

CS2100

<http://www.comp.nus.edu.sg/~cs2100/>

COMPUTER ORGANISATION

Lecture #12

The Processor: Control

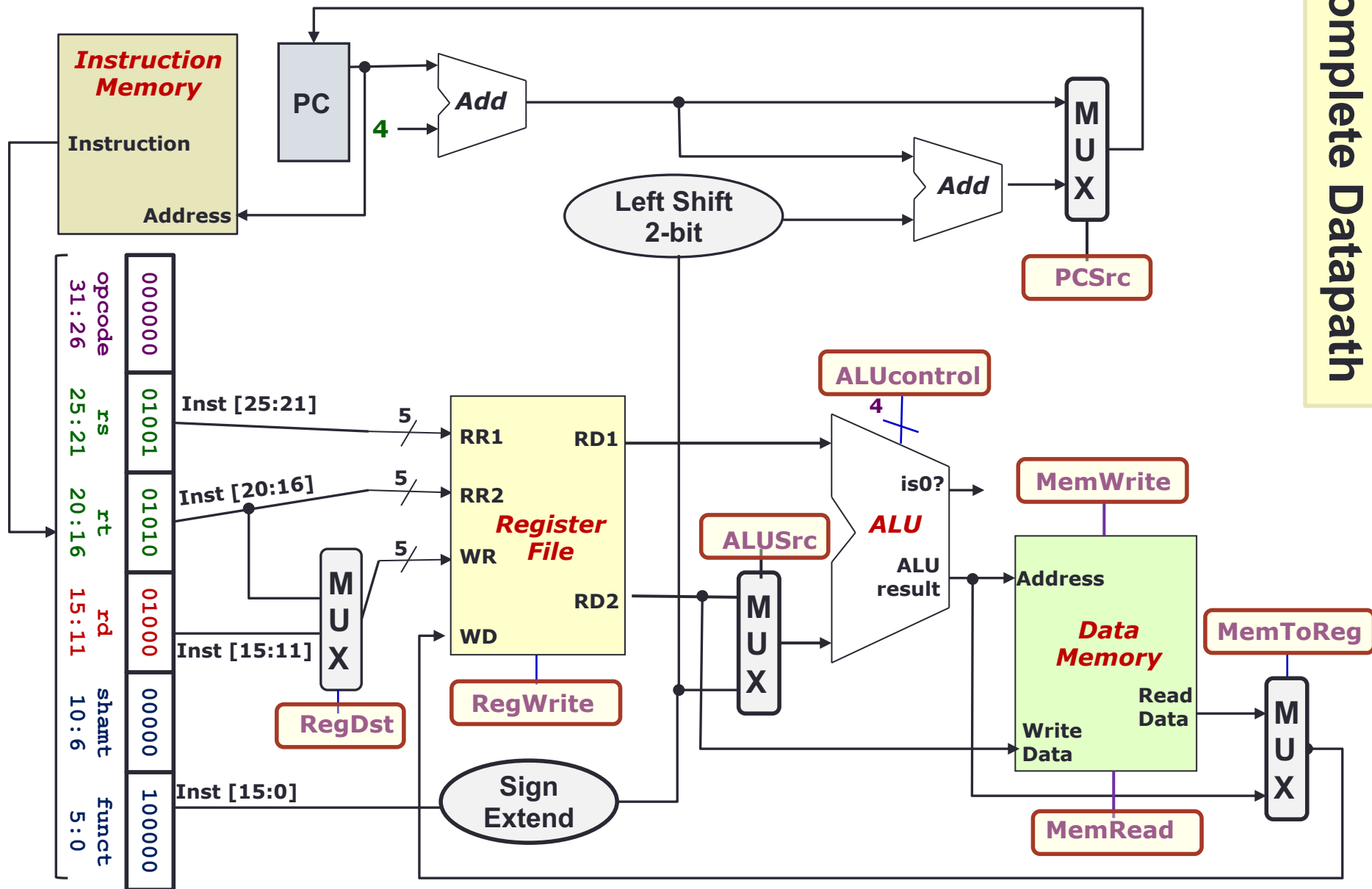


NUS
National University
of Singapore

School of
Computing

Lecture #12: Processor: Control

1. Identified Control Signals
2. Generating Control Signals: Idea
3. The Control Unit
4. Control Signals
5. ALU Control Signal
6. Instruction Execution

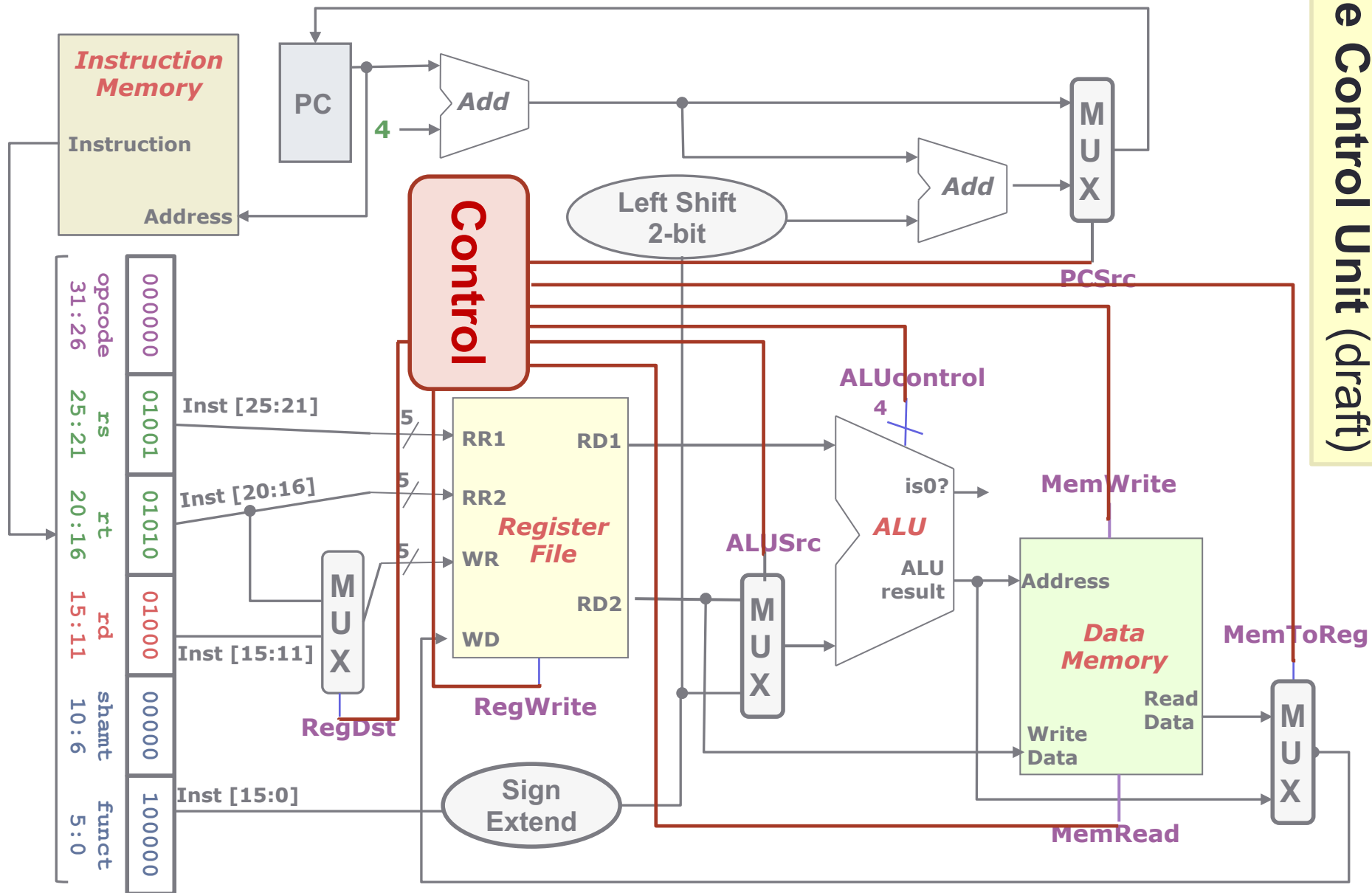


1. Identified Control Signals

Control Signal	Execution Stage	Purpose
RegDst	Decode/Operand Fetch	Select the destination register number
RegWrite	Decode/Operand Fetch RegWrite	Enable writing of register
ALUSrc	ALU	Select the 2 nd operand for ALU
ALUControl	ALU	Select the operation to be performed
MemRead / MemWrite	Memory	Enable reading/writing of data memory
MemToReg	RegWrite	Select the result to be written back to register file
PCSrc	Memory/RegWrite	Select the next PC value

