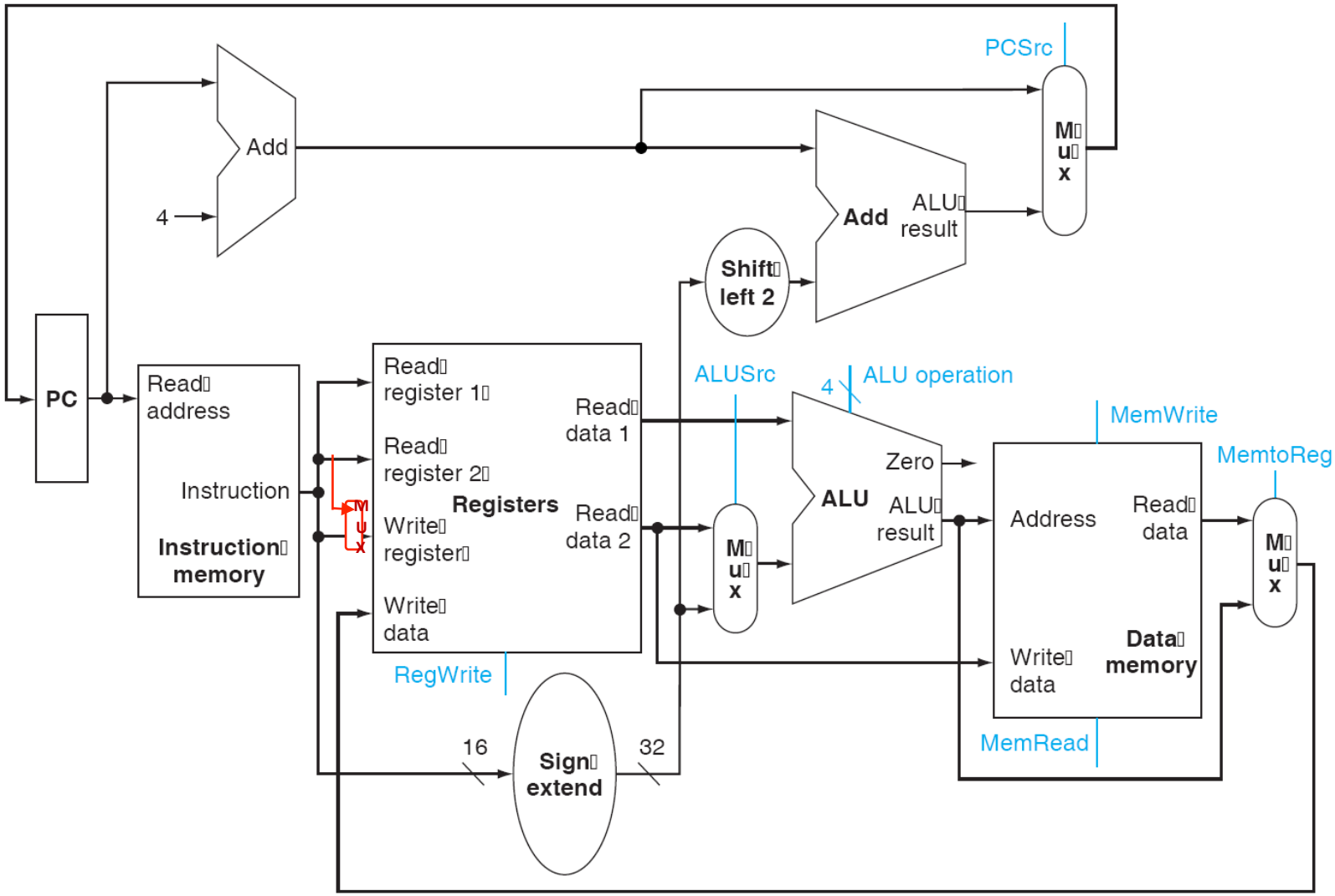
EECE 321: Computer Organization

Mohammad M. Mansour
Dept. of Electrical and Compute Engineering
American University of Beirut

Lecture 19: MIPS Single-Cycle Processor Implementation

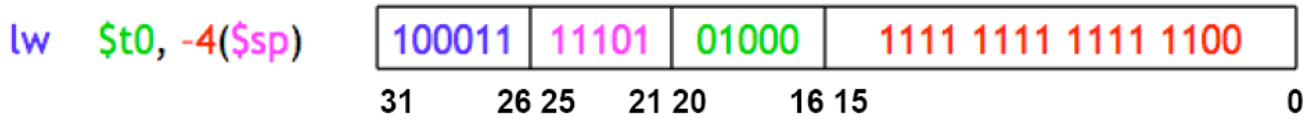
Appending the Instruction Fetch Portion to Combined Datapath

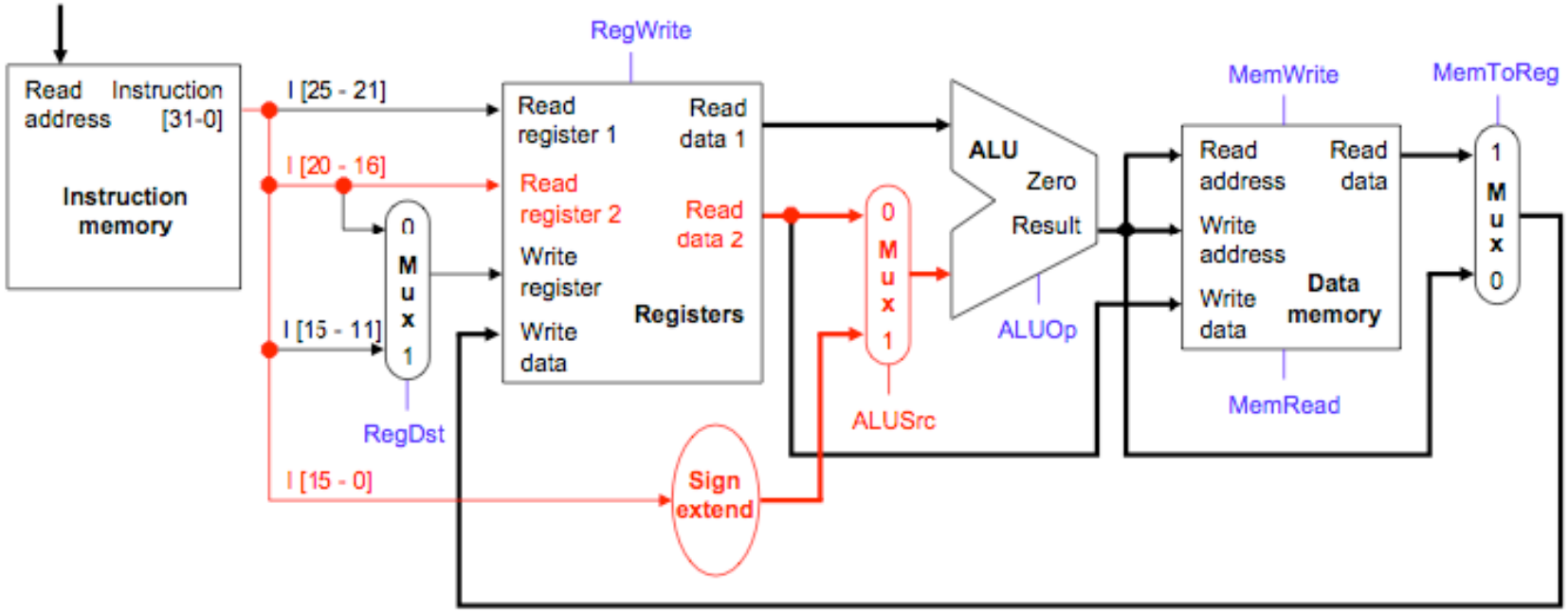
- Can't share adder and ALU.



