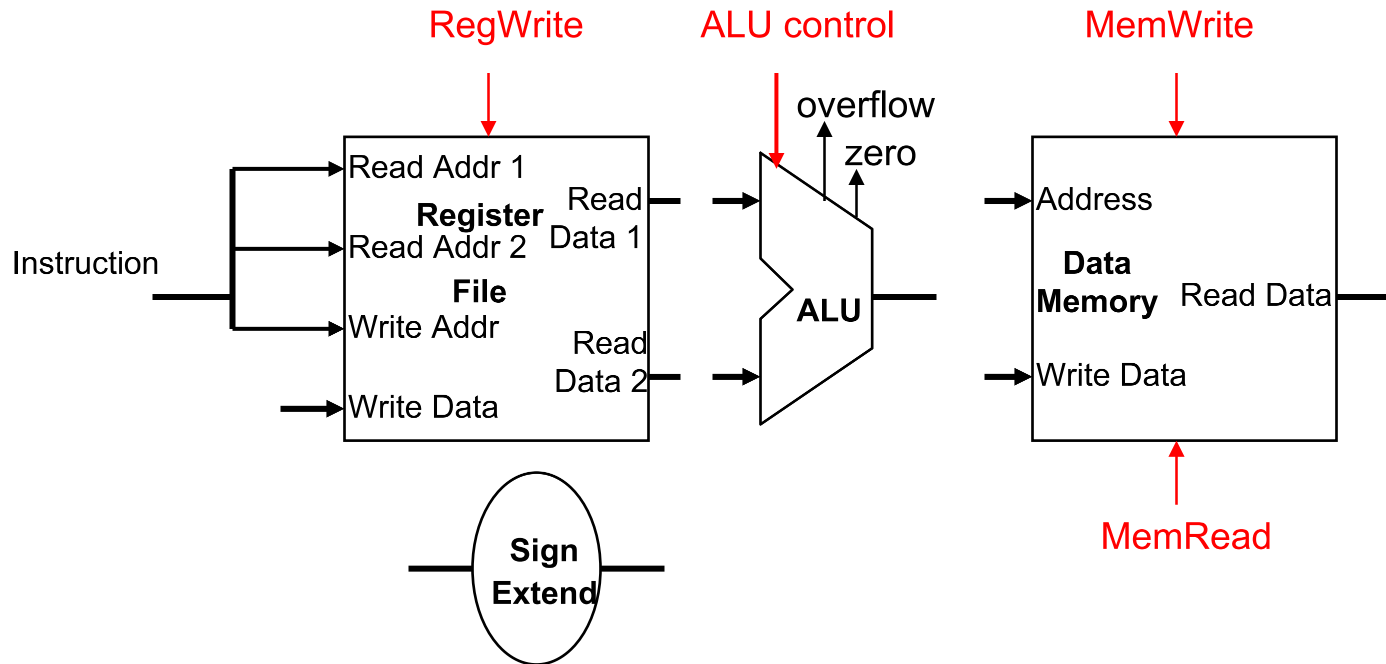
Executing Load and Store Operations

- Load and store operations



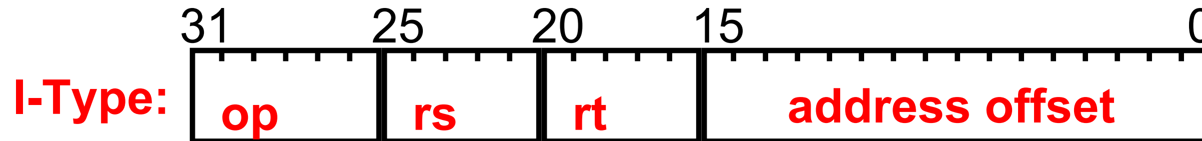
  - compute a memory address by adding the base register (in rs) to the 16-bit signed offset field in the instruction
    - base register was read from the Register File during decode
    - offset value in the low order 16 bits of the instruction must be sign extended to create a 32-bit signed value
  - store value, read from the Register File during decode, must be written to the Data Memory
  - load value, read from the Data Memory, must be stored in the Register File

Executing Load and Store Operations, con't



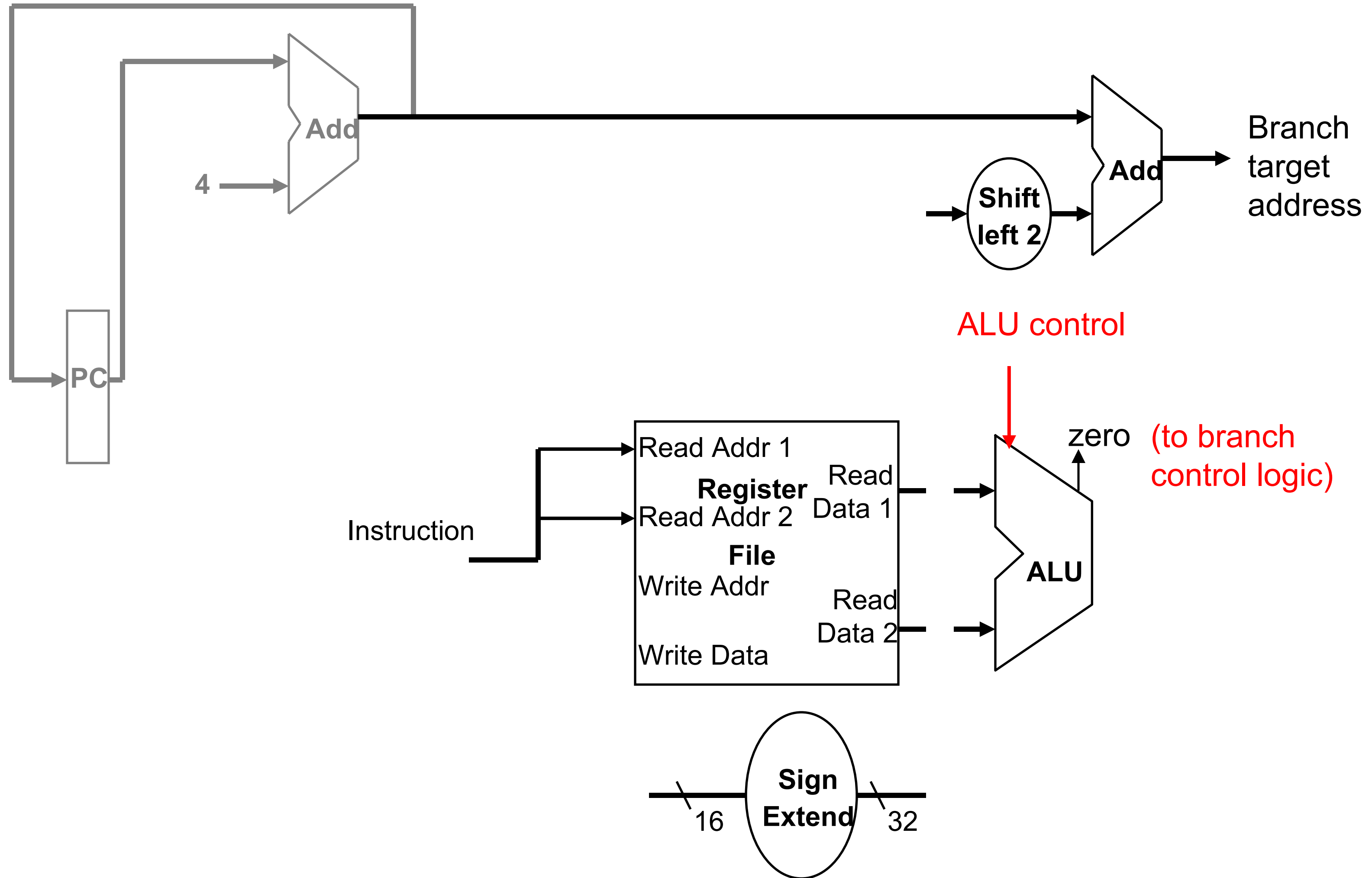
Executing Branch Operations

- Branch operations have to



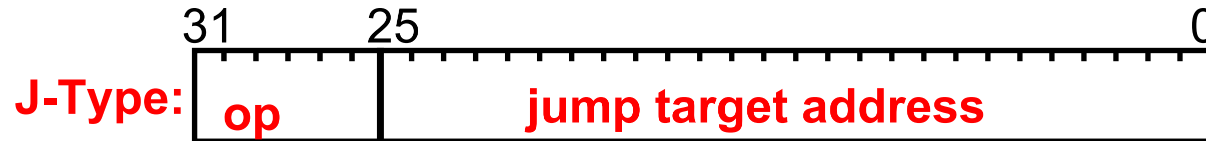
  - compare the operands read from the Register File during decode (rs and rt values) for equality (**`zero`** ALU output)
  - compute the branch target address by adding the updated PC to the sign extended16-bit signed offset field in the instruction
    - "base register" is the updated PC
    - offset value in the low order 16 bits of the instruction must be sign extended to create a 32-bit signed value and then shifted left 2 bits to turn it into a word address

Executing Branch Operations, con't

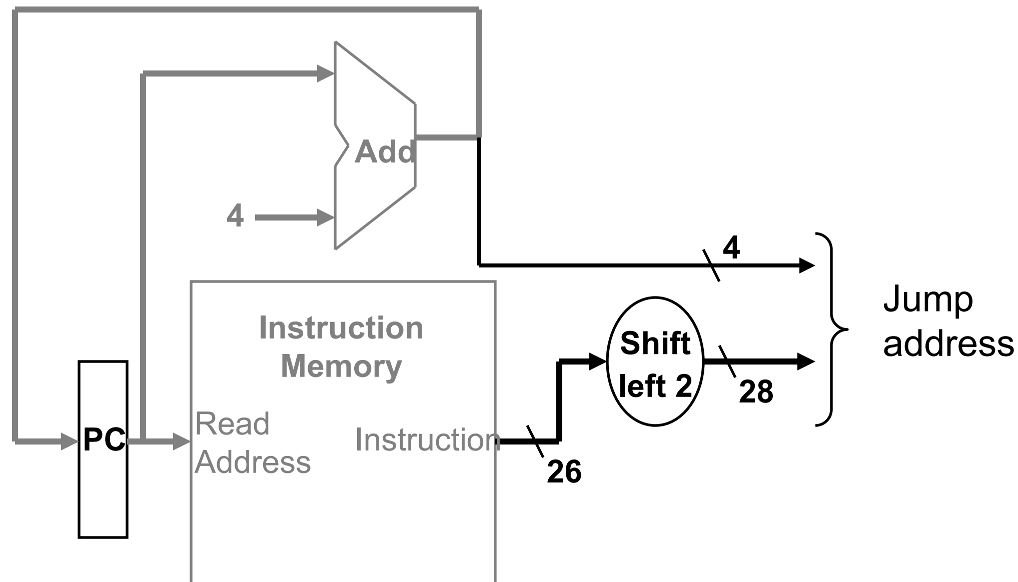


Executing Jump Operations

- Jump operations have to



  – replace the lower 28 bits of the PC with the lower 26 bits of the fetched instruction shifted left by 2 bits



