CSE 2021
Computer Organization

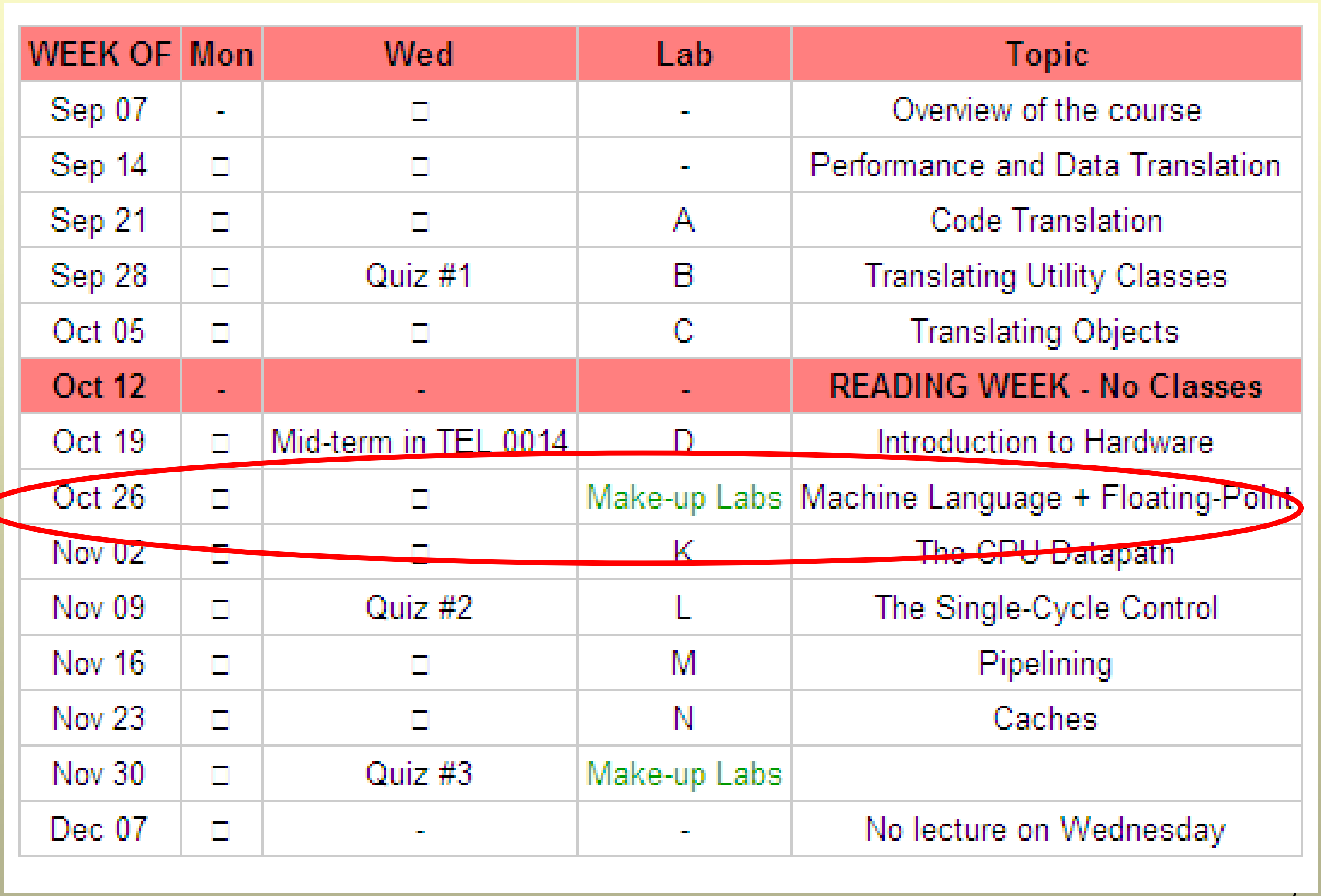
Hugh Chesser, CSEB 1012U

W8-M

Schedule, Reminders

Make-up Labs

- *Complete Labs A – D*
- *No additional "Lab X"*



WEEK OF	Mon	Wed	Lab	Topic
Sep 07	-	□	-	Overview of the course
Sep 14	□	□	-	Performance and Data Translation
Sep 21	□	□	A	Code Translation
Sep 28	□	Quiz #1	B	Translating Utility Classes
Oct 05	□	□	C	Translating Objects
Oct 12	-	-	-	**READING WEEK - No Classes**
Oct 19	□	Mid-term in TEL 0014	D	Introduction to Hardware
Oct 26	□	□	Make-up Labs	Machine Language + Floating-Point
Nov 02	□	□	K	The CPU Datapath
Nov 09	□	Quiz #2	L	The Single-Cycle Control
Nov 16	□	□	M	Pipelining
Nov 23	□	□	N	Caches
Nov 30	□	Quiz #3	Make-up Labs	
Dec 07	□	-	-	No lecture on Wednesday

Agenda

Topics:

1. Register files, Decoder, Data Memory, Instruction Memory – Building Blocks
2. Complete hardware implementation of goal instructions

Patterson: Appendix C, Section 4.1, 4.2, 4.3

Overview (1)

Goal: Implement a subset of core instructions from the MIPS instruction set, given below

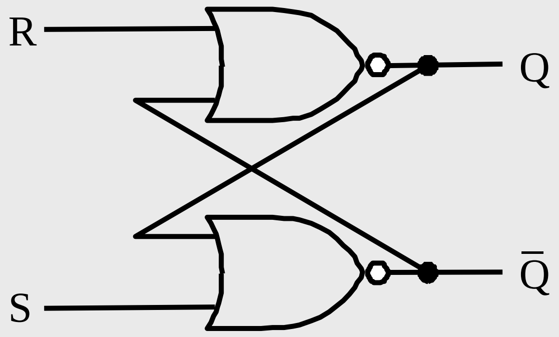
Category	Instruction	Example	Meaning	Comments
Arithmetic and Logical	add	`add $s1,$s2,$s3`	`$s1 ← $s2+$s3`	
	subtract	`sub $s1,$s2,$s3`	`$s1 ← $s2-$s3`	
	and	`add $s1,$s2,$s3`	`$s1 ← $s2&$s3`	& => and
	or	`or $s1,$s2,$s3`	`$s1 ← $s2\|$s3`	| => or
	slt	`slt $s1,$s2,$s3`	`If $s1 < $s3, $s1←1` `else $s1←0`	
Data Transfer	load word	`lw $s1,100($s2)`	`$s1 ← Mem[$s2+100]`	
	store word	`sw $s1,100($s2)`	`Mem[$s2+100] ← $s1`	
Branch	branch on equal	`beq $s1,$s2,L`	`if($s1==$s2) go to L`	
	unconditional jump	`j 2500`	`go to 10000`	

Basics: RS Latch (3)

Simplest memory elements are Flip-flops and Latches

— In clocked latches, state changes whenever input changes and the clock is asserted.