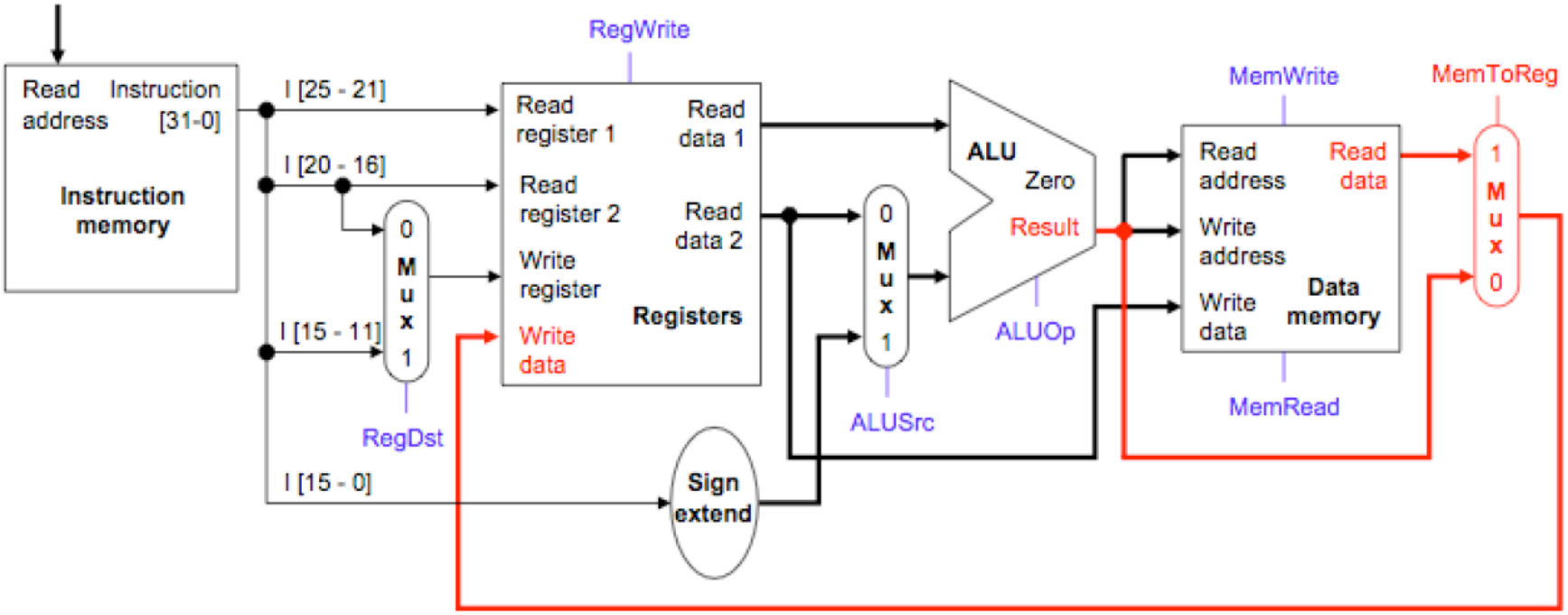
Executing a `lw` Instruction

- Example:

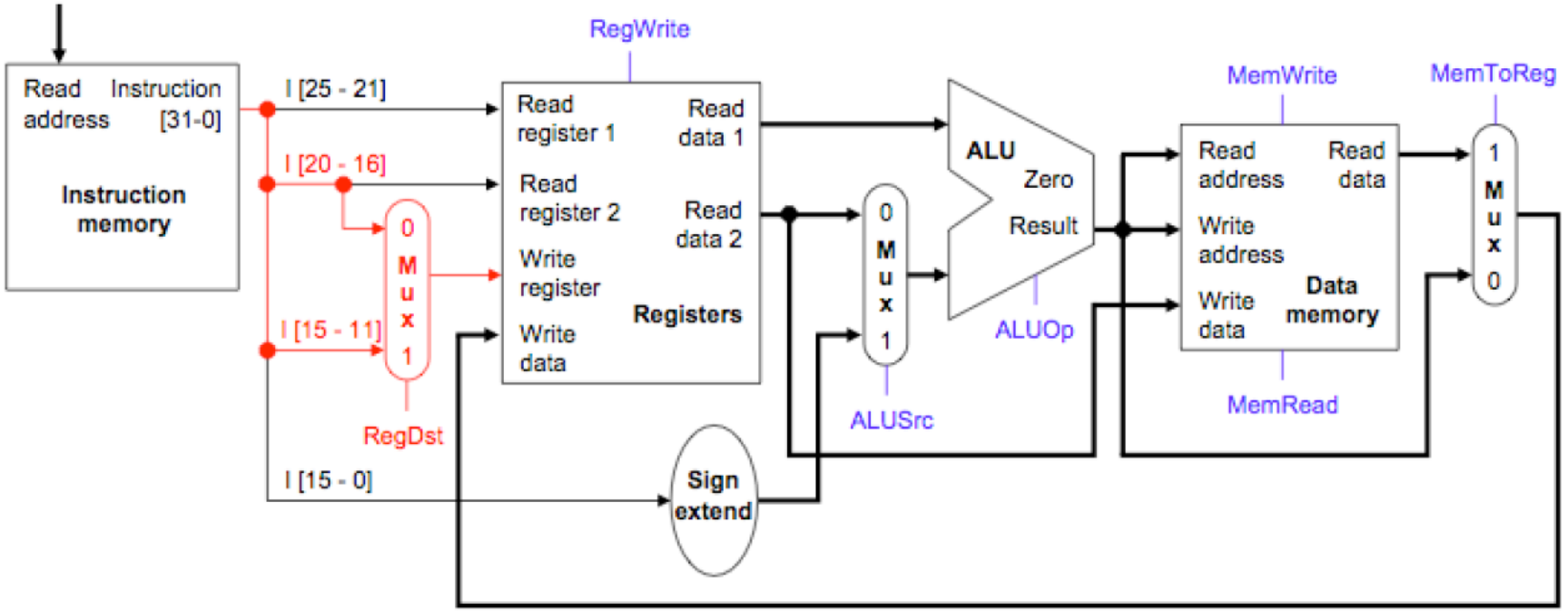




Executing a `lw` Instruction (cont'd)