Our Simple Control Structure

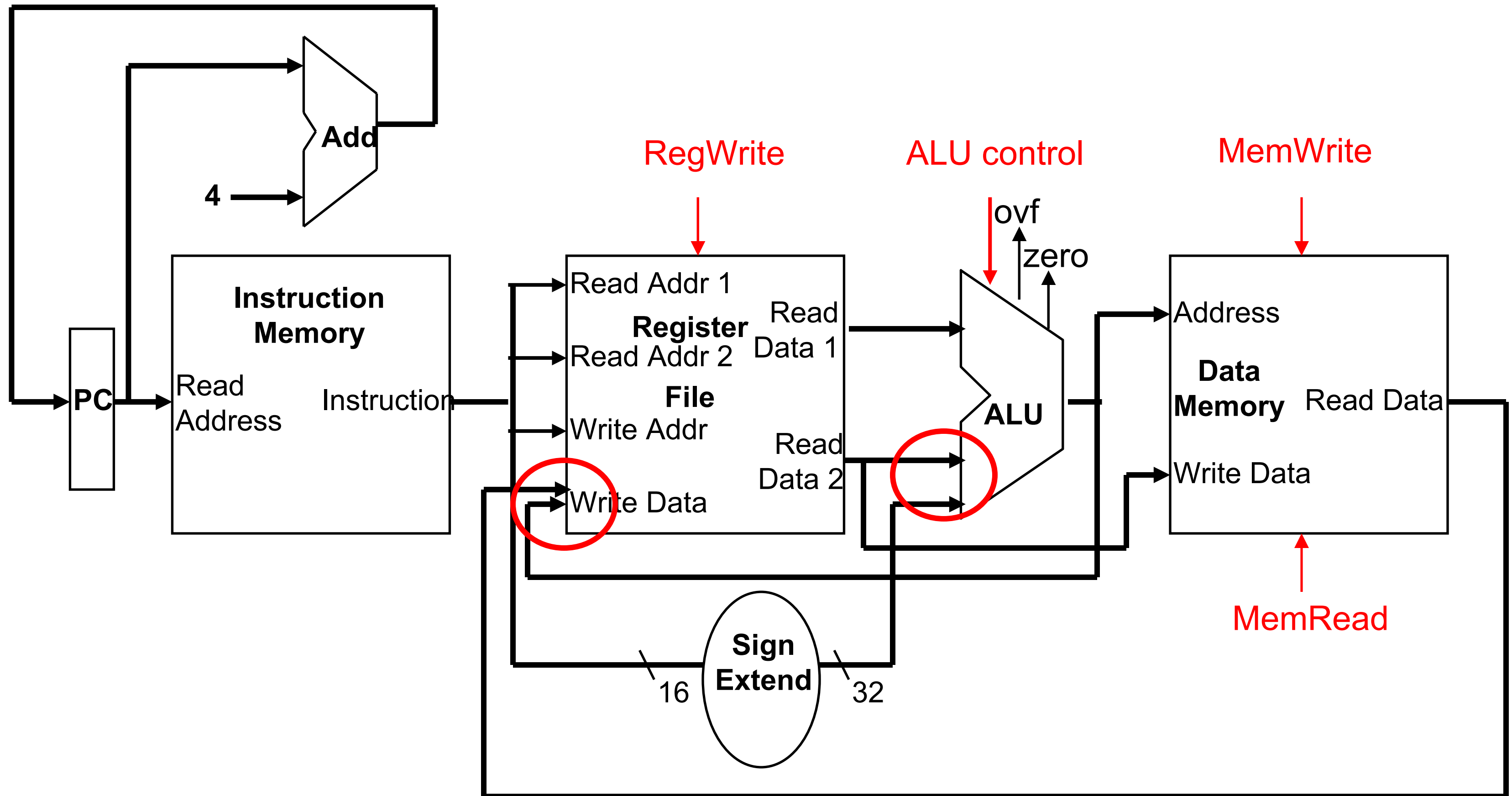
- We wait for everything to settle down
  - ALU might not produce “right answer” right away
  - we use write signals along with the clock edge to determine when to write (to the Register File and the Data Memory)
- Cycle time determined by length of the longest path

We are ignoring some details like register setup and hold times

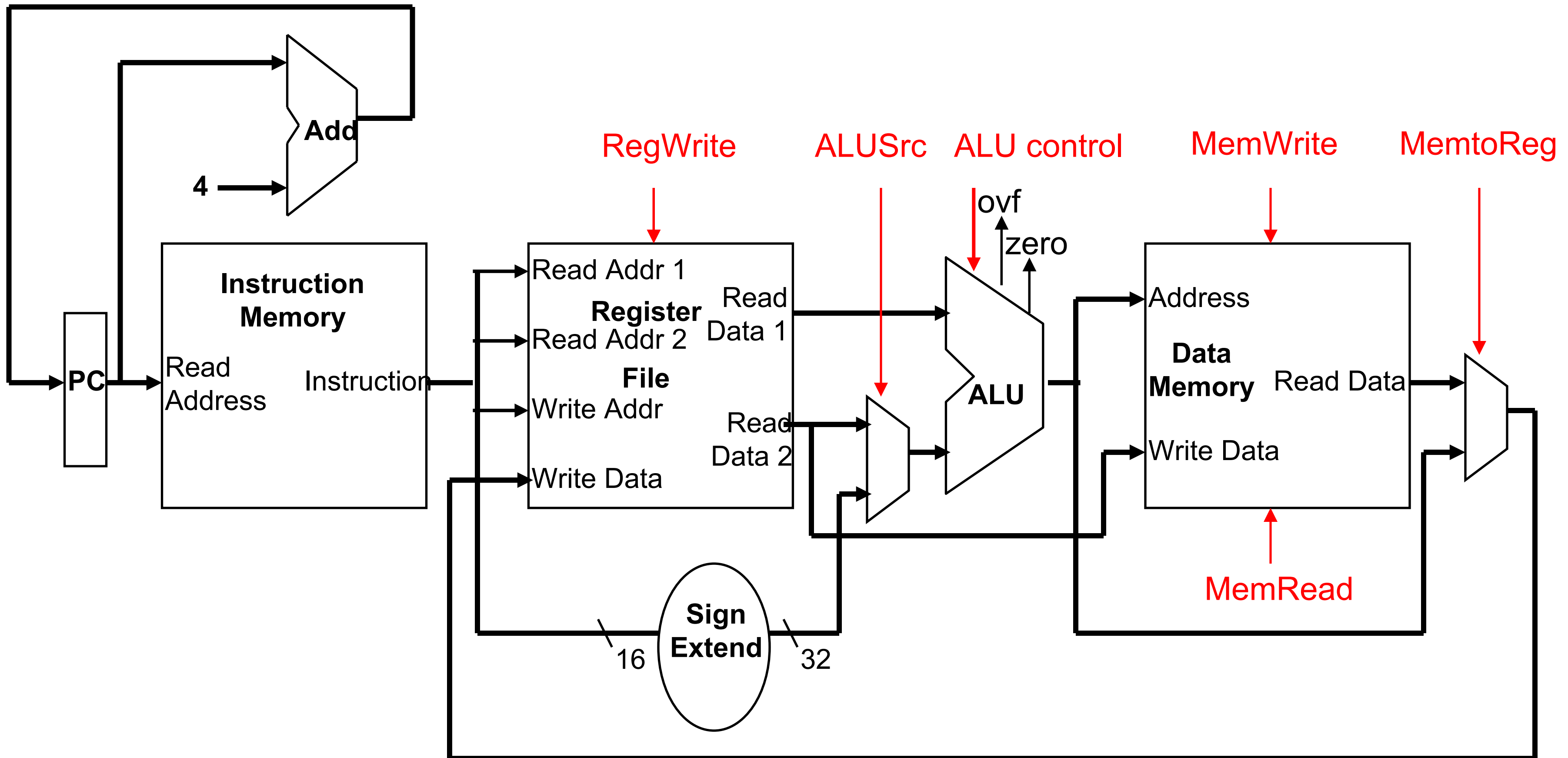
Creating a Single Datapath from the Parts

- Assemble the datapath segments, add control lines as needed, and design the control path
- Fetch, decode and execute each instructions 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 will 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

