Chapter 4 Review 1

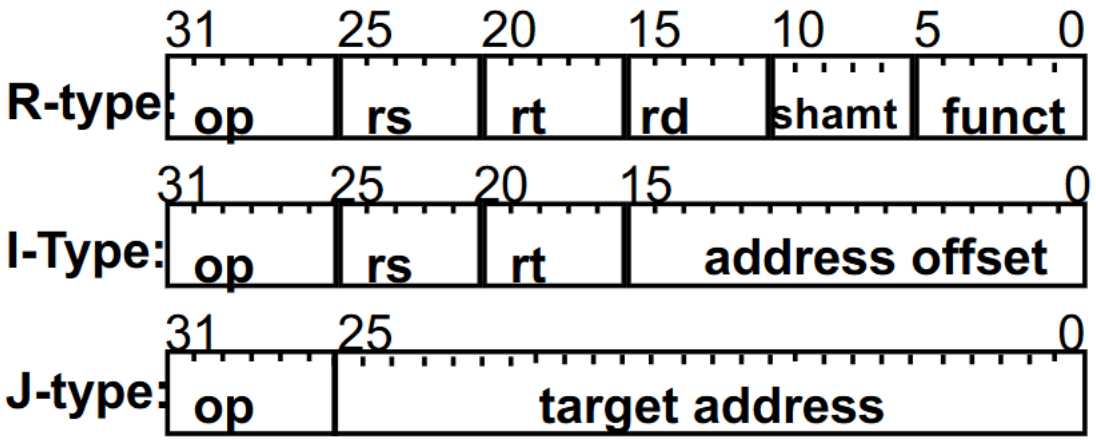
Yuanqing Miao

https://github.com/myuanqing/CMPEN331_Spring2025

In this exercise we examine in detail how an instruction is executed in a single-cycle datapath. Problems in this exercise refer to a clock cycle in which the processor fetches the following instruction word:
10101100011000100000000000010100
Assume that data memory is all zeros and that the processor's registers have the following values at the beginning of the cycle in which the above instruction word is fetched:

r0	r1	r2	r3	r4	r5	r6	r8	r12	r31
0	-1	2	-3	-4	10	6	8	2	-16

a. What are the outputs of the sign-extend and the "Shift left 2" unit (The simple control and datapath are extended to handle the jump instruction)) for this instruction word?



op field always in bits **31-26**

for lw and sw **rs** is the base

```
101011 00011 00010 0000000000010100
```

Step 1: Identify the Instruction Type

Breaking it into fields based on the MIPS instruction format:

- **Opcode (bits 31-26):** `101011` → This corresponds to **SW (Store Word)**
- **rs (bits 25-21):** `00011` → Register `$r3`
- **rt (bits 20-16):** `00010` → Register `$r2`
- **Immediate (bits 15-0):** `0000000000010100` → This is the offset.

Step 2: Sign Extend

The immediate field is `0000000000010100` (16-bit).

- This is `0x14` in hexadecimal, or **20 in decimal**.
- Since the most significant bit (MSB) is `0`, it is already non-negative.
- When sign-extended to 32 bits, it remains **00000000 00000000 00000000 00010100 (decimal 20).**

Thus, **Sign-extend output:** `00000000 00000000 00000000 00010100` (decimal `20`).

Step 3: Shift Left 2

The Shift Left 2 unit shifts the sign-extended immediate left by 2 bits.

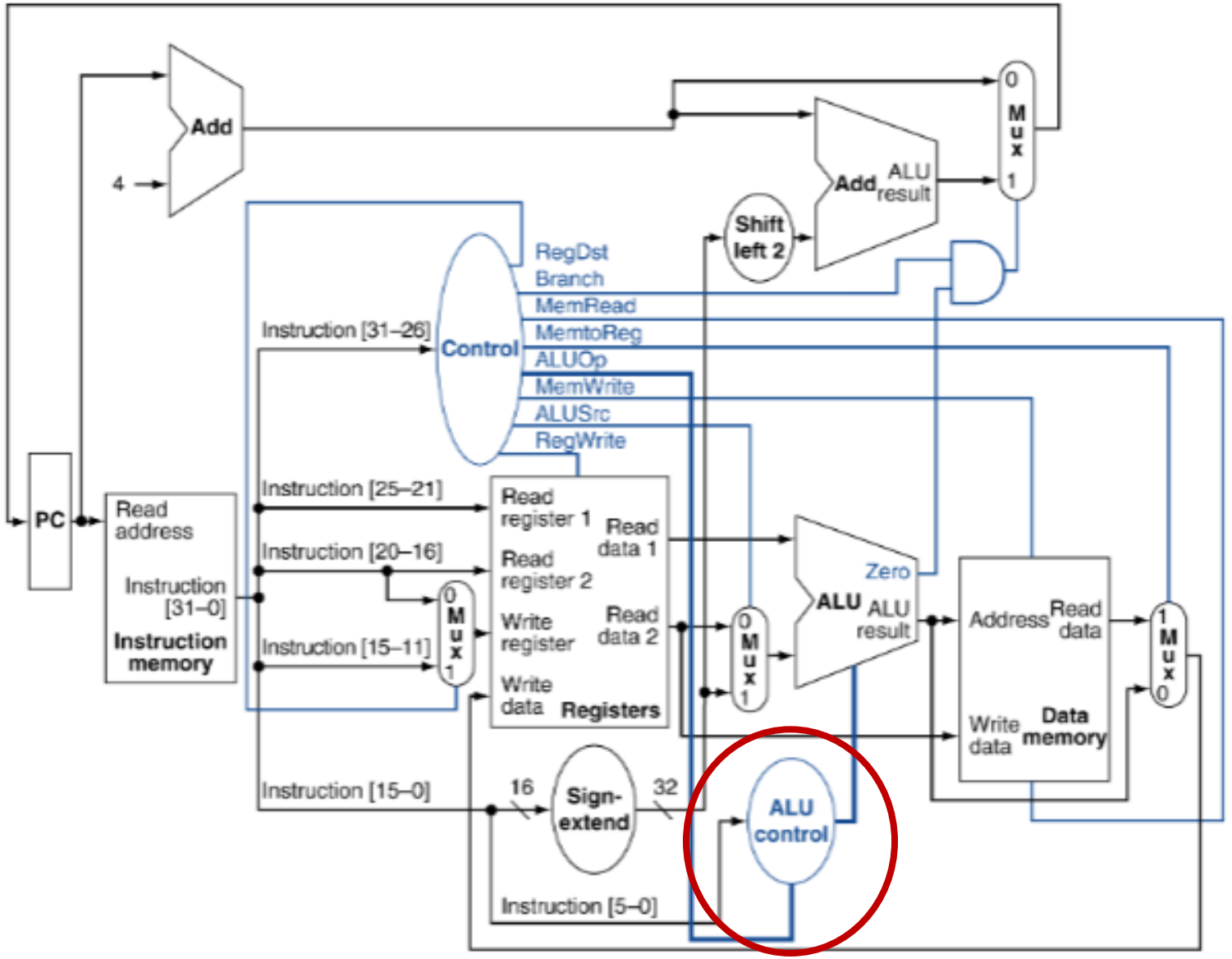
- `00000000 00000000 00000000 00010100` (decimal `20`)
- Shifting left by 2 bits (`<< 2`):
  - `00000000 00000000 00000000 01010000` (decimal **80**)

Thus, **Shift Left 2 output:** `00000000 00000000 00000000 01010000` (decimal `80`).

Final Answer:

- **Sign-extend output:** `00000000 00000000 00000000 00010100` (decimal **20**).
- **Shift Left 2 output:** `00000000 00000000 00000000 01010000` (decimal **80**).

b. What are the values of the ALU control unit's inputs for this instruction?



```
101011 00011 00010 0000000000010100
```

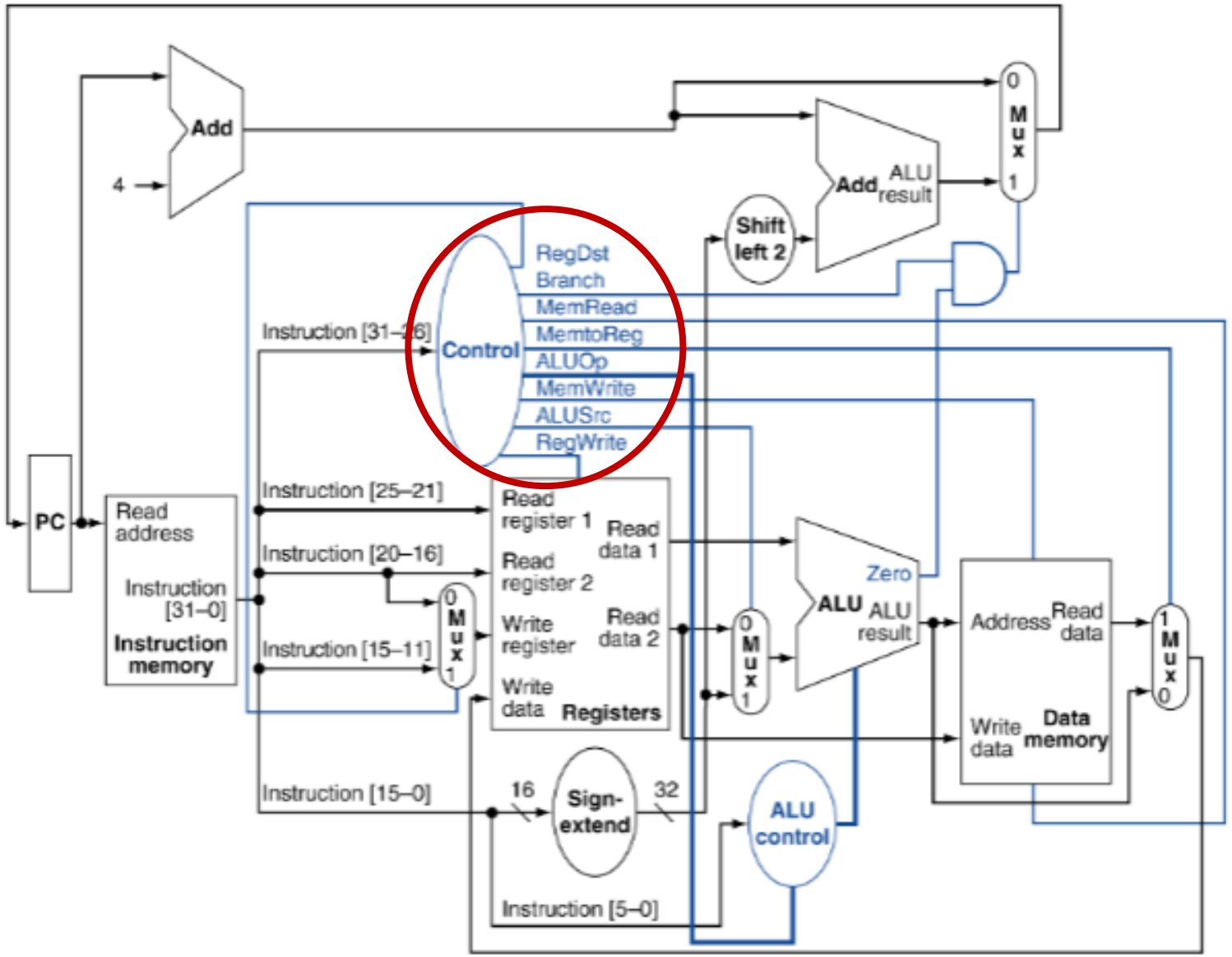
```
SW $r2, 20($r3)
```

- Assume 2-bit ALUOp derived from opcode
  - Combinational logic derives ALU control

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

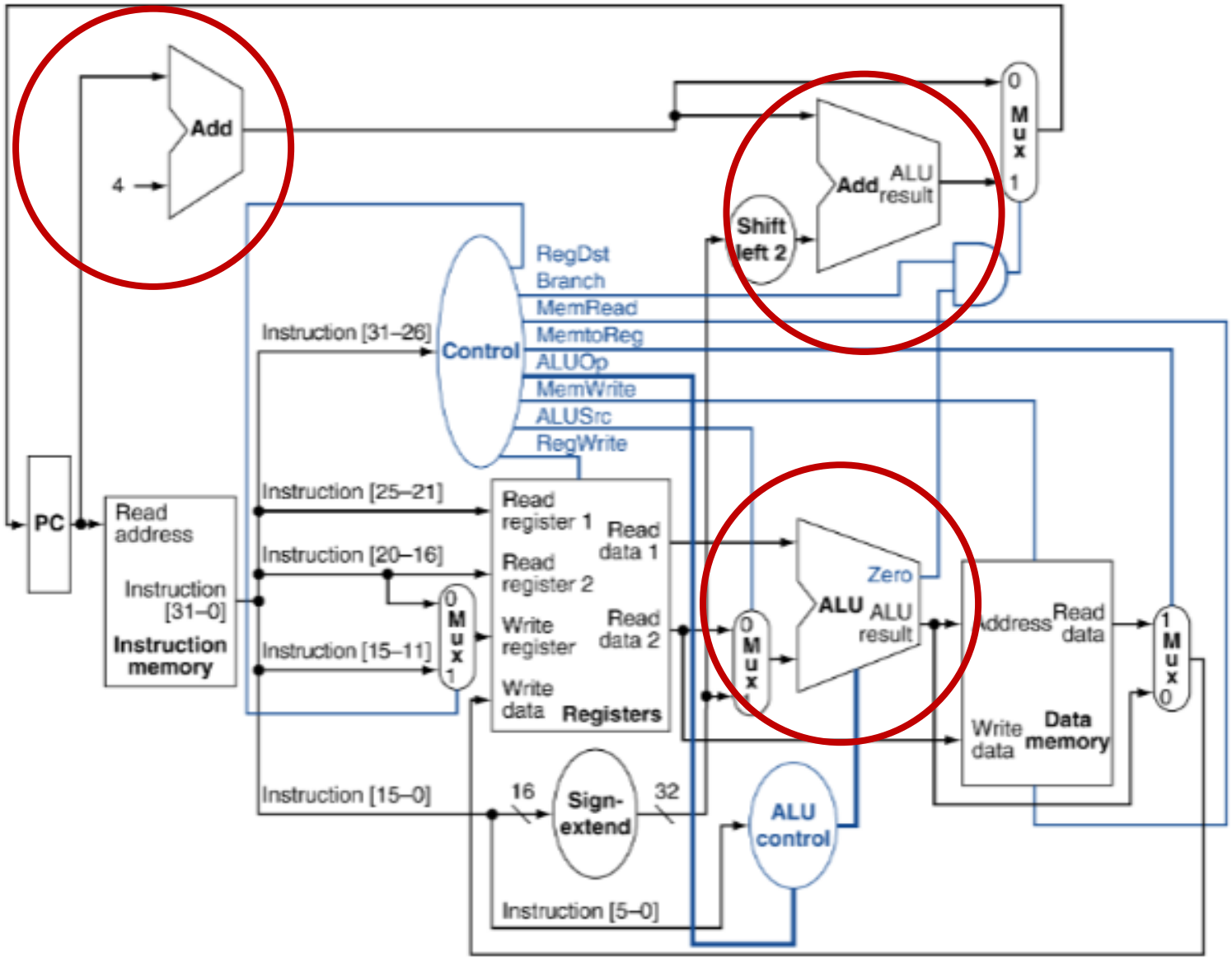
- **ALUOp[1-0]:** `00`
- **Instruction[5-0]:** `010100` (not used for I-type instructions like `SW`)
- **ALU Control Output:** `0010` (Addition)

c. Show the values of the control lines shown in the table.



Control Signal	Value for `SW`	Explanation
RegDst	`X` (Don't care)	`SW` does not write to a register, so destination selection is irrelevant.
Branch	`0`	`SW` is not a branch instruction, so no branching occurs.
MemRead	`0`	`SW` does not **read** from memory; it only writes data to memory.
MemtoReg	`X` (Don't care)	No register write occurs, so the source of register data is irrelevant.
ALUOp	`00`	The ALU performs **addition** (`rs + offset`) to calculate the memory address.
MemWrite	`1`	`SW` **writes data to memory**, so memory write is enabled.
ALUSrc	`1`	The second ALU operand comes from the **sign-extended immediate (offset)**, not a register.
RegWrite	`0`	`SW` does **not** write to a register, so register write is disabled.

d. For the ALU and the two add units, what are their data input values?




```
SW $r2, 20($r3)
```

r0	r1	r2	r3	r4	r5	r6	r8	r12	r31
0	-1	2	-3	-4	10	6	8	2	-16

1. **ALU Inputs:**
   - **First Operand:** `rs` (`$r3` = `-3`)
   - **Second Operand:** Sign-extended immediate (`20`)
   - **ALU Operation:** Addition (`-3 + 20 = 17`)
2. **First Adder (PC + 4):**
   - **Inputs:** `PC` and `4`
   - **Output:** `PC + 4` (new instruction address)
3. **Second Adder (Branch Target Calculation):**
   - **Inputs:** `Sign-Extended Immediate (shifted left 2)` and `PC + 4`
   - **Output:** Used only for branches (not applicable here)

e. What are all inputs for the "Registers" unit?

a. Registers

The **Registers Unit** has the following inputs:

- **Read Register 1** (`rs`): `$r3` (value = `-3`)
- **Read Register 2** (`rt`): `$r2` (value = `2`)
- **Write Register**: Not used in `SW`
- **Write Data**: Not used in `SW`
- **RegWrite**: `0` (disabled)