Designing

MIPS Processor

(Single-Cycle)

Introduction

- Implementation of the system that includes MIPS processor and memory

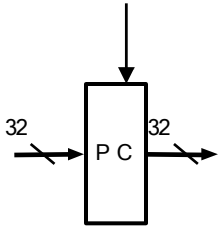
- The design will include support for execution of only:
  - memory-reference instructions: **lw** & **sw**
  - arithmetic-logical instructions: **add**, **sub**, **and**, **or**…
  - control flow instructions: **beq** & **j**

- But that design will provide us with principles, so many more instructions could be easily added such as: addu, lb, lbu, lui, addi, adiu, sltu, slti, andi, ori, xor, xori, jal, jr, jalr, bne, beqz, bgtz, bltz, nop, mfhi, mflo, mfepc, mfco, lwc1, swc1, etc.

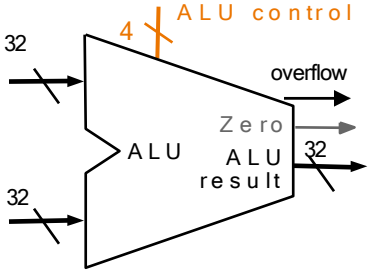
Single Cycle Design

- first design a simpler processor that executes each instruction in only one clock cycle time.
- This is not efficient from performance point of view, since:
  - a clock cycle time (i.e. clock rate) must be chosen such that the longest instruction can be executed in one clock cycle and
  - that makes shorter instructions execute in one unnecessarily long cycle.
- Additionally, no resource in the design may be used more than once per instruction, thus some resources will be duplicated.
- The singe cycle design will require:
  - two memories (instruction and data),
  - two additional adders.