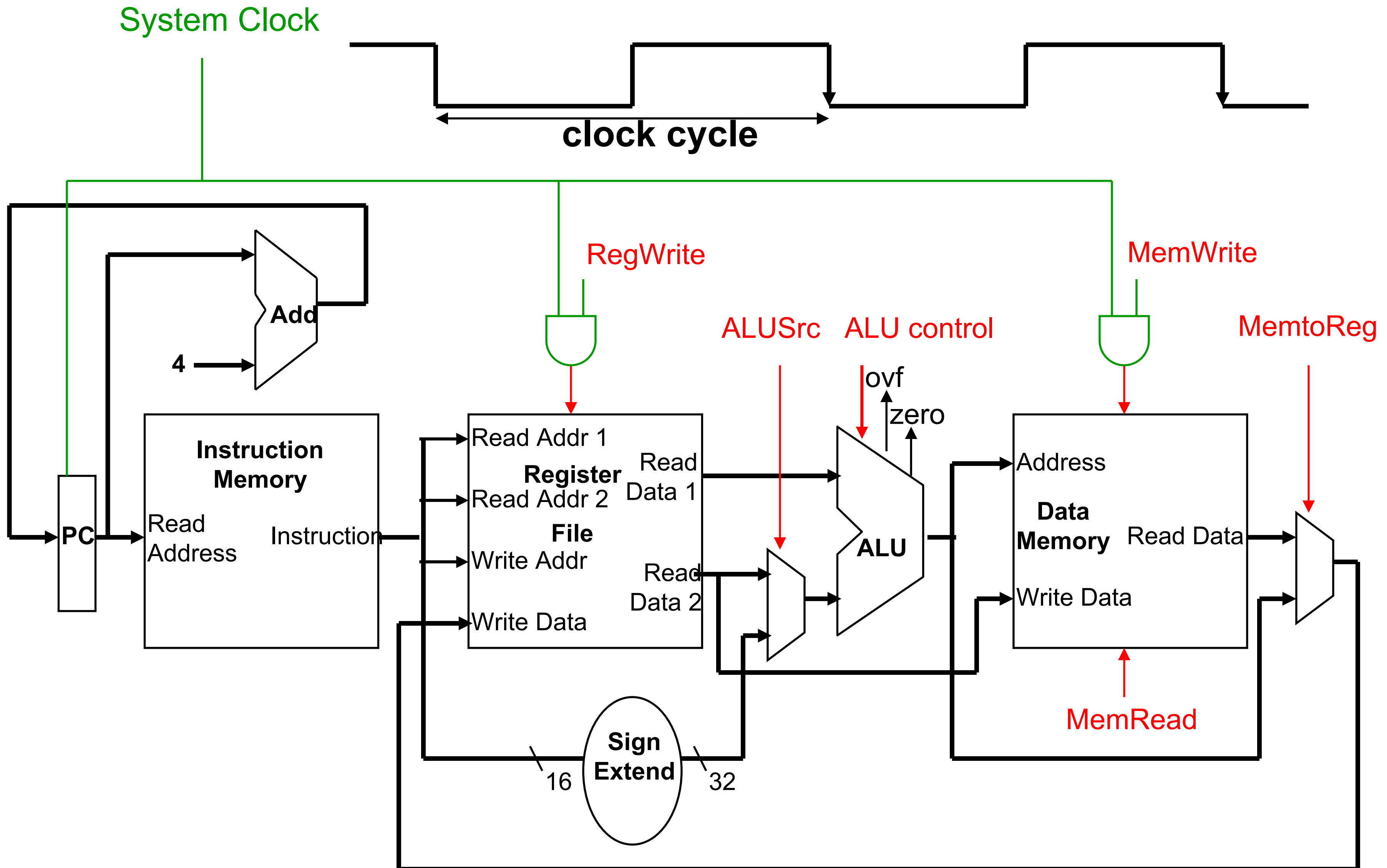
Fetch, R, and Memory Access Portions



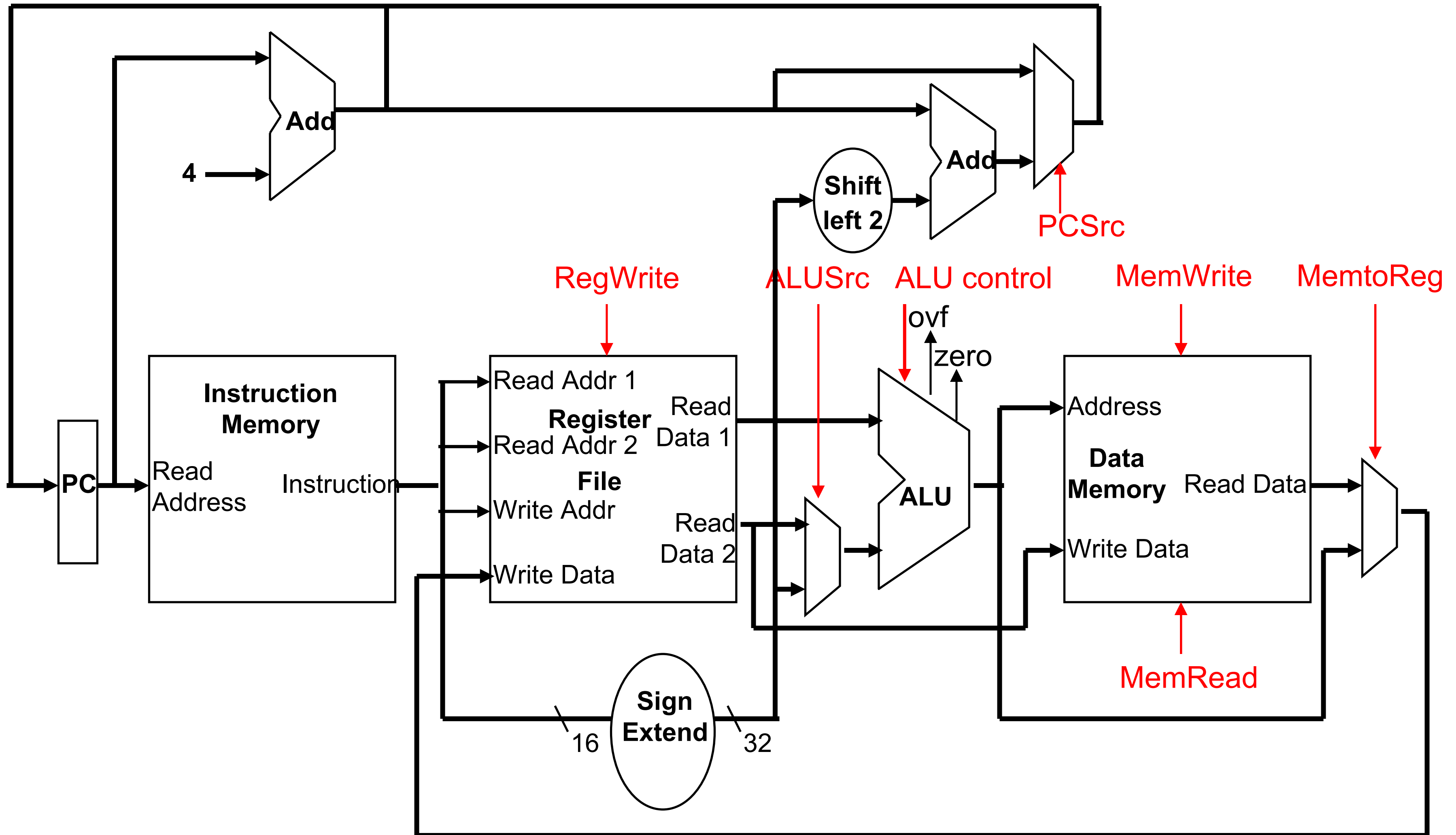
Multiplexor Insertion



Clock Distribution

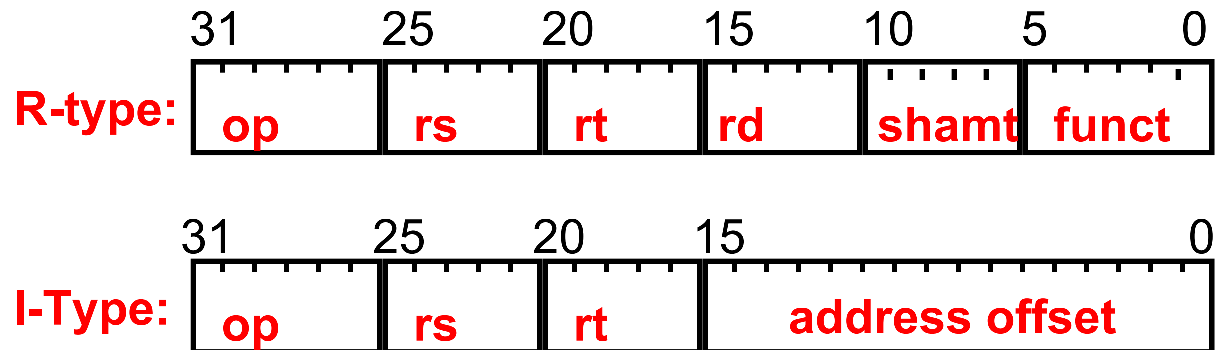


Adding the Branch Portion



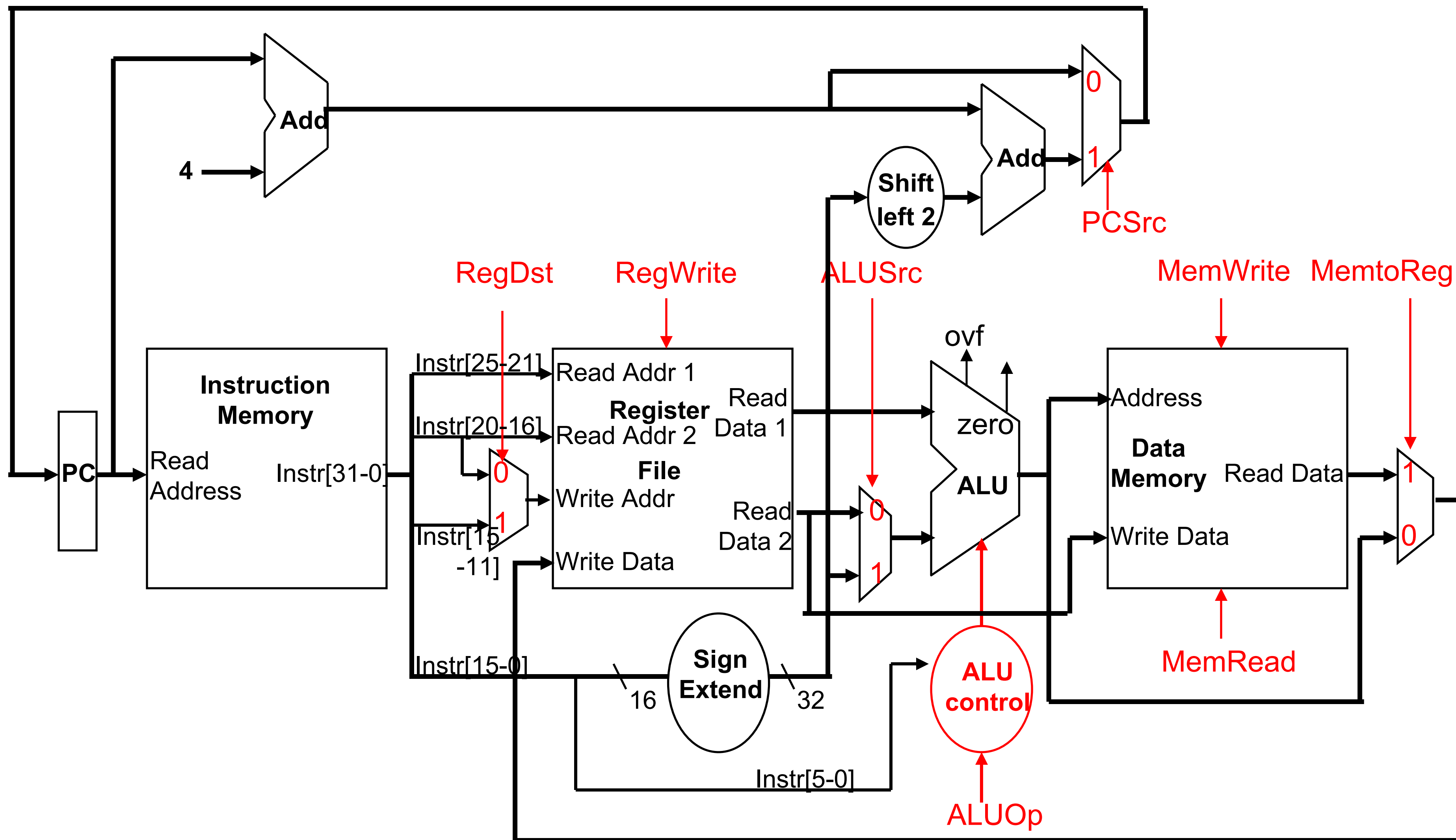
Adding the Control

- 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)
  - 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
  - base register for lw and sw always in rs (bits 25-21)
  - offset for beq, lw, and sw always in bits 15-0

(Almost) Complete Single Cycle Datapath



ALU Control

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

ALU control input	Function
000	and
001	or
010	add
110	subtract
111	set on less than

- Why is the code for subtract 110 and not 011?