— In flip-flops, state changes only at the trailing edge of the clock



Logic Diagram

Inputs		Outputs		Comments
S	R	Q	$\bar{Q}$	
		0	**1**	**Initial Condition**
1	**0**	**1**	**0**	
0	**0**	**1**	**0**	**After S = 1, R = 0**
0	**1**	**0**	**1**	
0	**0**	**0**	**1**	**After S = 0, R = 1**
1	**1**	**0**	**0**	**Undefined**

Function Table

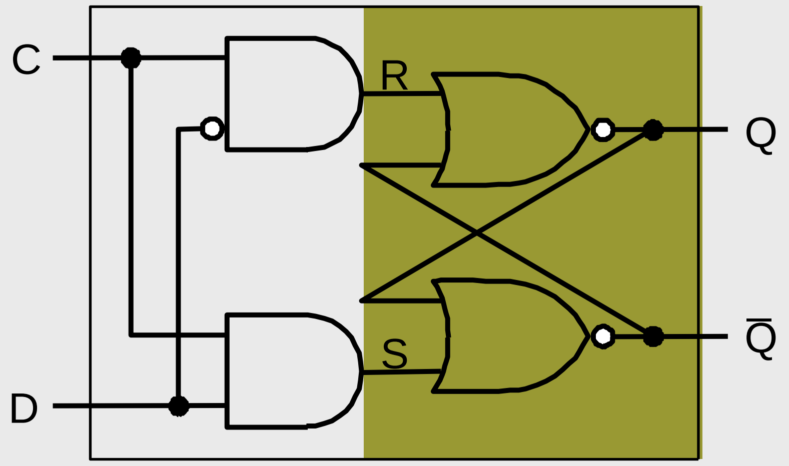
RS Unclocked Latch

— For a RS-latch: output Q = 1 when S = 1, R = 0 (set condition)

output Q = 0 when S = 0, R = 1 (reset condition)

Basics: Clocked D Latch (4)

1. For a D-latch: output Q = 1 when D = 1 (set condition)
   output Q = 0 when D = 0 (reset condition)

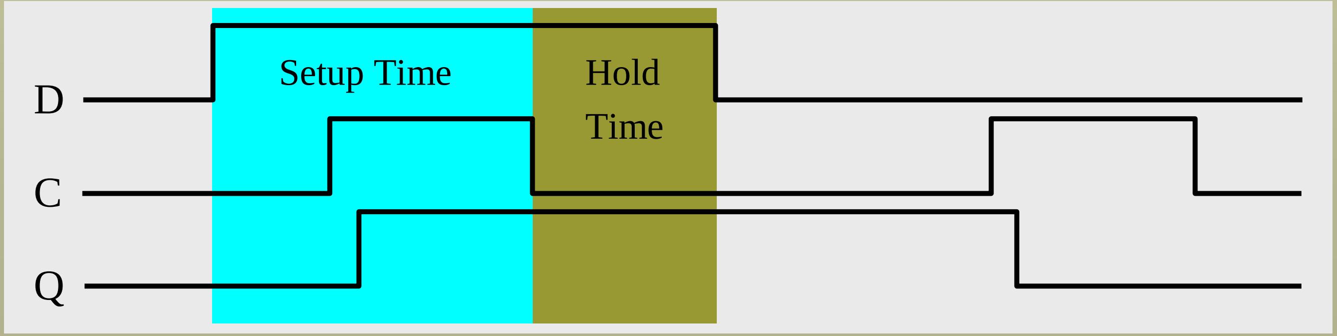


Logic Diagram

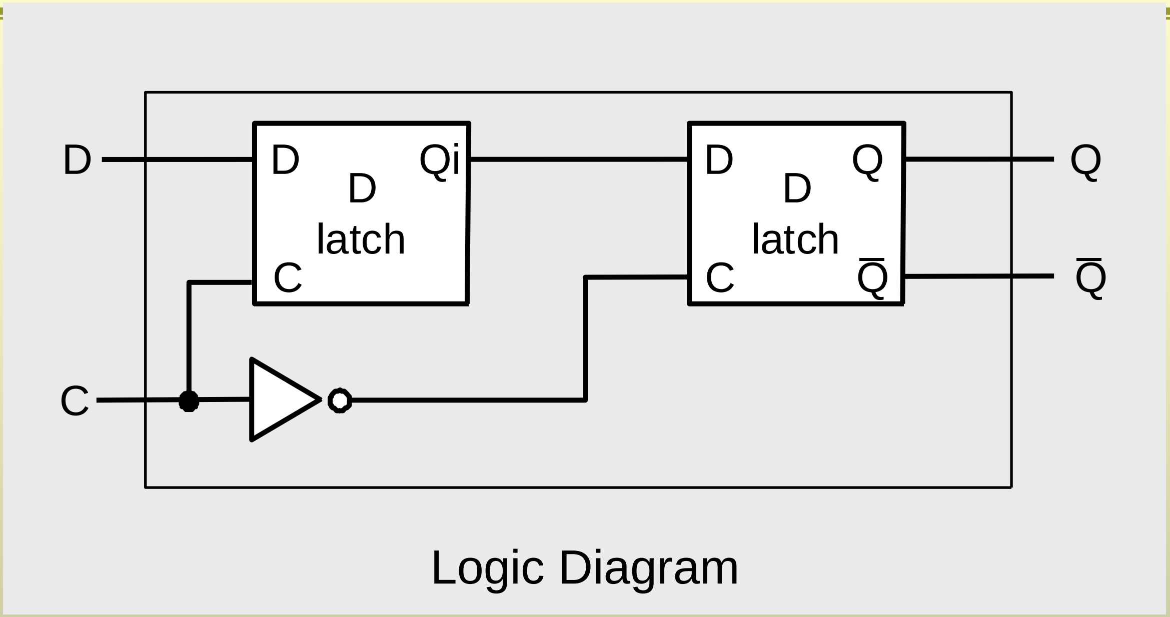
Inputs		Outputs		Comments
C	D	Q	$\bar{Q}$	
0	**X**	**Unchanged**		
1	**0**	**0**	**1**	**Reset**
1	**1**	**1**	**0**	**Set**

Function Table

2. D Latch requires clock to be asserted for output to change

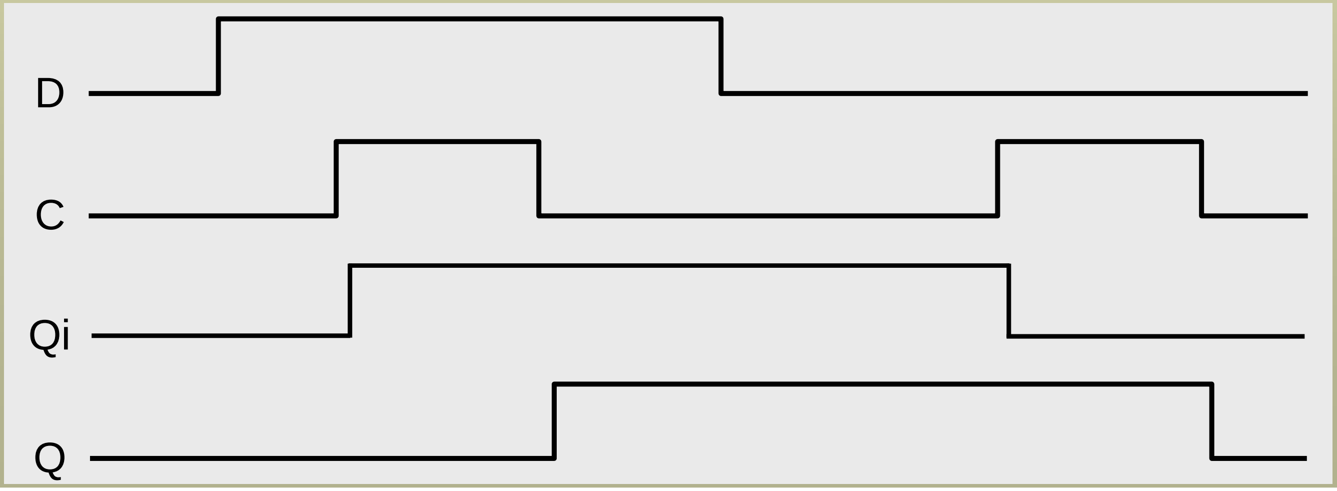


Basics: Falling Edge Triggered D flip-flop (5)



