Computer Architectures

MIPS Control

Zhu Wen-Jun

Institute of Computer

Outline

- Datapath Review
- Controller Implementation

Great Idea #1: Levels of Representation/Interpretation

Higher-Level Language Program (e.g. C)

Compiler

Assembly Language Program (e.g. MIPS)

Assembler

Machine Language Program (MIPS)

Machine Interpretation

Hardware Architecture Description (e.g. block diagrams)

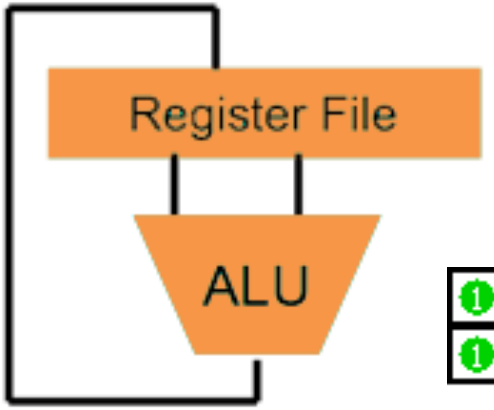
Architecture Implementation

Logic Circuit Description (Circuit Schematic Diagrams)

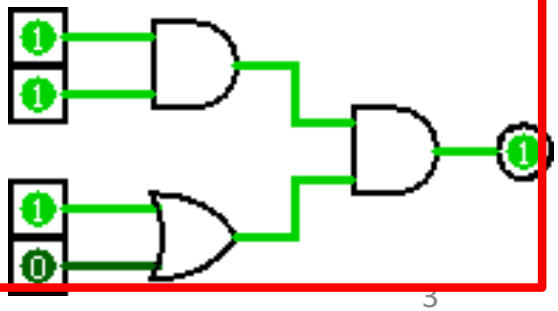
```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

```
lw   $t0, 0($2)
lw   $t1, 4($2)
sw   $t1, 0($2)
sw   $t0, 4($2)
```

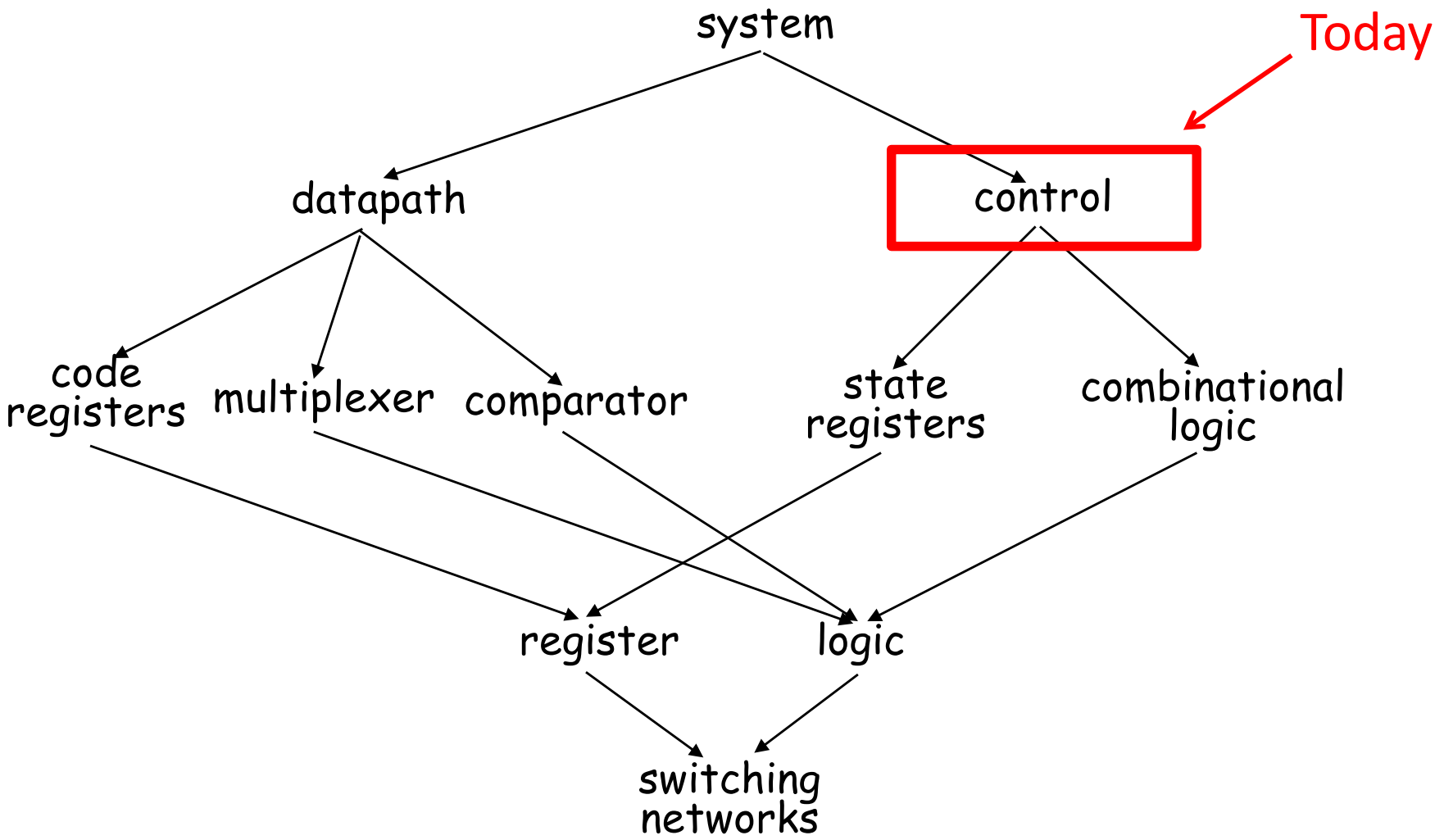
```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```