ALU Control, Con't

- Controlling the ALU makes use of multiple levels of decoding
  - main control unit generates the ALUOp bits
  - ALU control unit generates ALU control inputs

Instr op	funct	ALUOp	desired action	ALU control input
lw	xxxxxx	00		
sw	xxxxxx	00		
beq	xxxxxx	01		
add	100000	10	add	010
subt	100010	10	subtract	110
and	100100	10	and	000
or	100101	10	or	001
slt	101010	10	slt	111

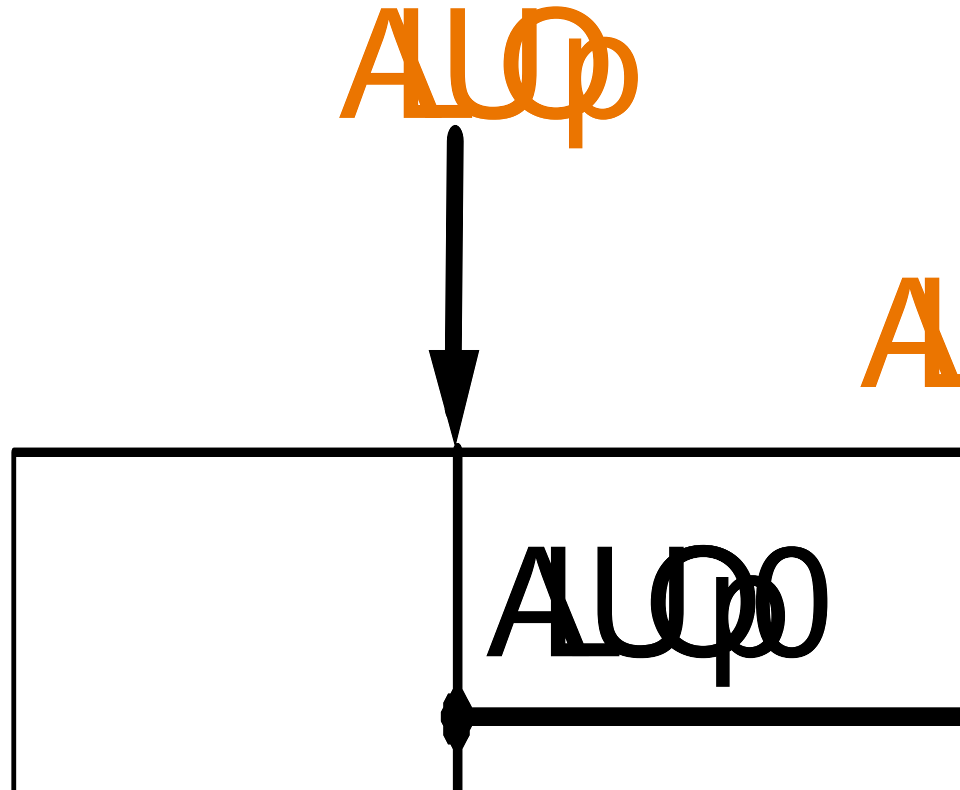
ALU Control Truth Table

F5	F4	F3	F2	F1	F0	ALUOp1	ALUOp0	Op2	Op1	Op0
X	X	X	X	X	X	0	0	0	1	0
X	X	X	X	X	X		1	1	1	0
X	X	0	0	0	0	1		0	1	0
X	X	0	0	1	0	1		1	1	0
X	X	0	1	0	0	1		0	0	0
X	X	0	1	0	1	1		0	0	1
X	X	1	0	1	0	1		1	1	1

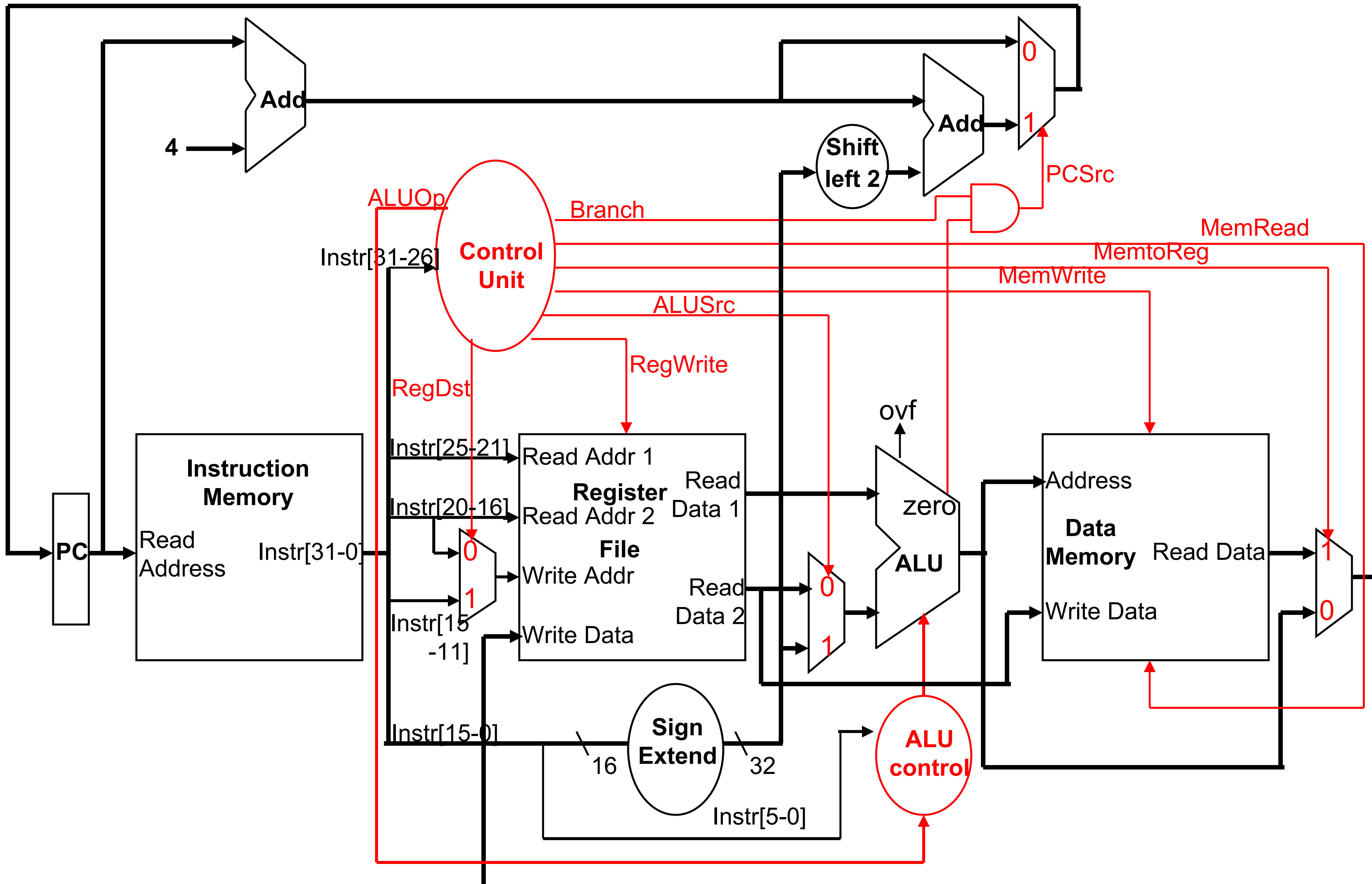
- Can make use of more don't cares
  - since ALUOp does not use the encoding 11
  - since F5 and F4 are always 10
- Logic comes from the K-maps …