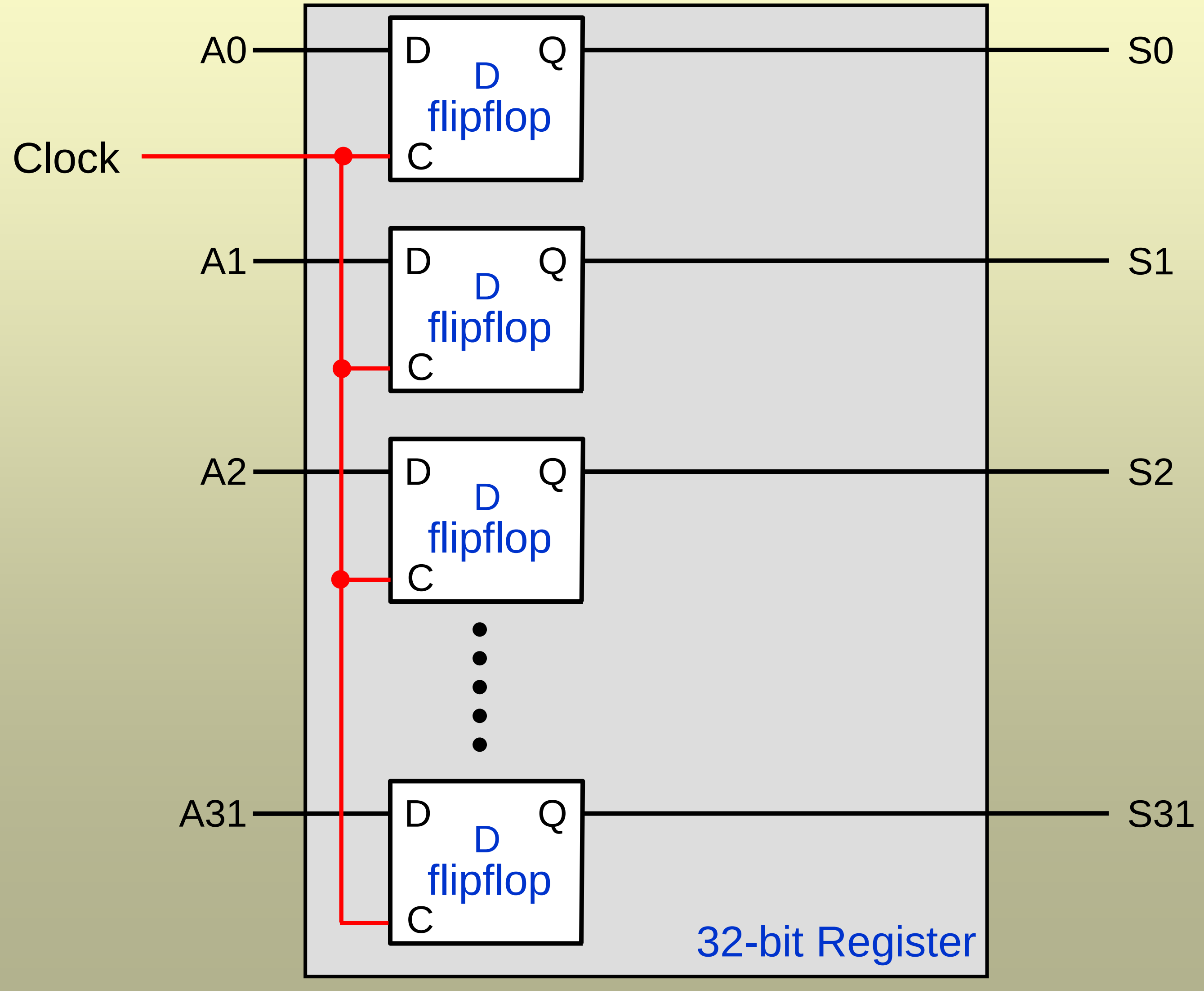
Logic Diagram

Output Q follows D but changes only at the falling edge



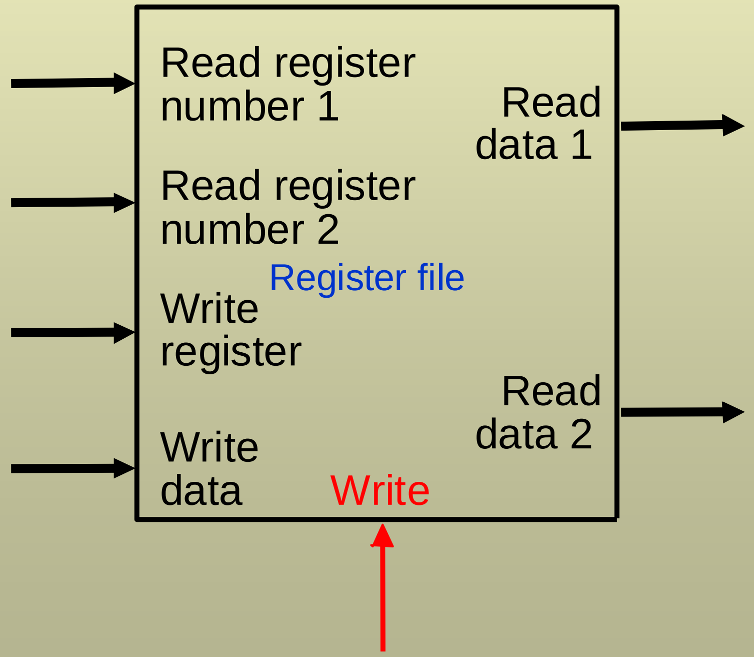
Basics: 32-bit Registers (6)

Falling edge triggered D flip-flops can be combined to form a register



Basics: Register Files (6)

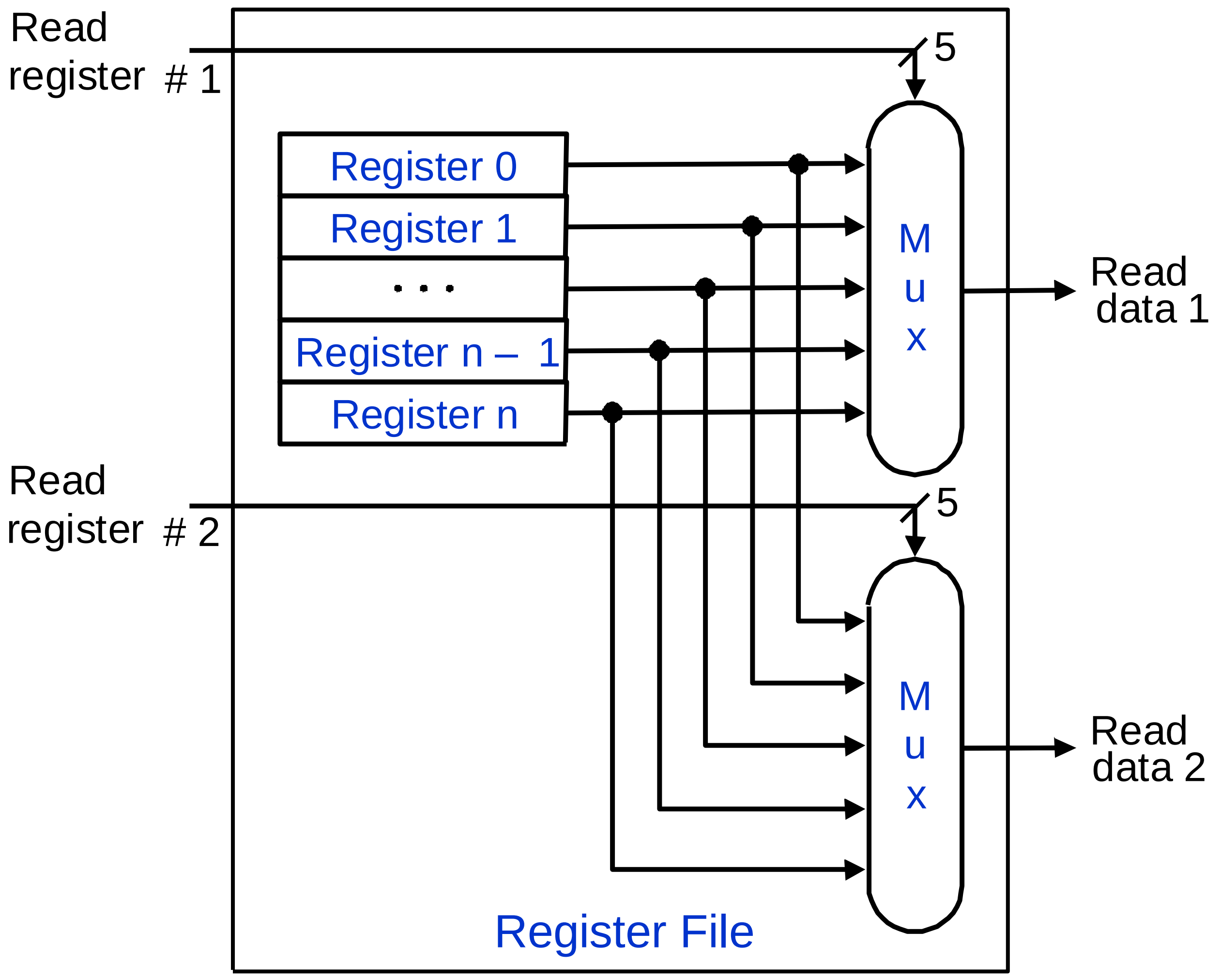
1. Register files consist of a set of registers that can be read or written individually
2. In MIPS, register file contains 32 registers
3. Two registers can be read simultaneously
4. One register can be written at one time