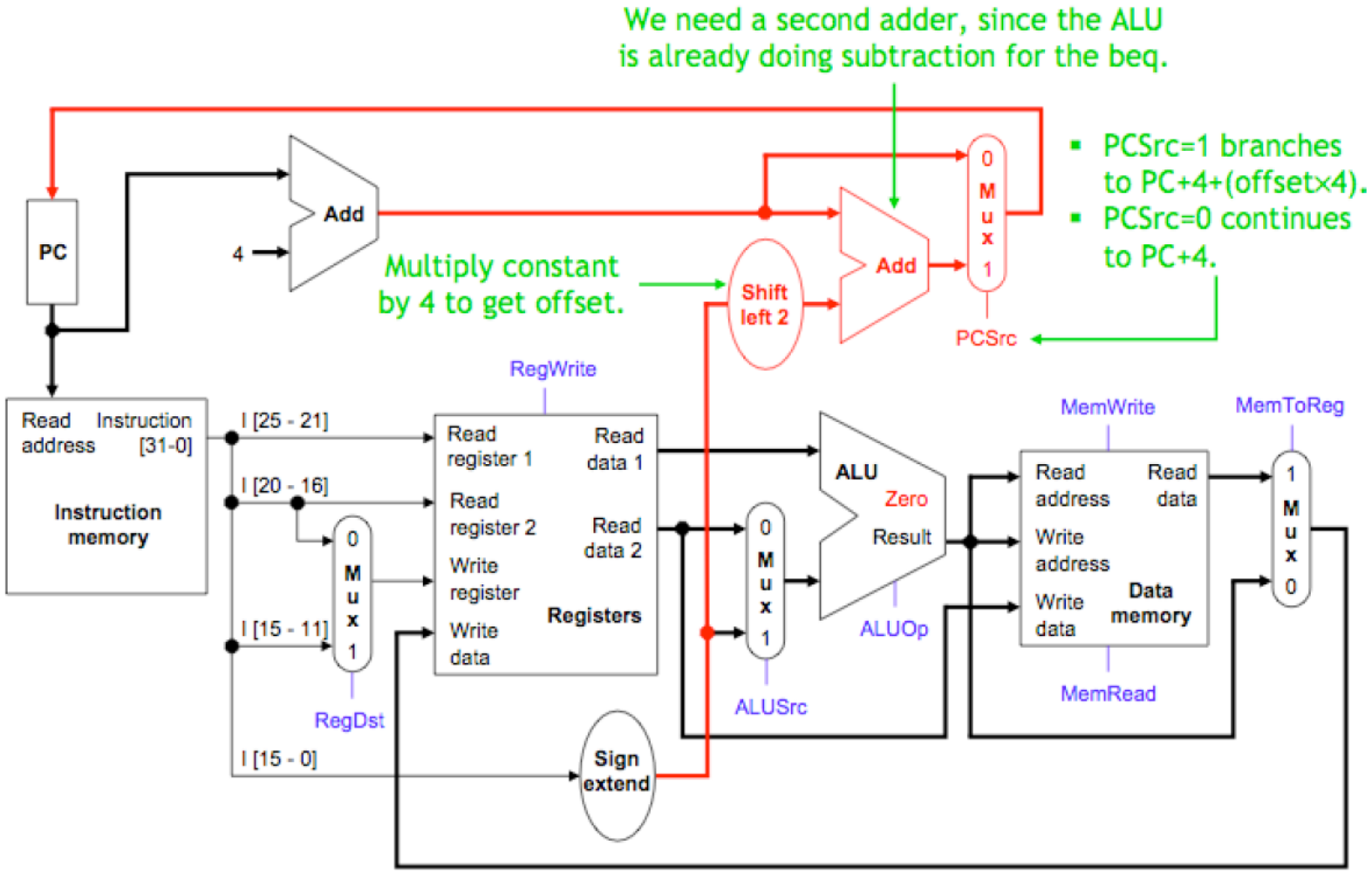


Executing a `lw` Instruction (cont'd)

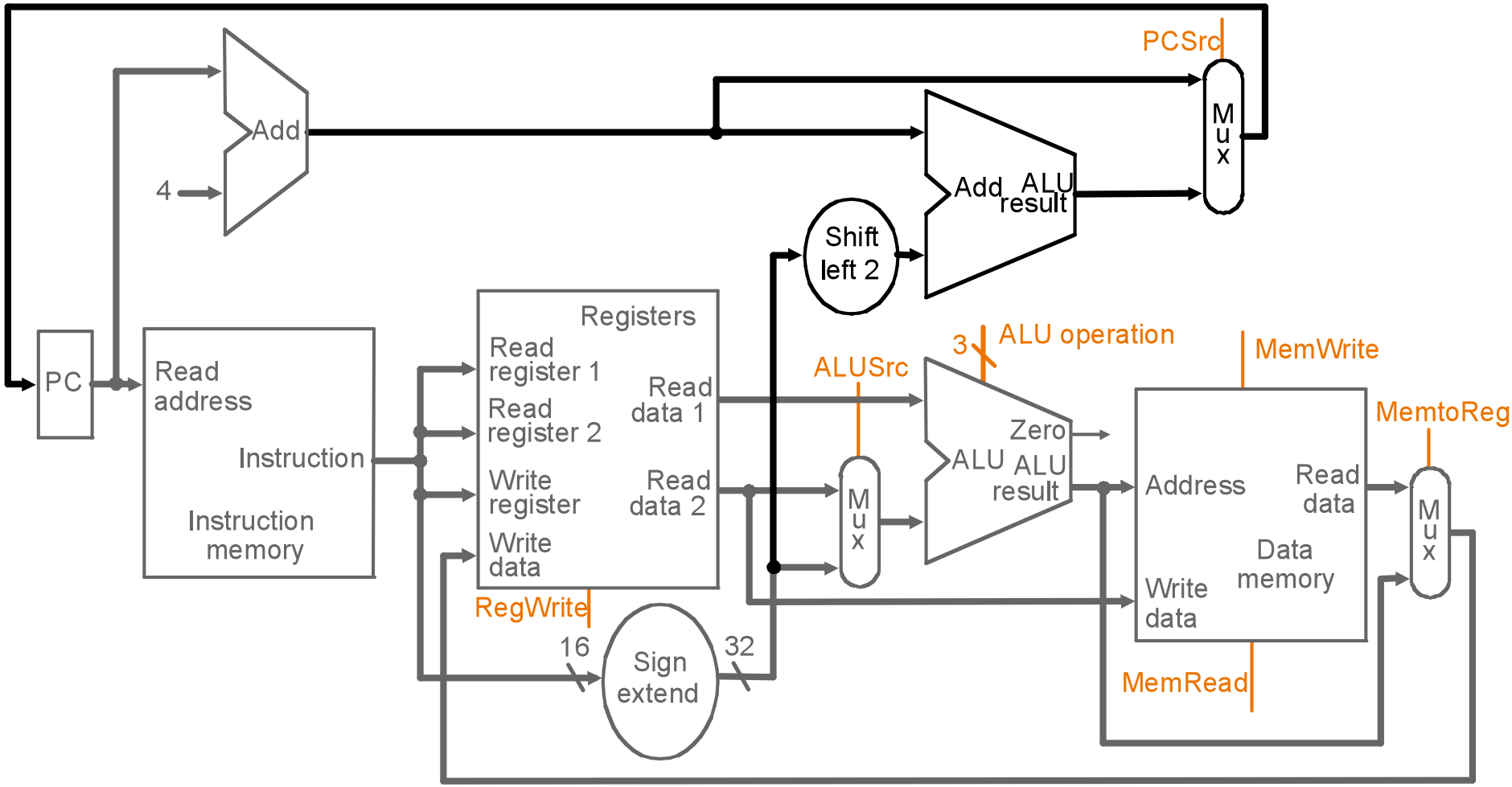


Adding the Branch Datapath: `beq $s0,$s1,label`

- Branch instructions use the main ALU for comparison, but needs separate adder for branch target address computation.

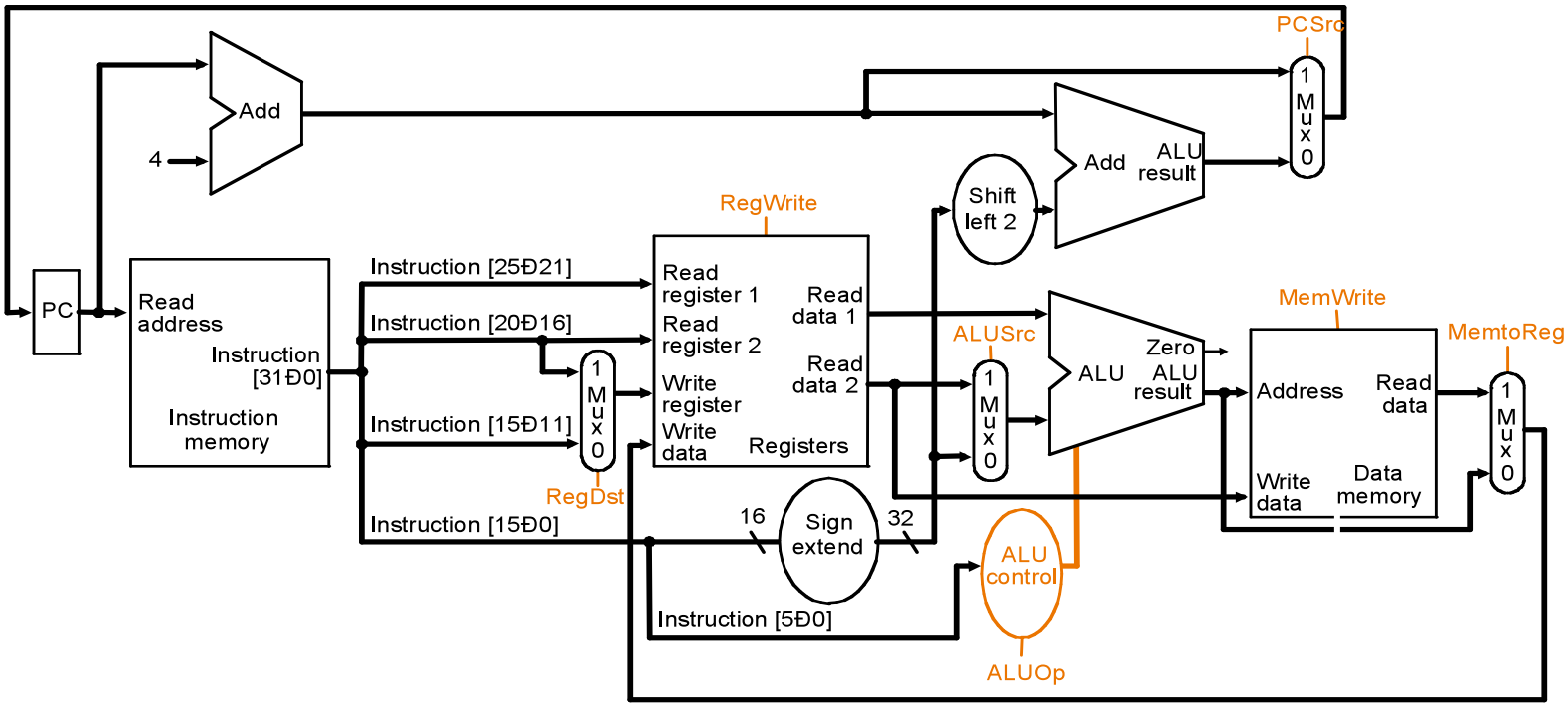


The Final Datapath



Controlling the Datapath

- The control unit is responsible for setting all the control signals in the datapath so that each instruction executes properly.
- Single-cycle implementation => Combinational controller.
- We'll construct a multi-level controller as follows:
  - An ALU controller
  - A Main controller that controls datapath and ALU controller



The ALU Control

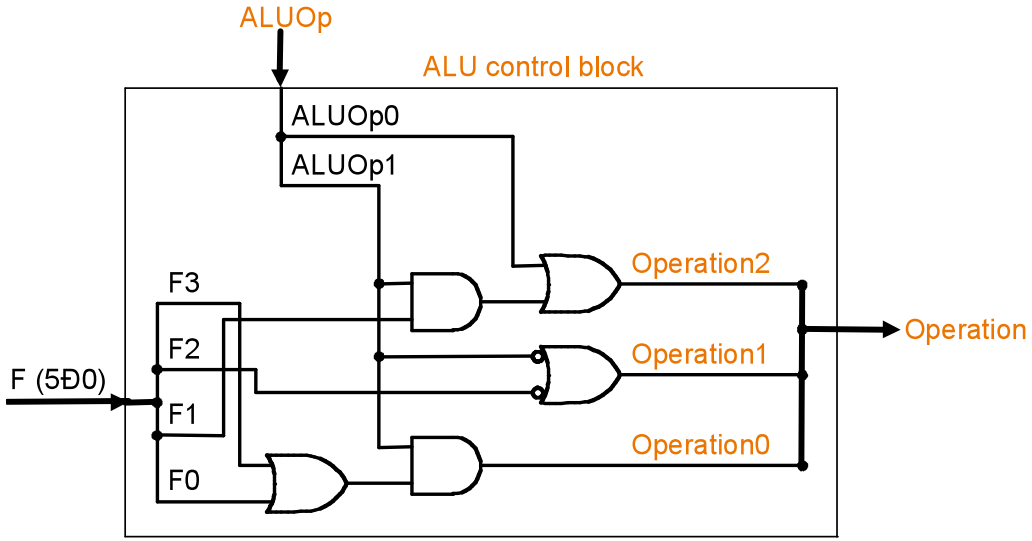
- Multi-levels of control (Main controller + ALU controller) is a common technique:
  - It reduces the size of the main controller.
  - Can potentially improve speed of the controller.
- Examining the datapath, the ALU must:
  - Add for lw/sw
  - Subtract for beq
  - Perform operation specified by function field for remaining instructions
- ALU controller inputs:
  - An ALUop field from main controller specifies a load/store (00), beq (01), R-Format (10).
  - 6-bit function field from instruction

ALU control input	Function
000	AND
001	OR
010	ADD
110	SUBTRACT
111	SLT

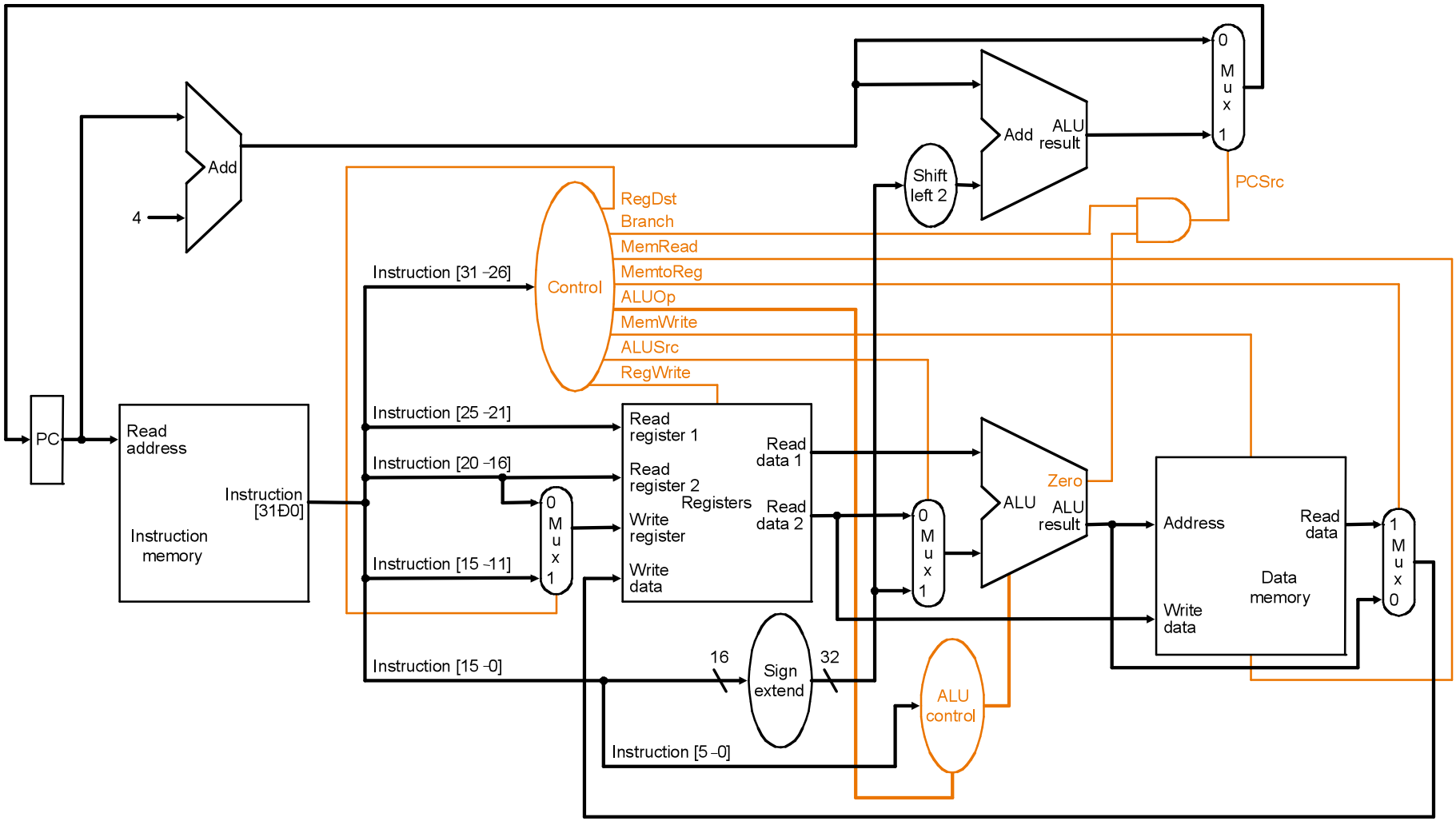
	input		input		output
Opcode	ALUop	Operation	Funct Field	Desired ALU action	ALU control input
lw	00	Load	xxxxxx	add	010
sw	00	Store	xxxxxx	add	010
beq	01	Branch EQ	xxxxxx	subtract	110
R-format	10	Add	100000	add	010
R-format	10	Subtract	100010	subtract	110
R-format	10	AND	100100	and	000
R-format	10	OR	100101	or	001
R-format	10	Set on LT	101010	set on LT	111

The ALU Control (cont'd)

ALUop[1:0]	Funct Field[5:0]	Operation
00	xxxxxx	010
00	xxxxxx	010
01	xxxxxx	110
10	100000	010
10	100010	110
10	100100	000
10	100101	001
10	101010	111



Datapath with Main Control



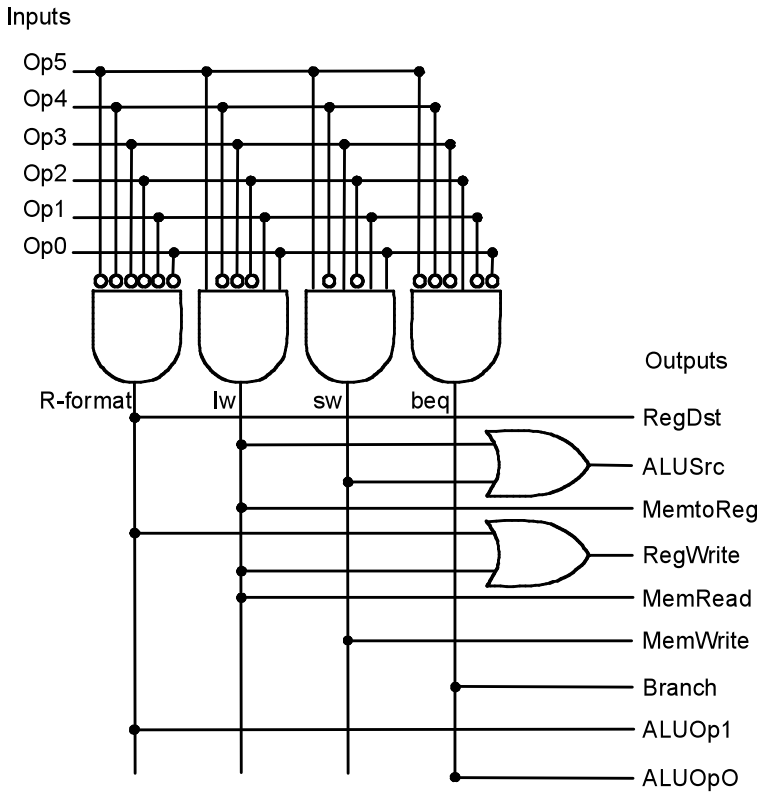
Designing the Main Controller

- Main controller takes as inputs the opcode field of the instruction.
- It controls the signals:
  - RegDst: Destination register address
  - RegWrite: Enables writing to register file
  - ALUsrc: selects appropriate second input to ALU
  - MemWrite/Read: Enables reading and writing from/to data memory
  - MemtoReg: Selects appropriate value to be written to register file.
  - PCsrc: Selects appropriate value to load next PC
    - This signal is set when there is a branch instruction AND zero output from ALU is 1
    - Controller generates a control signal called 'branch'

Instruction	opcode	RegDst	ALUsrc	MemtoReg	RegWrite	MemWrite	Branch	ALUop
R-Format	000000	1	0	0	1	0	0	10
lw	100011	0	1	1	1	0	0	00
sw	101011	x	1	X	0	1	0	00
beq	000100	x	0	X	0	0	1	01

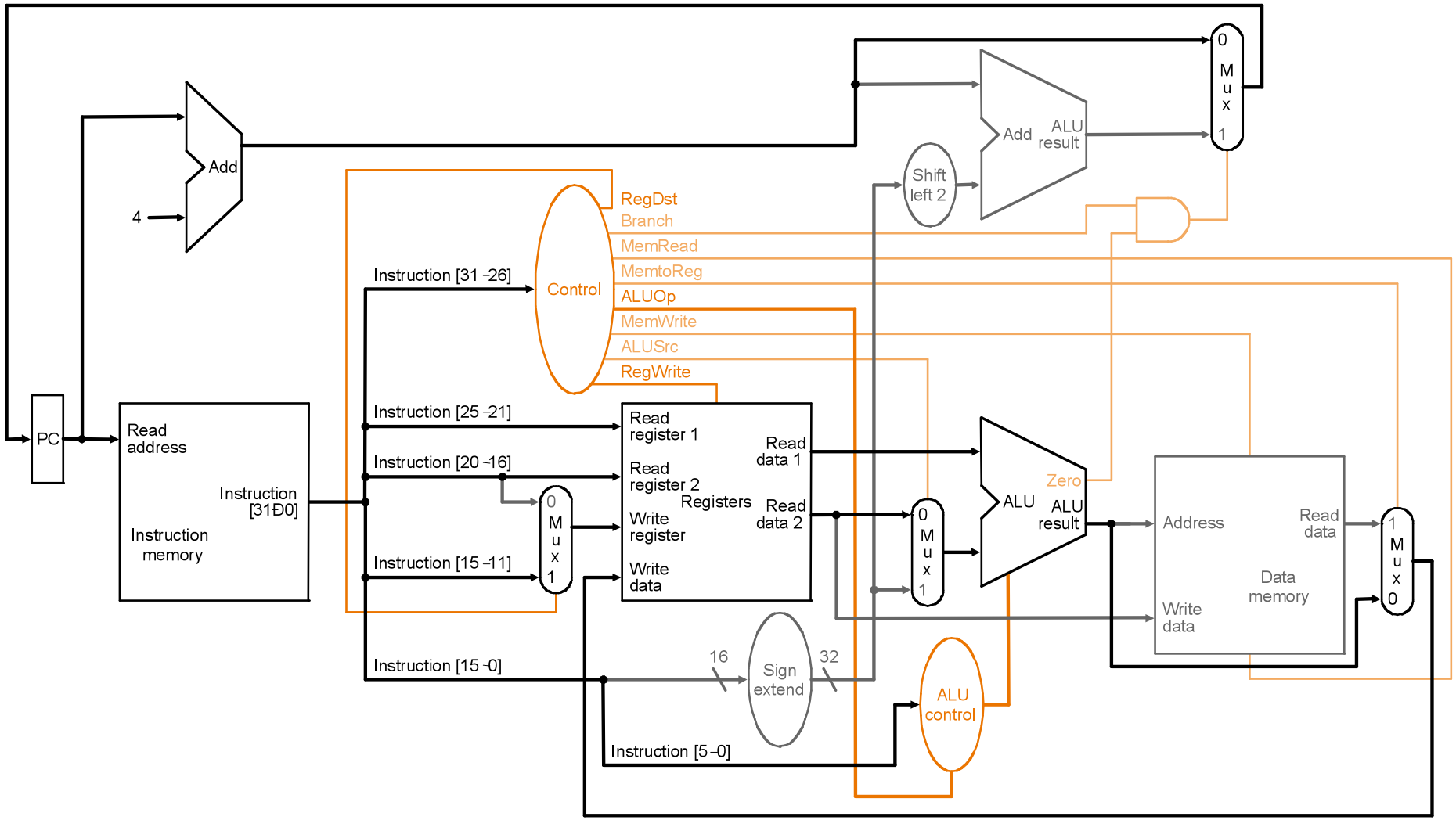
PLA Implementation of Main Controller

Instruction	opcode	RegDst	ALUsrc	MemtoReg	RegWrite	MemWrite	Branch	ALUop
R-Format	000000	1	0	0	1	0	0	10
lw	100011	0	1	1	1	0	0	00
sw	101011	x	1	X	0	1	0	00
beq	000100	x	0	X	0	0	1	01



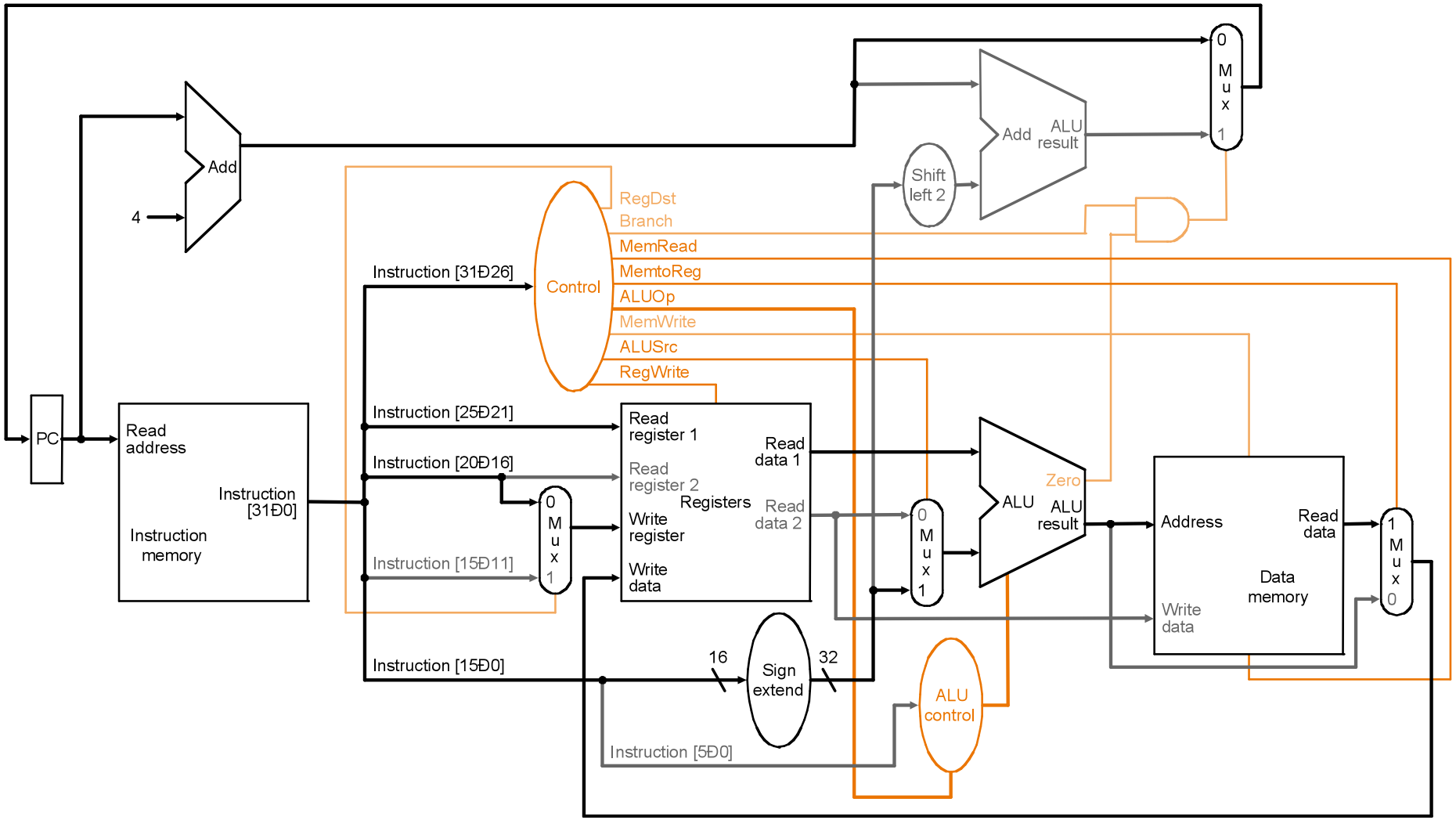
Example: Execution of `add $t1,$t2,$t3` on Datapath

- Divide operations into 4 steps: inst. fetch, operand fetch, exec., write-back



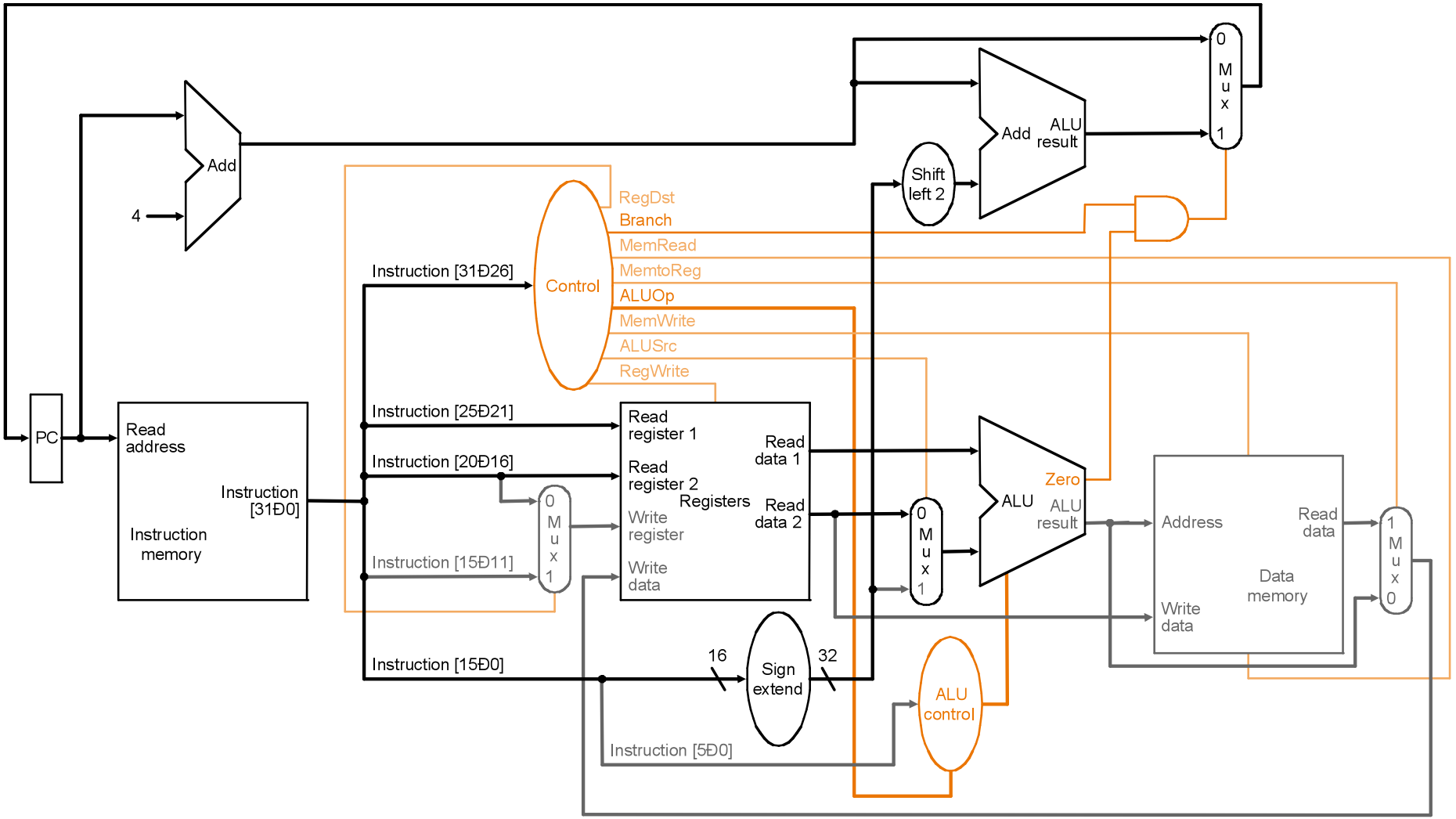
Example: Execution of `lw $t1,offset($t2)` on Datapath

- Divide into 5 steps: inst fetch, op fetch, addition, mem read, write-back.



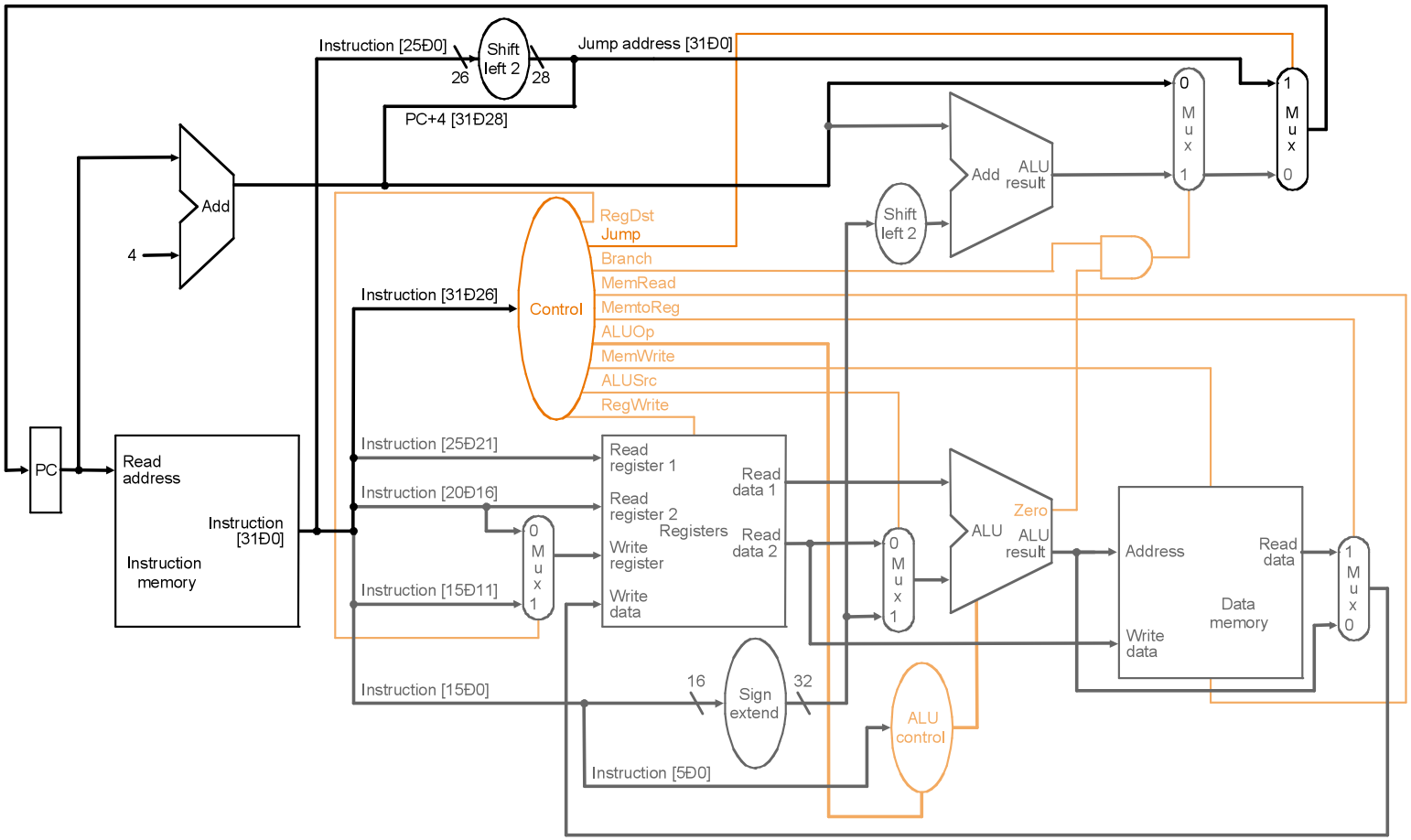
Example: Execution of `beq $t1,$t2,offset` on Datapath

- Divide into 4: inst fetch, op fetch, branch target address computation, branch decision.



Implementing Jumps

- We'll extend the datapath to include jump instructions: J Label
  - Jump is similar to branch but computes the target PC differently and is unconditional.
  - Jump address: 4 MSBs from PC || 26 bits from Label field || 00
  - Need an additional control signal called 'jump' and an additional MUX



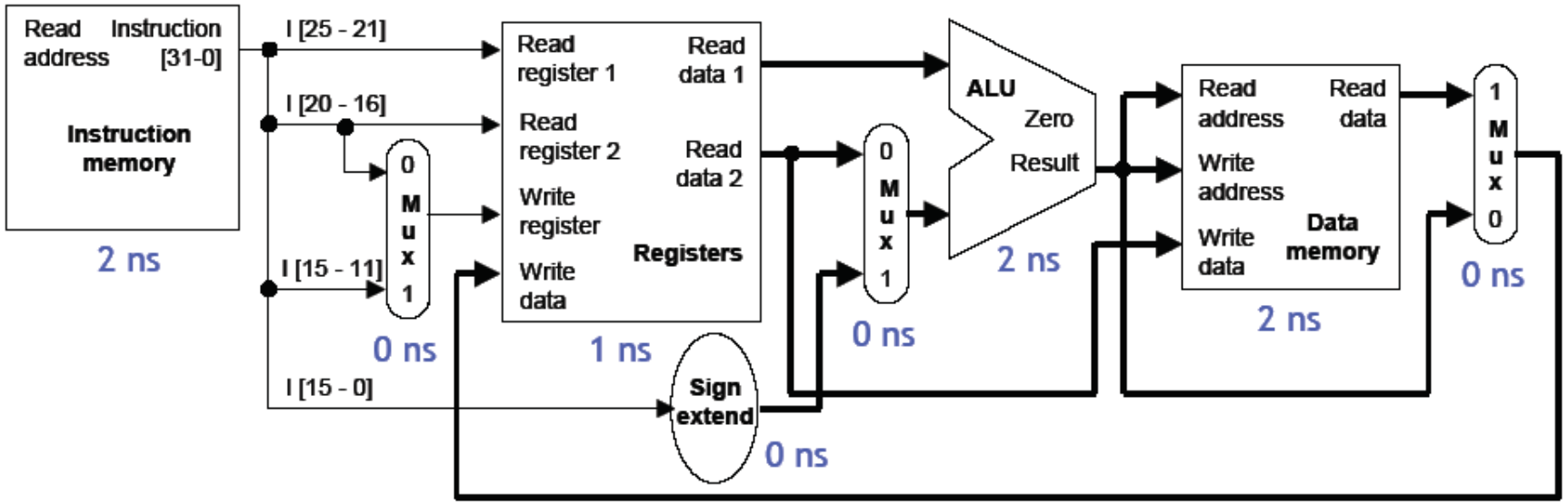
Performance of Single-Cycle Machines

- Single-Cycle operations recap:
  - On a positive clock edge, the PC is updated with a new address.
  - A new instruction can then be loaded from memory. The control unit sets the datapath signals appropriately so that:
    - Registers are read,
    - ALU output is generated,
    - Data memory is read or written, and
    - Branch target addresses are computed.
  - Several things happen on the next positive clock edge.
    - The register file is updated for arithmetic or lw instructions.
    - Data memory is written for a sw instruction.
    - The PC is updated to point to the next instruction.
- In a single-cycle datapath operations in Step 2 must complete within one clock cycle, before the next positive clock edge.
  - Assume that the operation time for the major functional units are:
  - memory (2ns,ultra-optimisitic), ALU and adders (2ns), register file access (1ns)
- Ignore delay of all other units
- If all instructions must complete within one clock cycle, then the cycle time has to be large enough to accommodate the slowest instruction.

The Slowest Instruction

- For example, lw $t0,–4($sp) needs 8ns:

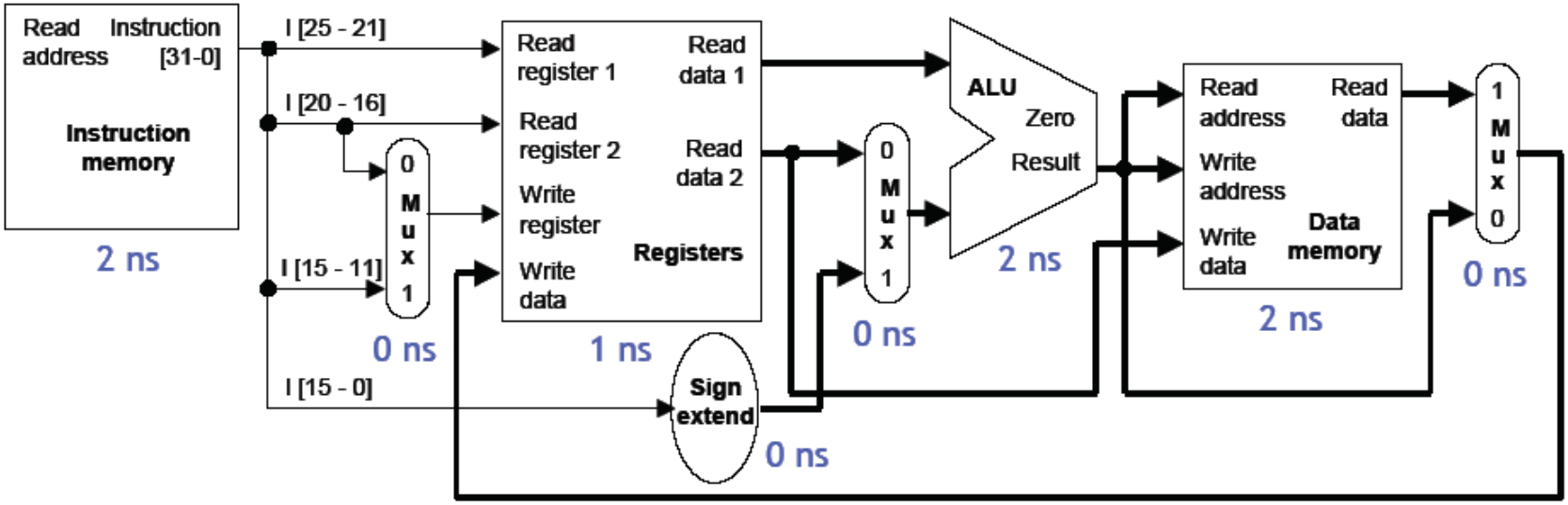
reading the instruction memory	2ns	8ns
reading the base register $sp	1ns	
computing memory address $sp-4	2ns	
reading the data memory	2ns	
storing data back to $t0	1ns	



The Slowest Instruction Determines the Clock-Cycle Time

- If we make the cycle time 8ns then every instruction will take 8ns, even if they don't need that much time.
- For example, the instruction add $s4, $t1, $t2 really needs just 6ns.

reading the instruction memory	2ns	6ns
reading registers $t1 and $t2	1ns	
computing $t1 + $t2	2ns	
storing the result into $s0	1ns	



Impact on Performance

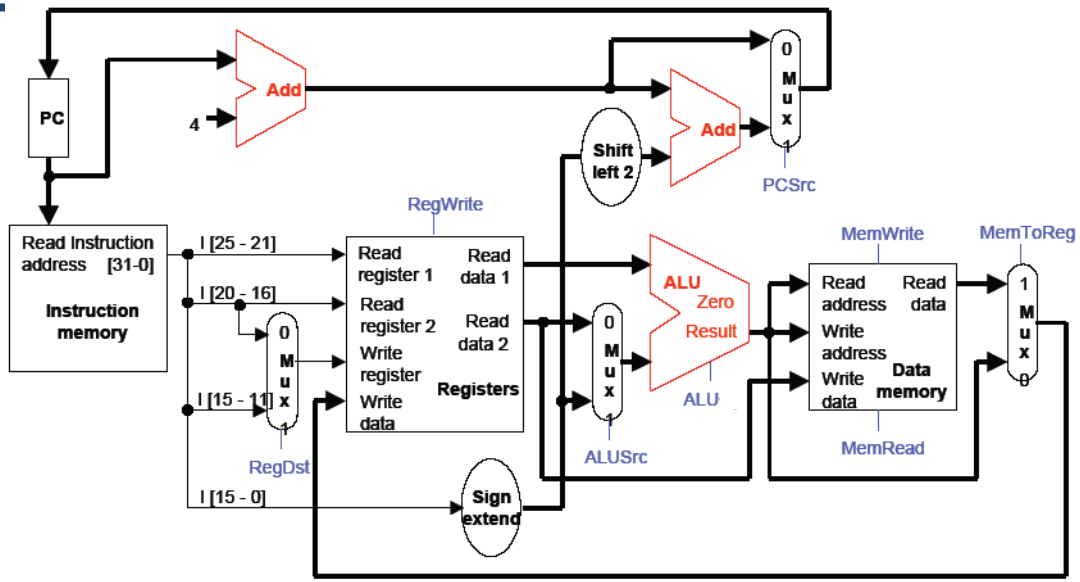
- With these same component delays, a sw instruction would need 7ns, and beq would need just 5ns.
- Let's consider the 'gcc' instruction mix:

Instruction	Frequency
Arithmetic	48%
Loads	22%
Stores	11%
Branches	19%

- With a single-cycle datapath, each instruction would require 8ns.
- But if we could execute instructions as fast as possible, the average time per instruction for gcc would be:
  - (48% x 6ns) + (22% x 8ns) + (11% x 7ns) + (19% x 5ns) = 6.36ns
- The single-cycle datapath is about 1.26 times slower!
- We've made very optimistic assumptions about memory latency:
  - Main memory accesses on modern machines is >50ns.
    - For comparison, an ALU on the Pentium4 takes ~0.3ns.
- Our worst case cycle (loads/stores) includes 2 memory accesses
  - A modern single cycle implementation would be stuck at <10Mhz.
  - Caches will improve common case access time, not worst case.
- Tying frequency to worst case path violates first law of performance!!

What About Hardware-Efficiency?

- A single-cycle datapath also uses extra hardware—one ALU is not enough, since we must do up to three calculations in one clock cycle for a beq.



- Remember we had to use a Harvard architecture with two memories to avoid requiring a memory that can handle two accesses in one cycle.