2. Generating Control Signals: Idea

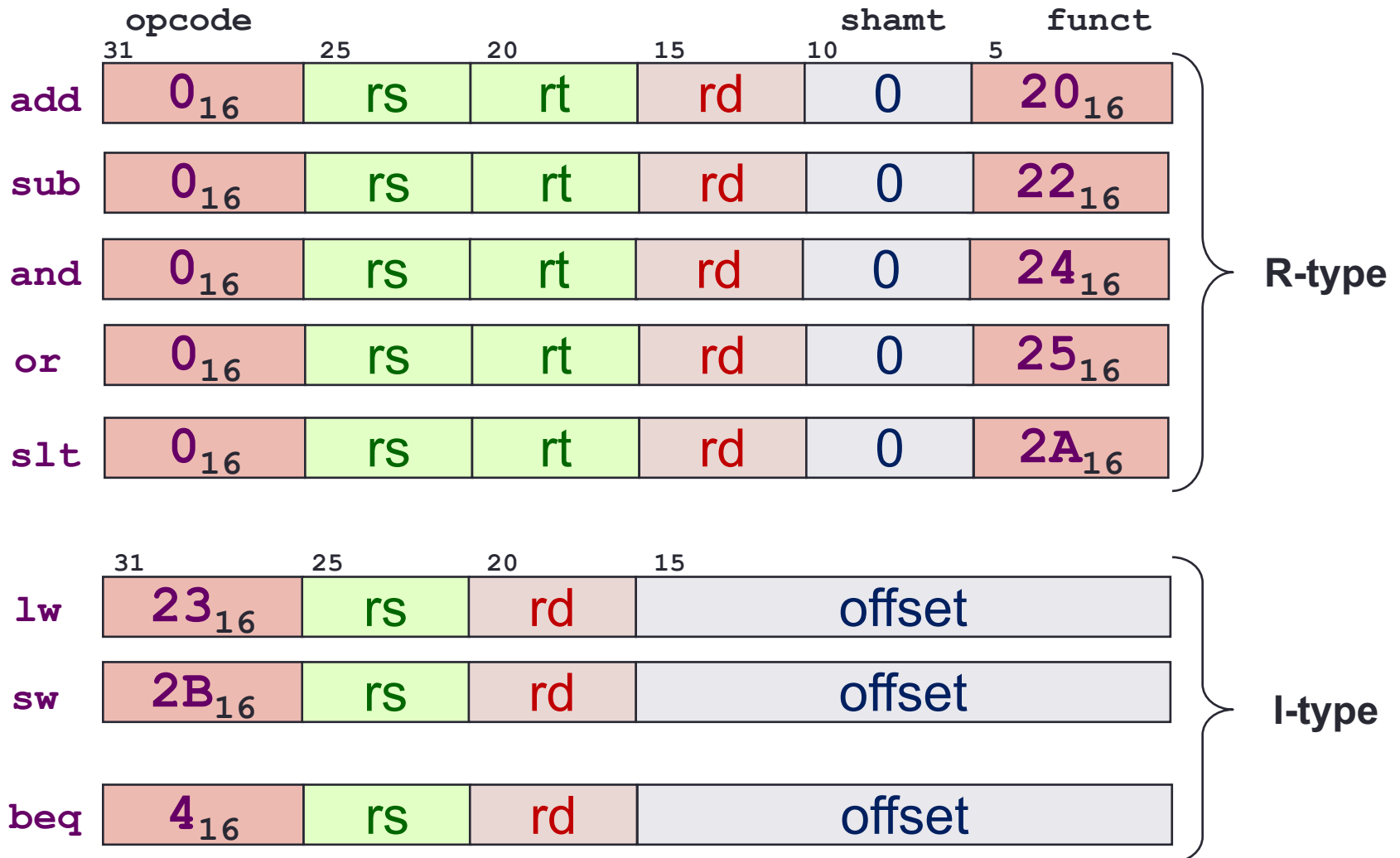
- The control signals are generated based on the instruction to be executed:
 - Opcode → Instruction Format
 - Example:
 - R-Format instruction → **RegDst** = 1 (use **Inst**[**15:11**])
 - R-Type instruction has additional information:
 - The 6-bit "**funct**" (function code, **Inst**[**5:0**]) field
- **Idea:**
 - Design a combinational circuit to generate these signals based on Opcode and possibly Function code
 - A **control unit** is needed (a draft design is shown next)



3. Let's Implement the Control Unit!

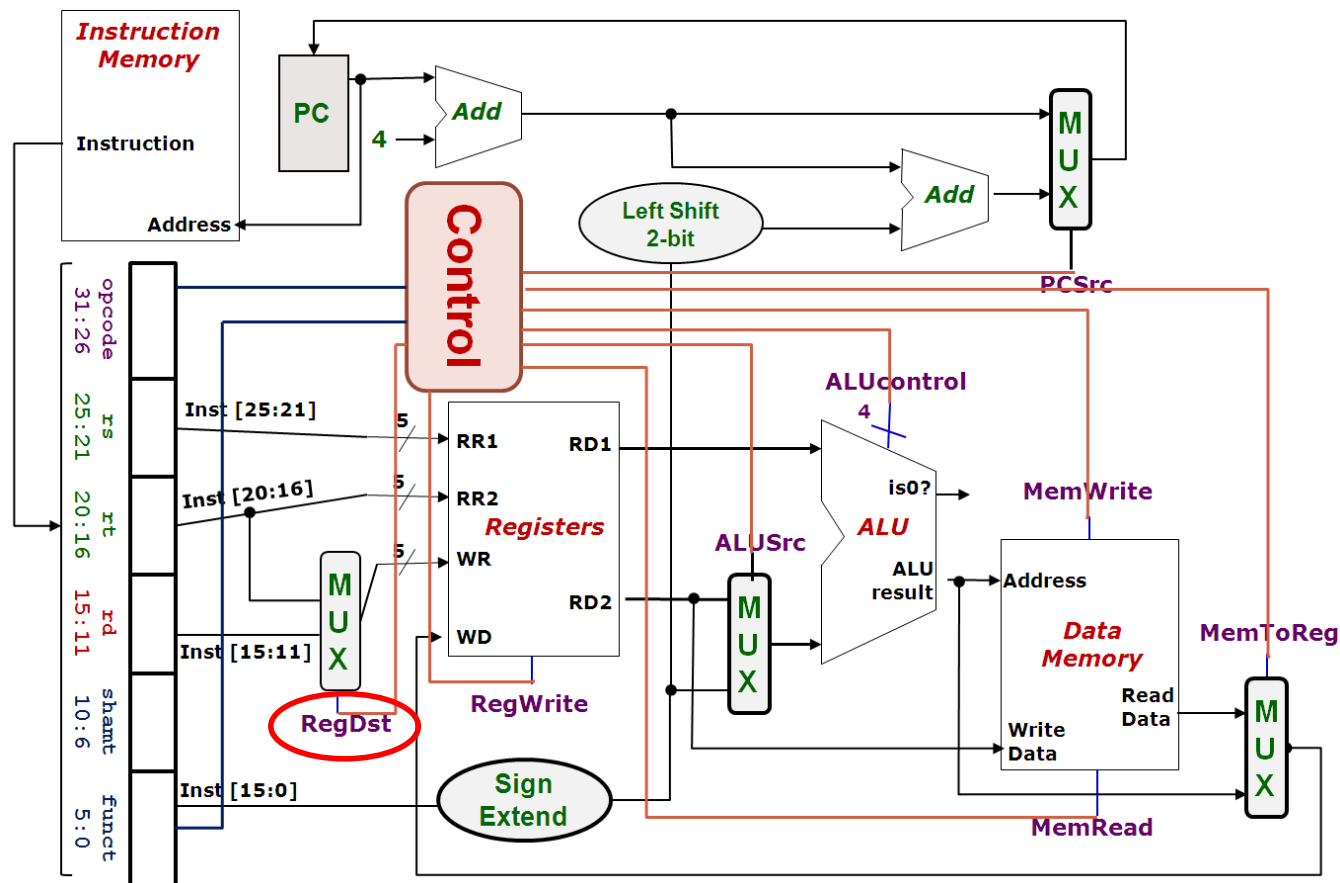
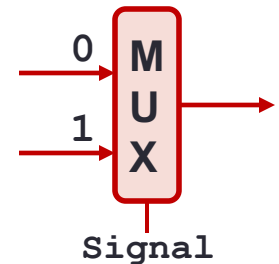
- Approach:
 - Take note of the instruction subset to be implemented:
 - Opcode and Function Code (if applicable)
 - Go through each signal:
 - Observe how the signal is generated based on the instruction opcode and/or function code
 - Construct truth table
 - Design the control unit using logic gates

3. MIPS Instruction Subset (Review)



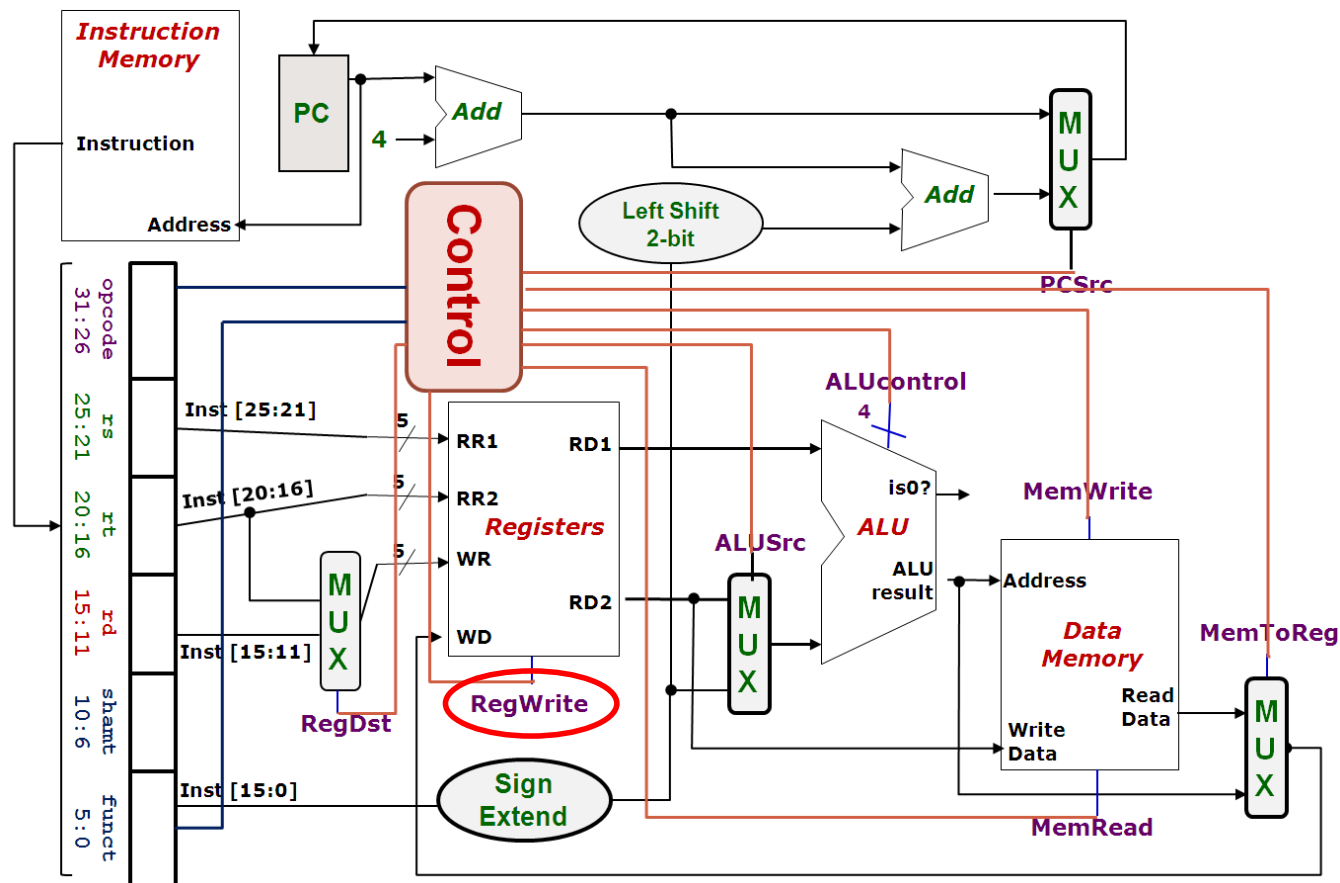
4. Control Signal: RegDst

- False (0): Write register = $Inst[20:16]$
- True (1): Write register = $Inst[15:11]$



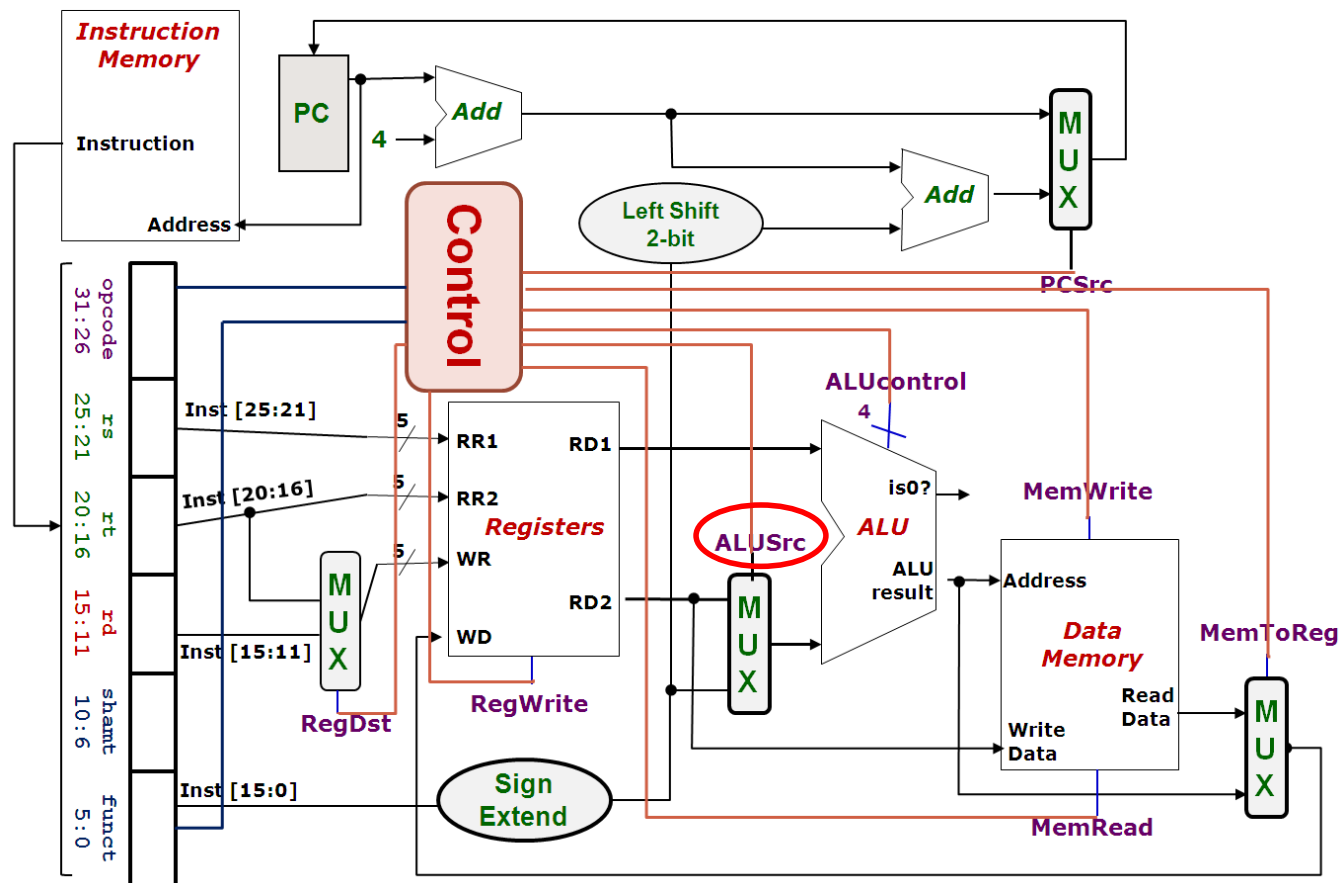
4. Control Signal: RegWrite

- **False (0):** No register write
- **True (1):** New value will be written



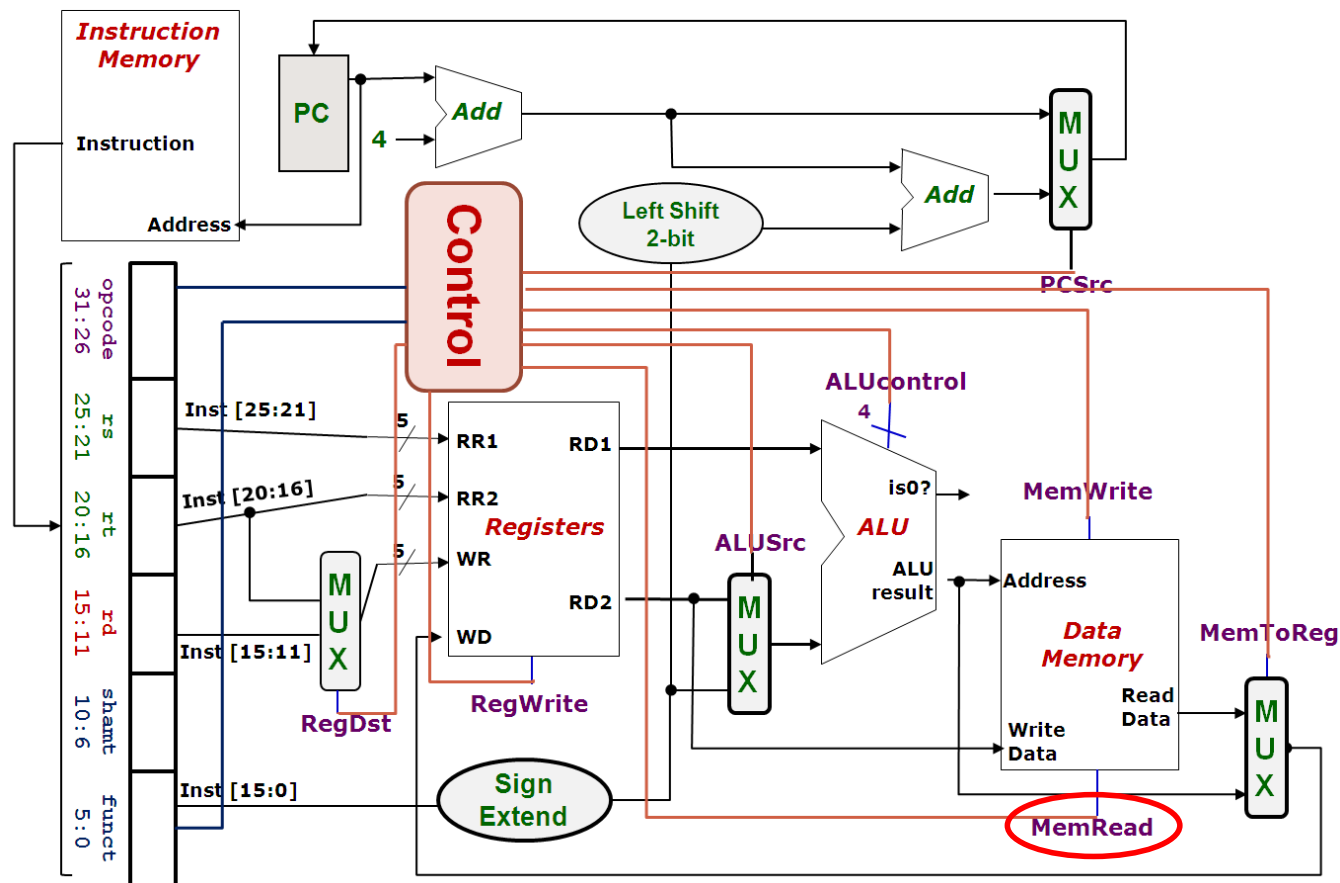
4. Control Signal: ALUSrc

- **False (0):** Operand2 = Register Read Data 2
- **True (1):** Operand2 = SignExt(**Inst** [**15:0**])



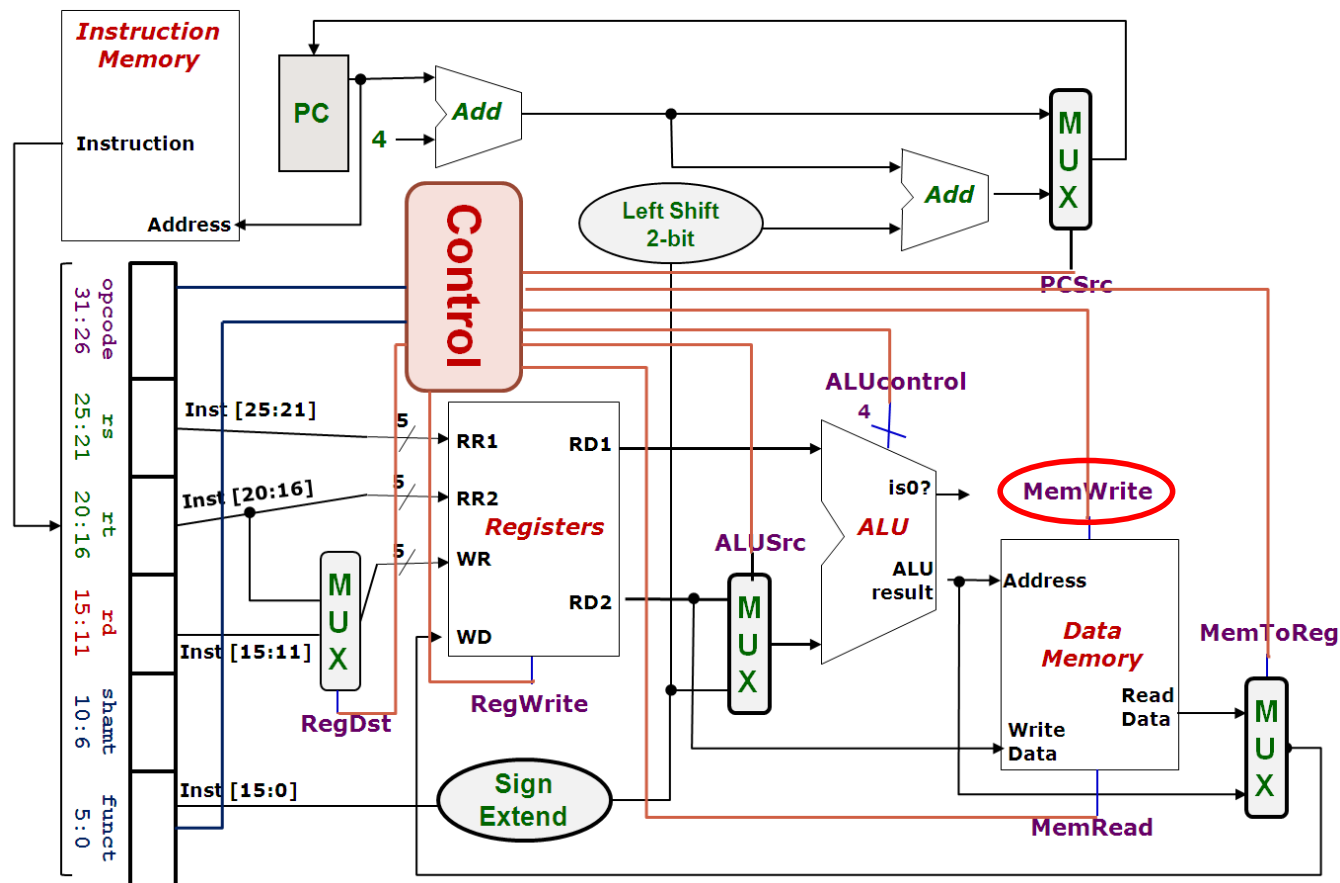
4. Control Signal: MemRead

- **False (0):** Not performing memory read access
- **True (1):** Read memory using *Address*



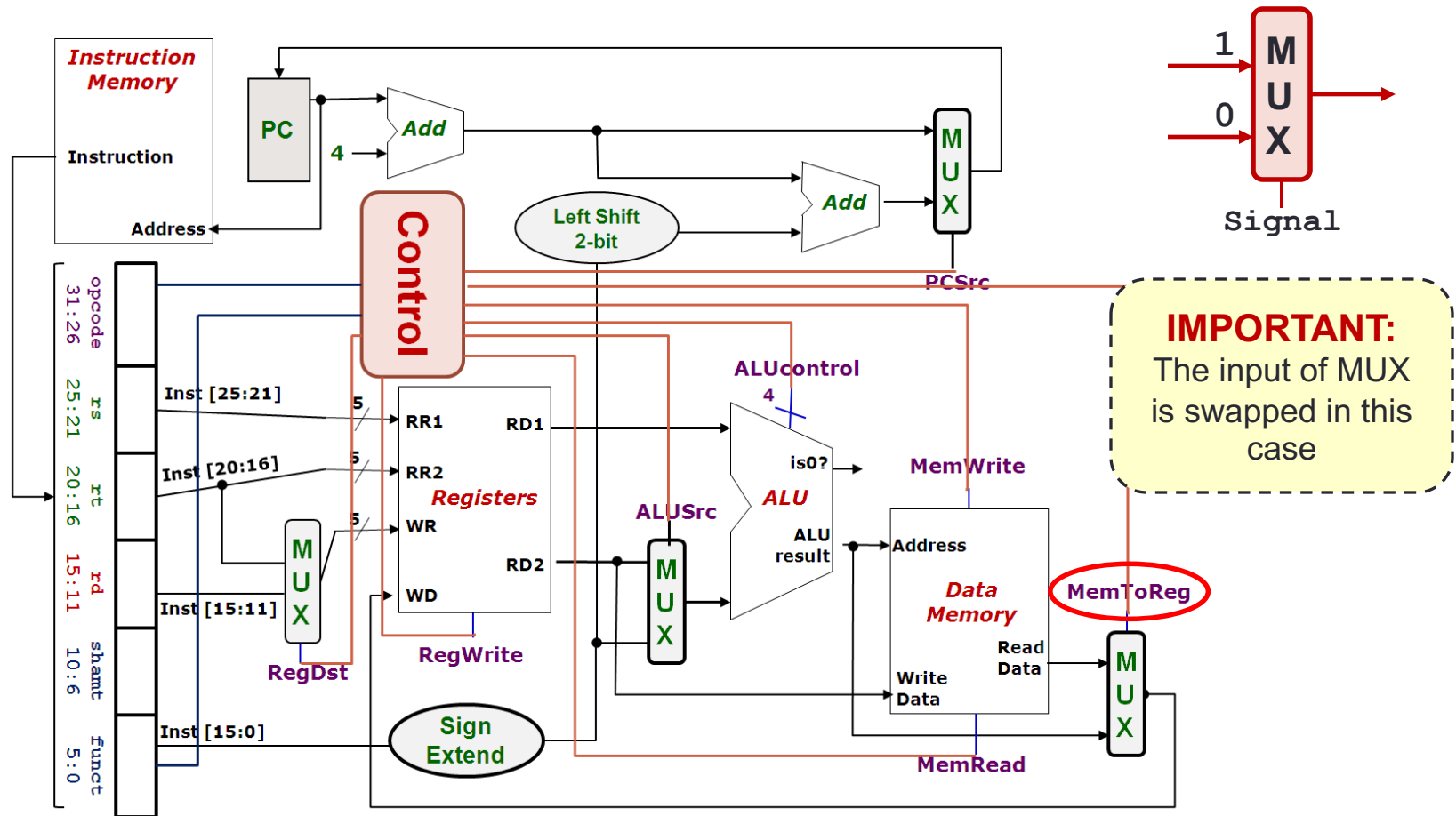
4. Control Signal: MemWrite

- **False (0):** Not performing memory write operation
- **True (1):** $\text{memory}[\text{Address}] \leftarrow \text{Register Read Data 2}$



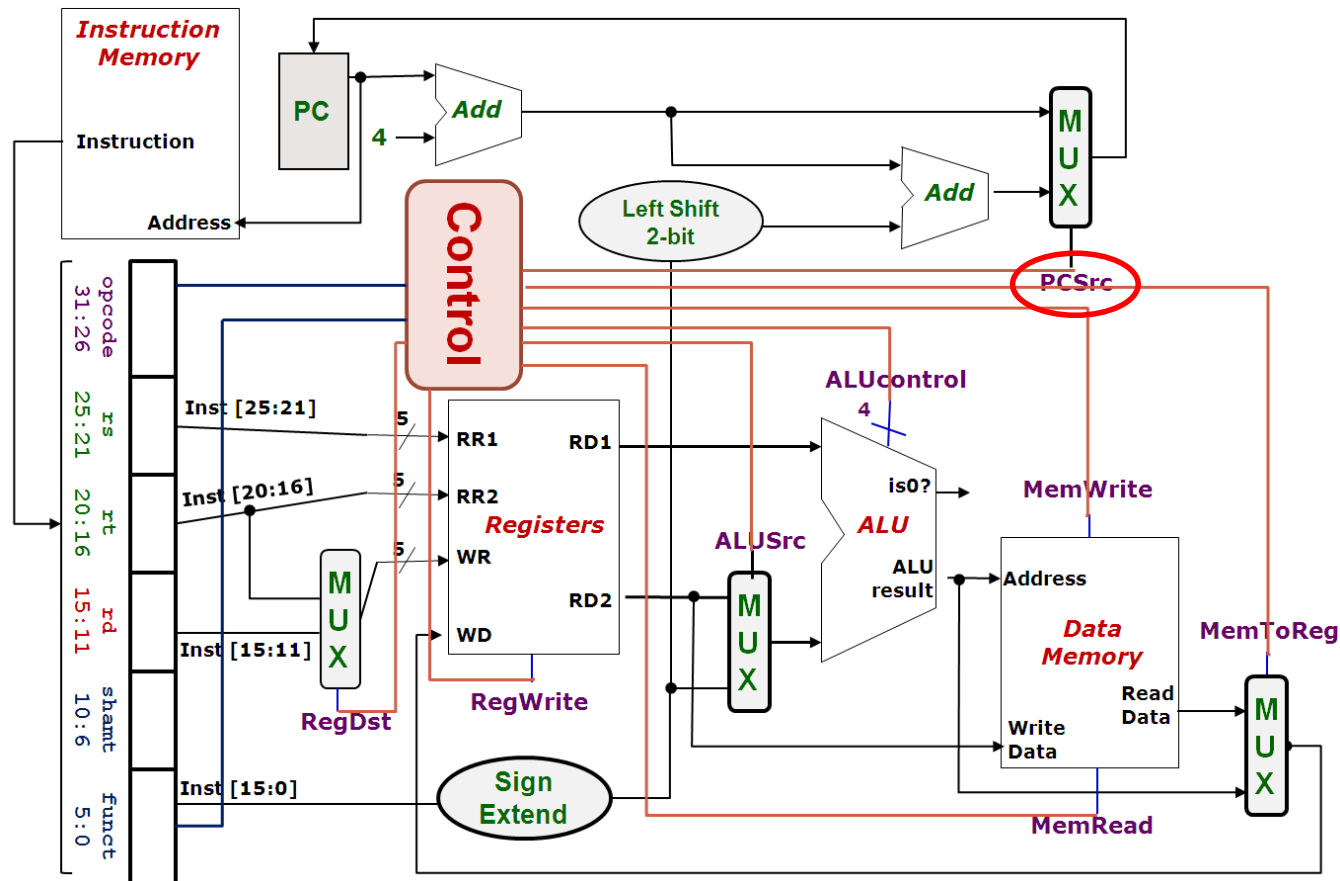
4. Control Signal: MemToReg

- **True (1):** Register write data = Memory read data
- **False (0):** Register write data = ALU result



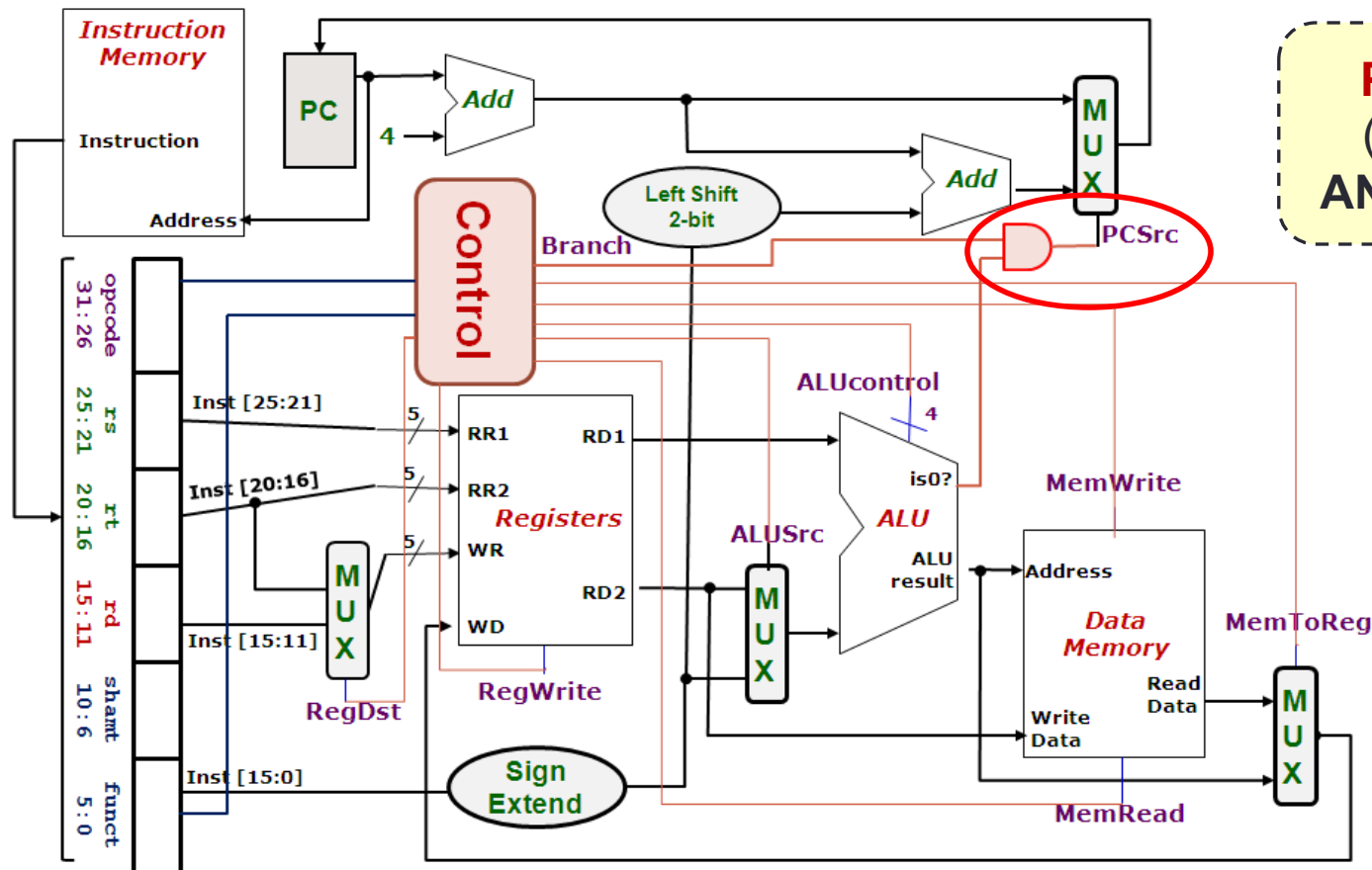
4. Control Signal: PCSrc (1/2)

- The "isZero?" signal from the ALU gives us the actual branch outcome (taken/not taken)
- **Idea:** "If instruction is a branch **AND** taken, then..."



4. Control Signal: PCSrc (2/2)

- **False (0):** Next PC = PC + 4
- **True (1):** Next PC = $\text{SignExt}(\text{Inst}[15:0]) \ll 2 + (\text{PC} + 4)$



PCSrc =
(Branch
AND isZero)

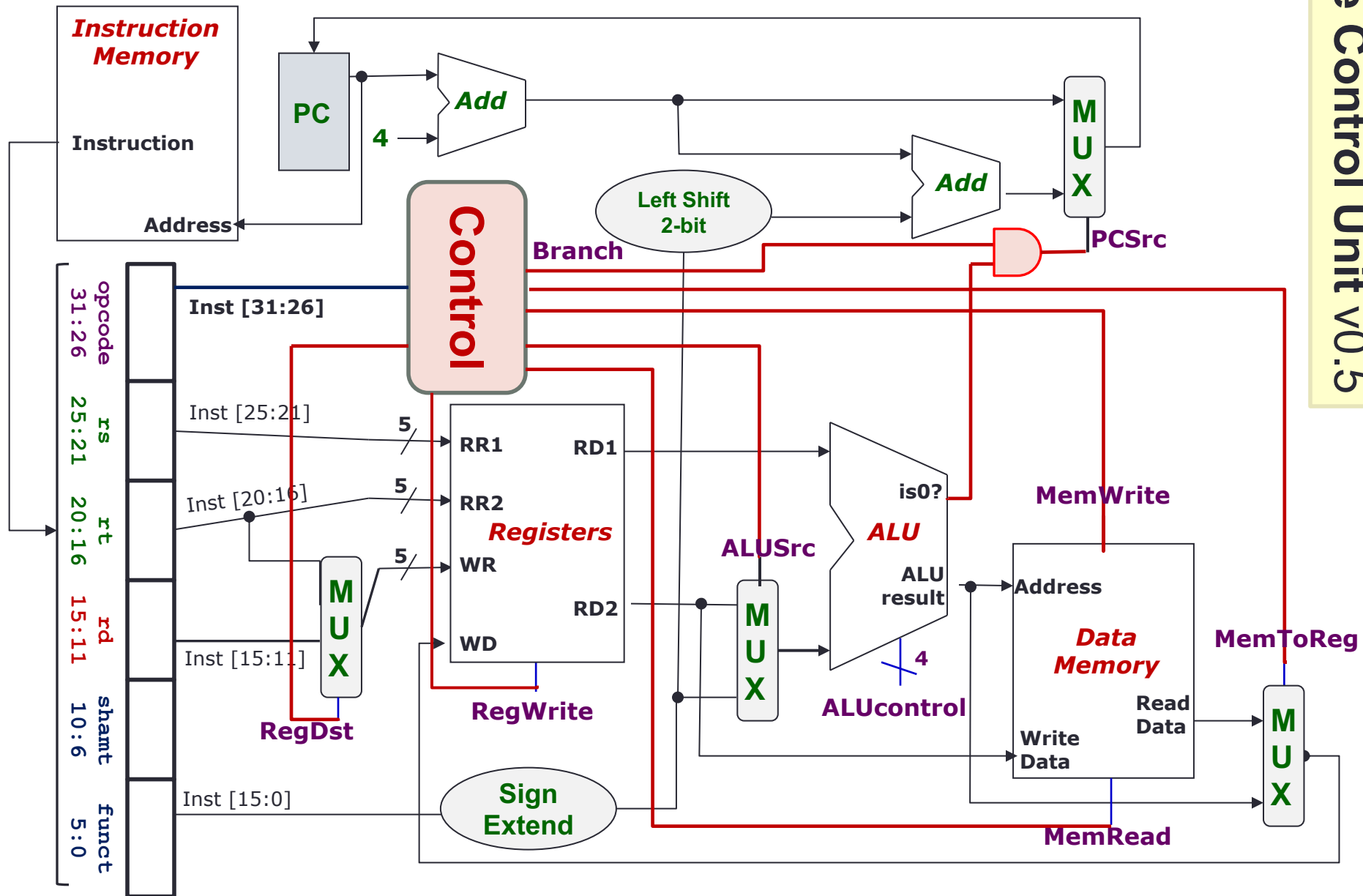
4. Midpoint Check

- We have gone through almost all of the signals:

- Left with the more challenging **ALUControl** signal

Control Signal	Execution Stage	Purpose
RegDst	Decode/Operand Fetch	Select the destination register number
RegWrite	Decode/Operand Fetch RegWrite	Enable writing of register
ALUSrc	ALU	Select the 2 nd operand for ALU
ALUControl	ALU	Select the operation to be performed
MemRead / MemWrite	Memory	Enable reading/writing of data memory
MemToReg	RegWrite	Select the result to be written back to register file
PCSrc	Memory/RegWrite	Select the next PC value

- Observation so far:
 - The signals discussed so far can be generated by *opcode* directly
 - Function code is not needed up to this point
- A major part of the controller can be built based on *opcode* alone



5. Closer Look at ALU

Note: We will cover combinational circuits after the recess.

- The ALU is a combinational circuit:
 - Capable of performing several arithmetic operations
- In Lecture #11:
 - We noted the required operations for the MIPS subset

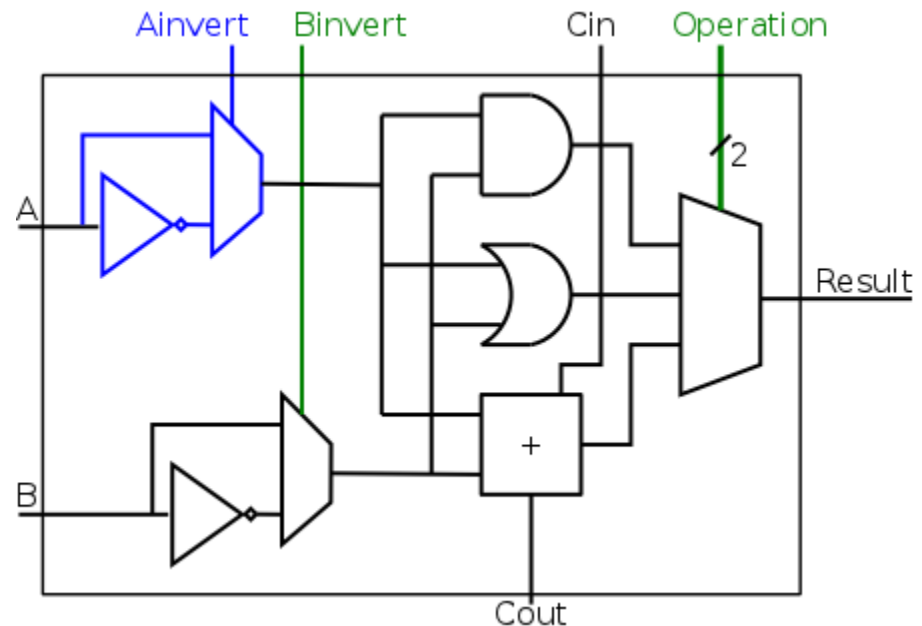
- Question:
 - How is the **ALUcontrol** signal designed?

ALUcontrol	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	slt
1100	NOR

5. One Bit At A Time

Note: We will revisit this when we cover combinational circuits later.

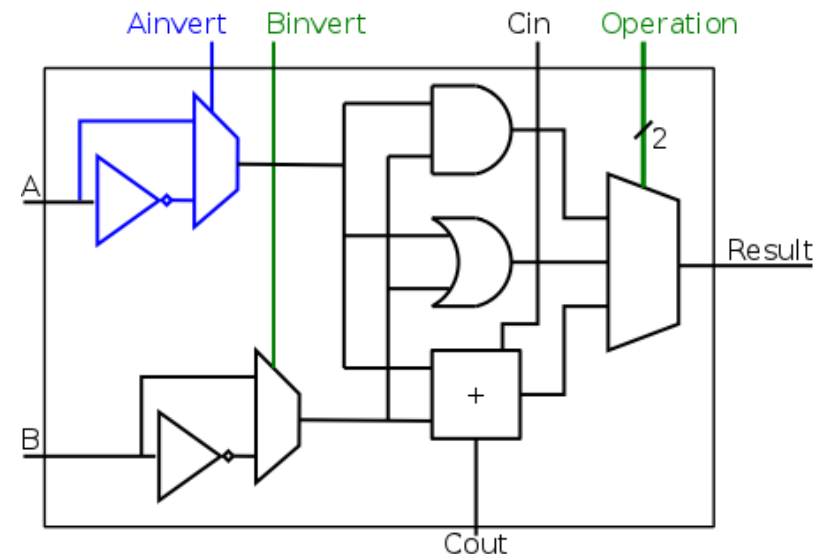
- A simplified 1-bit MIPS ALU can be implemented as follows:
- 4 control bits are needed:
 - **Ainvert**:
 - 1 to invert input A
 - **Binvert**:
 - 1 to invert input B
 - **Operation** (2-bit)
 - To select one of the 3 results



5. One Bit At A Time (Aha!)

- Can you see how the **ALUcontrol** (4-bit) signal controls the ALU?
 - Note: implementation for **slt** not shown

ALUcontrol			Function
Ainvert	Binvert	Operation	
0	0	00	AND
0	0	01	OR
0	0	10	add
0	1	10	subtract
0	1	11	slt
1	1	00	NOR



5. Multilevel Decoding

- Now we can start to design for **ALUcontrol** signal, which depends on:
 - Opcode (6-bit) field **and** Function Code (6-bit) field
- **Brute Force approach:**
 - Use Opcode and Function Code directly, i.e. finding expressions with 12 variables
- **Multilevel Decoding approach:**
 - Use some of the input to reduce the cases, then generate the full output
 - Simplify the design process, reduce the size of the main controller, potentially speedup the circuit

5. Intermediate Signal: **ALUop**

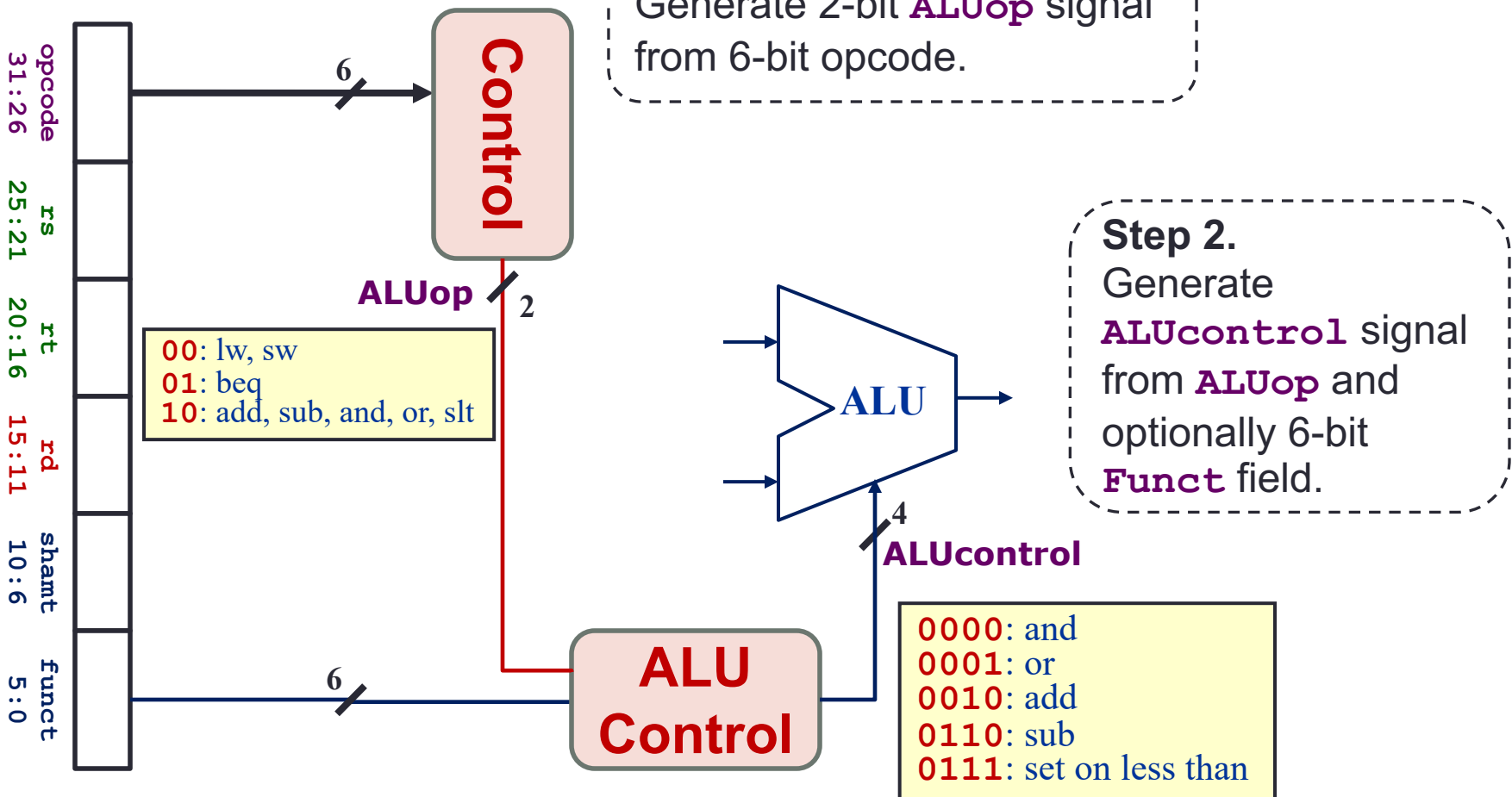
■ Basic Idea:

1. Use Opcode to generate a 2-bit **ALUop** signal
 - Represents classification of the instructions:

Instruction type	ALUop
lw / sw	00
beq	01
R-type	10

2. Use **ALUop** signal and Function Code field (for R-type instructions) to generate the 4-bit **ALUcontrol** signal

5. Two-level Implementation



5. Generating ALUcontrol Signal

Opcode	ALUop	Instruction Operation	Funct field	ALU action	ALU control
lw		load word		add	
sw		store word		add	
beq		branch equal		subtract	
R-type		add		add	
R-type		subtract		subtract	
R-type		AND		AND	
R-type		OR		OR	
R-type		set on less than		set on less than	

Instruction Type	ALUop
lw / sw	00
beq	01
R-type	10

ALUcontrol	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	slt
1100	NOR

Generation of 2-bit **ALUop** signal will be discussed later

5. Design of ALU Control Unit (1/2)

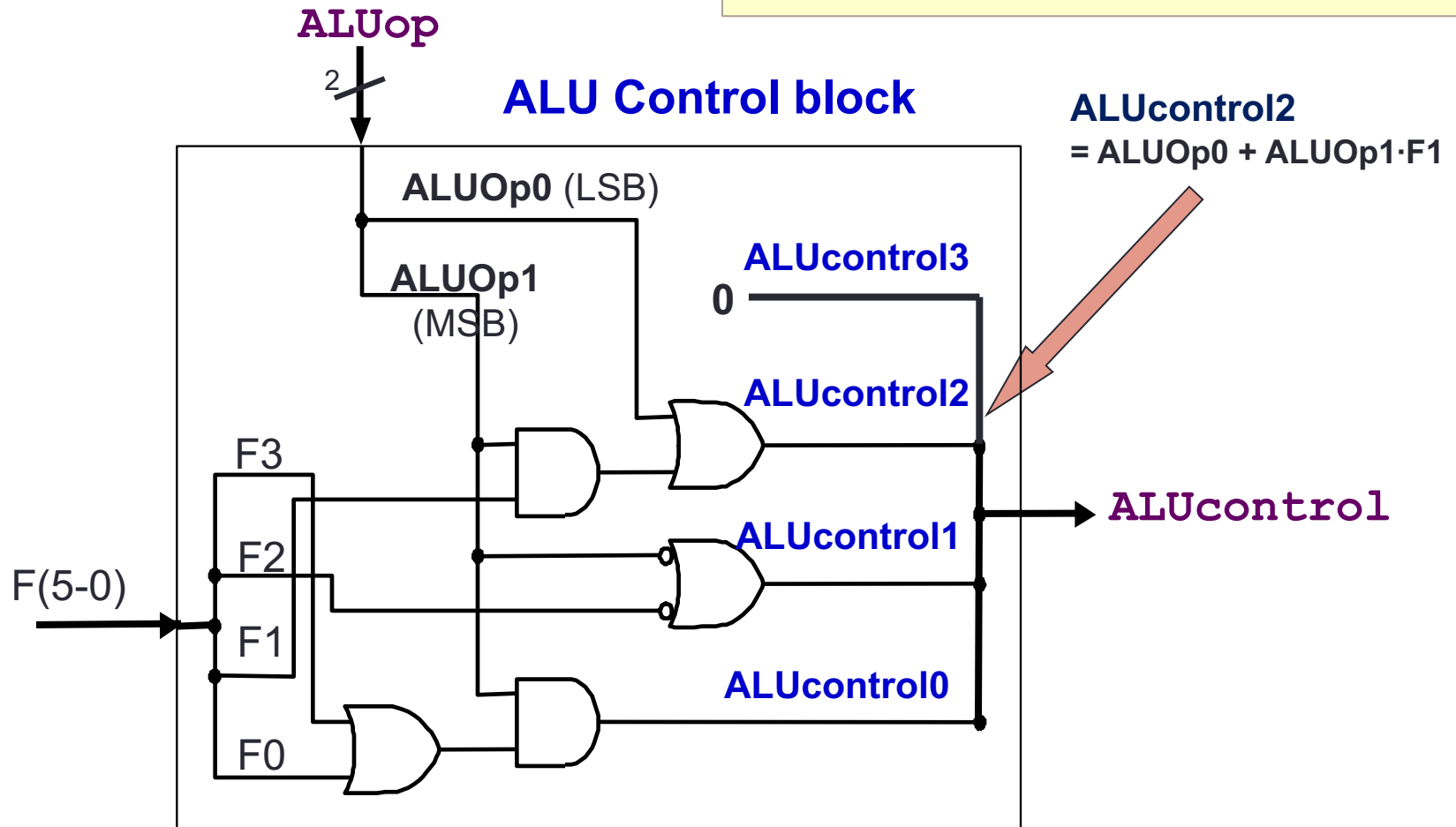
- **Input:** 6-bit **Funct** field and 2-bit **ALUop**
- **Output:** 4-bit **ALUcontrol**
- Find the simplified expressions

	ALUop		Funct Field (F[5:0] == Inst[5:0])						ALU control
	MSB	LSB	F5	F4	F3	F2	F1	F0	
lw									
sw									
beq									
add									
sub									
and									
or									
slt									

5. Design of ALU Control Unit (2/2)

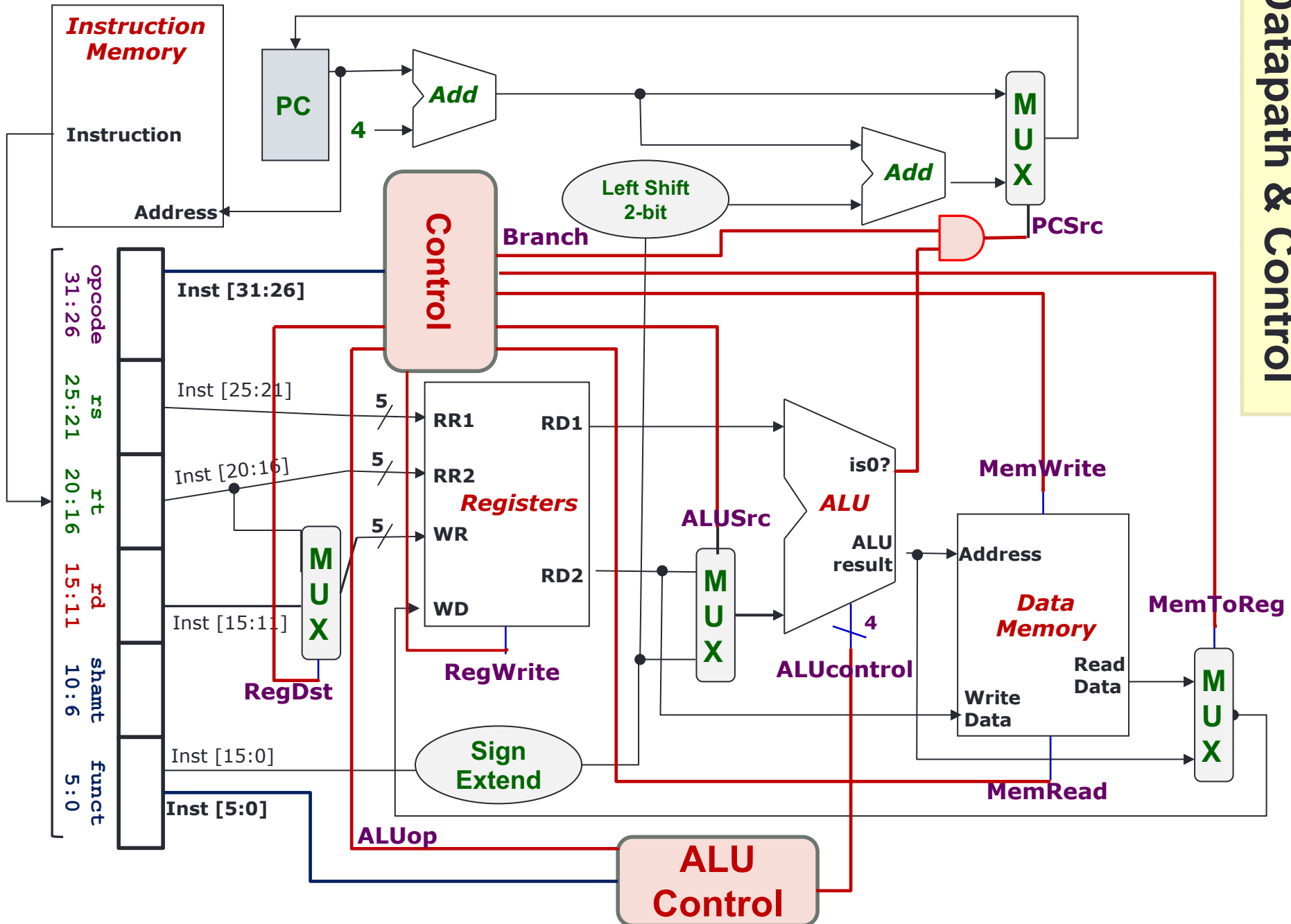
- Simple combinational logic

Note: We will revisit this when we cover combinational circuits later.



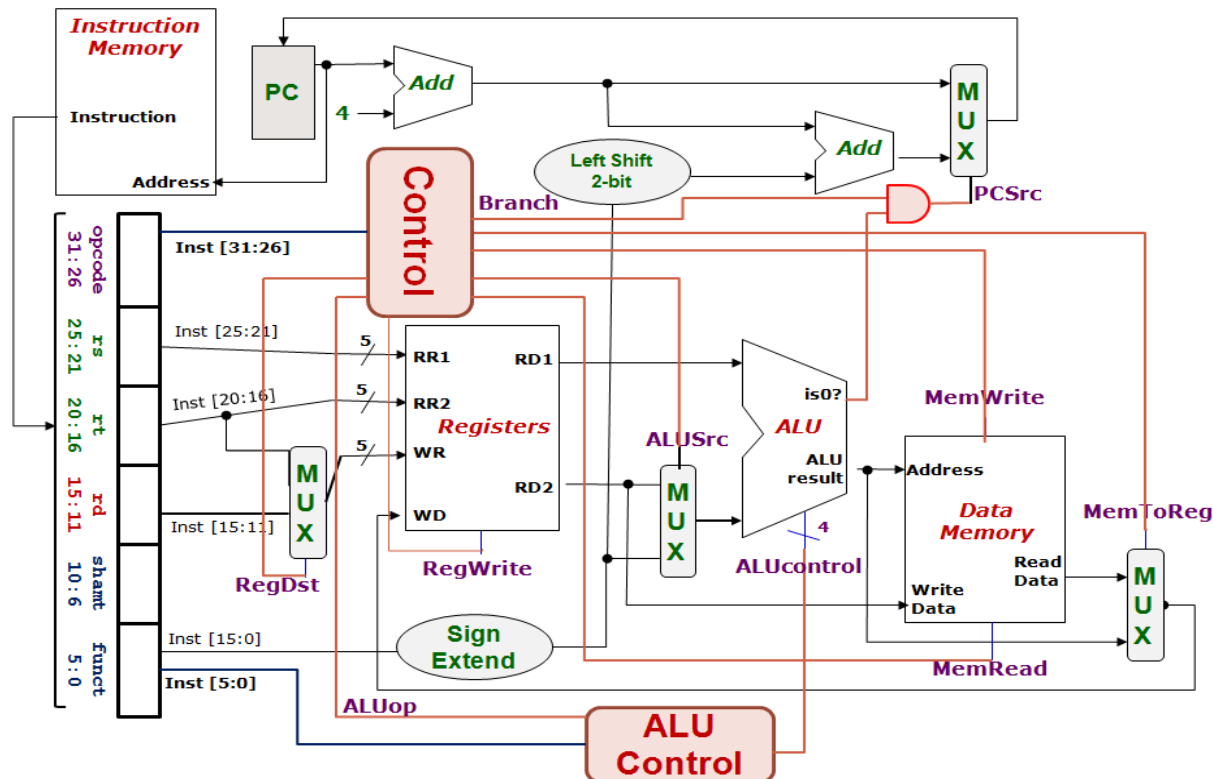
5. Finale: Control Design

- We have now considered all individual signals and their expected values
 - ➔ Ready to design the controller itself
- Typical digital design steps:
 - Fill in truth table
 - **Input:** Opcode
 - **Output:** Various control signals as discussed
 - Derive simplified expression for each signal



5. Control Design: Outputs

	RegDst	ALUSrc	MemTo Reg	Reg Write	Mem Read	Mem Write	Branch	ALUop	
								op1	op0
R-type	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

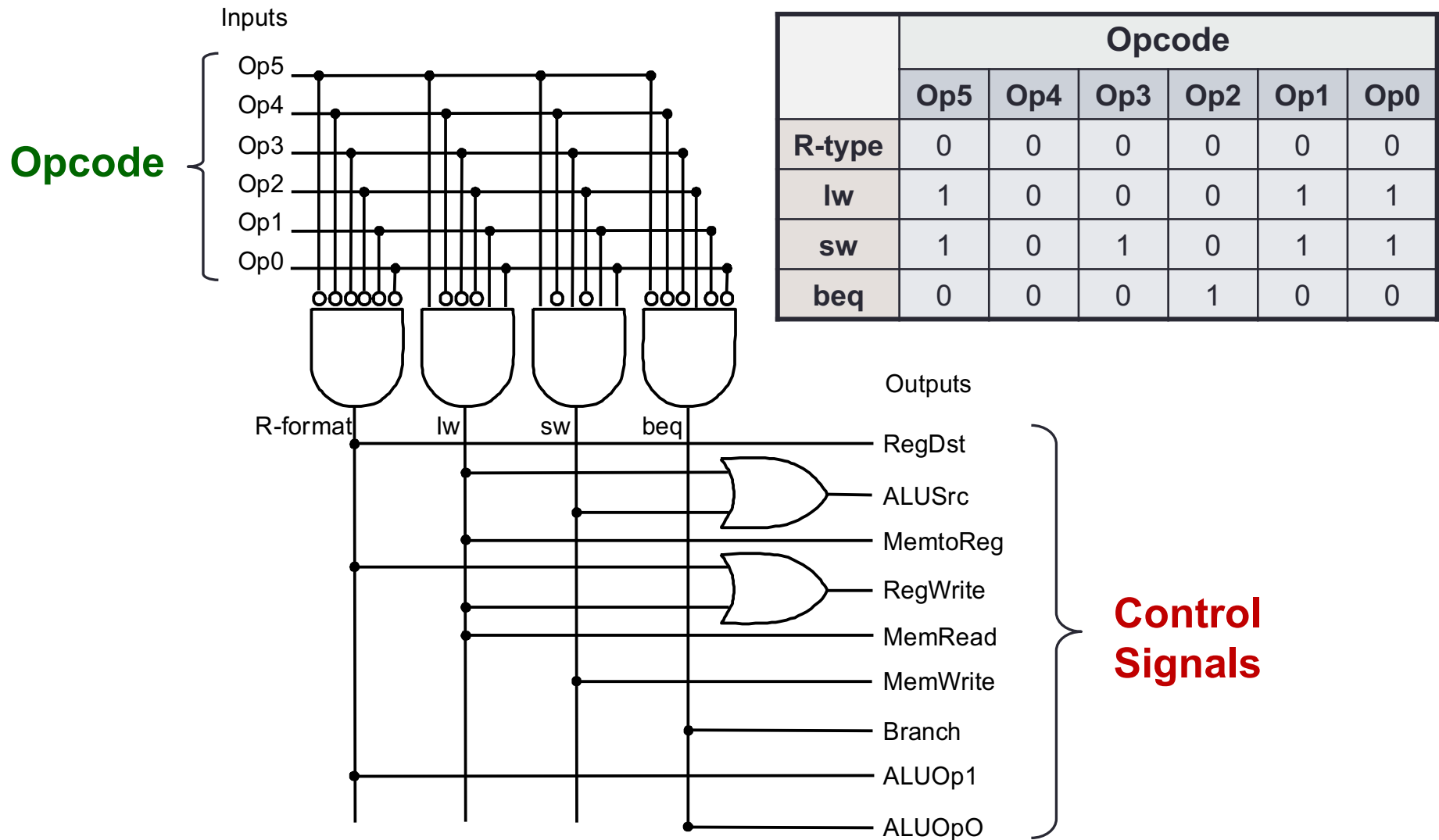


5. Control Design: Inputs

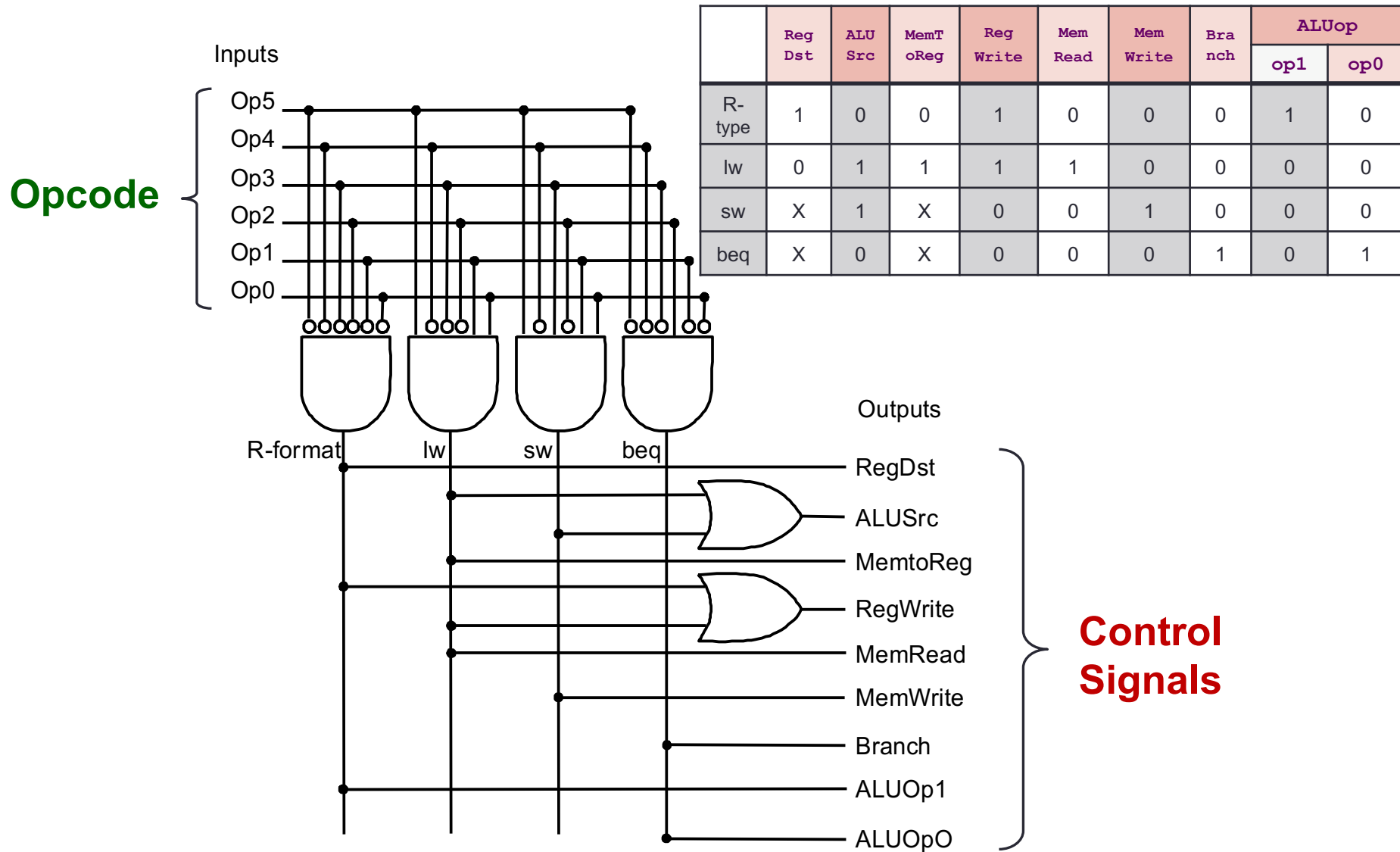
	Opcode (Op[5:0] == Inst[31:26])						
	Op5	Op4	Op3	Op2	Op1	Op0	Value in Hexadecimal
R-type	0	0	0	0	0	0	0
lw	1	0	0	0	1	1	23
sw	1	0	1	0	1	1	2B
beq	0	0	0	1	0	0	4

- With the input (opcode) and output (control signals), let's design the circuit

5. Combinational Circuit Implementation

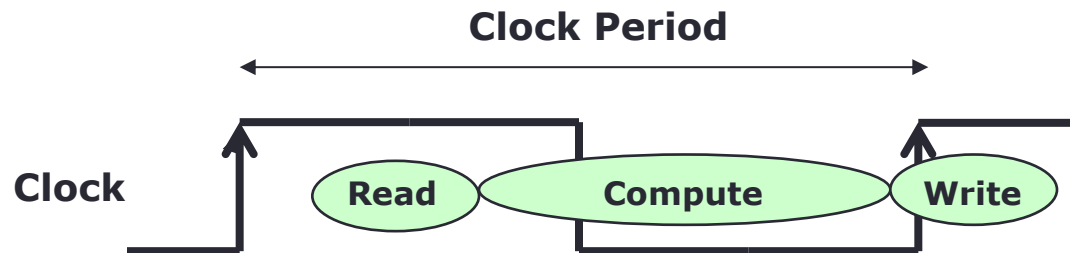


5. Combinational Circuit Implementation



6. Big Picture: Instruction Execution

- Instruction Execution =
 1. Read contents of one or more storage elements (register/memory)
 2. Perform computation through some combinational logic
 3. Write results to one or more storage elements (register/memory)
- All these performed **within a clock period**



Don't want to read a storage element when it is being written.

6. Single Cycle Implementation: Shortcoming

- Calculate cycle time assuming negligible delays: memory (2ns), ALU/adders (2ns), register file access (1ns)

Instruction	Inst Mem	Reg read	ALU	Data Mem	Reg write	Total
ALU	2	1	2		1	6
lw	2	1	2	2	1	8
sw	2	1	2	2		7
beq	2	1	2			5

- All instructions take as much time as the slowest one (i.e., load)
 - ➔ Long cycle time for each instruction

6. Solution #1: Multicycle Implementation

- Break up the instructions into execution steps:
 1. Instruction fetch
 2. Instruction decode and register read
 3. ALU operation
 4. Memory read/write
 5. Register write
- Each execution step **takes one clock cycle**
 - ➔ **Cycle time is much shorter, i.e., clock frequency is much higher**
- Instructions take variable number of clock cycles to complete execution
- Not covered in class:
 - See Section 5.5 of COD if interested

6. Solution #2: Pipelining

- Break up the instructions into execution steps one per clock cycle
- Allow different instructions to be in different execution steps simultaneously
- Covered in a later lecture

Summary

- A very simple implementation of MIPS datapath and control for a subset of its instructions
- Concepts:
 - An instruction executes in a single clock cycle
 - Read storage elements, compute, write to storage elements
 - Datapath is shared among different instructions types using MUXs and control signals
 - Control signals are generated from the machine language encoding of instructions

Reading

- **The Processor: Datapath and Control**
 - COD Chapter 5 Sections 5.4 (3rd edition)
 - COD Chapter 4 Sections 4.4 (4th edition)
- **Exploration:**
 - ALU design and implementation:
 - 4th edition (MIPS): Appendix C
 - <http://cs.nyu.edu/courses/fall11/CSCI-UA.0436-001/class-notes.html>



End of File