We are here



Hardware Design Hierarchy



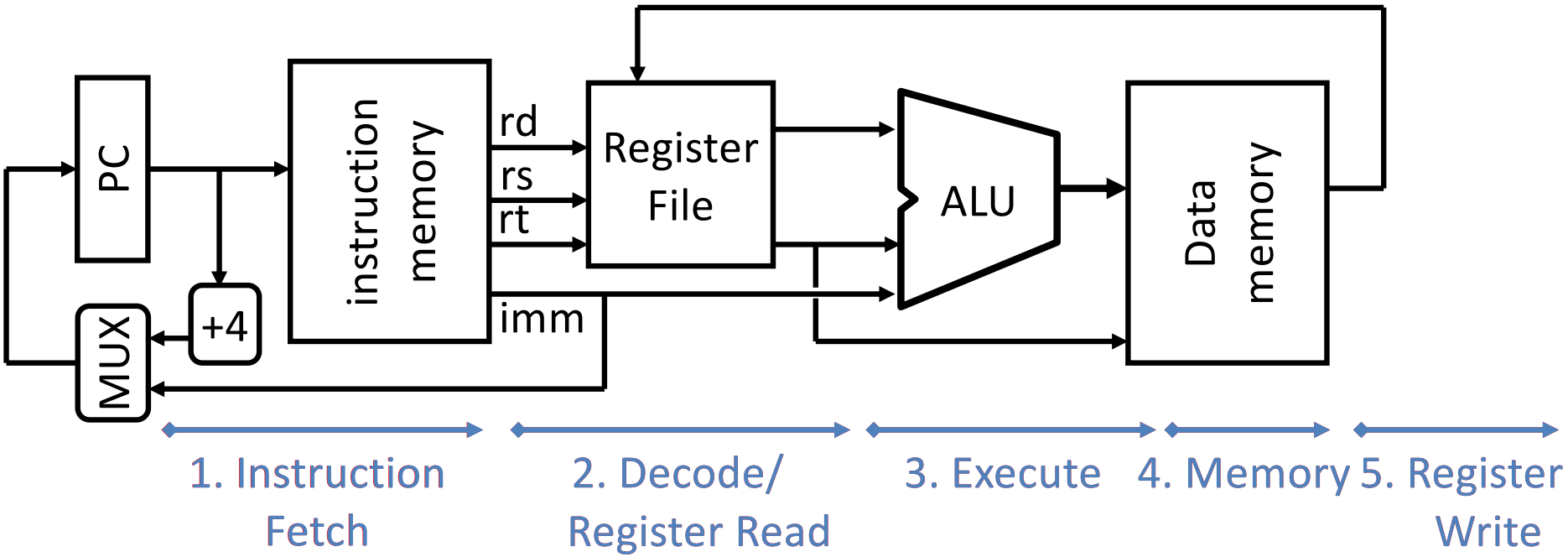
Outline

- Datapath Review
- Controller Implementation

Datapath Review

- Part of the processor; the *hardware* necessary to perform *all* operations required
  - Depends on exact ISA, RTL of instructions
- Major components:
  - PC and Register File (RegFile holds registers)
  - Instruction and Data Memory
  - ALU for operations (on two operands)
  - Extender (sign/zero extend)