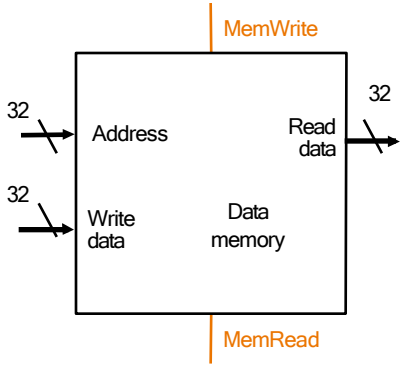
Elements for Datapath Design



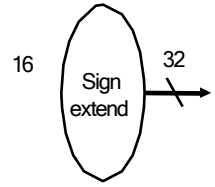
a. Program counter



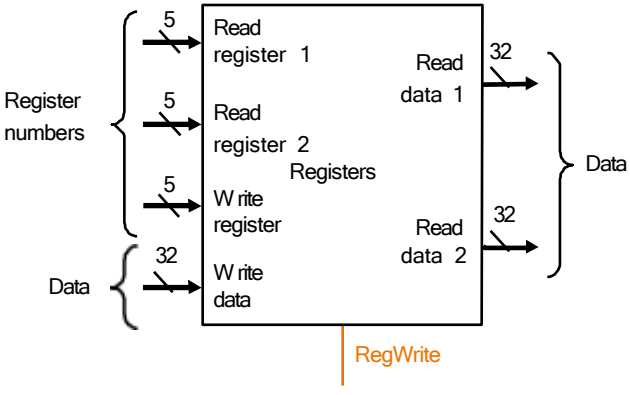
c. ALU



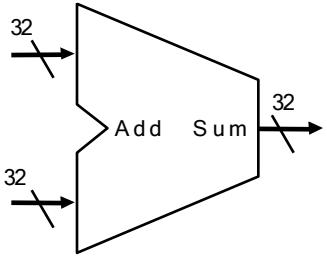
e. Data memory unit



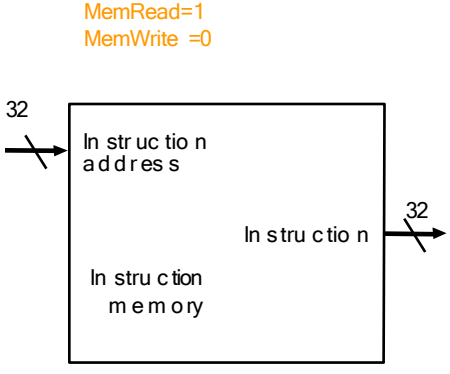
g. Sign-extension unit



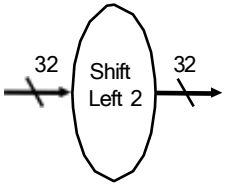
b. Register File



d. Adder

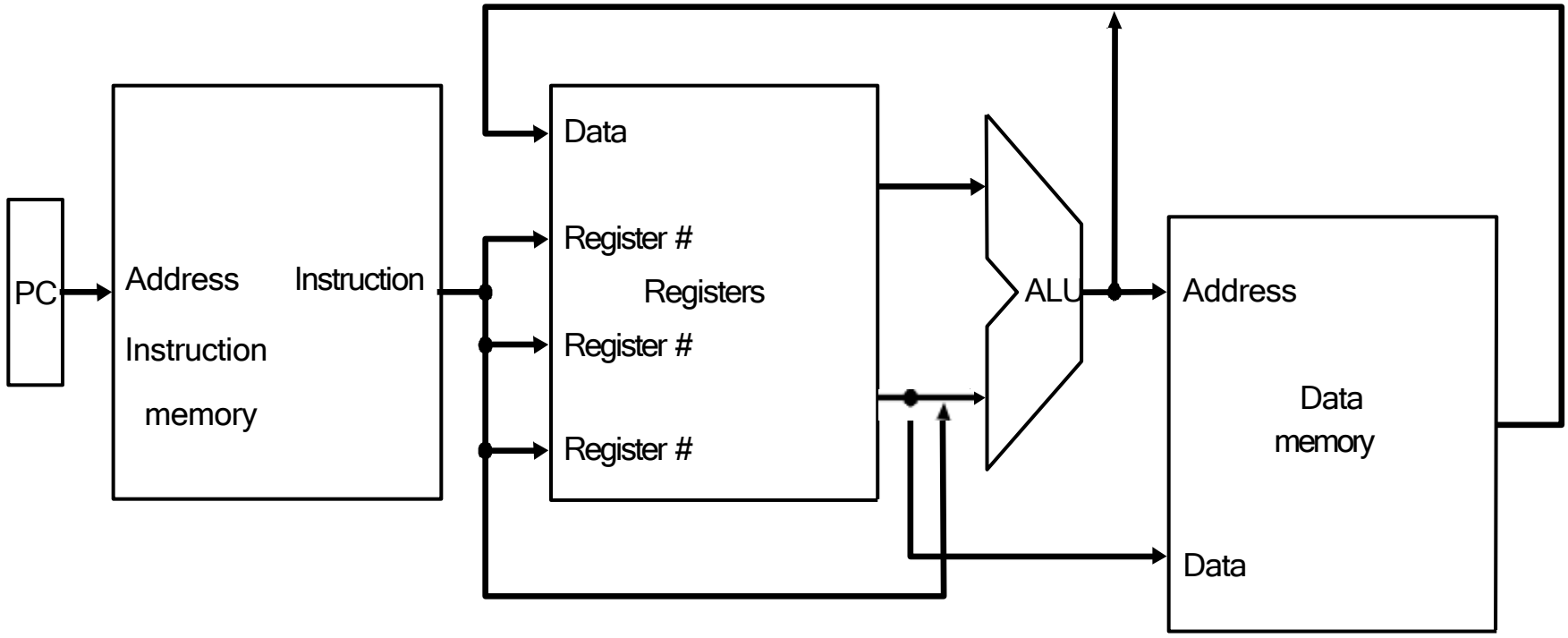


f. Instruction memory



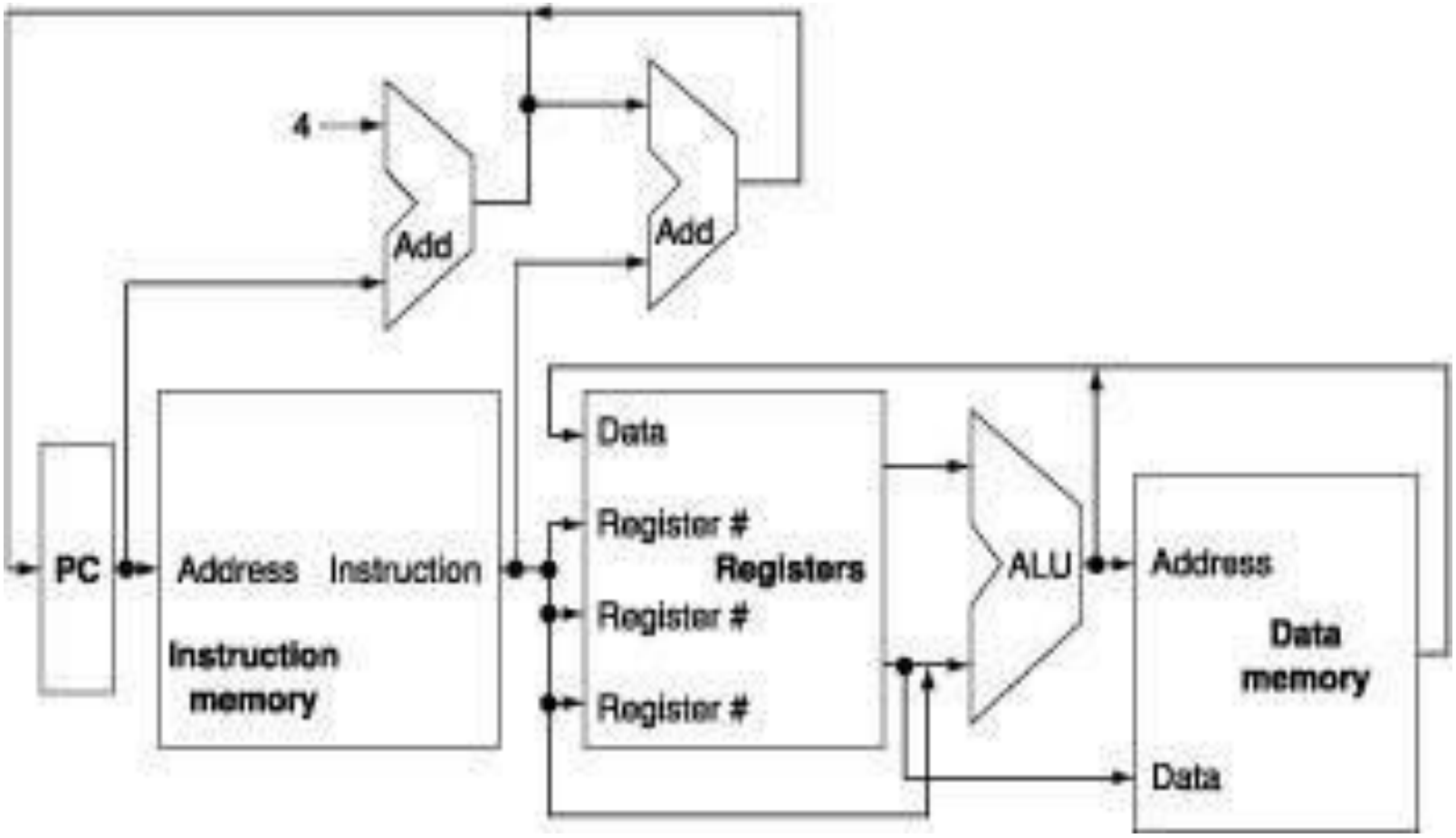
h. Shift left 2

Abstract /Simplified View (1st look)



- This generic implementation:
  - uses the program counter (PC) to supply instruction address,
  - gets the instruction from memory,
  - reads registers,
  - uses the instruction opcode to decide exactly what to do.

Abstract /Simplified View (2nd look)

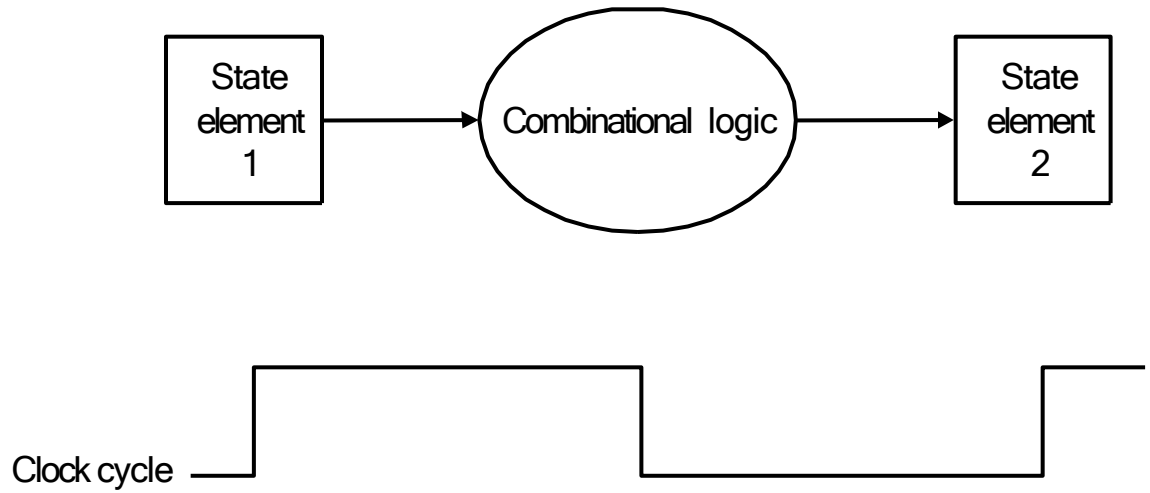


- **PC is incremented by 4 by most instructions, and 4 + 4×offset by branch instructions.**
- **Jump instructions change PC differently (not shown)**

.