ALU Control Combinational Logic

- From the truth table can design the ALU Control logic



(Almost) Complete Datapath with Control Unit

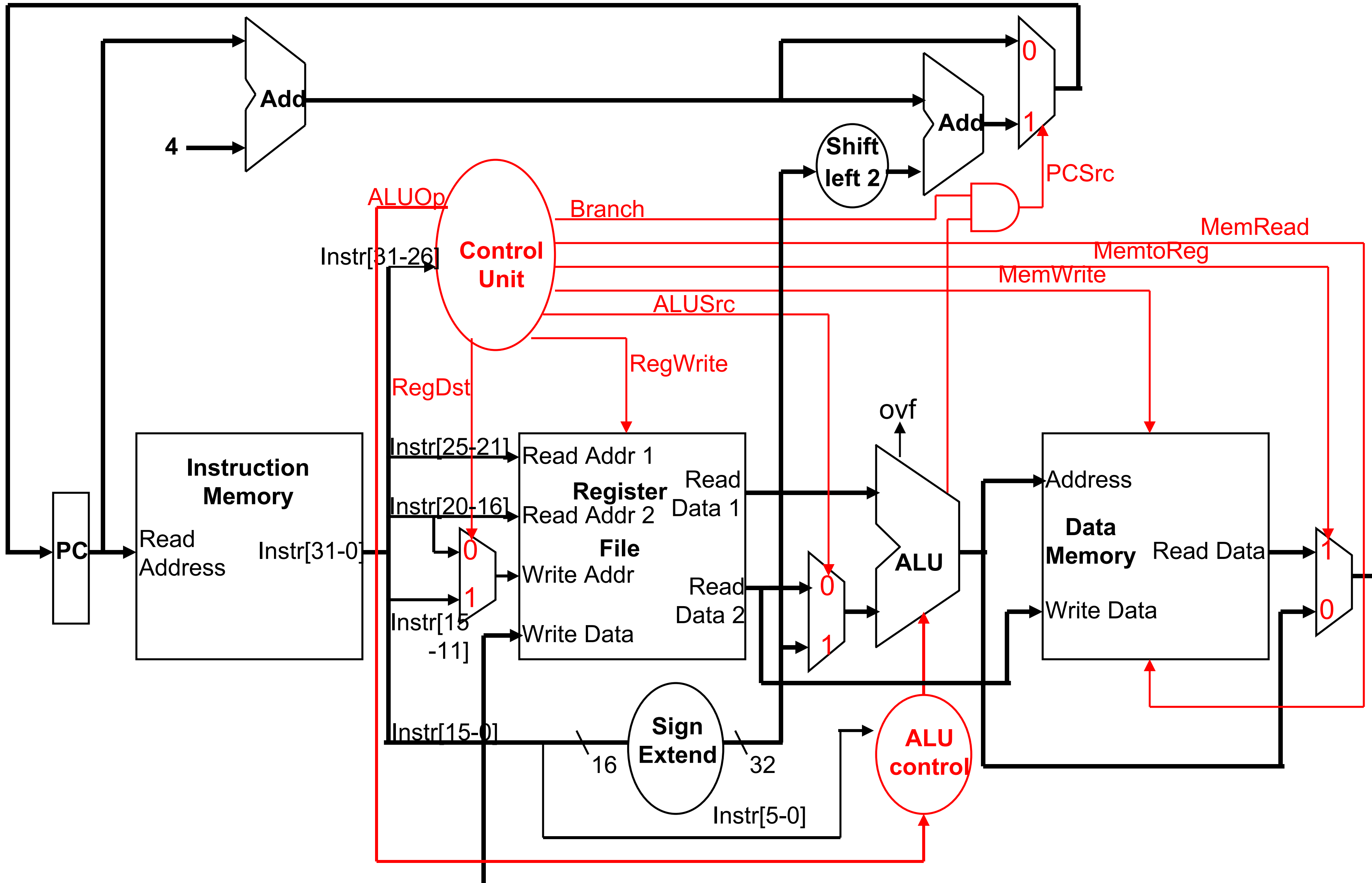


Main Control Unit

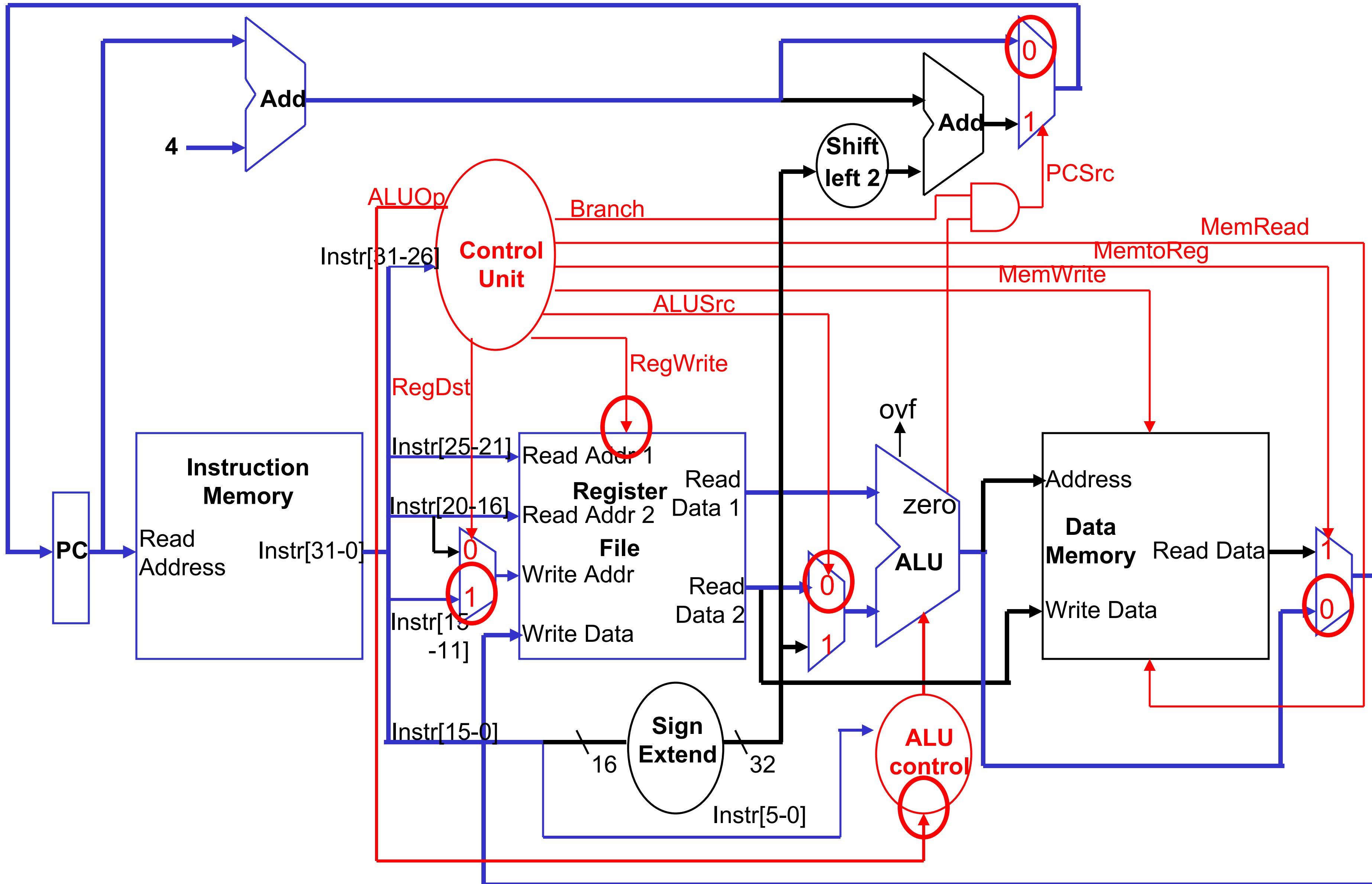
Instr	RegDst	ALUSrc	MemReg	RegWr	MemRd	MemWr	Branch	ALUOp1	ALUOp2
R-type 000000									
lw 100011									
sw 101011									
beq 000100									