Basics: Read Operation in Register Files (7)

Read Operation:

- Register number of the register to be read is provided as input
- Content of the read register is the output of the register file
- Multiplexers are used in the read operation



Basics: Write Operation in Register Files (8)

Write Operation:

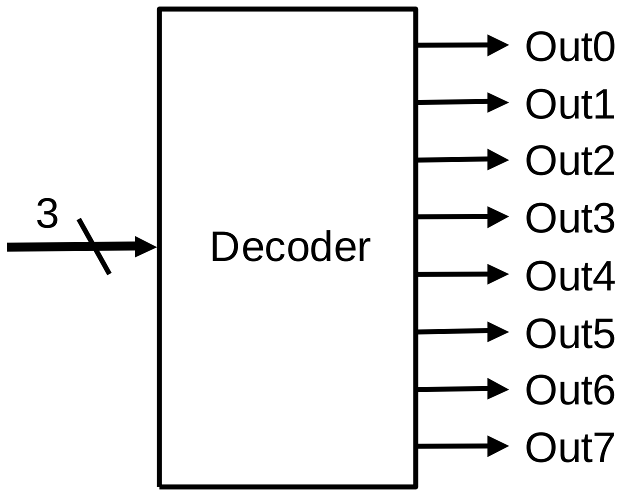
— Register number of the register to be written is one input

— Data to be written is the second input

— Clock that controls the write operation is the third input

— Decoders are used in the write operation

What is a Decoder?



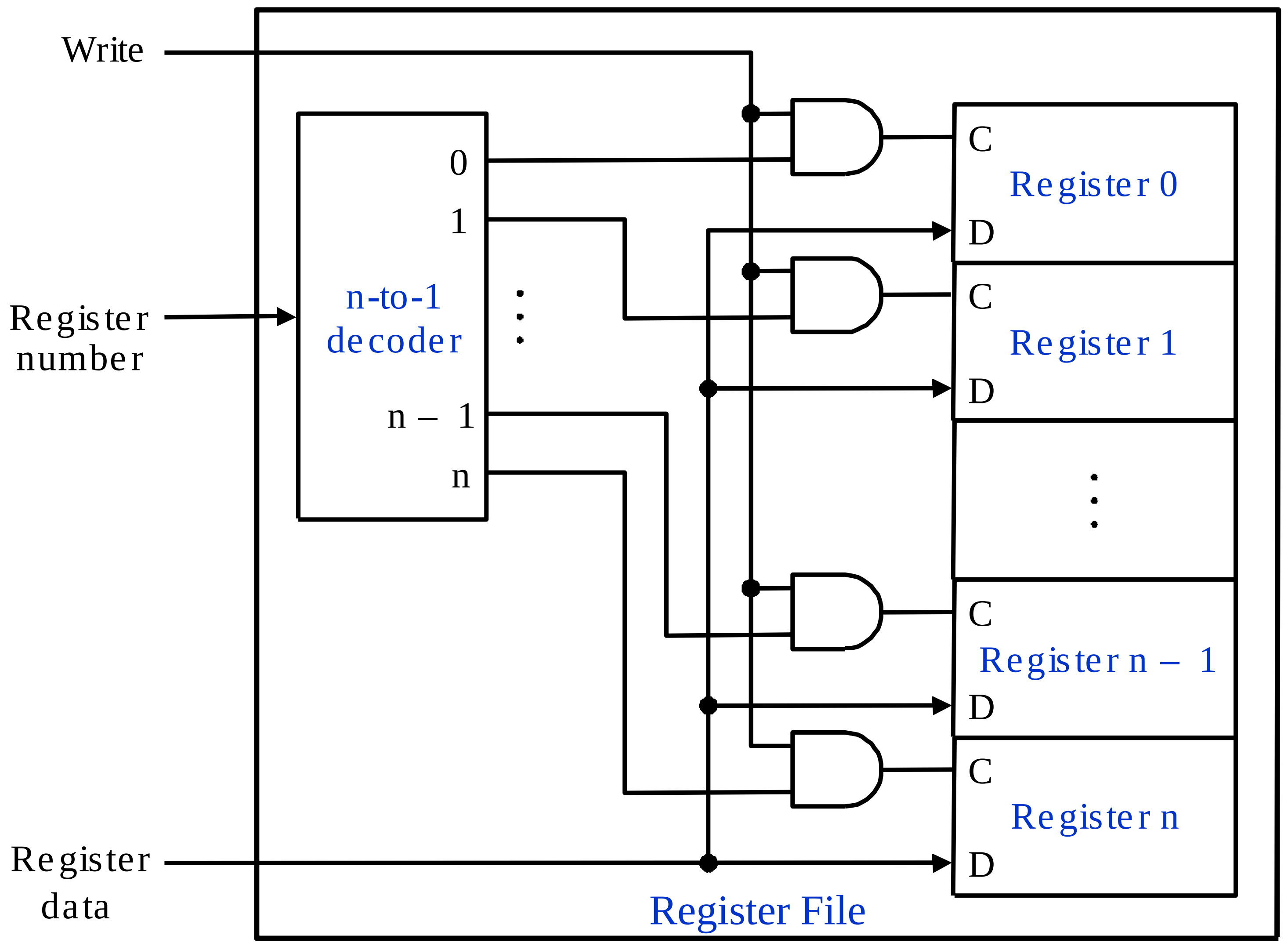
3-bit Decoder

Inputs			Outputs							
In2	In1	In0	Out7	Out6	Out5	Out4	Out3	Out2	Out1	Out0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

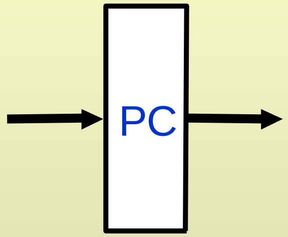
Basics: Write Operation in Register Files (9)

Write Operation:

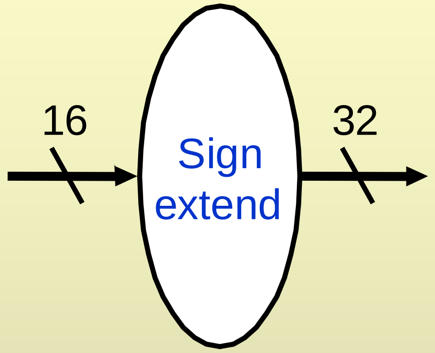
- Register number of the register to be written is one input
- Data to be written is the second input
- Clock that controls the write operation is the third input
- Decoders are used in the write operation



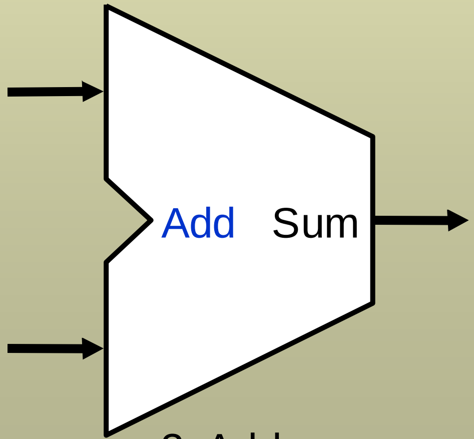
Basic Building Blocks (1)



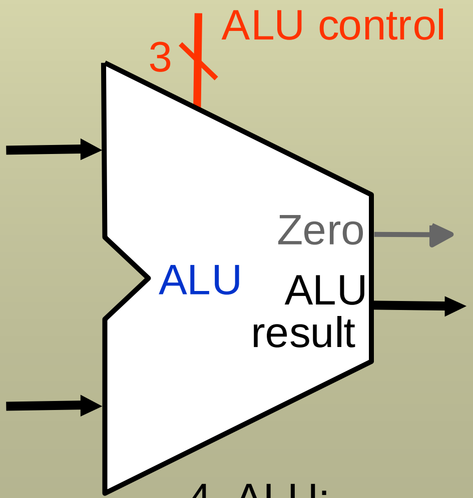
1. Program counter:
contains address of next instruction



2. Sign-extension unit:
extends a 16-bit integer to a 32-bit integer

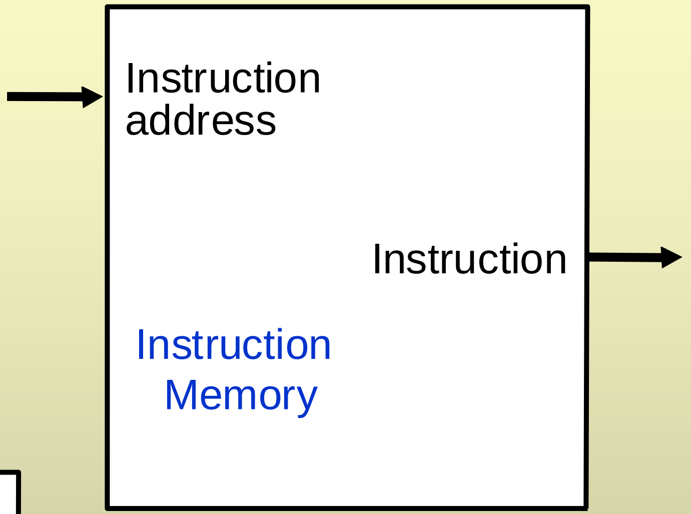


3. Adder:
adds two 32-bit integers

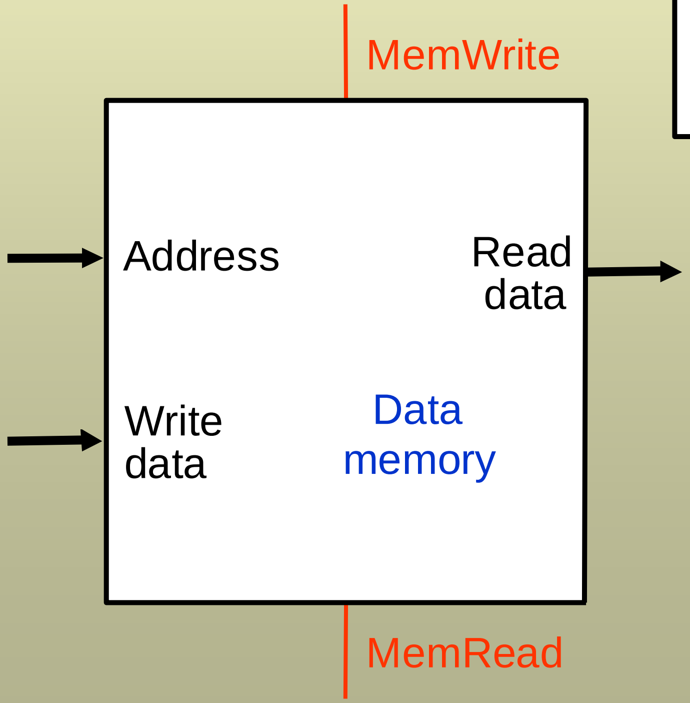


4. ALU:
add/subtract/and/or/compare two 32-bit integers

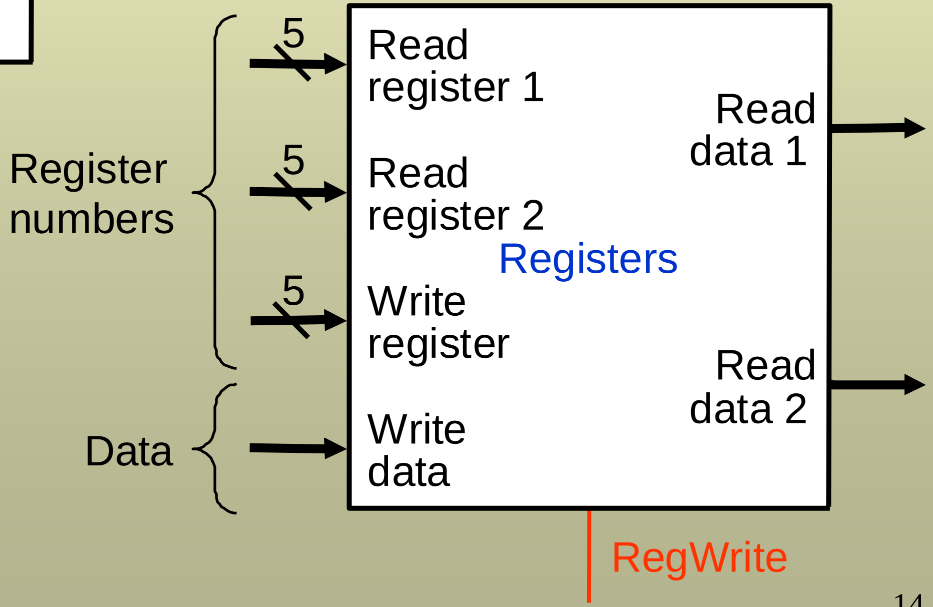
Basic Building Blocks (2)



5. Instruction memory



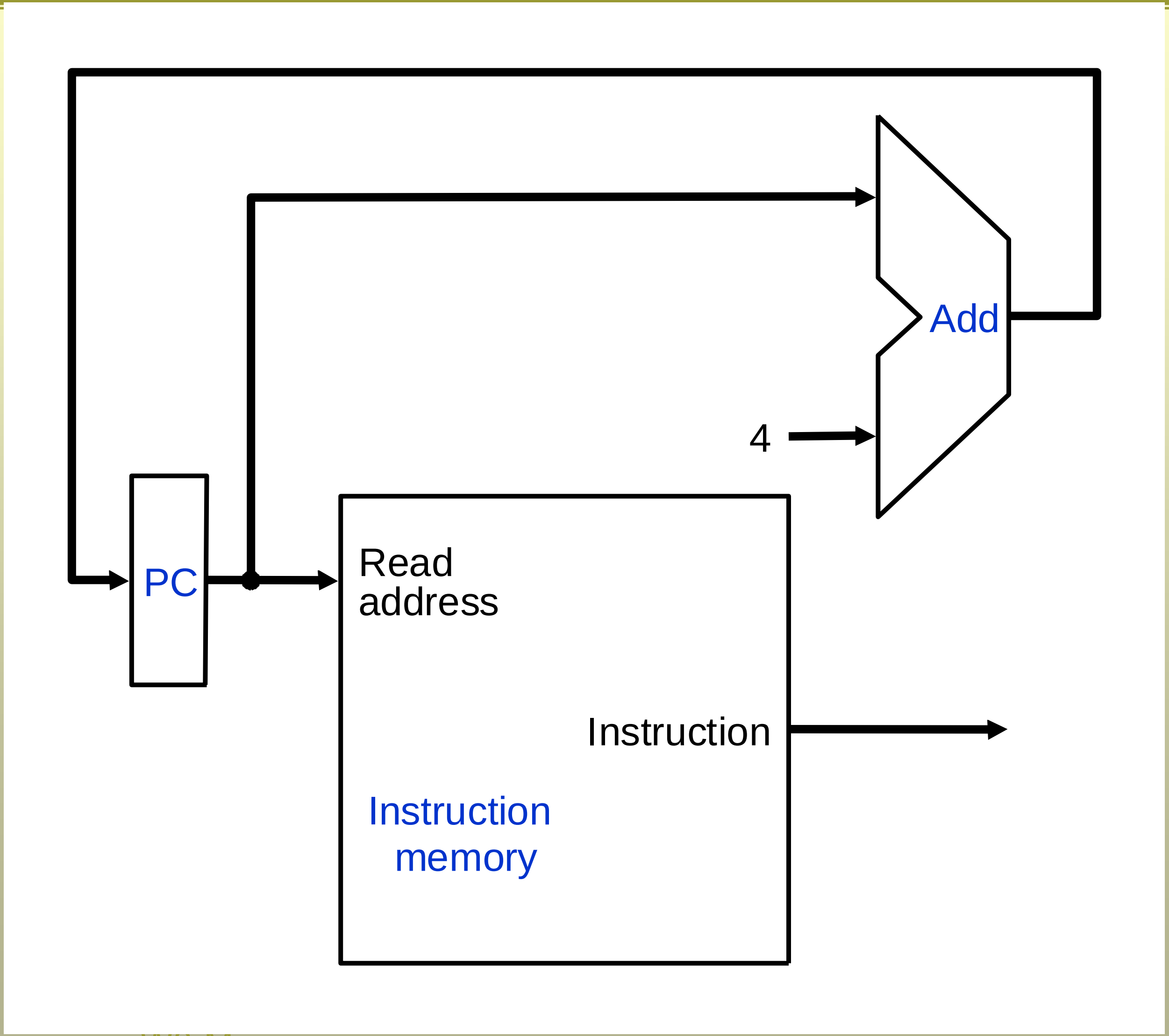
6. Data memory unit



7. Register Files

Datapath: Fetch Instruction

1. Provide address from PC to Instruction Memory
2. Increment PC by 1 word (4 bytes)
3. Fetch the instruction

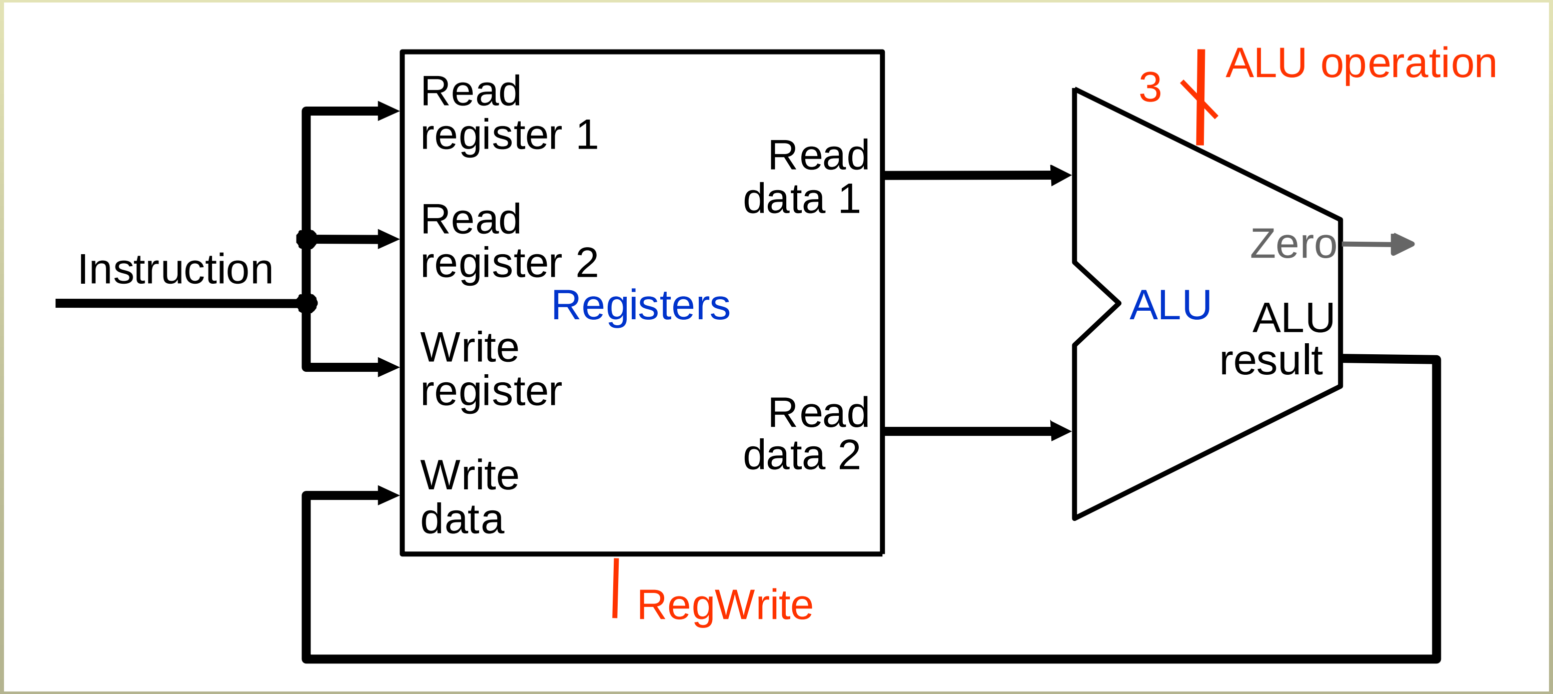


Datapath: R-type Instructions

R-type instructions include arithmetic and logical instructions (add, sub, or, and, slt)

Example: `add $s1,$s2,$s3`

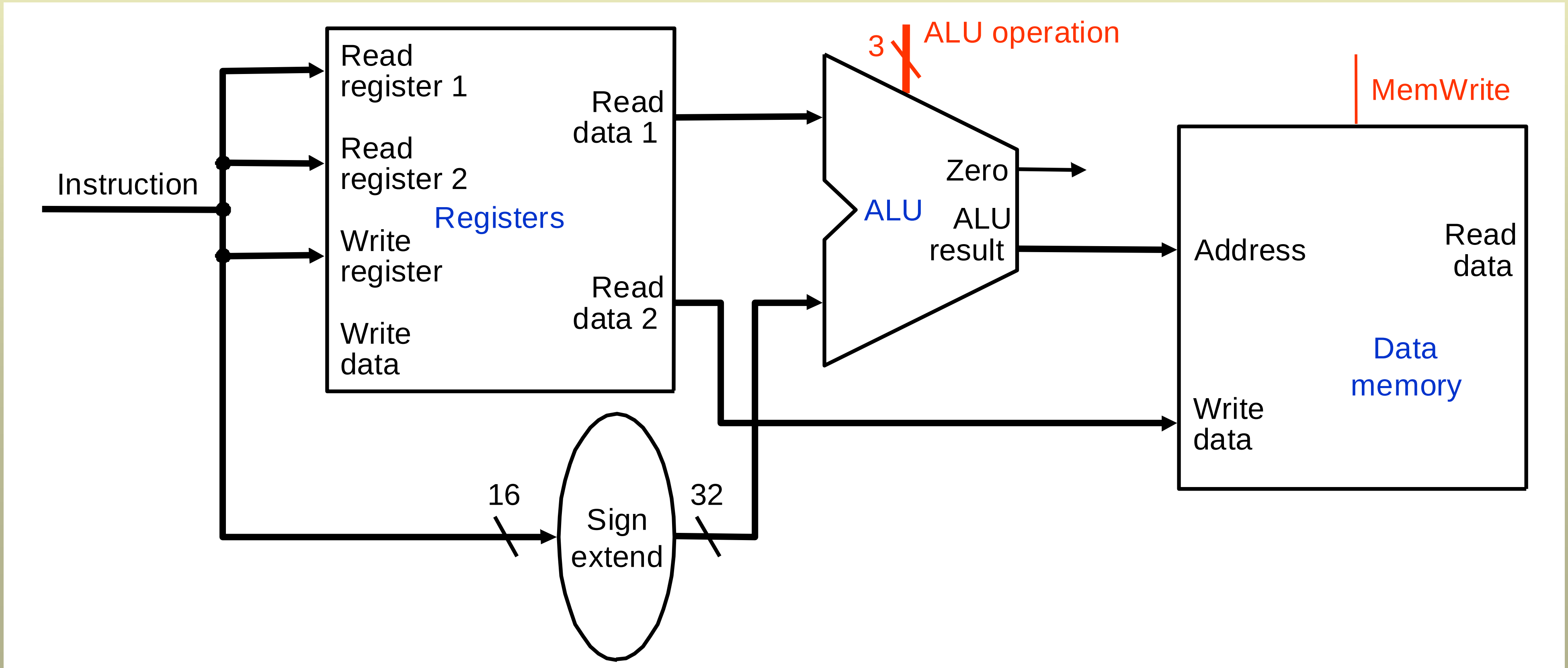
1. Read two registers `($s2,$s3)` specified in the instruction
2. ALU performs the required operation `(add)` on the two operands
3. Output of ALU is written to the specified register `($s1)`



Datapath: Data transfer Instruction (1)

Store instruction `sw $s1,offset($s2)`

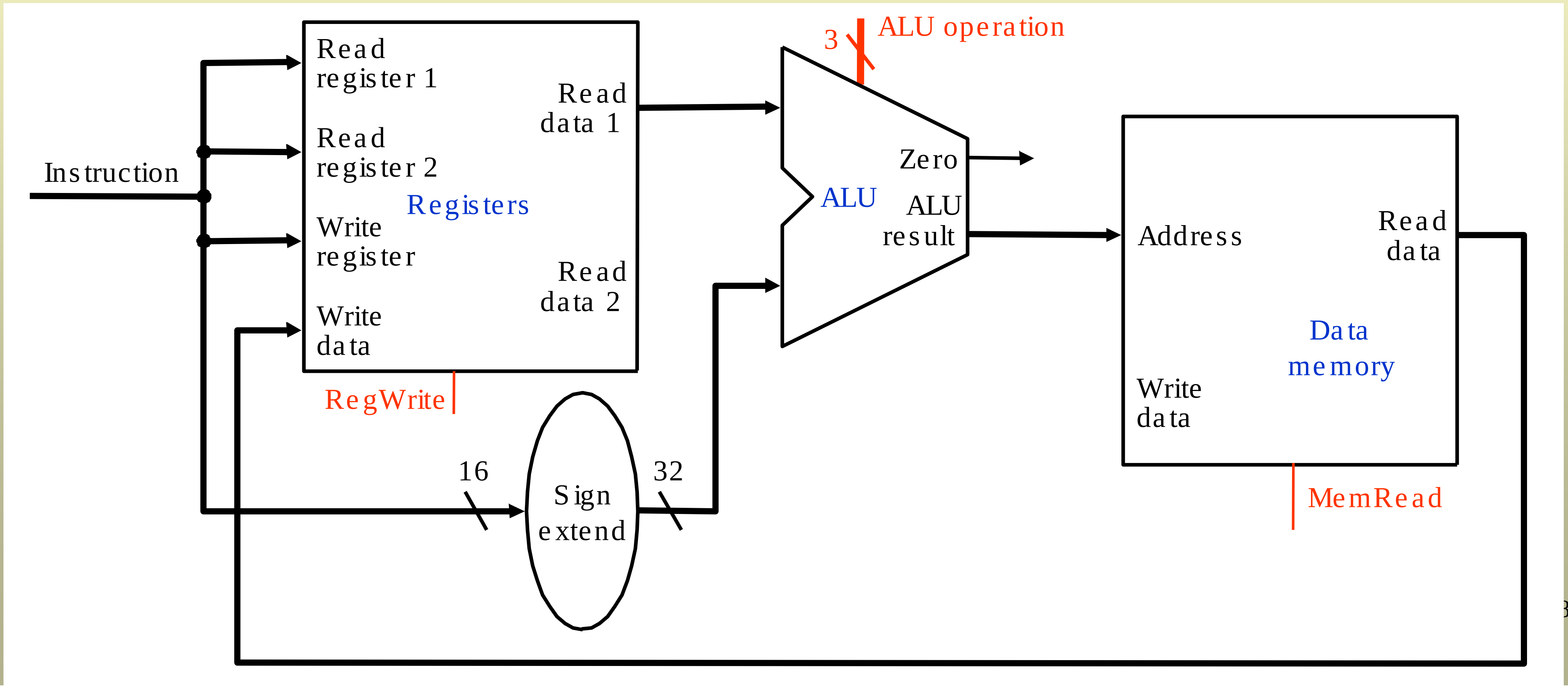
1. Read two registers ($s1,$s2) specified in the instruction.
2. Offset is extended to 32 bits.
3. ALU adds offset with specified register ($s2) to obtain data memory address.
4. Address along with data of the register ($s1) to be stored passed to data memory.



Datapath: Data transfer Instruction (2)

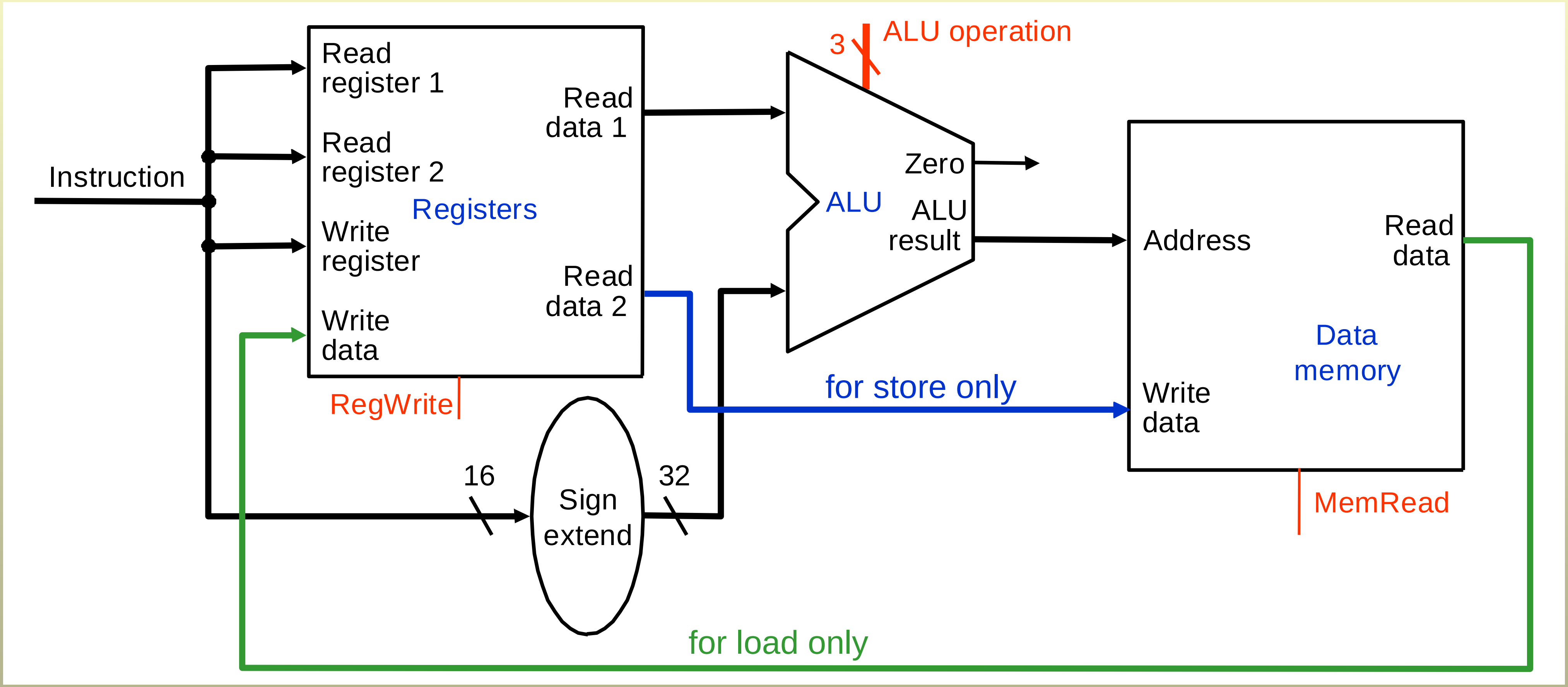
Load instruction `lw $s1,offset($s2)`

1. Read register ($s2) specified in the instruction. 2. Offset is extended to 32 bits.
3. ALU adds offset with specified register ($s2) to obtain data memory address.
4. Data memory transfers data from provided address to Register file where it is stored in the specified register ($s1).



Datapath: Data transfer Instruction (3)

Load and store instruction combined



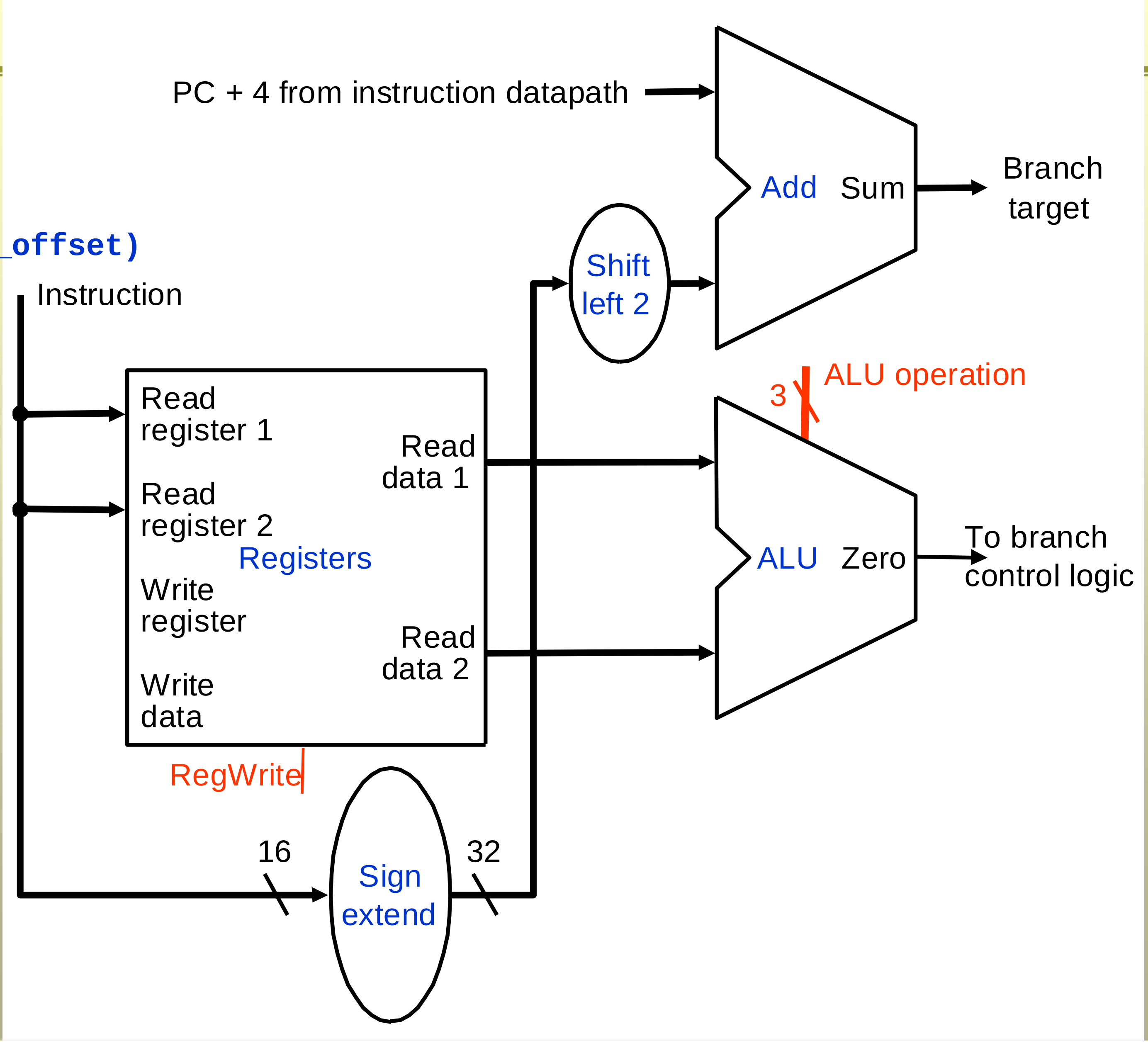
Datapath: Branch Instructions

Example: `beq $s1,$s2,Loop`

Compiler translation:

```
beq $s1,$s2,w_offset
#if $s1==$s2, goto (PC+4+4*w_offset)
```

1. Read two registers ($s2,$s3) specified in the instruction
2. ALU compares content of specified registers ($s1,$s2)
3. Adder computes the branch address
4. If equal (zero = 1), branch address is copied to PC


PC + 4 from instruction datapath
Add
Sum
Branch target
Shift left 2
Instruction
ALU operation
3
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
ALU
Zero
To branch control logic
RegWrite
16
Sign extend
32