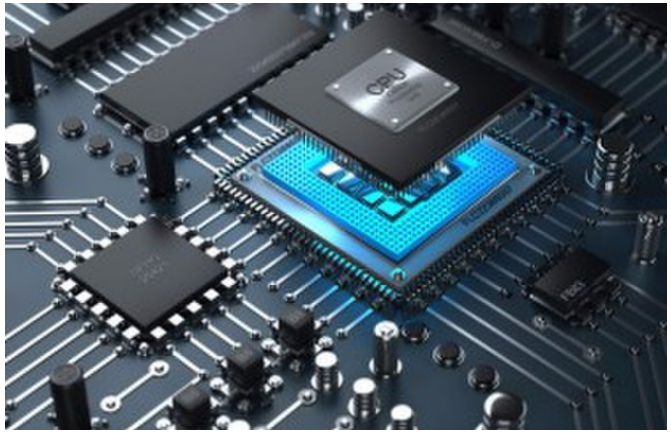


CPU Datapaths 2: Single Cycle

CS 154: Computer Architecture

Lecture #12

Winter 2020

Ziad Matni, Ph.D.

Dept. of Computer Science, UCSB

Administrative

- Talk next week – must attend
  - Details to follow

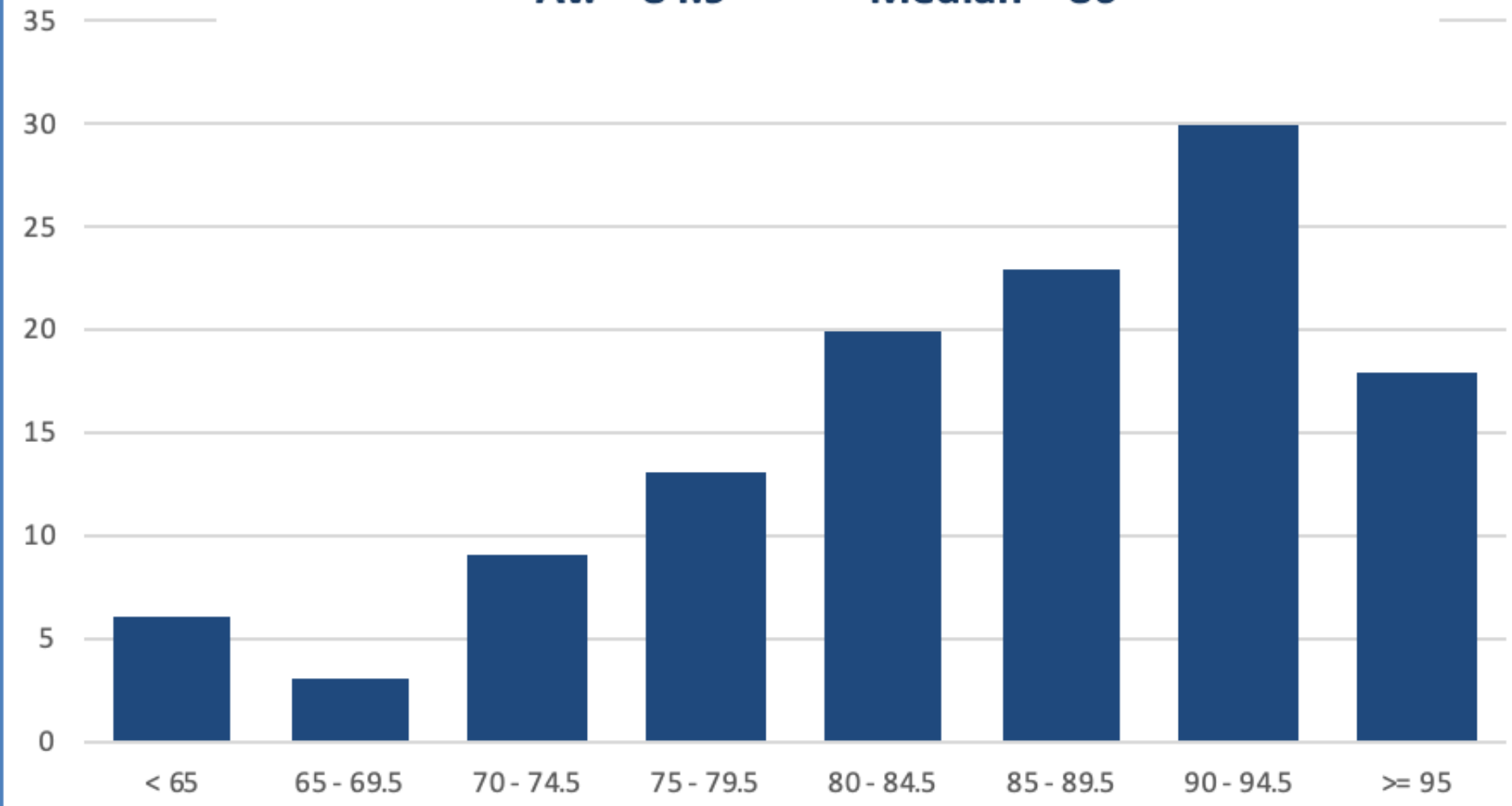
Reviewing Your Midterm Exams

- You can review your midterm with a TA during office hours
  - *Last name*: **A thru L** **George T.** **Tu 10:30 am – 12:30 pm**
  - *Last name*: **M thru Z** **Sid S.** **Mo 3:00 pm – 5:00 pm**
  - If you can't go to these o/hs, you can see me instead, but let me know ***many days ahead of time*** first so I can get your exam from the TA…

- When reviewing your exams:
  - Do **not** take pictures, do not copy the questions
  - TA cannot change your grade
    - If you have a legitimate case for grade change, the prof. will decide
    - Legitimate = When we graded, we added the total points wrong
    - Not legitimate = Why did you take off *N* points on this question????