- 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

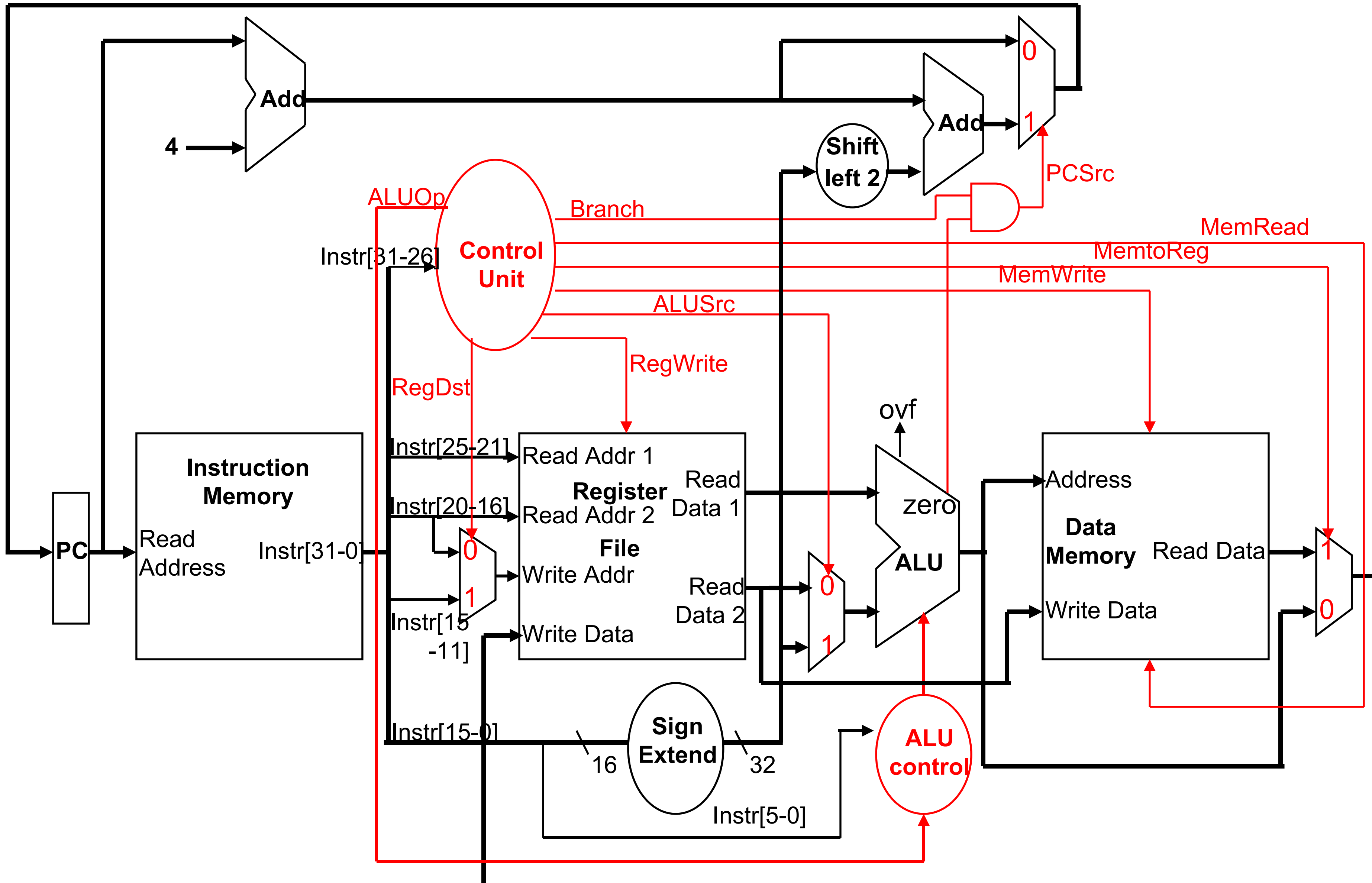
R-type Instruction Data/Control Flow



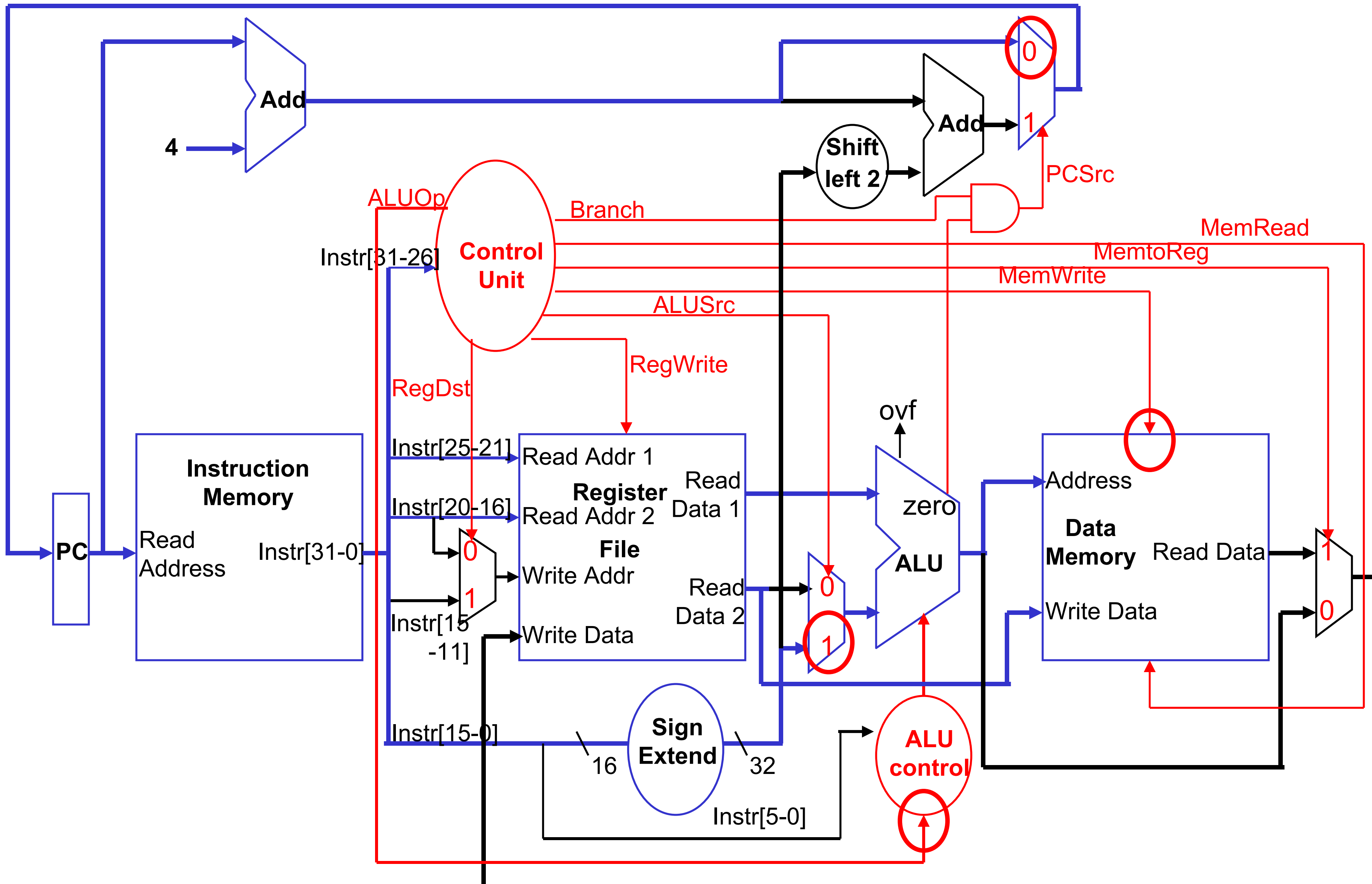
R-type Instruction Data/Control Flow



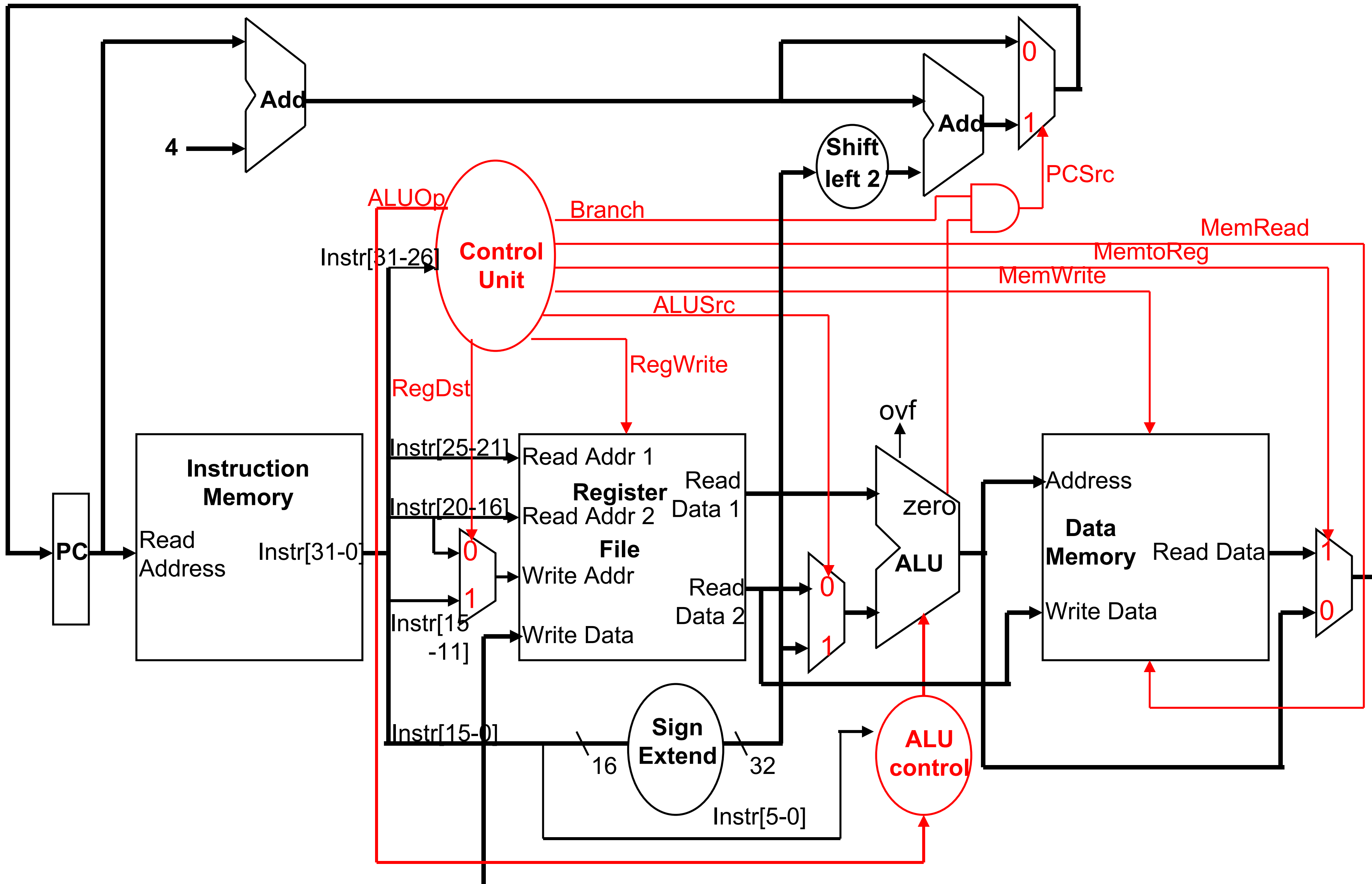
Store Word Instruction Data/Control Flow



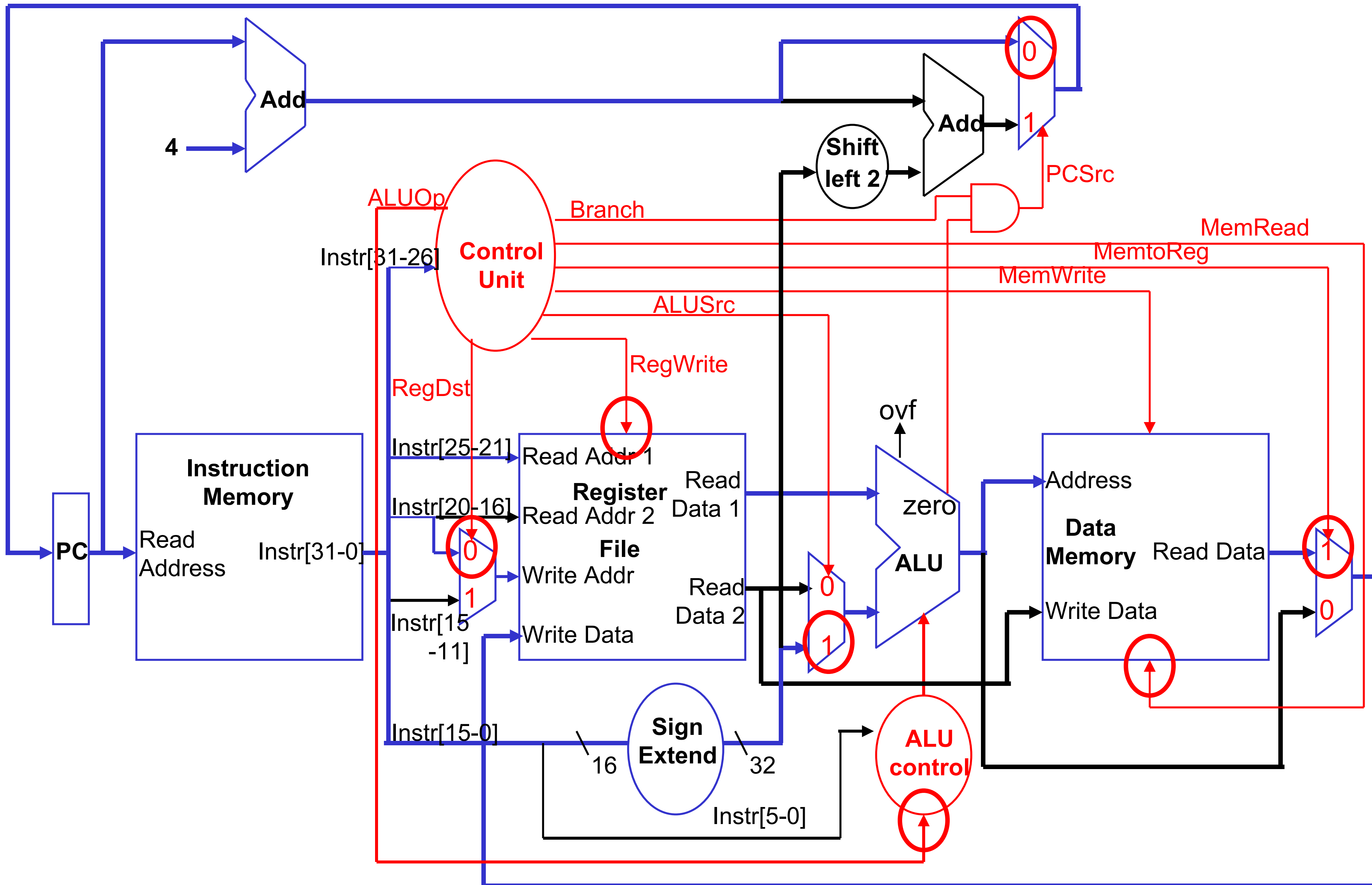
Store Word Instruction Data/Control Flow