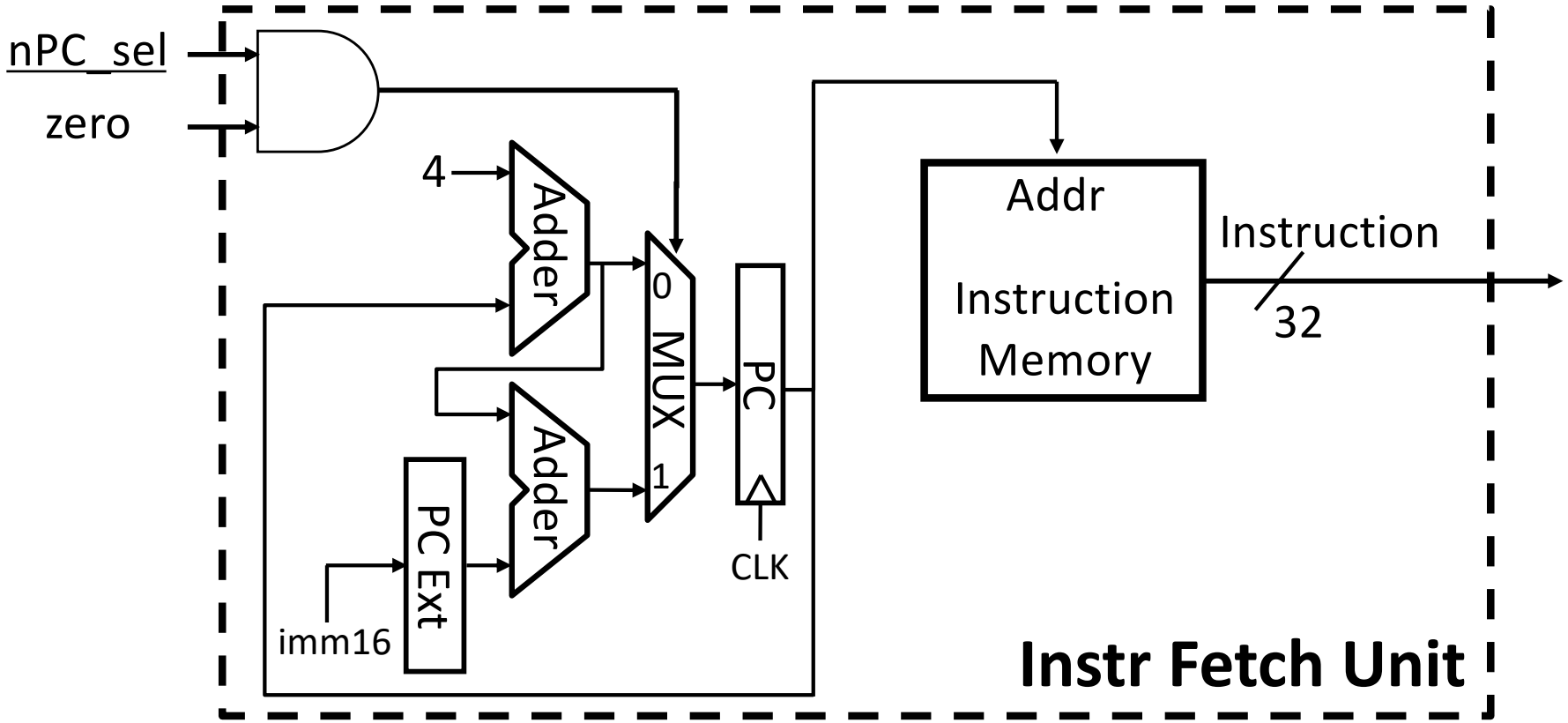
Five Stages of the Datapath



Datapath and Control

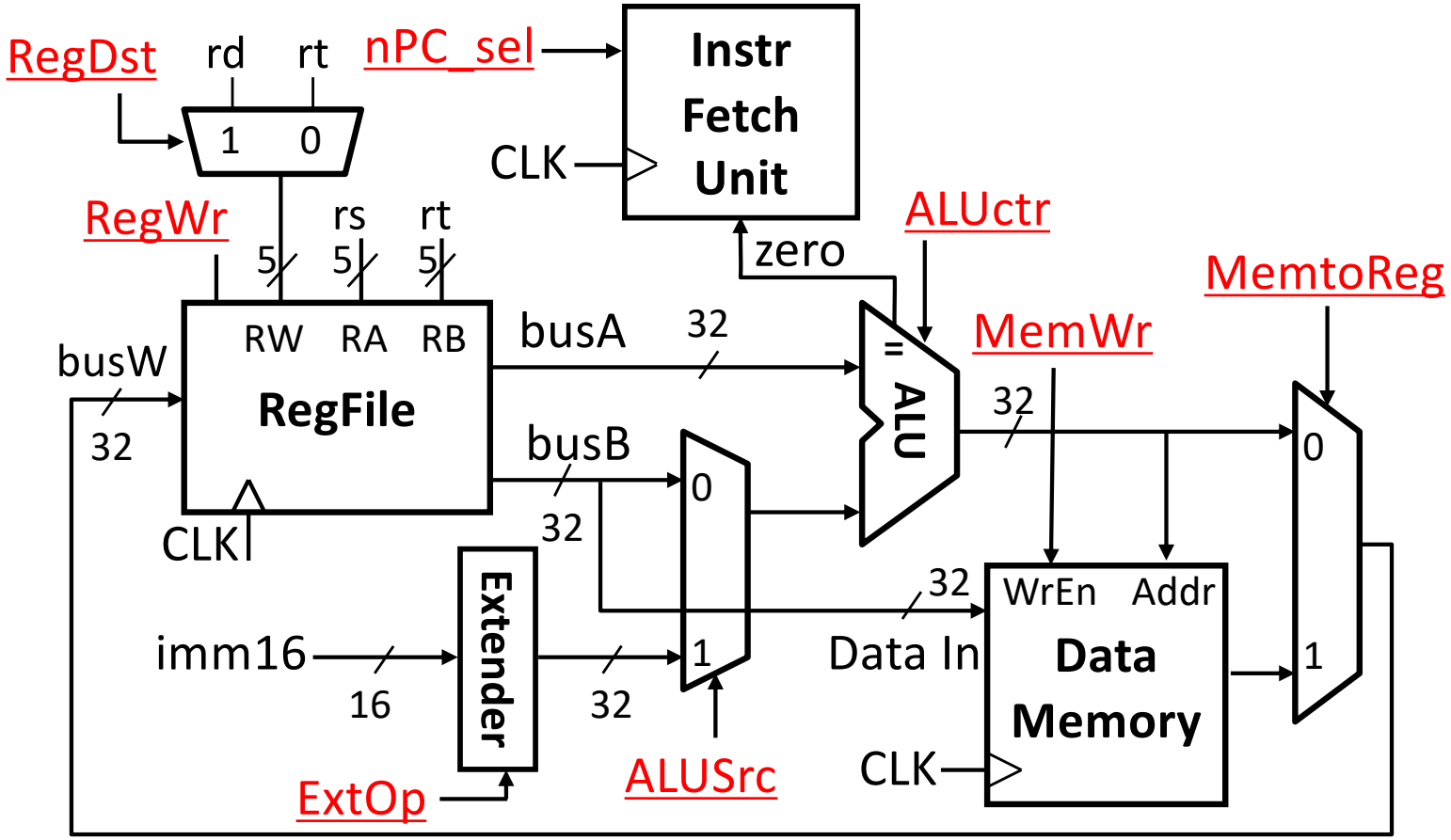
- Route parts of datapath based on ISA needs
  - Add MUXes to select from multiple inputs
  - Add *control signals* for component inputs and MUXes
- Analyze control signals
  - How wide does each one need to be?
  - For each instruction, assign appropriate value for correct routing

MIPS-lite Instruction Fetch



MIPS-lite Datapath Control Signals

- **ExtOp:** 0 → "zero"; 1 → "sign"
- **ALUsrc:** 0 → busB; 1 → imm16
- **ALUctr:** "ADD", "SUB", "OR"
- **nPC_sel:** 0 → +4; 1 → branch
- **MemWr:** 1 → write memory
- **MemtoReg:** 0 → ALU; 1 → Mem
- **RegDst:** 0 → "rt"; 1 → "rd"
- **RegWr:** 1 → write register