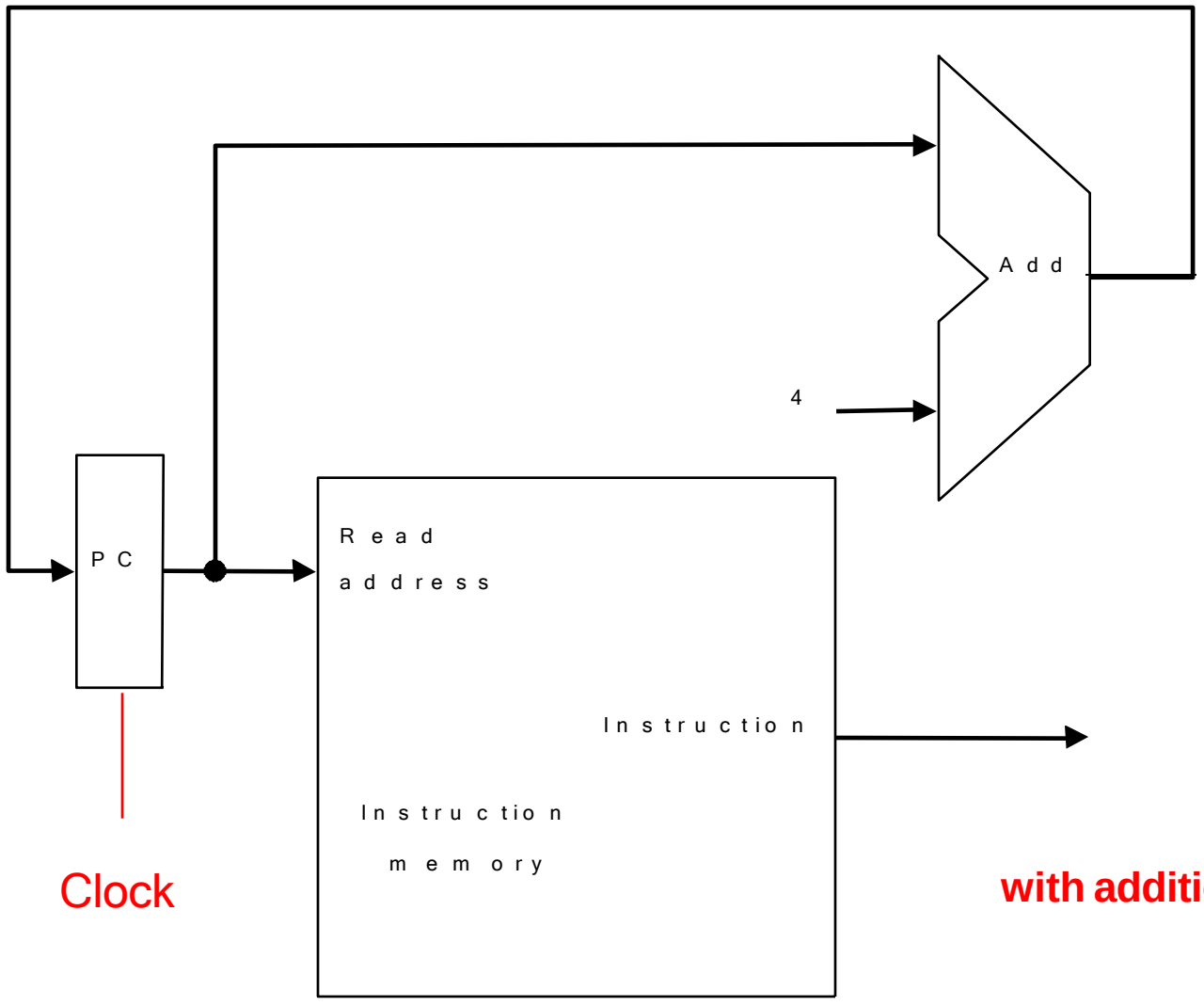
Our Implementation

- An edge triggered methodology
- Typical execution:
  - read contents of some state elements at the beginning of the clock cycle,
  - send values through some combinational logic,
  - write results to one or more state elements at the end of the clock cycle.



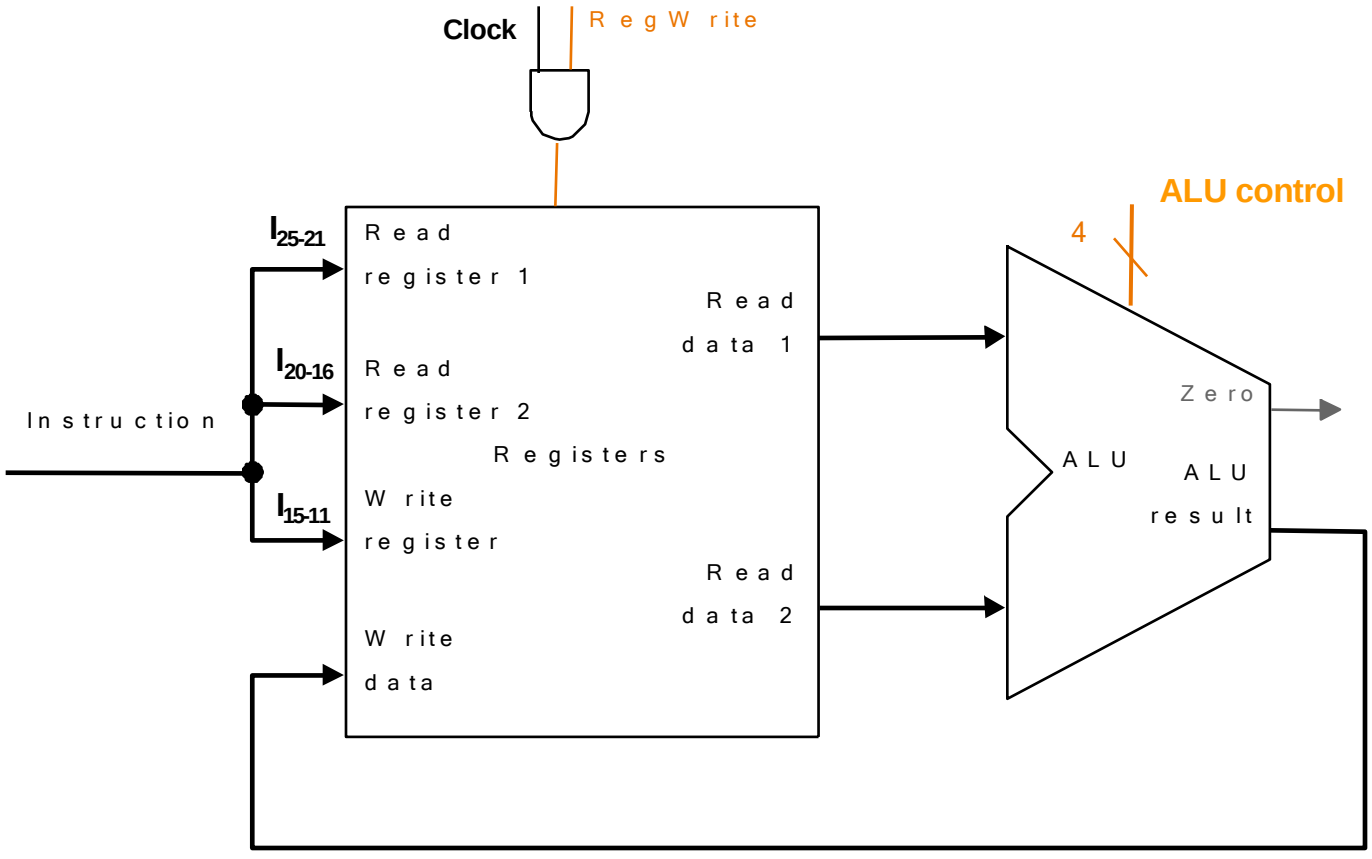
- An edge triggered methodology allows a state element to be read and written in the same clock cycle.

Incrementing PC & Fetching Instruction

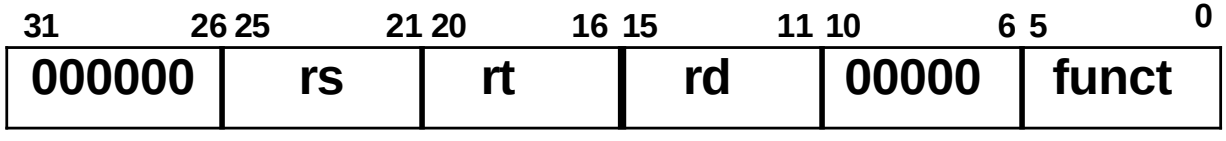


with addition in red

Datapath for R-type Instructions



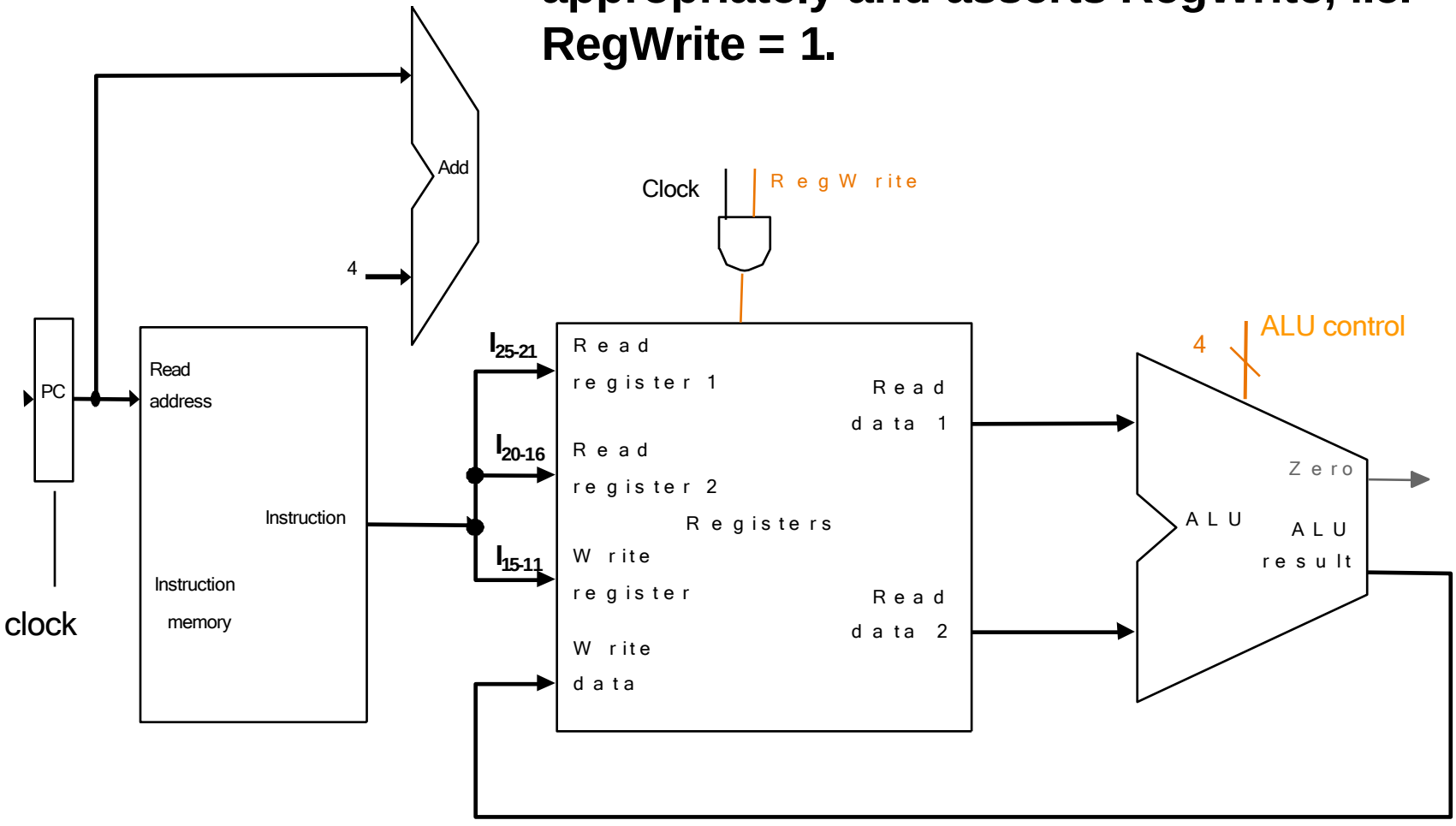
R-type



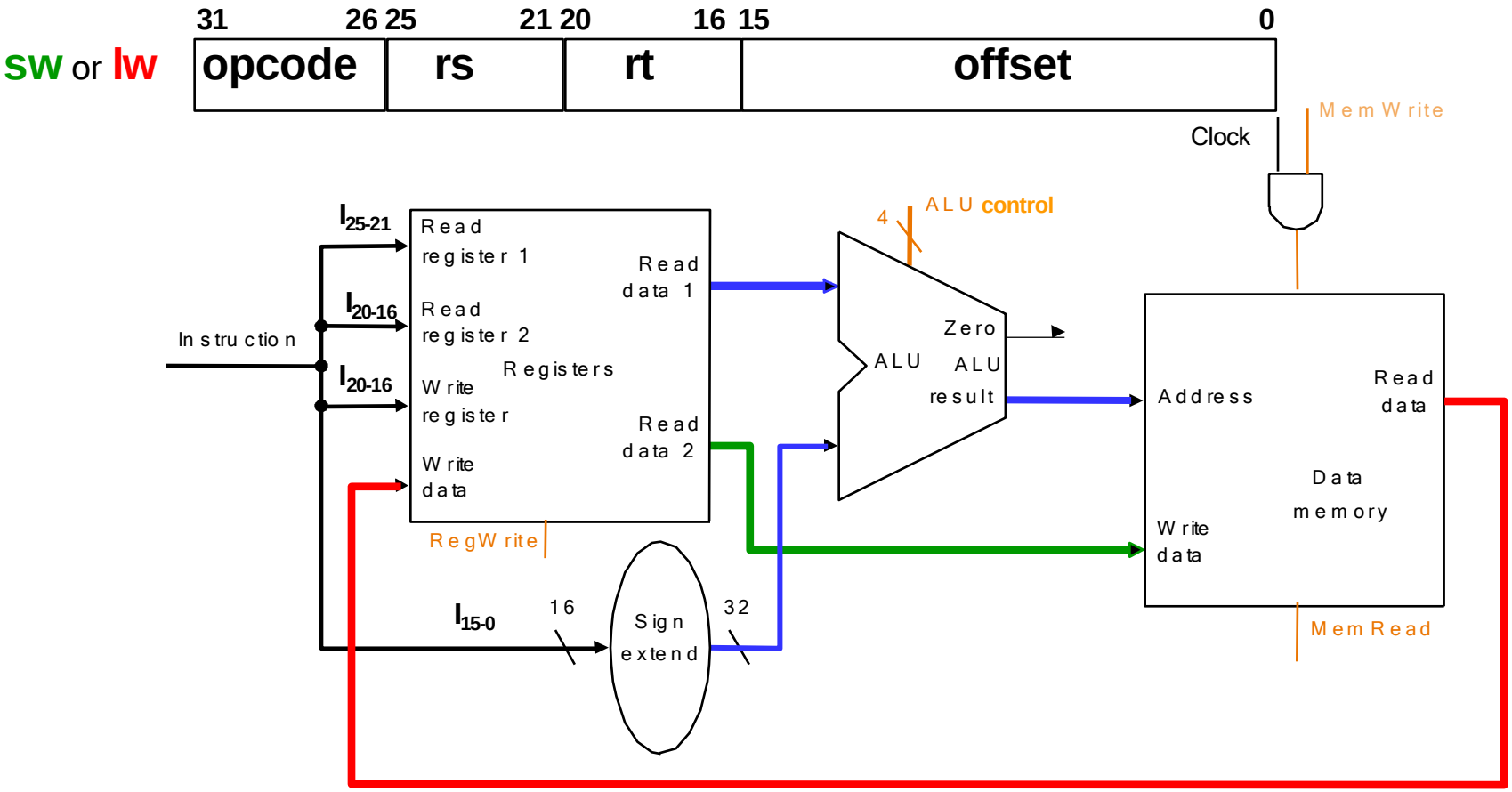
add = 32
sub = 34
slt = 42
and = 36
or = 37
nor = 39

Complete Datapath for R-type Instructions

Based on contents of op-code and funct fields, Control Unit sets ALU control appropriately and asserts RegWrite, i.e. RegWrite = 1.



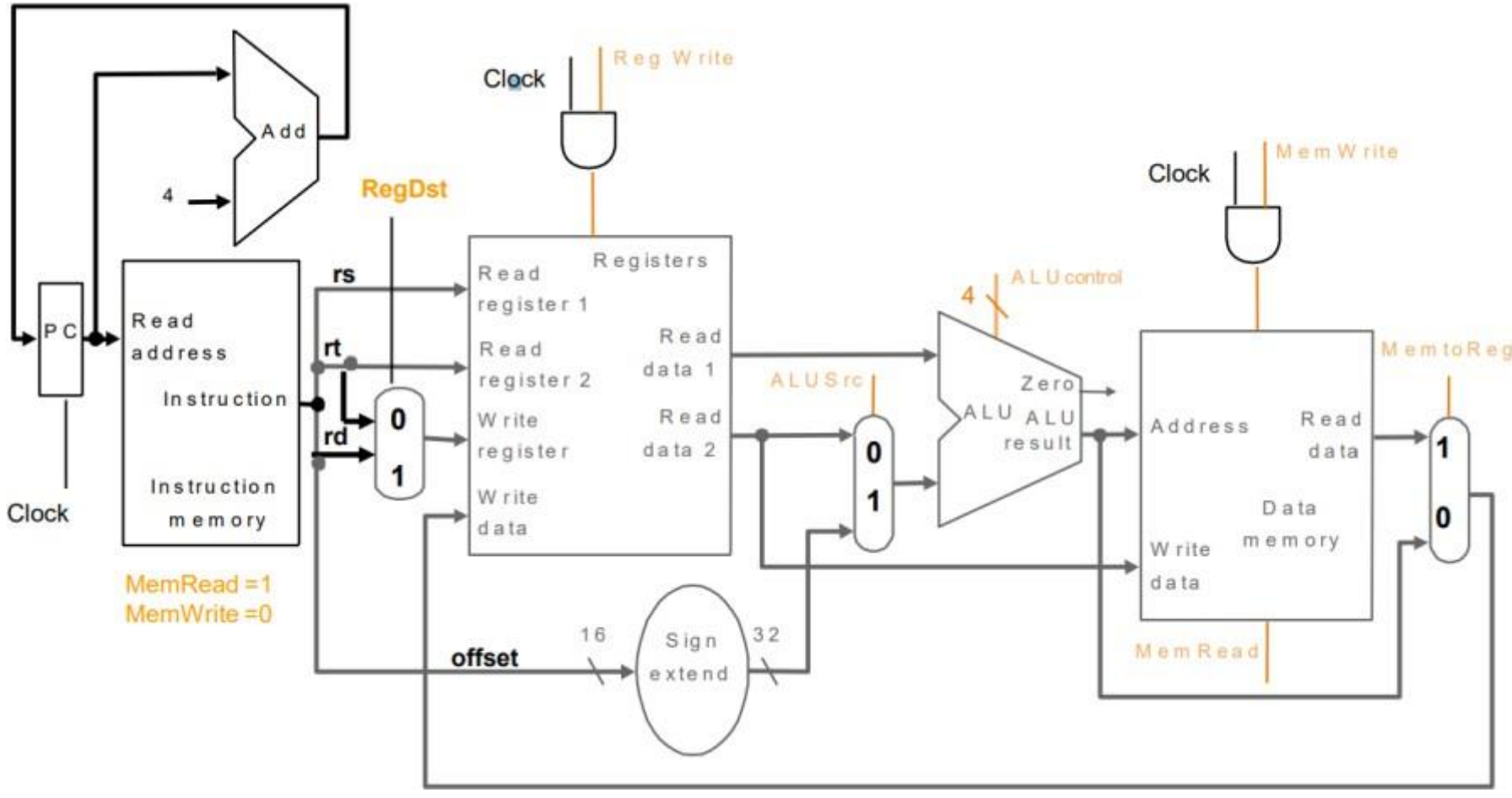
Datapath for LW and SW Instructions



Control Unit sets:

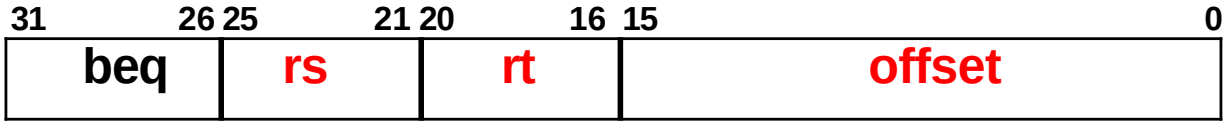
- **ALU control = 0010 (add) for address calculation for both lw and sw**
- **MemRead=0, MemWrite=1 and RegWrite=0 for sw**
- **MemRead=1, MemWrite=0 and RegWrite=1 for lw**

Datapath for R-type, LW & SW Instructions

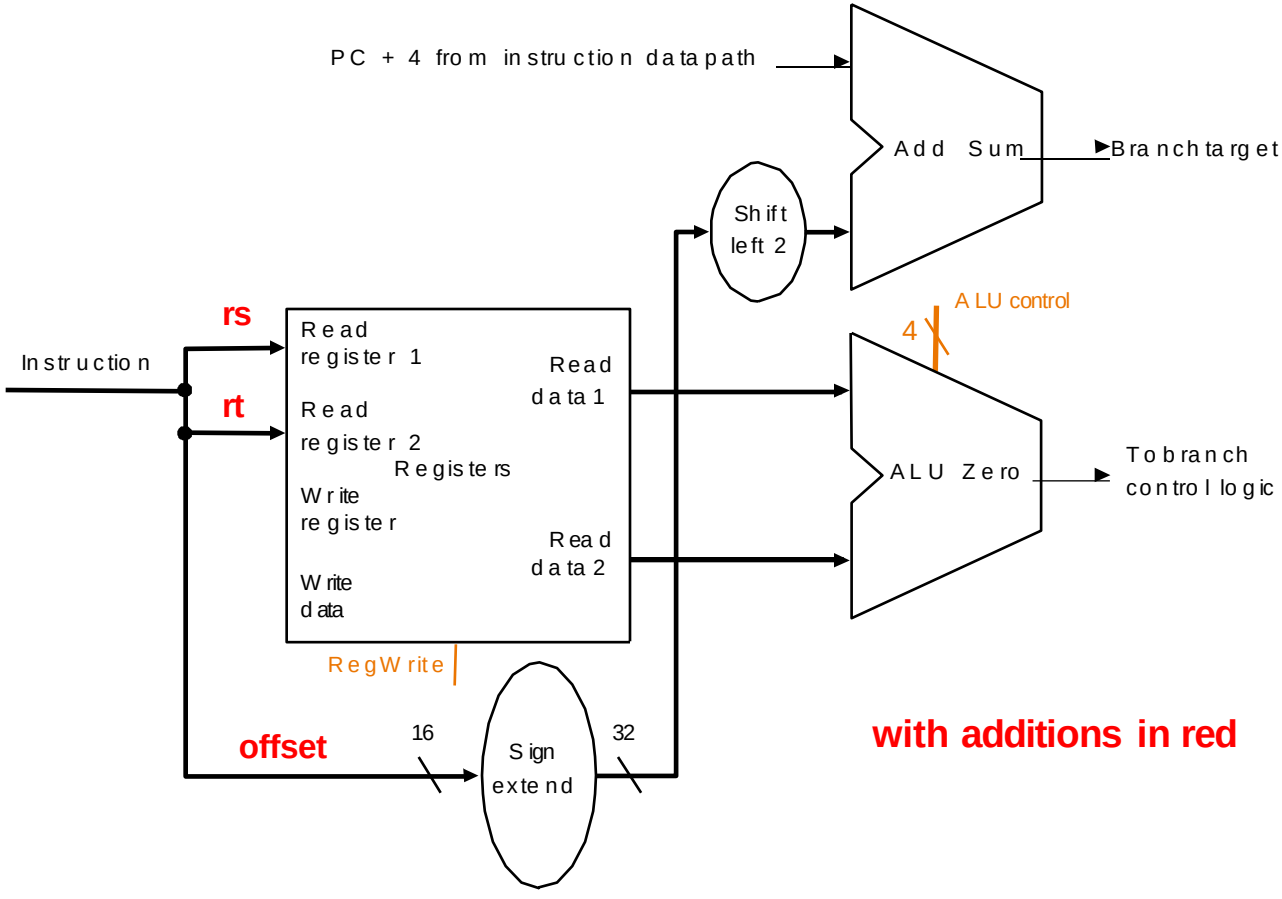


Let us determine setting of control lines for R-type, lw & sw instructions.

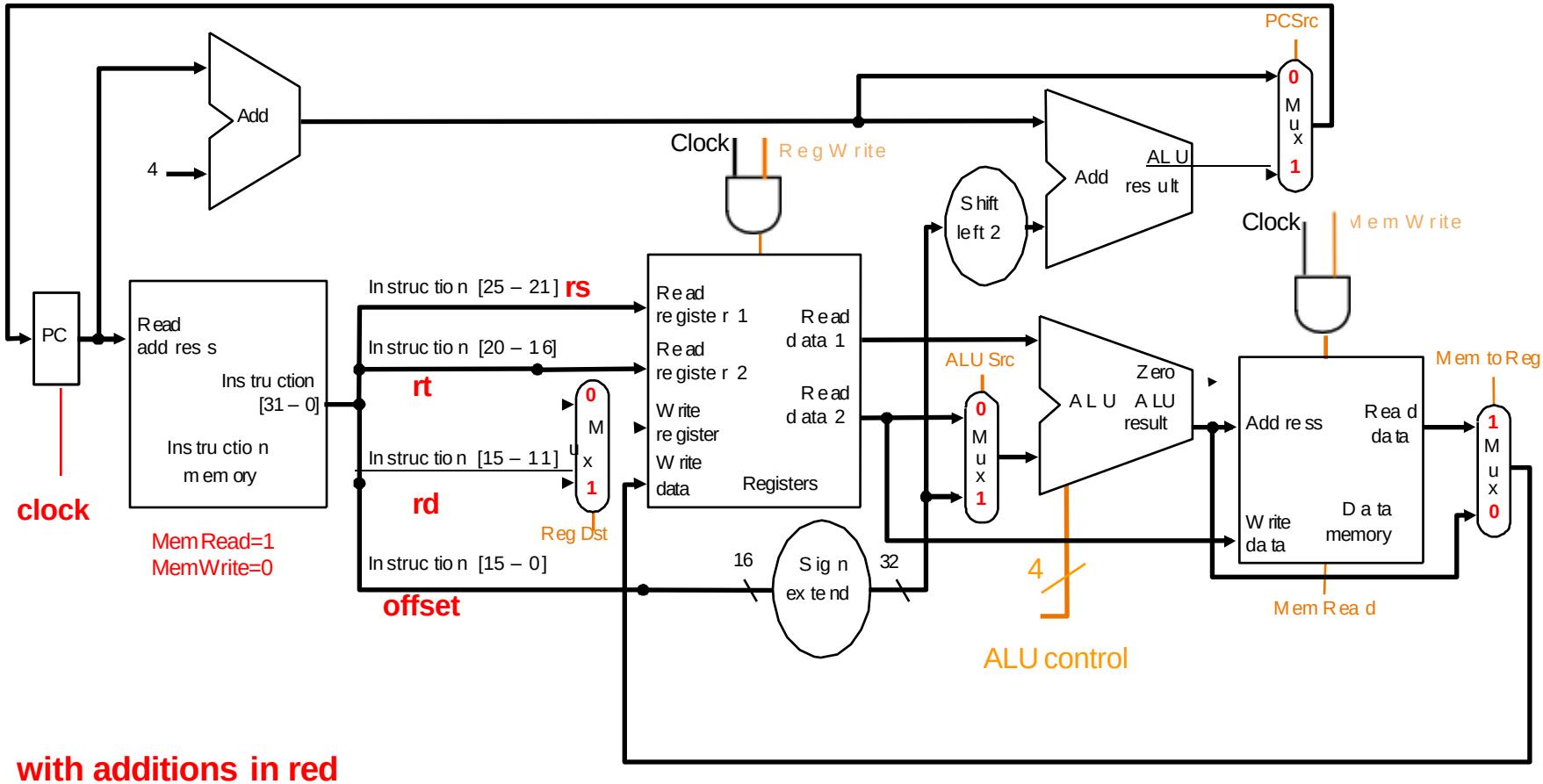
Datapath for BEQ Instruction



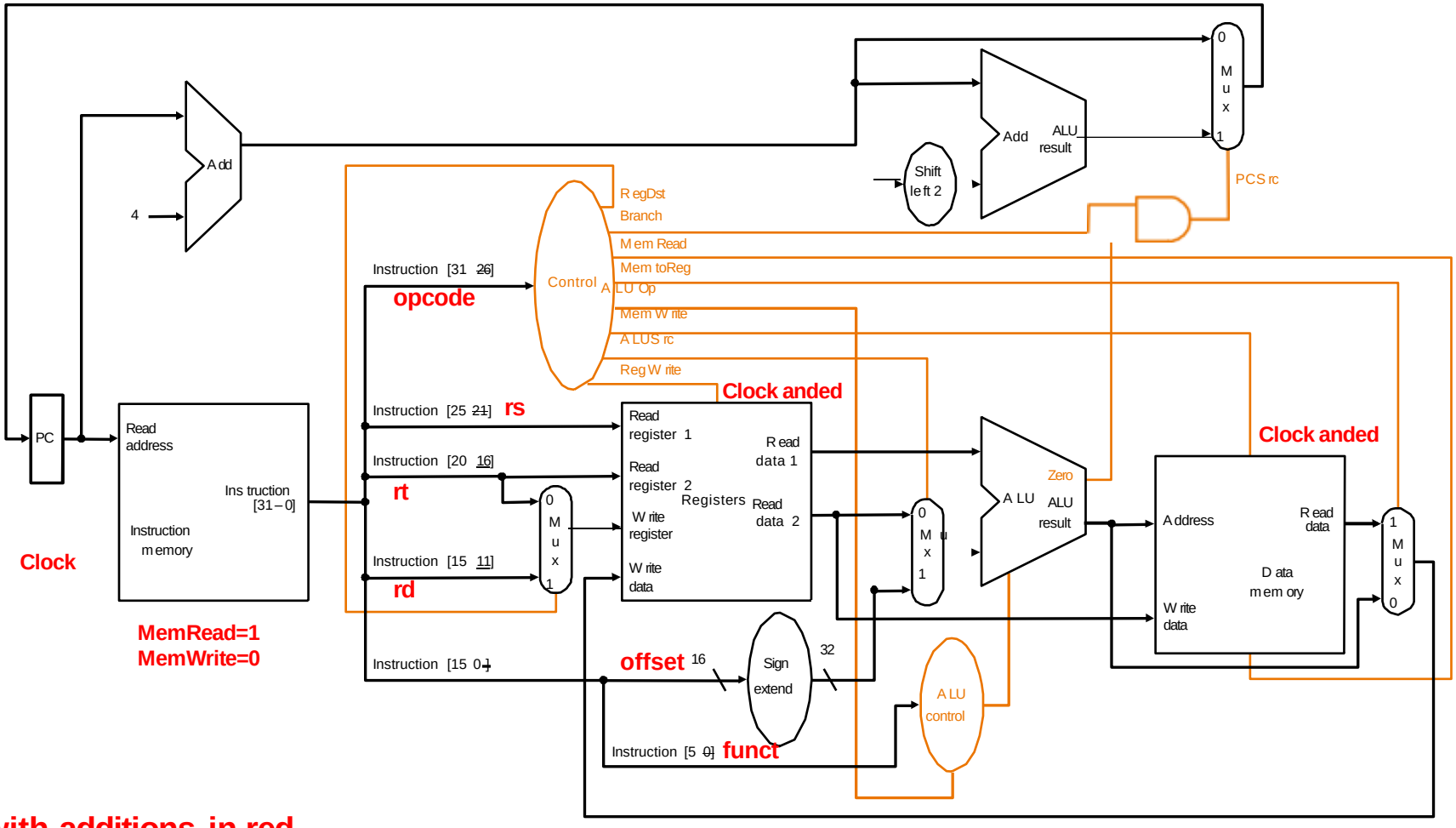
Branch target = [PC] + 4 + 4×offset



Datapath for R-type, LW, SW & BEQ



Control Unit and Datapath



Truth Table for (Main) Control Unit

	Input	Output								
	Op-code	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUp0
R-type	000000	1	0	0	1	**d**	0	0	1	0
lw	100011	0	1	1	1	1	0	0	0	0
sw	101011	d	1	d	0	0	1	0	0	0
beq	000100	d	0	d	0	**d**	0	1	0	1

- **ALUOp[1-0] = 00 → signal to ALU Control unit for ALU to perform add function, i.e. set Ainvert = 0, Binvert=0 and Operation=10**
- **ALUOp[1-0] = 01 → signal to ALU Control unit for ALU to perform subtract function, i.e. set Ainvert = 0, Binvert=1 and Operation=10**
- **ALUOp[1-0] = 10 → signal to ALU Control unit to look at bits $I_{[5-0]}$ and based on its pattern to set Ainvert, Binvert and Operation so that ALU performs appropriate function, i.e. add, sub, slt, and, or & nor**

Truth Table of ALU Control Unit

Input								Output	
ALUOp		Funct field						ALU Control	
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0		
0	0	d	d	d	d	d	d	0 0 10	add
0	1	d	d	d	d	d	d	0 1 10	sub
1	0	1	0	0	0	0	0	0 0 10	add
1	0	1	0	0	0	1	0	0 1 10	sub
1	0	1	0	0	1	0	0	0 0 00	and
1	0	1	0	0	1	0	1	0 0 01	or
1	0	1	0	1	0	1	0	0 1 11	slt
1	0	1	0	0	1	1	1	1 1 00	nor

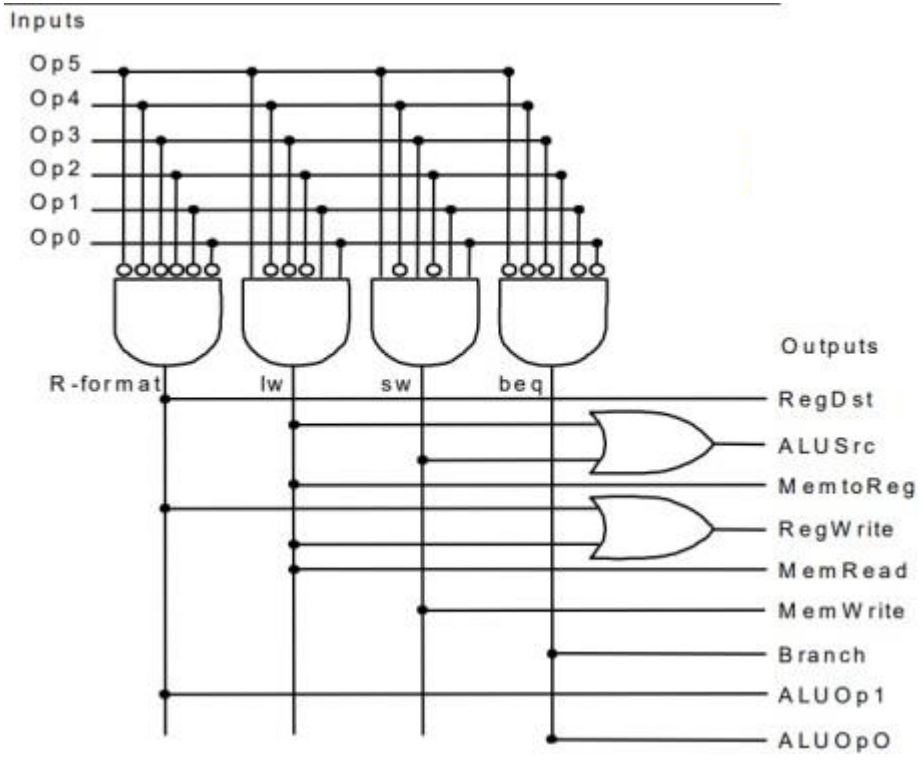
Ainvert Bivert Operation

Design of (Main) Control Unit

Op-code bits 5 4 3 2 1 0	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUp0
0 0 0 0 0 0	1	0	0	1	d	0	0	1	0
1 0 0 0 1 1	0	1	1	1	1	0	0	0	0
1 0 1 0 1 1	d	1	d	0	0	1	0	0	0
0 0 0 1 0 0	d	0	d	0	d	0	1	0	1

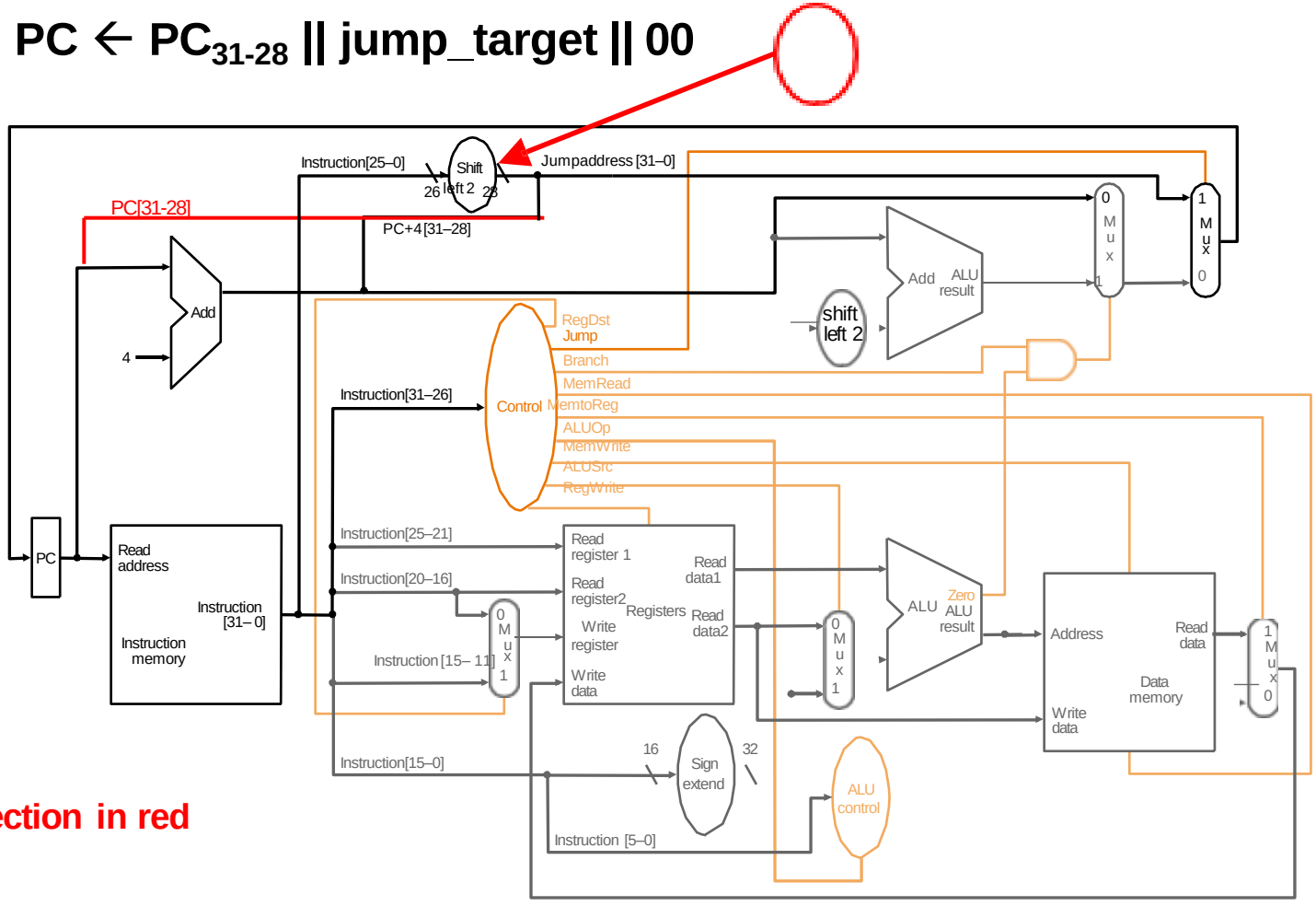
$$RegDst = \overline{Op_5}\,\overline{Op_4}\,\overline{Op_3}\,\overline{Op_2}\,\overline{Op_1}\,\overline{Op_0}$$

$$ALUSrc = Op_5\overline{Op_4}\,\overline{Op_3}\,\overline{Op_2}\,Op_1\,Op_0 + \overline{Op_5}\,Op_4\,Op_3\,\overline{Op_2}\,Op_1\,Op_0$$



Datapath for R-type, LW, SW, BEQ & J

$PC \leftarrow PC_{31\text{-}28} \,||\, jump\_target \,||\, 00$



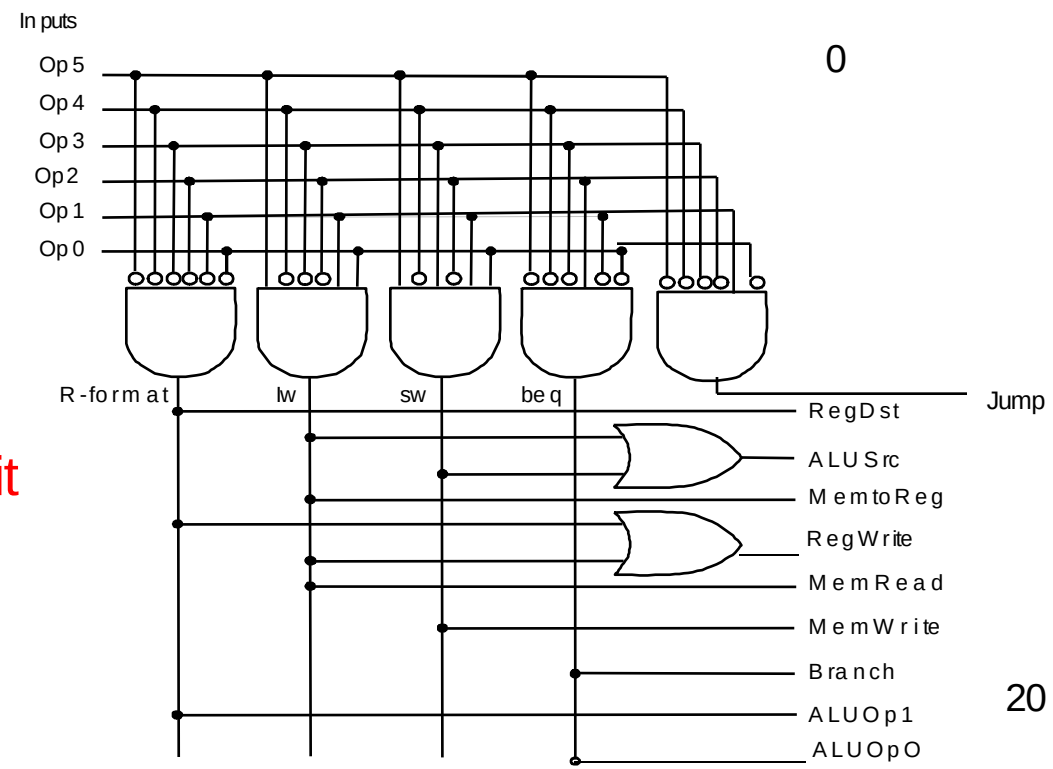
with correction in red

Design of Control Unit (J included)

Op-code bits 5 4 3 2 1 0	RegDst	ALUSrc	Memto - Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUp0	Jump
0 0 0 0 0 0	1	0	0	1	d	0	0	1	0	**0**
1 0 0 0 1 1	0	1	1	1	1	0	0	0	0	**0**
1 0 1 0 1 1	d	1	d	0	0	1	0	0	0	**0**
0 0 0 1 0 0	d	0	d	0	d	0	1	0	1	**0**
0 0 0 0 1 0	d	d	d	0	d	0	d	d	d	…**1**

J

$$Jump = \overline{Op_5}\,\overline{Op_4}\,Op_3\,\overline{Op_2}\,Op_1\,\overline{Op_0}$$



No changes in ALU Control unit

Cycle Time Calculation

- Let us assume that the **only** delays introduced are by the following tasks:
  - Memory access (read and write time = 3 nsec)
  - Register file access (read and write time = 1 nsec)
  - ALU to perform function (= 2 nsec)
- Under those assumption here are instruction execution times:

Instr			Reg		ALU		Data		Reg	
	fetch		read		oper		memory		write	Total
R-type	3	+	1	+	2	+			1	= 7 nsec
lw	3	+	1	+	2	+	3	+	1	= 10 nsec
sw	3	+	1	+	2	+	3			= 9 nsec
branch	3	+	1	+	2					= 6 nsec
jump	3									= 3 nsec

- Thus a clock cycle time has to be 10nsec, and clock rate = 1/10 nsec = 100MHz

Single Cycle Processor: Conclusion

- Single Cycle Problems:
  - what if we had a more complicated instruction like floating point?
  - a clock cycle would be much longer,
  - thus for shorter and more often used instructions, such as add & lw, wasteful of time.
- One Solution:
  - use a “smaller” cycle time, and
  - have different instructions take different numbers of cycles.
- And that is a “multi-cycle” processor.