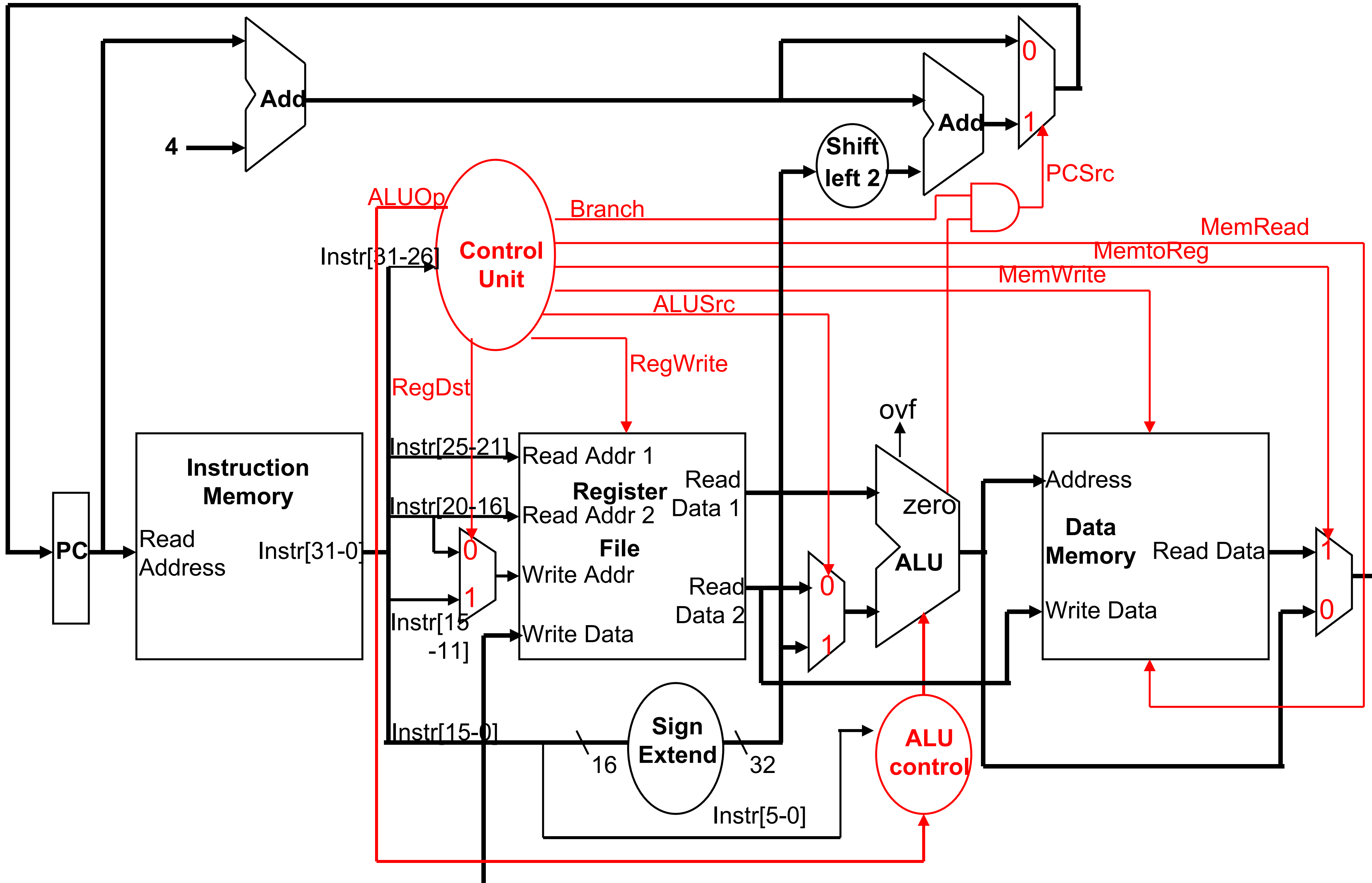
Load Word Instruction Data/Control Flow



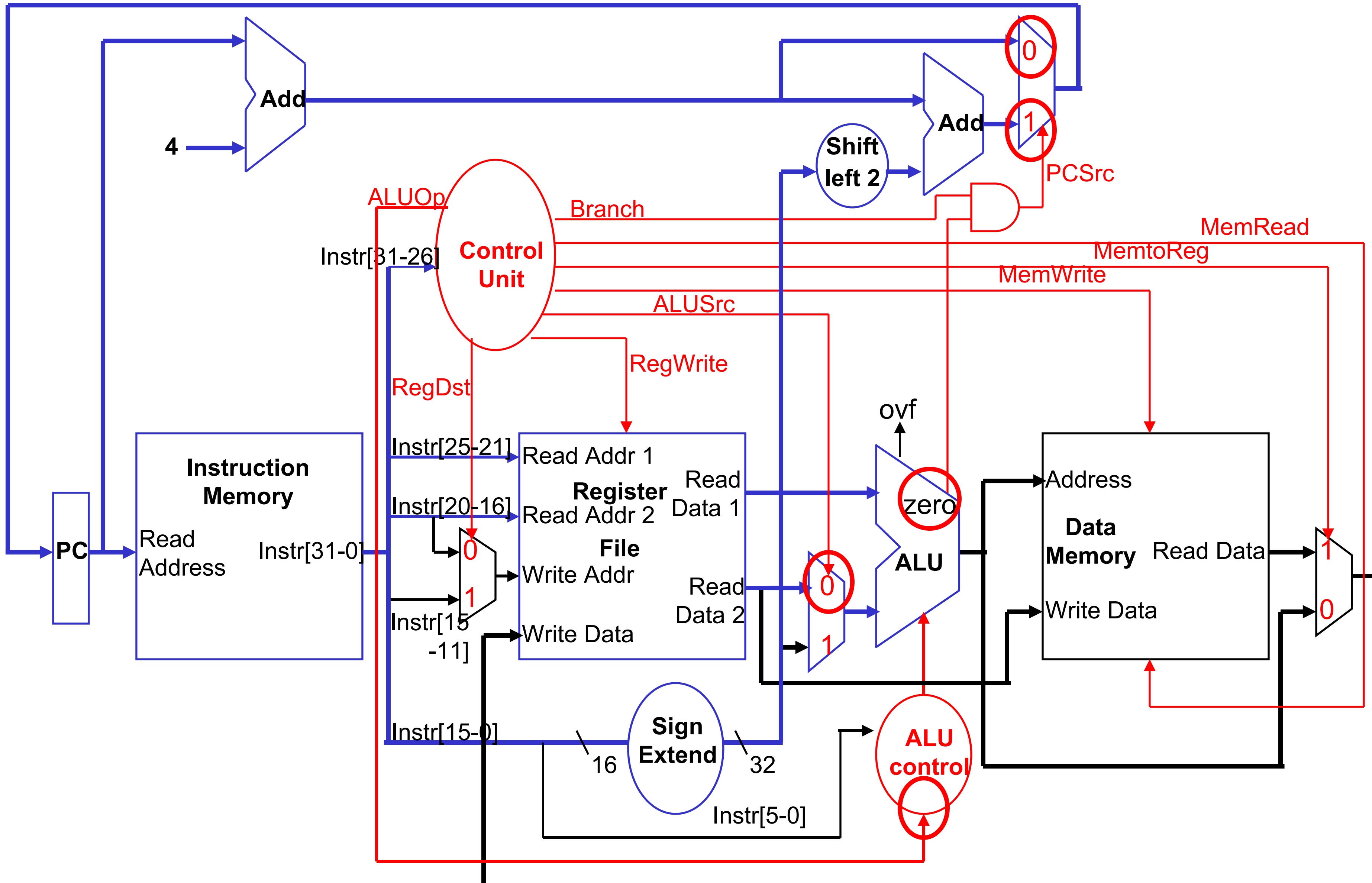
Load Word Instruction Data/Control Flow



Branch Instruction Data/Control Flow



Branch Instruction Data/Control Flow



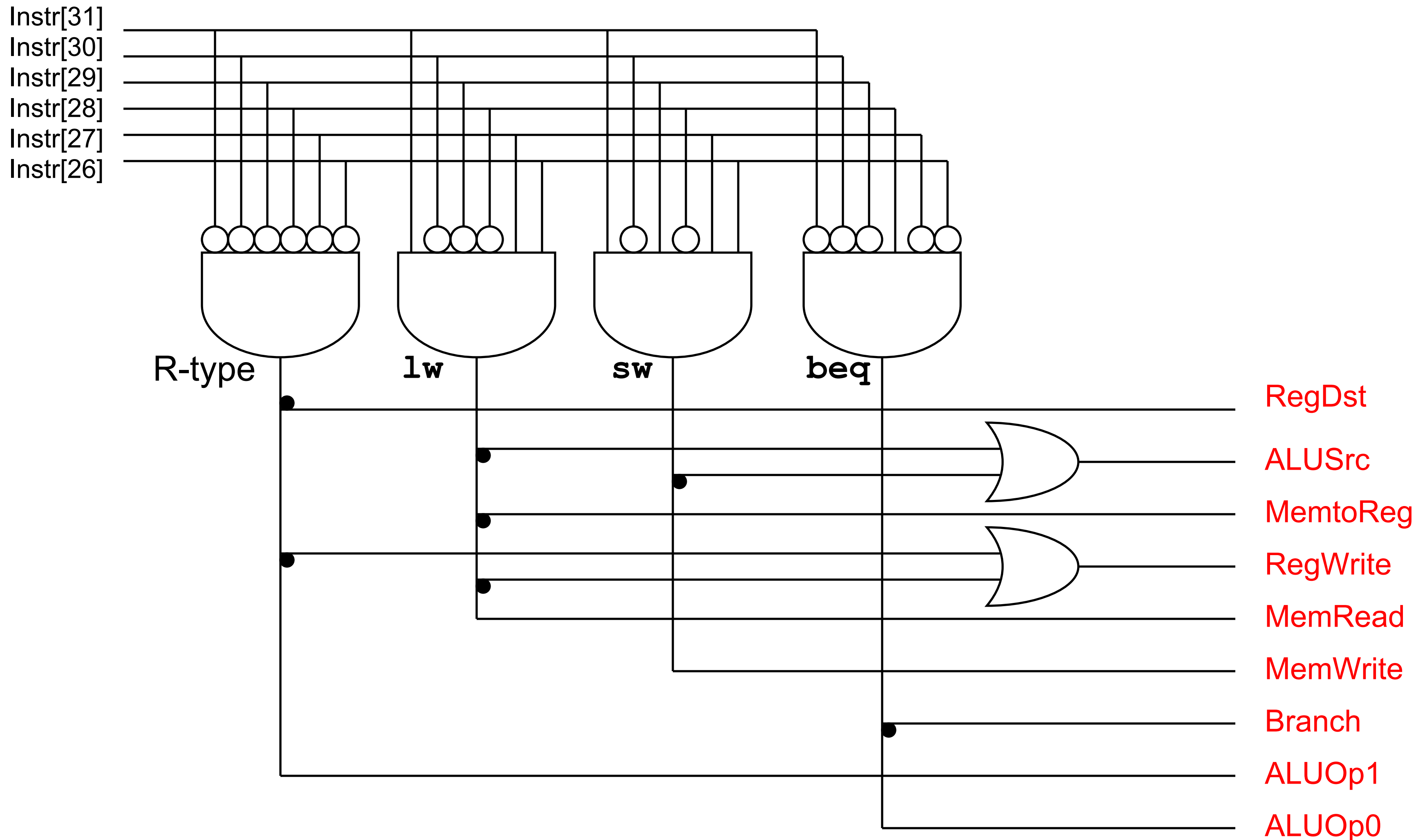
Main Control Unit

Instr	RegDst	ALUSrc	MemReg	RegWr	MemRd	MemWr	Branch	ALUOp1	ALUOp0
R-type 000000	1	0	0	1	X	0	0	1	X
lw 100011	0	1	1	1	1	0	0	0	0
sw 101011	X	1	X	0	X	1	0	0	0
beq 000100	X	0	X	0	X	0	1	X	1