CS154, W20, Midterm Exam Grade Distribution

Av. = 84.9 **Median = 86**



Lecture Outline

- A Simplified Datapath for all Instructions
  - Single Cycle

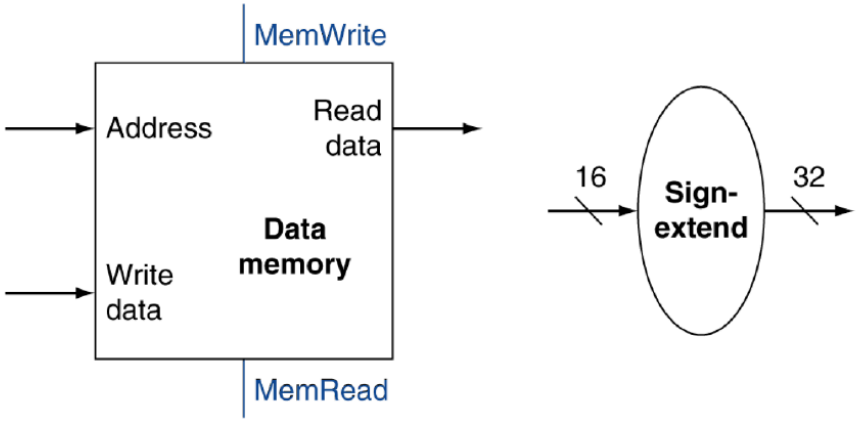
- ALU Design and Control
  - FYI: **Read the appendix section B.5 (pp. B-26 thru B-38)** for review / reference

Load/Store Instructions

includes lw, sw
e.g.: **`Lw $t0, 4($sp)`**

- Read register operands
- Calculate address using 16-bit offset (immediate)
  - First take the offset and sign-extend it to 32-bits
  - Then use ALU
- Load: Read memory and update register
- Store: Write register value to memory

We'll also need…



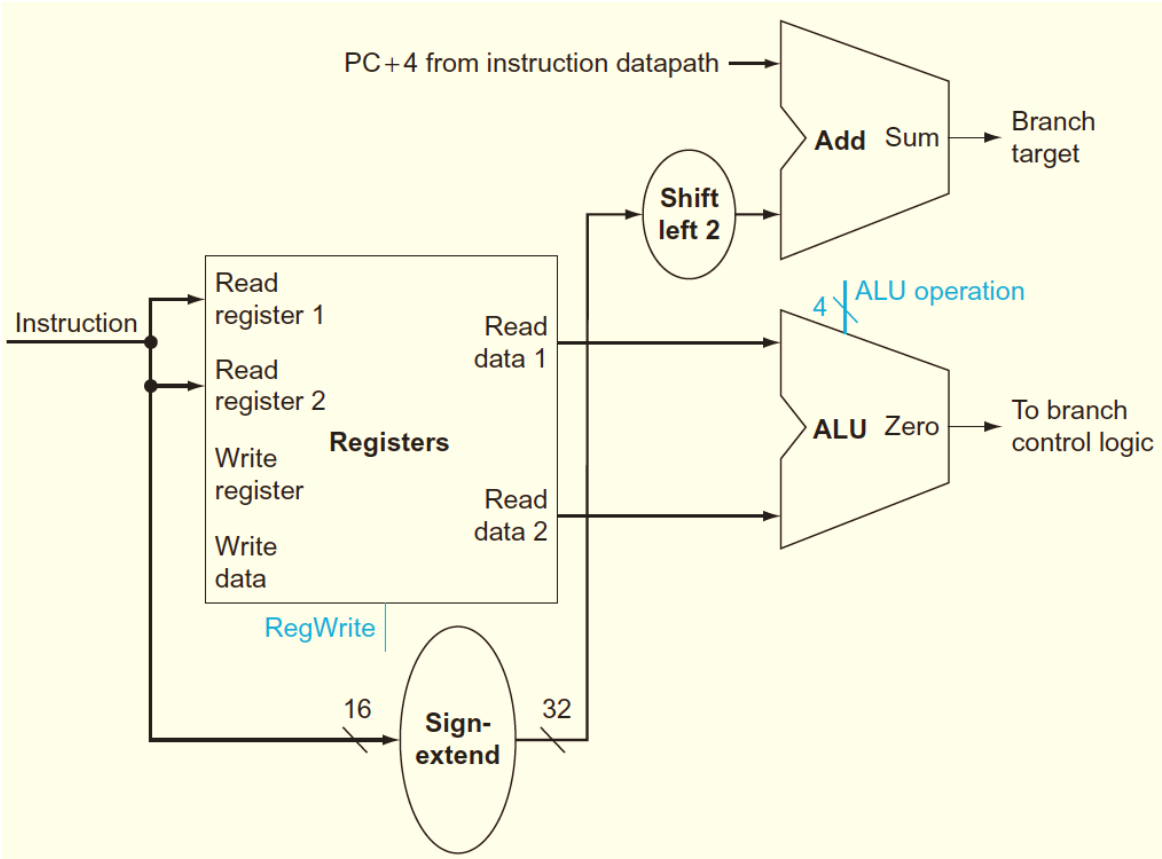
a. Data memory unit

b. Sign extension unit

Branch Instructions

includes beq, bne
e.g.: **`beq $t1, $t2, Label`**

- Read register operands
- Compare operands
  - Use ALU, subtract
    and check Zero output
- Calculate target address
  - Sign-extend displacement
  - Shift left 2 places
  - Add to PC + 4
    (already calculated
    by instruction fetch)



Putting the Elements Together

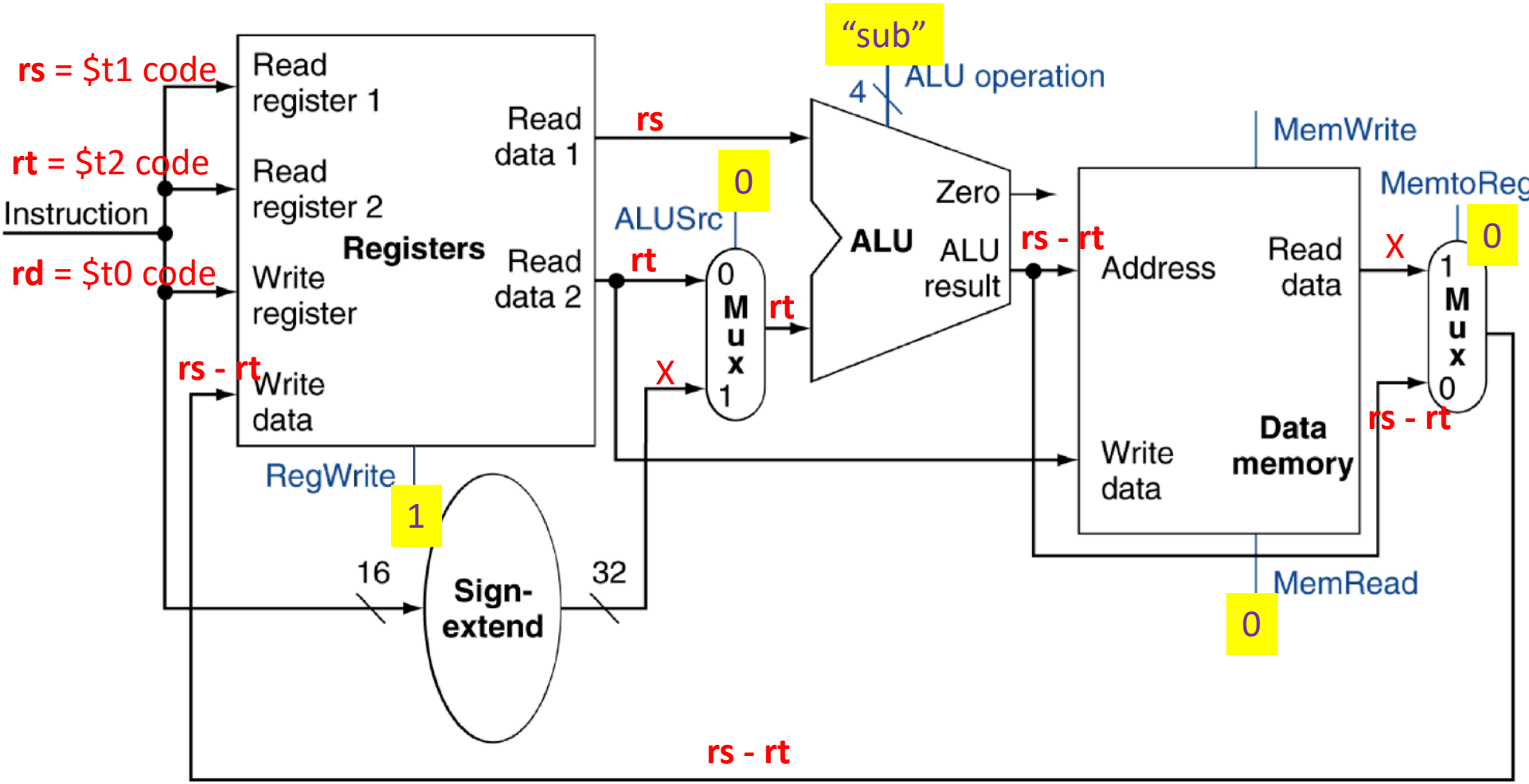
- These ”simple” data paths perform
  one instruction in **one** clock cycle
  - Each datapath element can only do one function at a time
  - Hence, we need separate instruction and data memories
  - In the next lesson(s), we will see how we can perform parallel-like processing, i.e. pipelining
- Use multiplexers where alternate data sources are used for different instructions

R-Type / Load/Store Datapath

EXAMPLE:

sub $t0, $t1, $t2

R[rd] = R[rs] – R[rt]

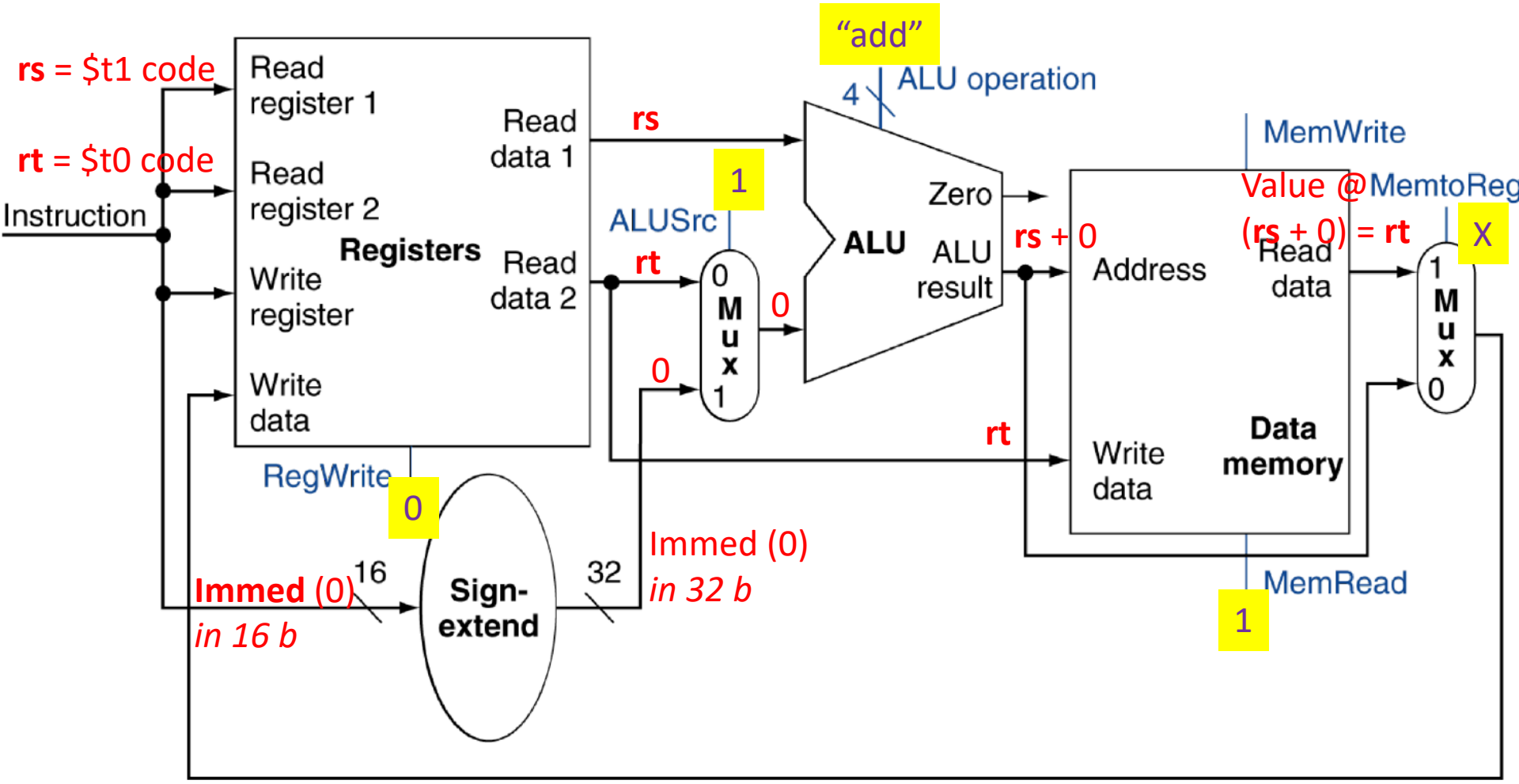


R-Type / Load/Store Datapath

EXAMPLE:

sw $t0, 0($t1)

R[rs]+SignExtImm = R[rt]

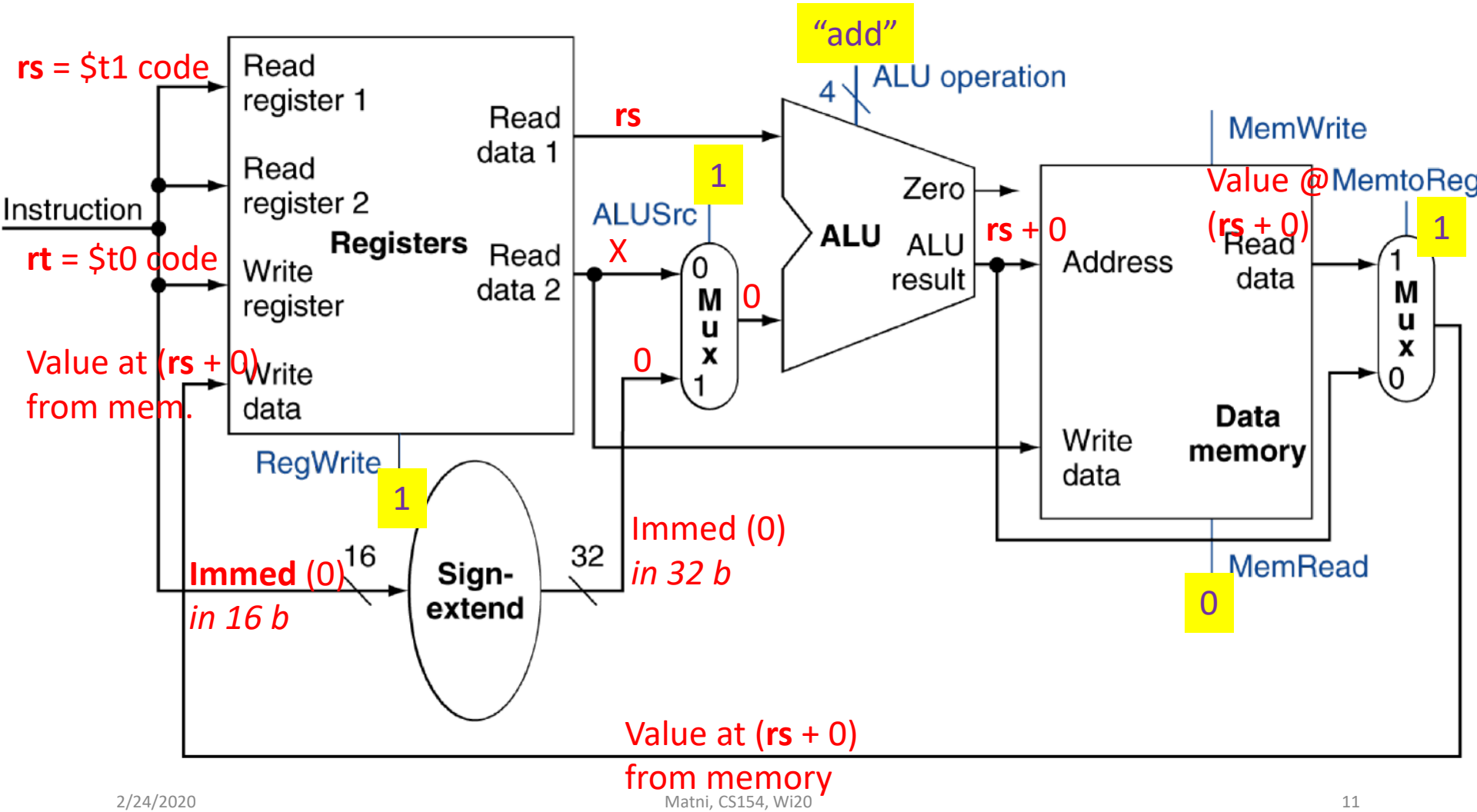


R-Type / Load/Store Datapath

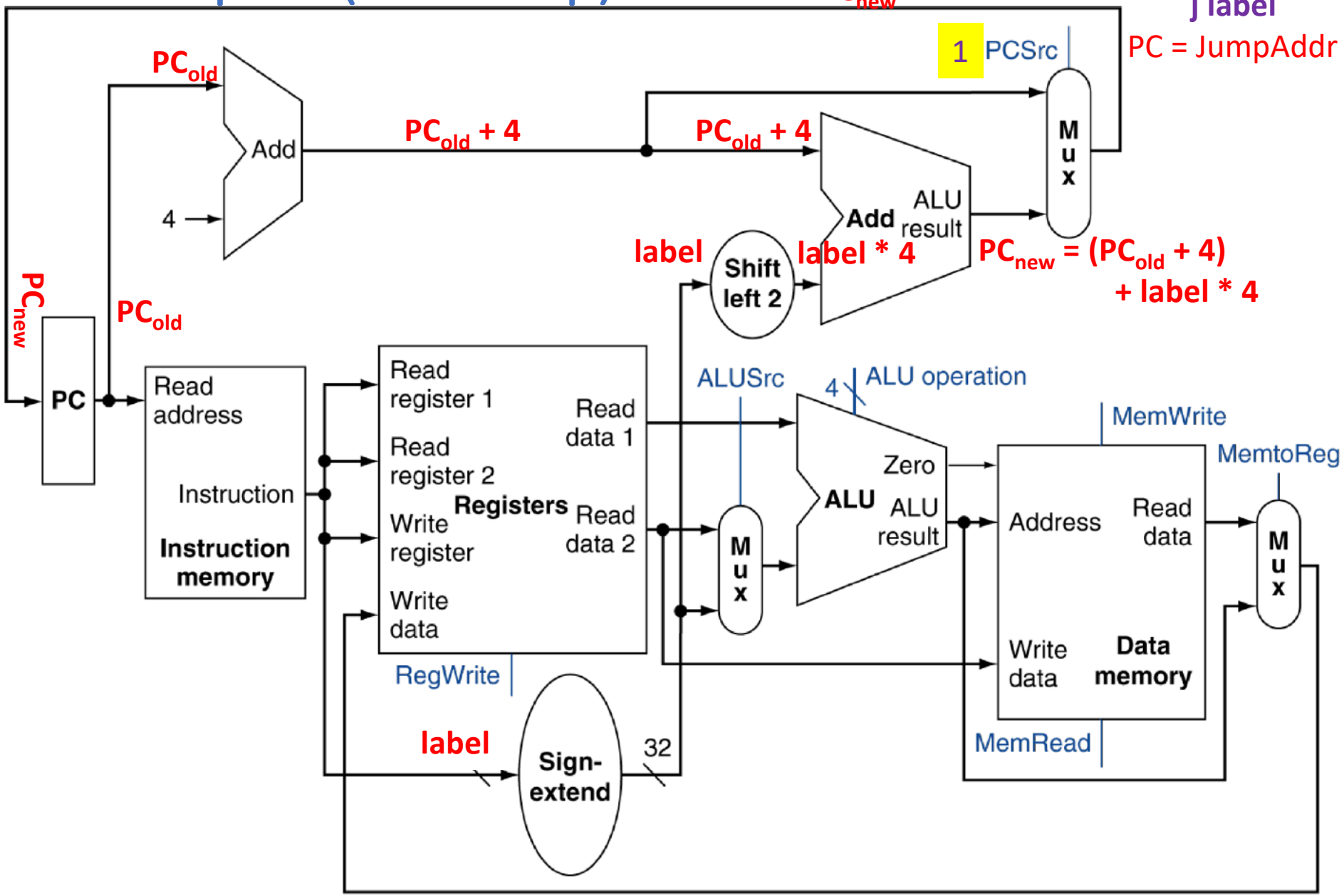
EXAMPLE:

lw $t0, 0($t1)

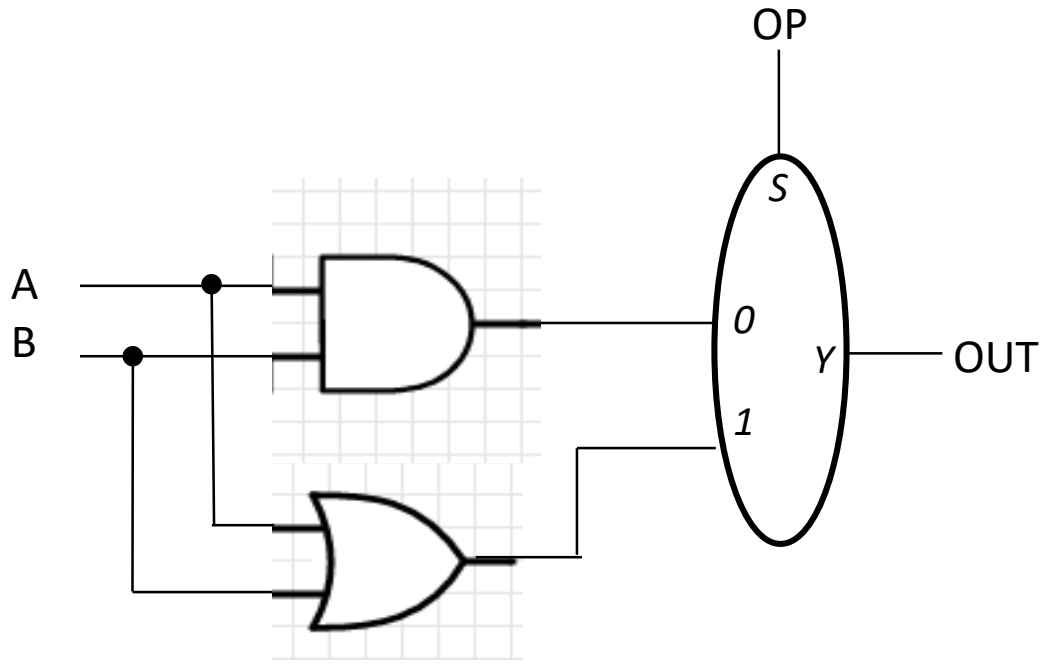
R[rt] = R[rs]+SignExtImm



Full Datapath (add Jump)



What Does This Logic Block Do?



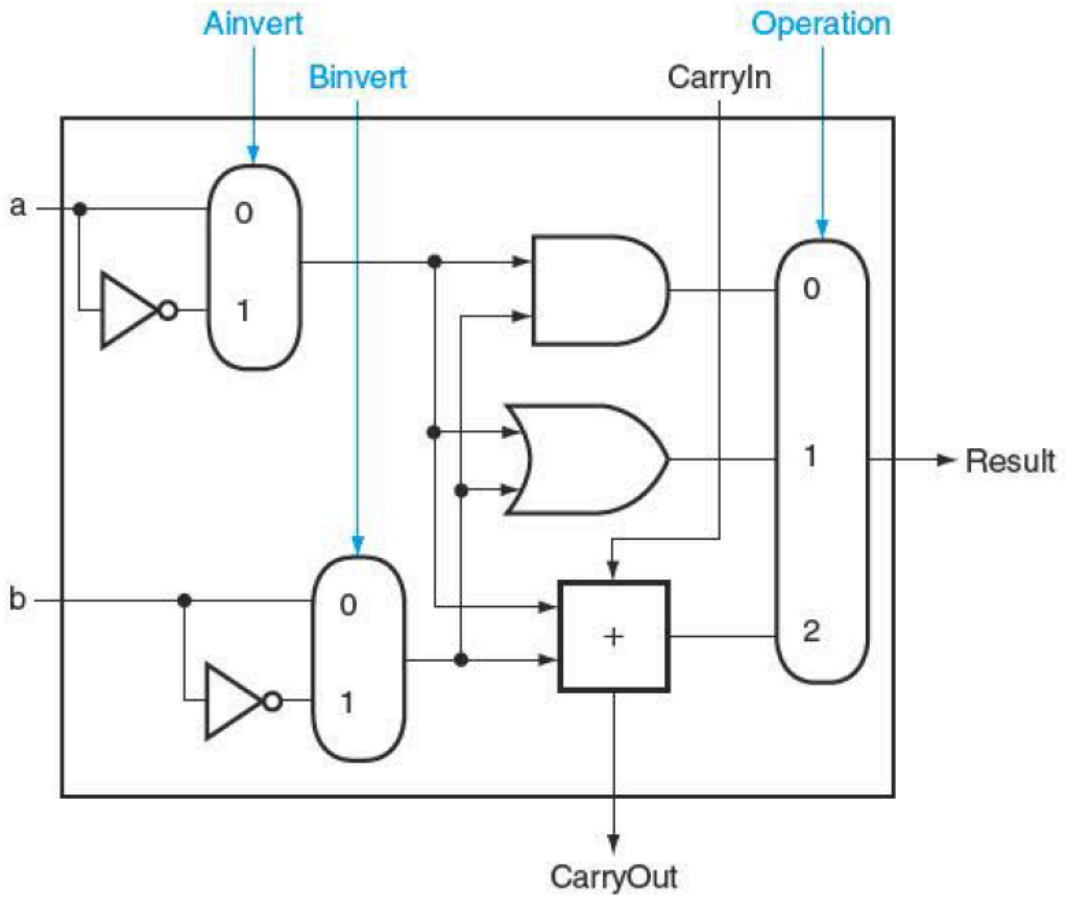
What Does This Logic Block Do?

Class Ex.

Ainv	Binv	OP[1:0]	Result
0	0	00	AND
1	1	00	NOR
0	*1*	*00*	*A.!B^*
1	*0*	*00*	*!A.B^*
0	0	01	OR
1	1	01	NAND
0	0	10	add
0	1	10	subtract*

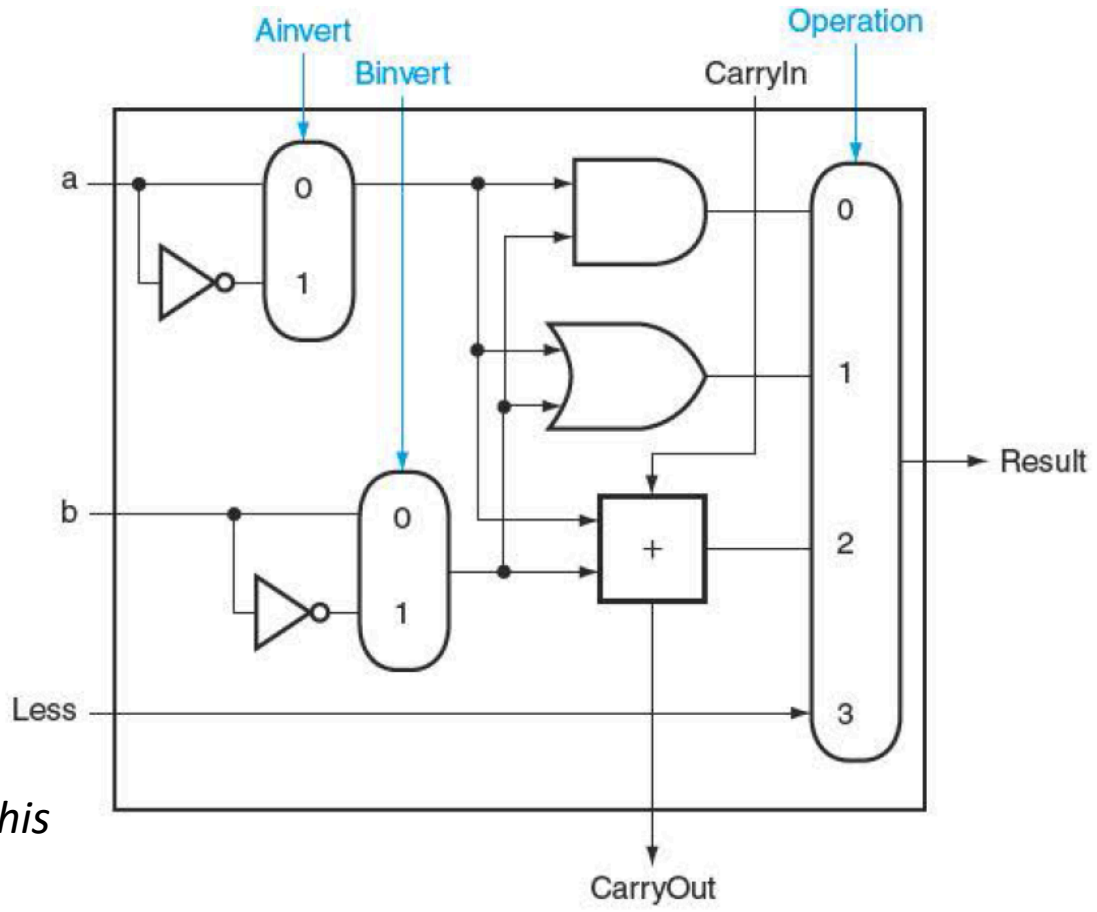
^ *Unpopular combo – not used much*

* *To do this, Cin has to be 1*



What Does This Logic Block Do?

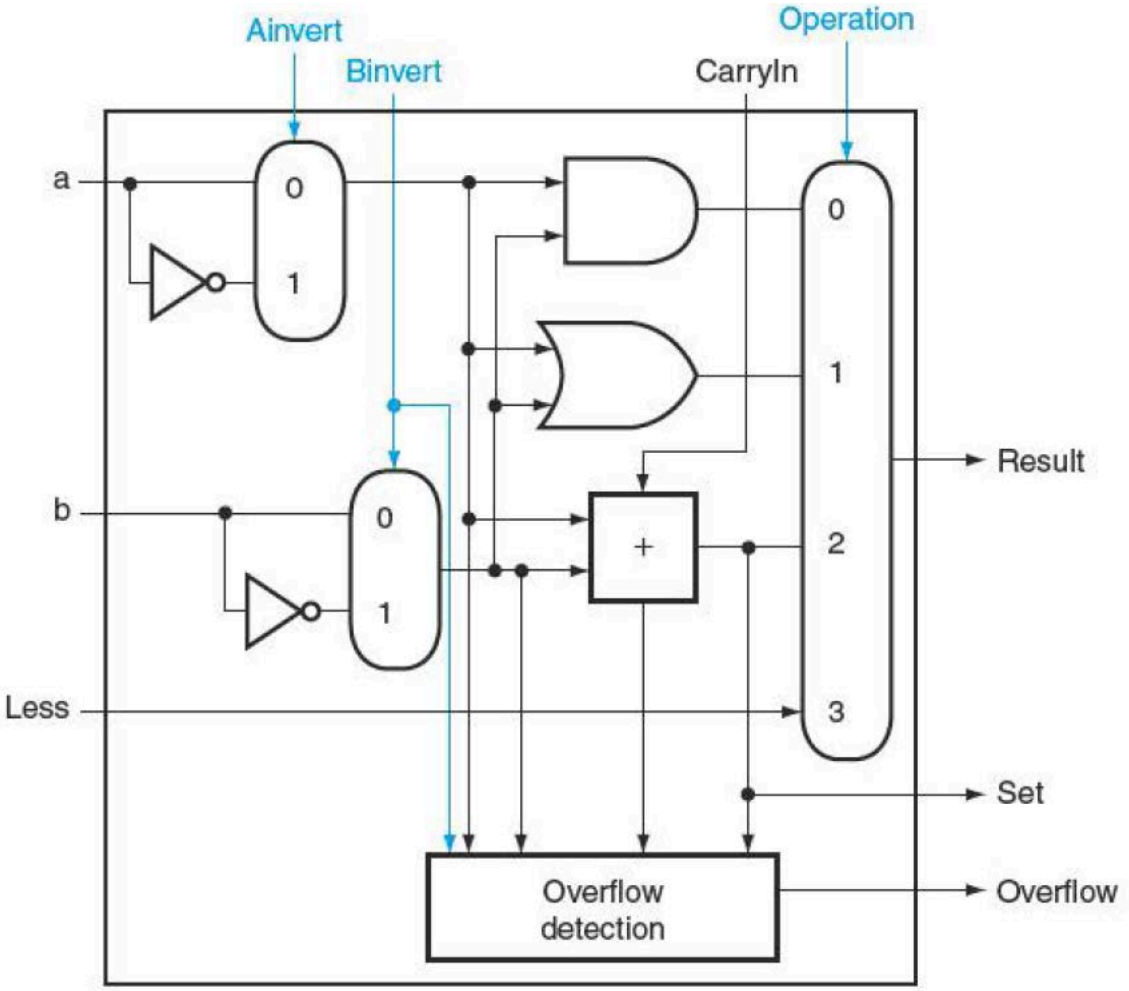
Ainv	Binv	OP[1:0]	Result
0	0	00	AND
1	1	00	NOR
0	0	01	OR
1	1	01	NAND
0	0	10	add
0	1	10	subtract
X	1	11	less*



* **less**: *as in a flag for "set-if-less-than". Note that Binv has to be 1 when using this*

What Does This Logic Block Do?

Same as previous block, but now with ***"overflow detection"*** and with another output, **set**, used only with `slt`



A 32-bit ALU Using 1-bit ALUs as building blocks

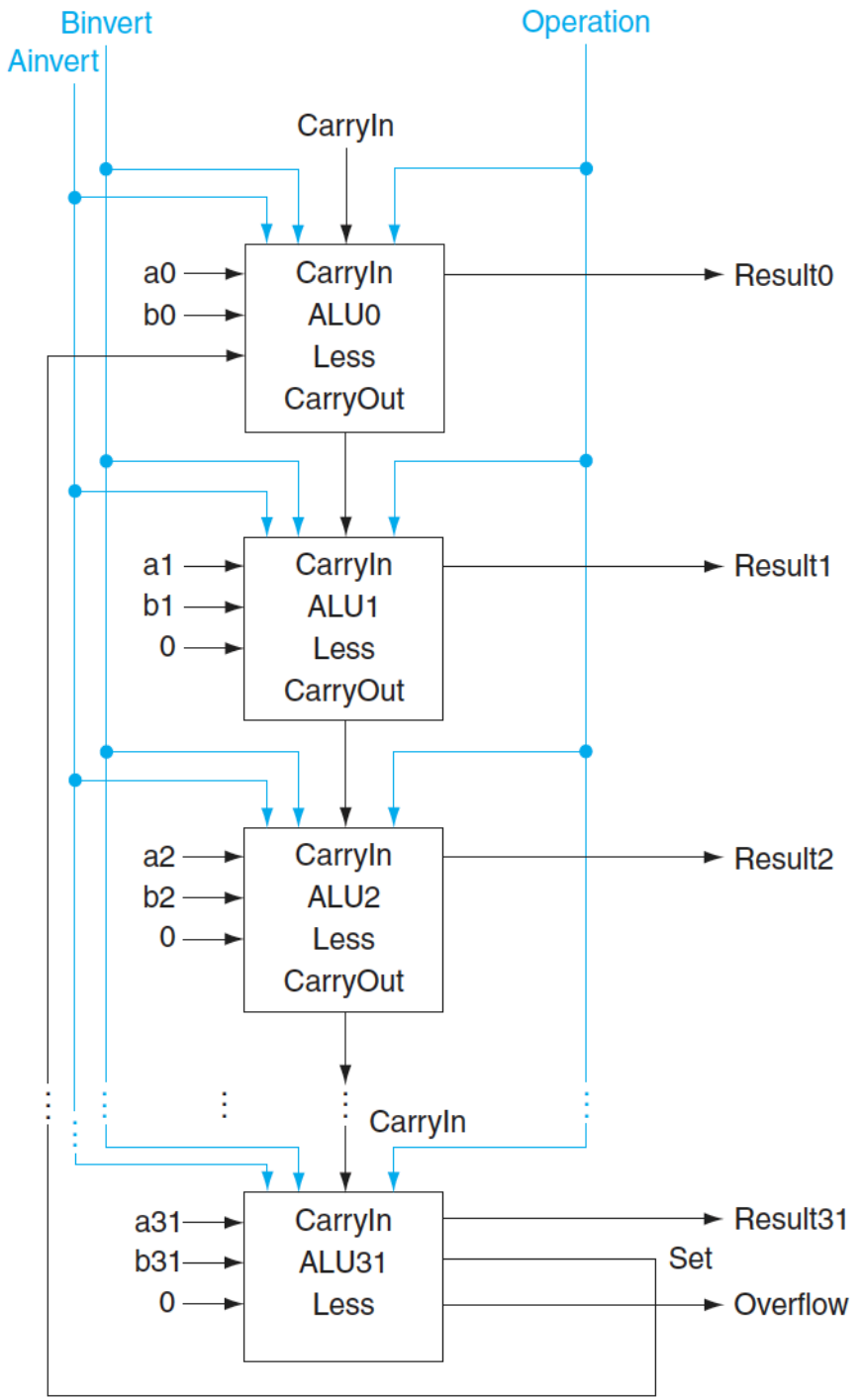
Important Notes/Observations:

1. **slt** and ***overflow*** are decisions made in bit 31 (MSB)
2. Bits **CarryIn** and **Binvert** work the same way (redundant) and so can be combined into one bit called **Bnegate**
3. To support branching ops, we need an "equality" function. This can be done by doing subtraction and seeing if the result is zero (i.e. $a = b \leftrightarrow a - b = 0$)

   So, we need an output that says "the answer at the Result is Zero".

   Best done as:

   $Zero = \overline{(Result1 + Result2 + \ldots + Result31)}$



MIPS ALU Control

ALU when used for

- Load/Store: F = add
- Branch: F = subtract
- R-type: F depends on funct field

ALU_CONTROL[3:0] is **Ainv**, **Bnegate**, **OP1**, **OP0** (in that order)

ALU_CONTROL[3:0]	FUNCTION
0000	AND
0001	OR
0010	add
0110	subtract
0111	set-on-less-than
1100	NOR

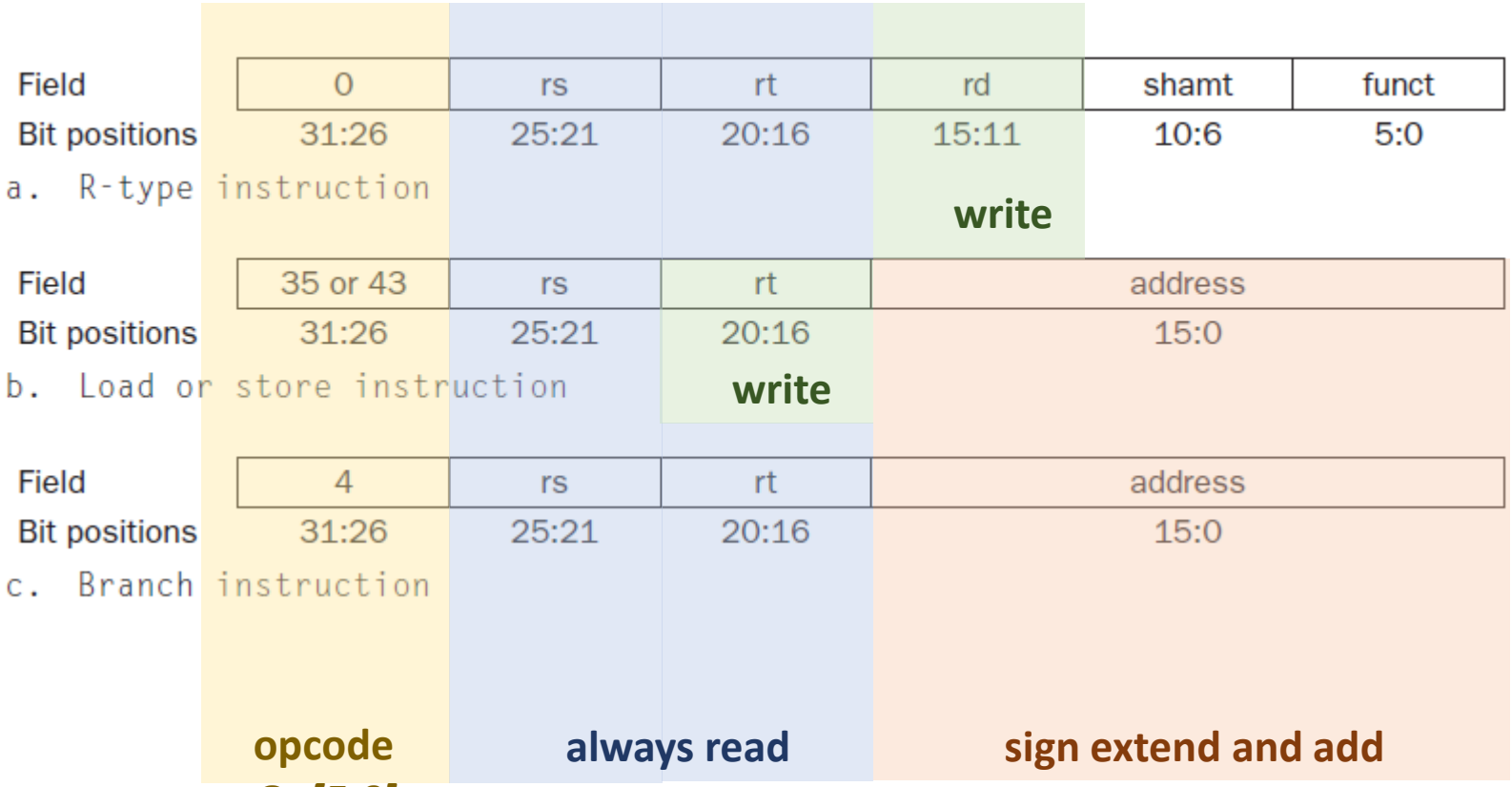
Generating the ALU_Control

- We get **ALUOp** from a decode of the **opcode** field of the instruction
- We can further refine choices by looking at **funct** field

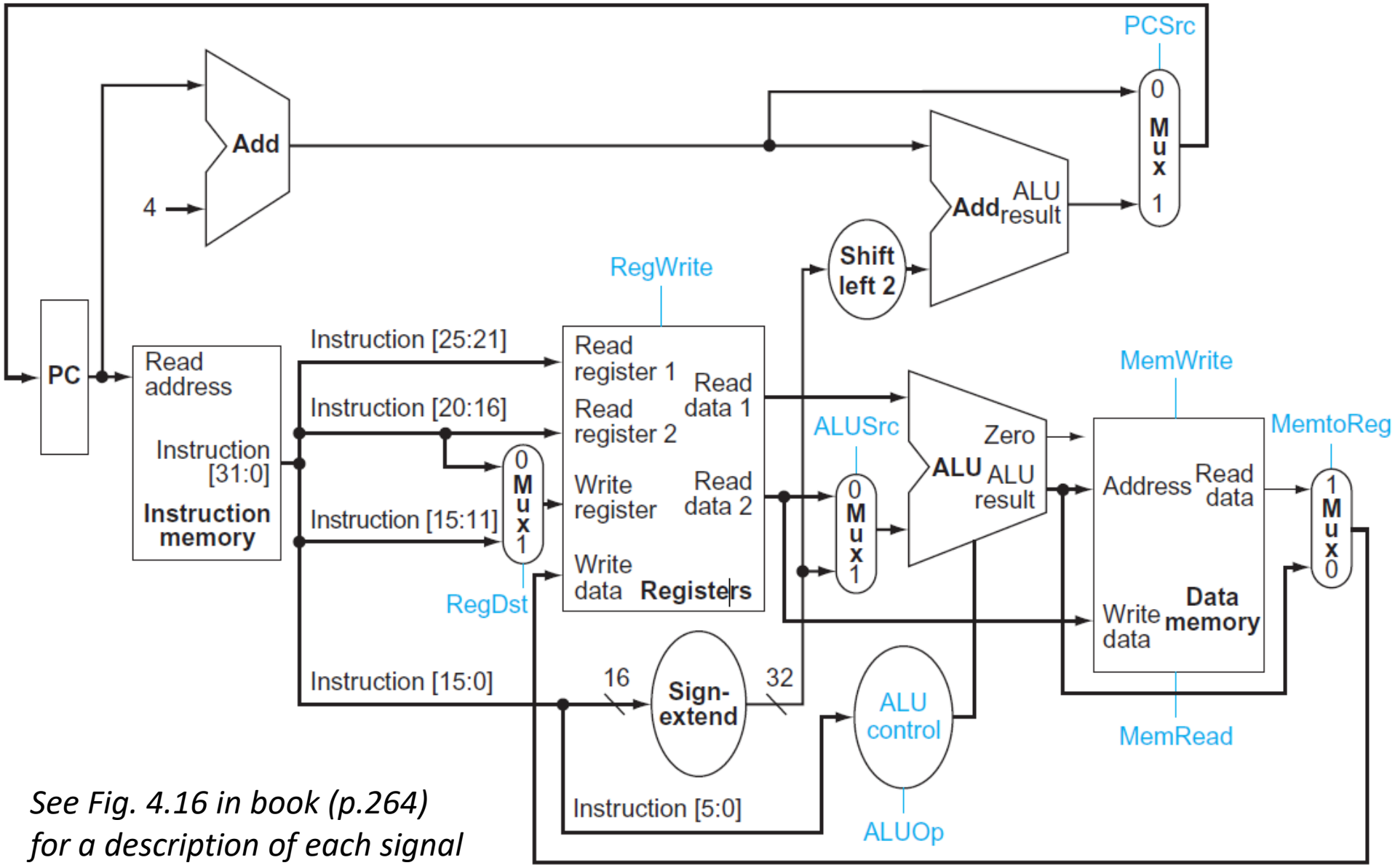
opcode	ALUOp	Operation	funct	ALU function	ALU control
lw	00	load word	XXXXXX	add	0010
sw	00	store word	XXXXXX	add	0010
beq	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
		subtract	100010	subtract	0110
		AND	100100	AND	0000
		OR	100101	OR	0001
		set-on-less-than	101010	set-on-less-than	0111

The Main Control Unit

- Control signals derived (i.e. decoded) from instruction



Full Datapath showing 7 Control Signals



See Fig. 4.16 in book (p.264) for a description of each signal

YOUR TO-DOs for the Week

- Lab 6 due soon…

</LECTURE>