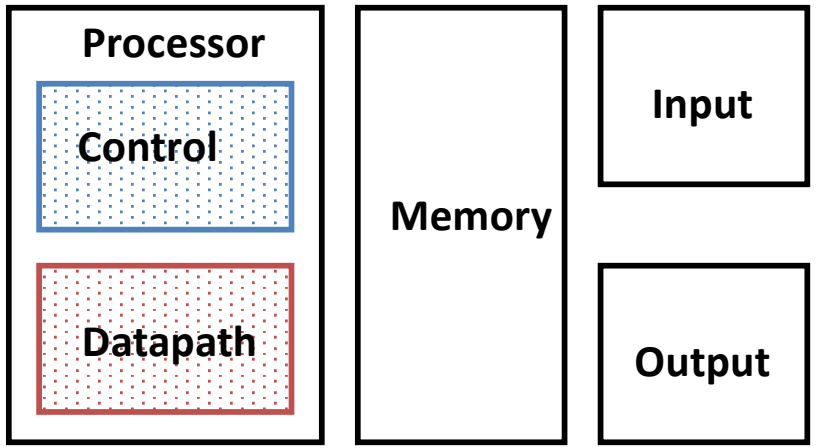


Outline

- Datapath Review
- Controller Implementation

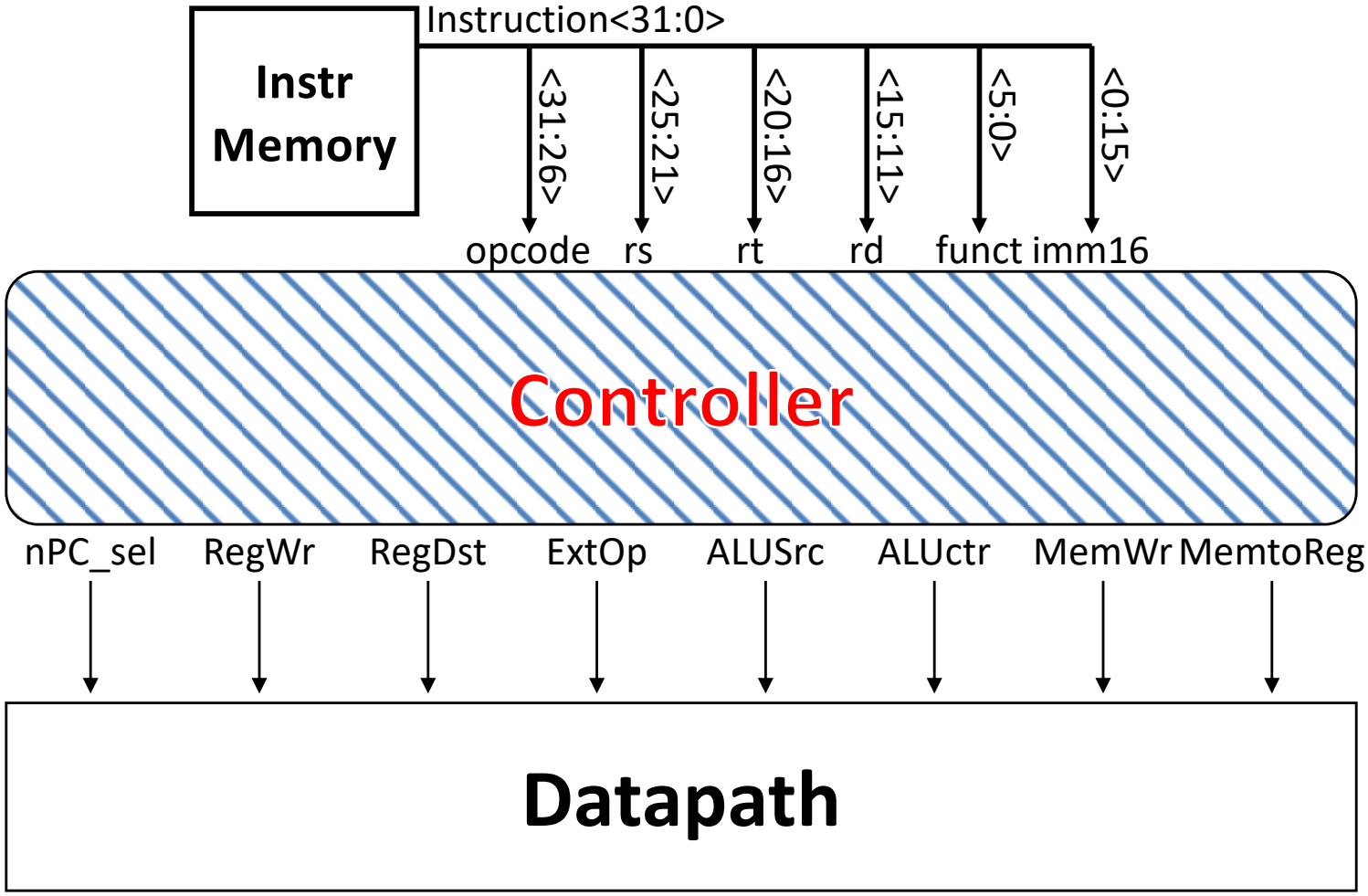
Processor Design Process

- Five steps to design a processor:
  1. Analyze instruction set → datapath requirements
  2. Select set of datapath components & establish clock methodology
  3. Assemble datapath meeting the requirements
  4. Analyze implementation of each instruction to determine setting of control points that effects the register transfer
  5. Assemble the control logic
     - Formulate Logic Equations
     - Design Circuits



Now

Purpose of Control



MIPS-lite Instruction RTL

Instr	**Register Transfer Language**
addu	R[rd]←R[rs]+R[rt]; PC←PC+4
subu	R[rd]←R[rs]−R[rt]; PC←PC+4
ori	R[rt]←R[rs]+zero_ext(imm16); PC←PC+4
lw	R[rt]←MEM[R[rs]+sign_ext(imm16)]; PC←PC+4
sw	MEM[R[rs]+sign_ext(imm16)]←R[rs]; PC←PC+4
beq	if(R[rs]==R[rt]) then PC←PC+4+[sign_ext(imm16)||00] else PC←PC+4

MIPS-lite Control Signals (1/2)

Instr	**Control Signals**
addu	ALUsrc=RegB, ALUctr="ADD", RegDst=rd, RegWr, nPC_sel="+4"
subu	ALUsrc=RegB, ALUctr="SUB", RegDst=rd, RegWr, nPC_sel="+4"
ori	ALUsrc=Imm, ALUctr="OR", RegDst=rt, RegWr, ExtOp="Zero", nPC_sel="+4"
lw	ALUsrc=Imm, ALUctr="ADD", RegDst=rt, RegWr, ExtOp="Sign", MemtoReg, nPC_sel="+4"
sw	ALUsrc=Imm, ALUctr="ADD", MemWr, ExtOp="Sign", nPC_sel="+4"
beq	ALUsrc=RegB, ALUctr="SUB", nPC_sel="Br"

MIPS-lite Control Signals (2/2)

See MIPS Green Sheet

		add	sub	ori	lw	sw	beq
	func	10 0000	10 0010	**n/a**			
	op	00 0000	00 0000	00 1101	10 0011	10 1011	00 0100
Control Signals	**RegDst**	1	1	0	0	X	X
	ALUSrc	0	0	1	1	1	0
	MemtoReg	0	0	0	1	X	X
	RegWrite	1	1	1	1	0	0
	MemWrite	0	0	0	0	1	0
	nPC_sel	0	0	0	0	0	1
	ExtOp	X	X	0	1	1	X
	ALUctr<2:0>	Add	Subtract	Or	Add	Add	Subtract

All Supported Instructions

- Now how do we implement this table with CL?

Generating Boolean Expressions

- **Idea #1:** Treat instruction names as Boolean variables!
  - `opcode` and `funct` bits are available to us
  - Use gates to generate signals that are 1 when it is a particular instruction and 0 otherwise
- Examples:

```
beq = op[5]'·op[4]'·op[3]'·op[2]·op[1]'·op[0]'

Rtype = op[5]'·op[4]'·op[3]'·op[2]'·op[1]'·op[0]'

add = Rtype·funct[5]·funct[4]'·funct[3]'
      ·funct[2]'·funct[1]'·funct[0]'
```

Generating Boolean Expressions

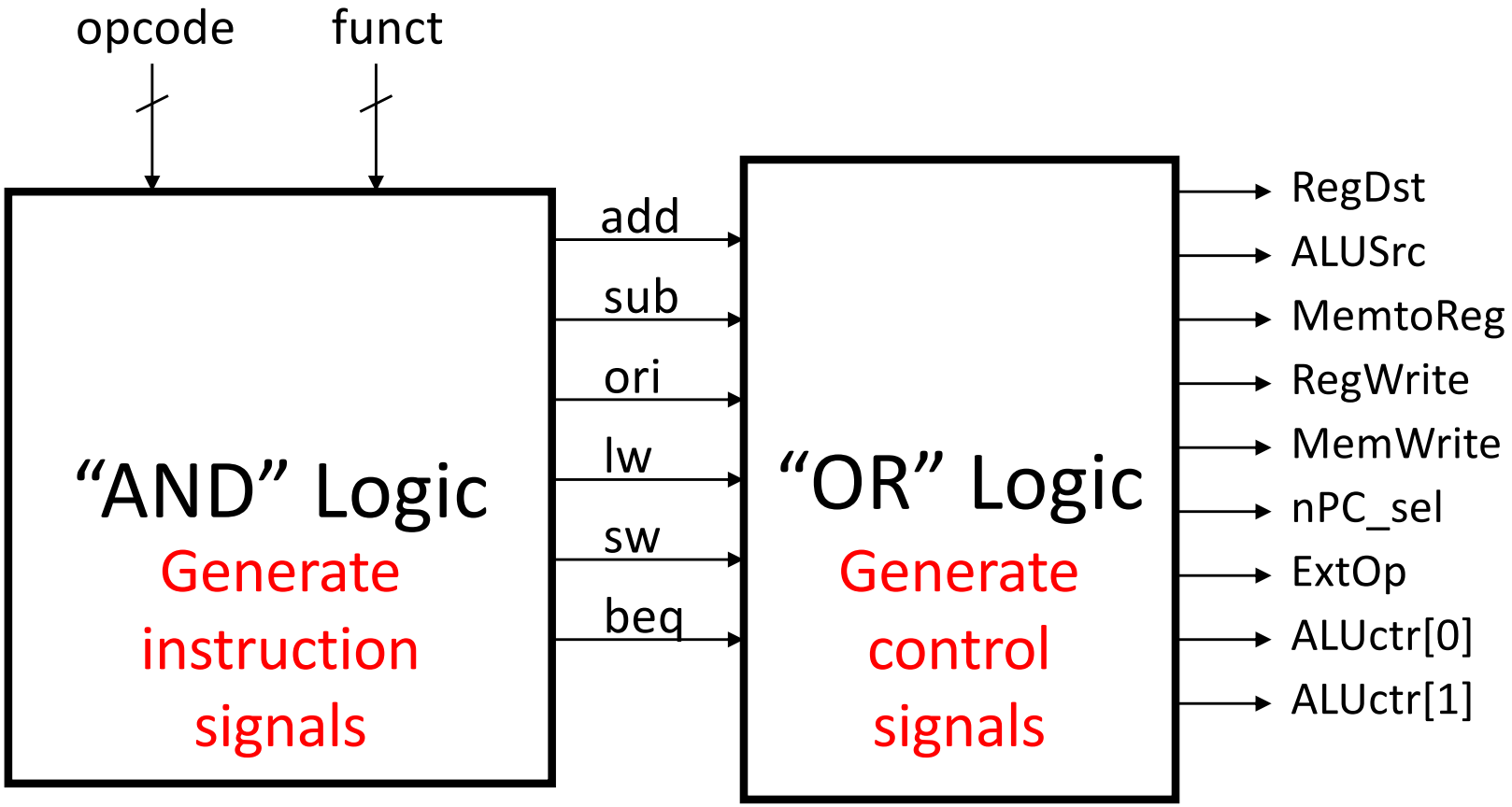
- **Idea #2:** Use instruction variables to generate control signals
  - Make each control signal the combination of all instructions that need that signal to be a 1
- Examples:
  - `MemWrite = sw`
  - `RegWrite = add + sub + ori + lw`

Read from row of table

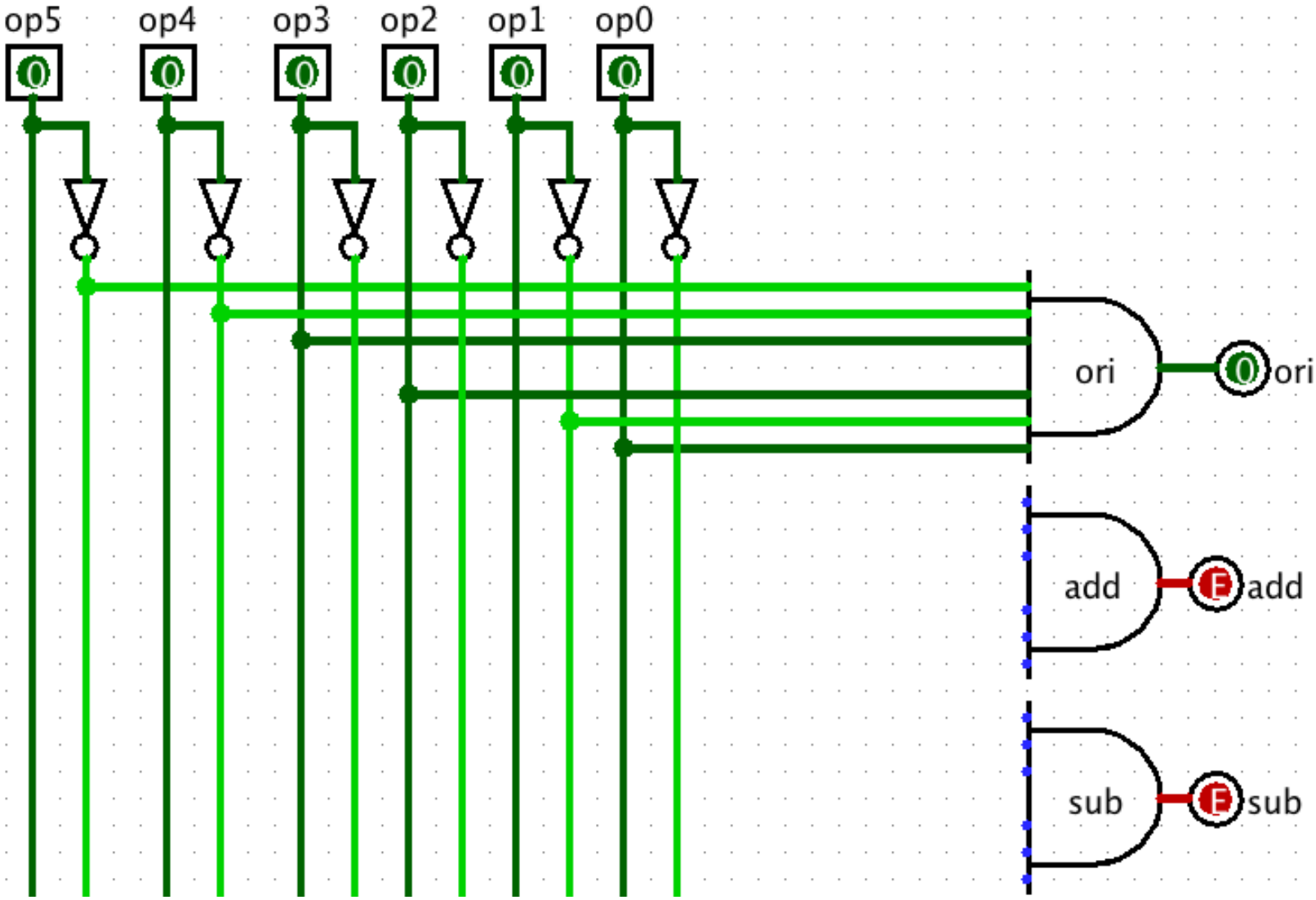
- What about don't cares (X's)?
  - Want simpler expressions; set to 0!

Controller Implementation

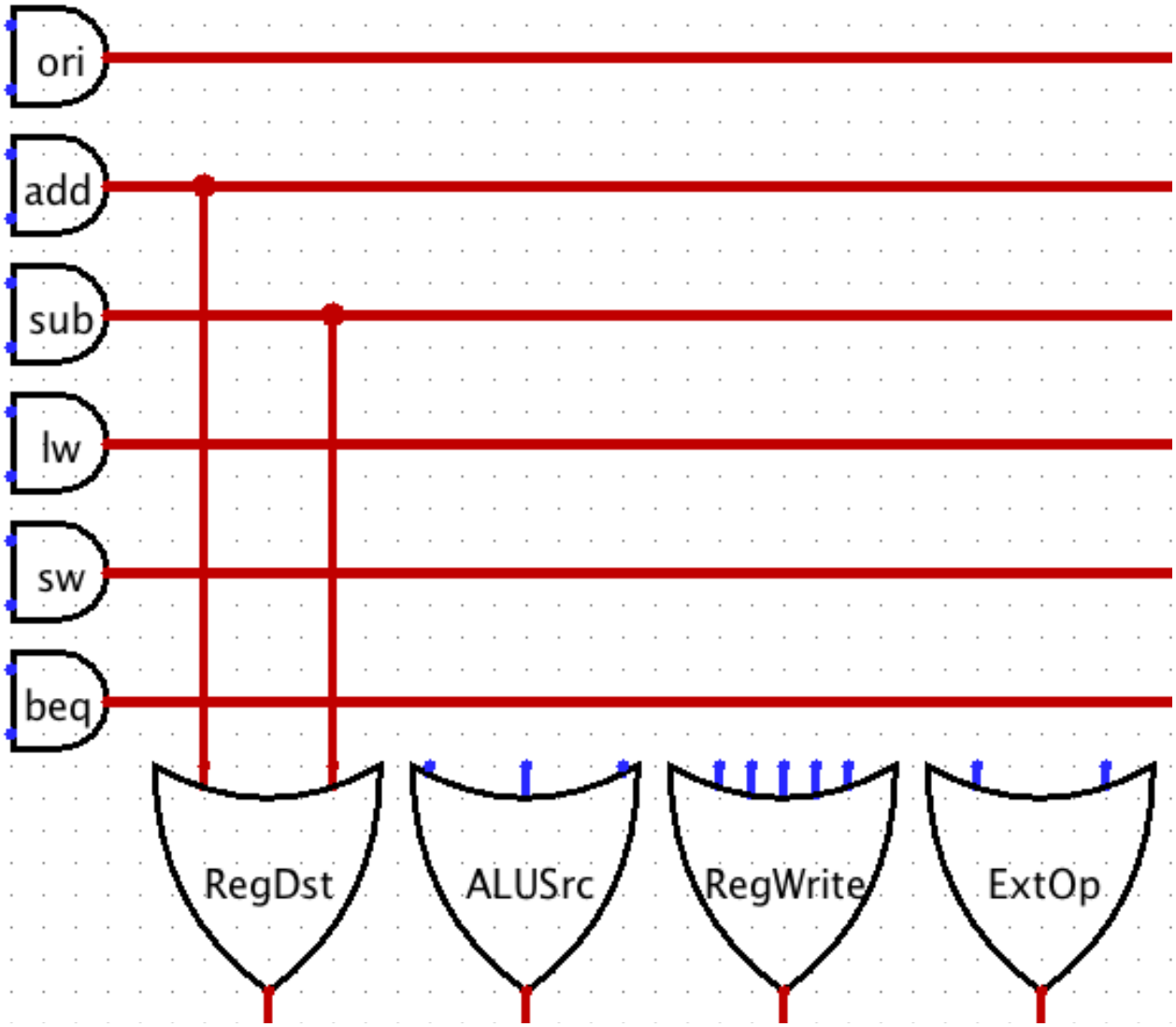
- Use these two ideas to design controller:



AND Control Logic in Logisim



OR Control Logic in Logisim



Great Idea #1: Levels of Representation/Interpretation