- 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

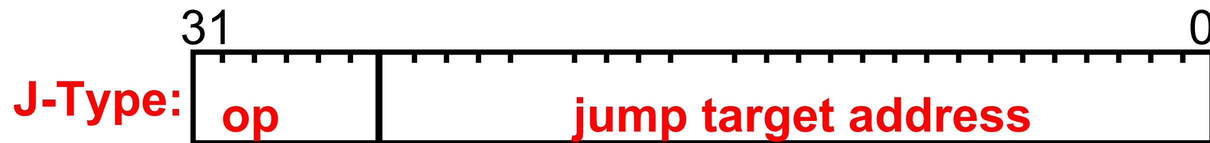
Control Unit Logic

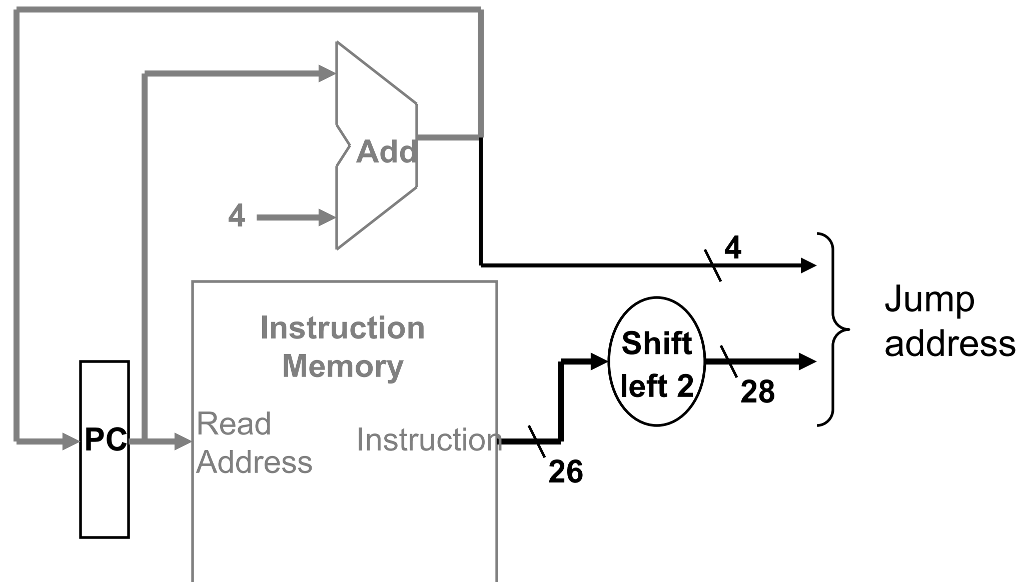
- From the truth table can design the Main Control logic



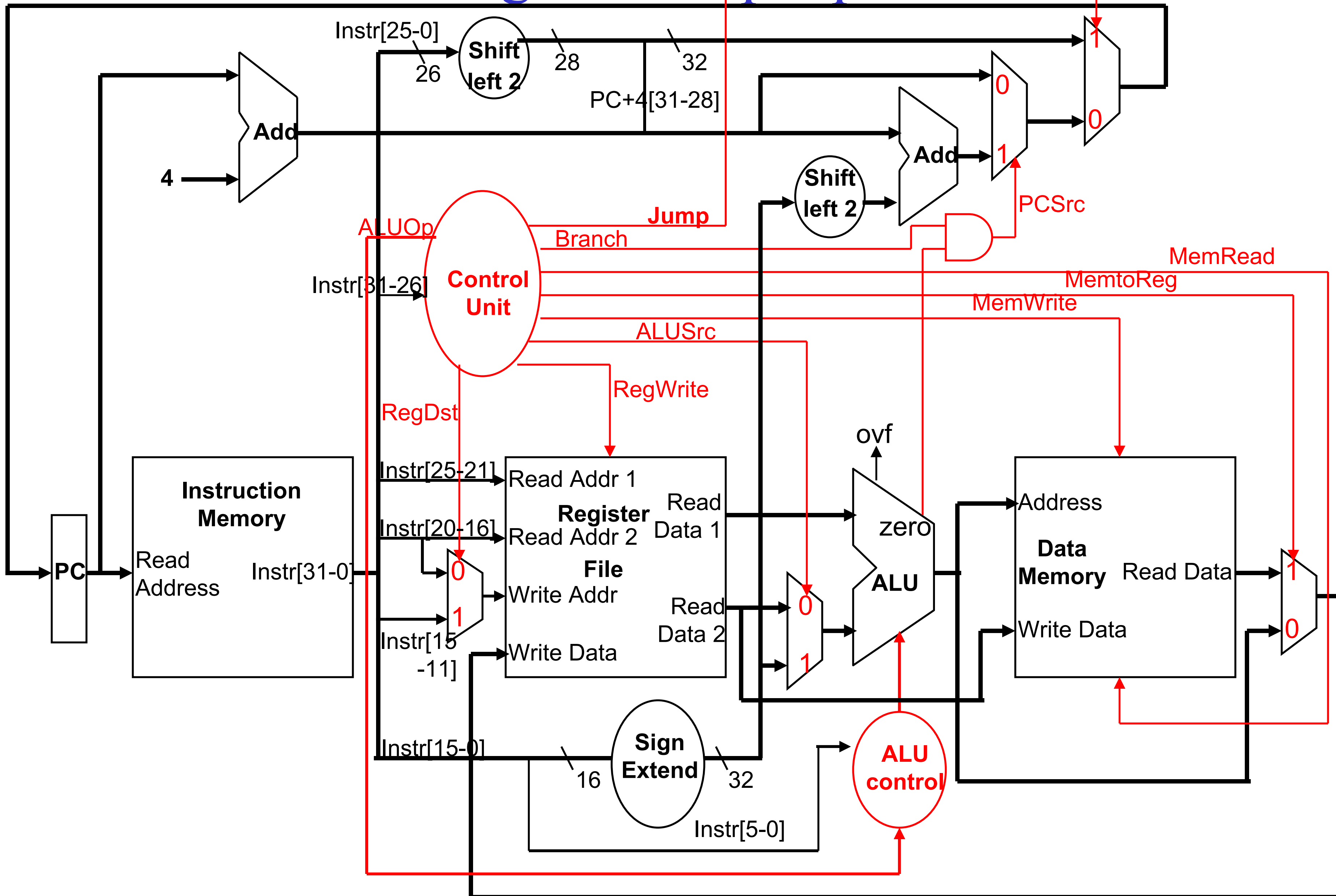
Review: Handling Jump Operations

- 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

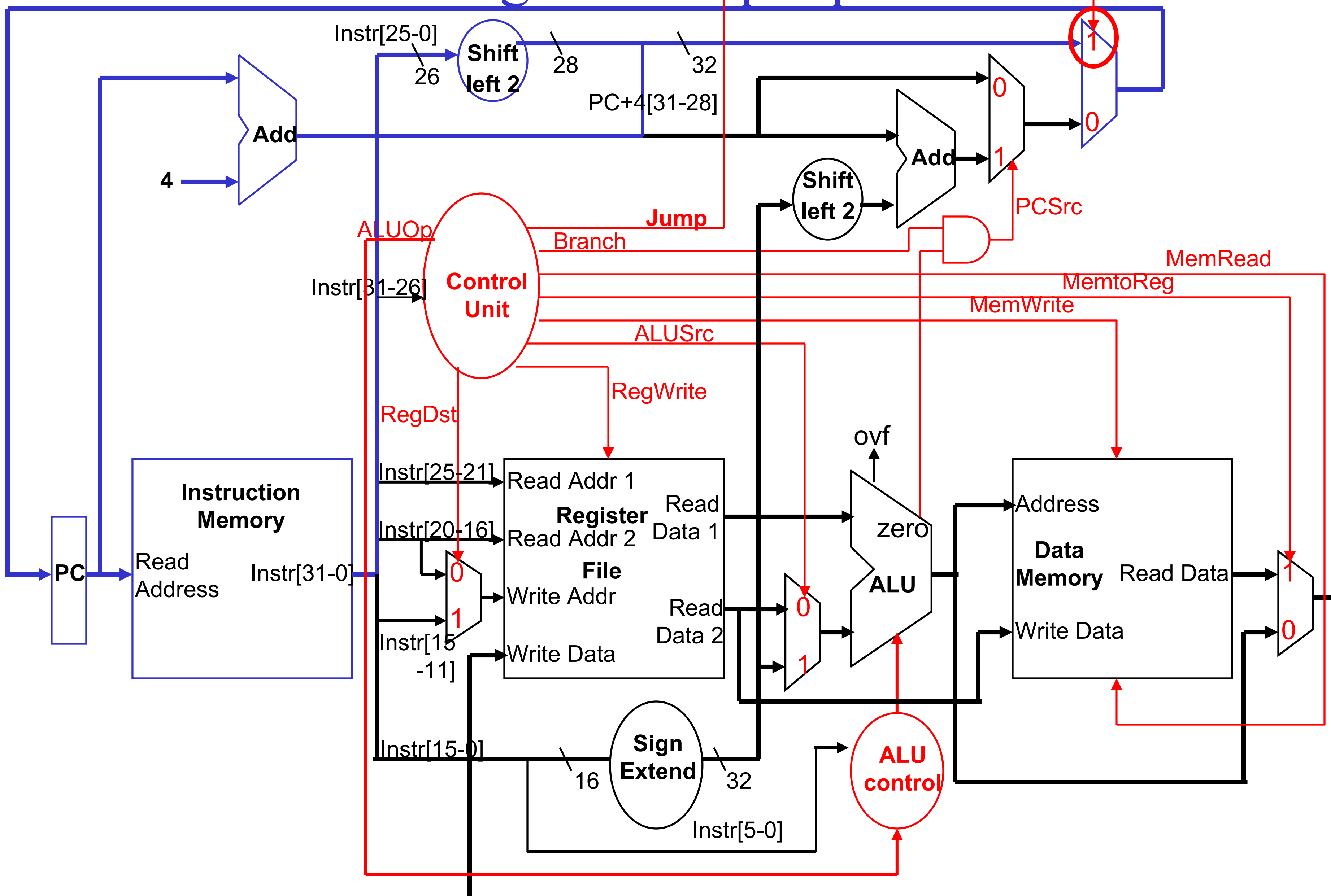




Adding the Jump Operation



Adding the Jump Operation



Single Cycle Implementation Cycle Time

- Unfortunately, though simple, the single cycle approach is not used because it is inefficient
- 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 (2ns)
  - ALU and adders (2ns)
  - Register File access (reads or writes) (1ns)

Instr.	I Mem	Reg Rd	ALU Op	D Mem	Reg Wr	Total
R-type						
load						
store						
beq						
jump						

Where We are Headed

- Problems with single cycle datapath design
  - uses clock cycle inefficiently
  - and what if we had a more complicated instruction like floating point multiply?
  - wasteful of area
- Another approach
  - use a "smaller" cycle time
  - have different instructions take different numbers of cycles
  - a "multicycle" datapath: