计算机学院课程

计算机组成

MIPS控制

高小鹏

北京航空航天大学计算机学院
系统结构研究所

提纲

- 内容主要取材
  - CS617的21讲
- 快速回顾数据通路
- 实现控制
- 时钟方法

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)

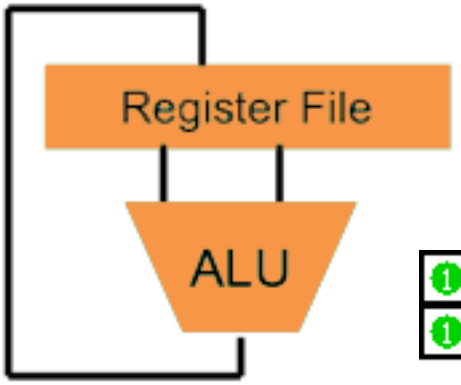
Architecture Implementation

Logic Circuit Description (Circuit Schematic Diagrams)

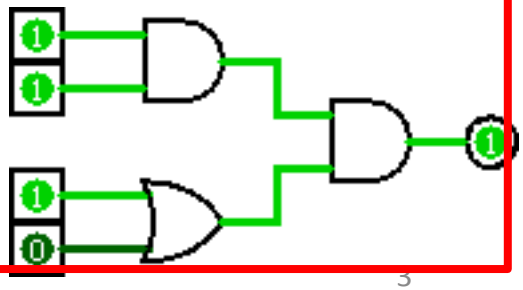
```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

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

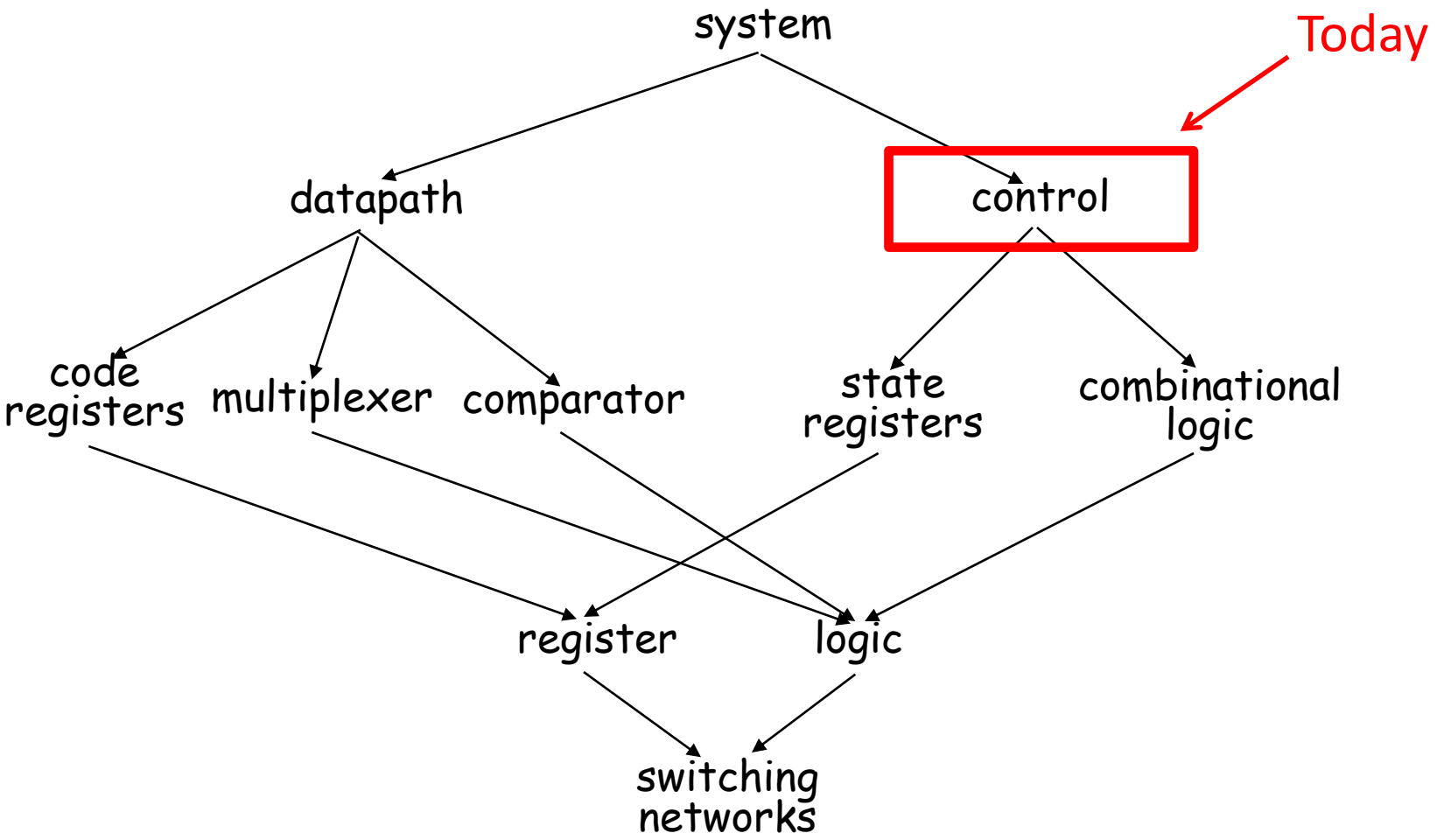
```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```



We are here



Hardware Design Hierarchy



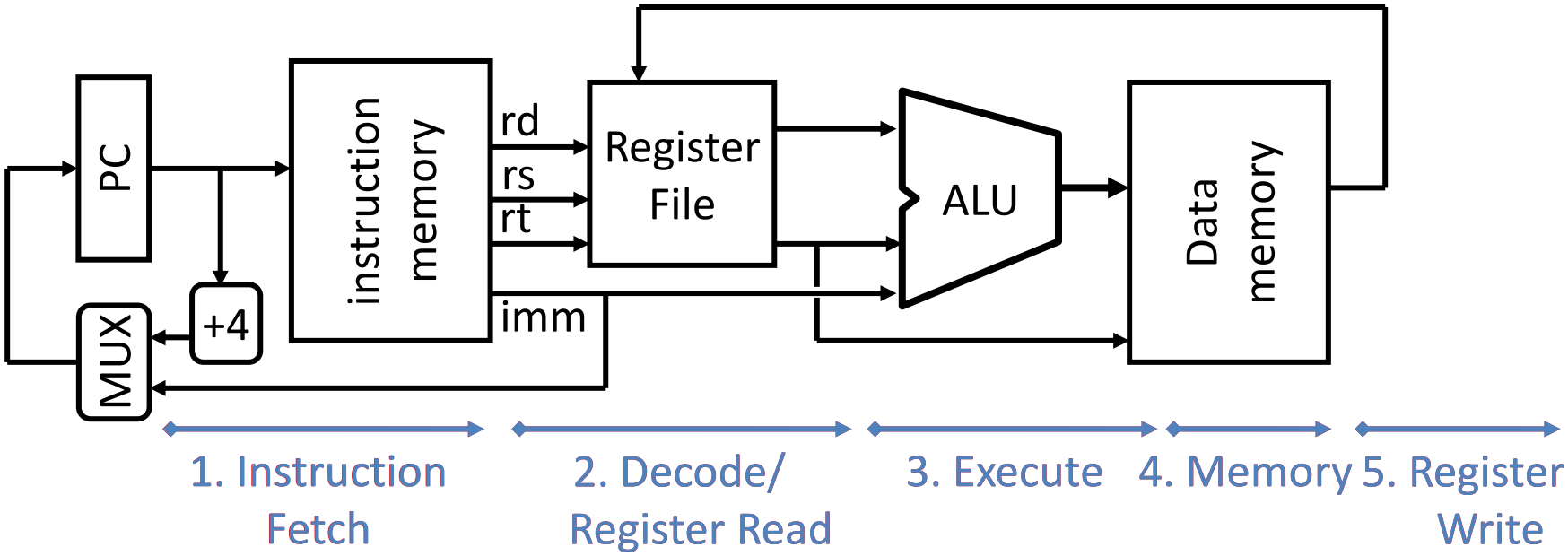
提纲

- 内容主要取材
  - CS617的21讲
- 快速回顾数据通路
- 实现控制
- 时钟方法

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)

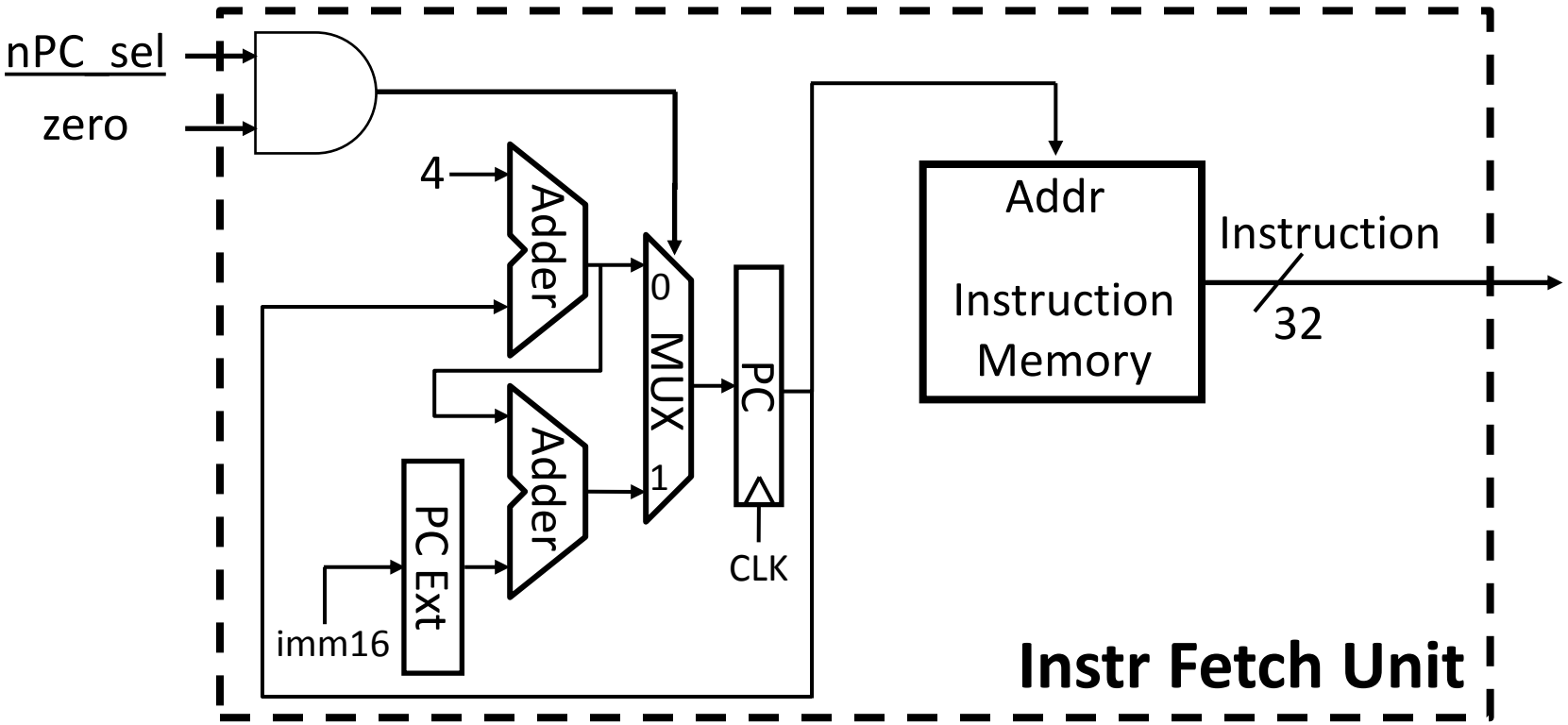
Five Stages of the Datapath



Datapath and Control

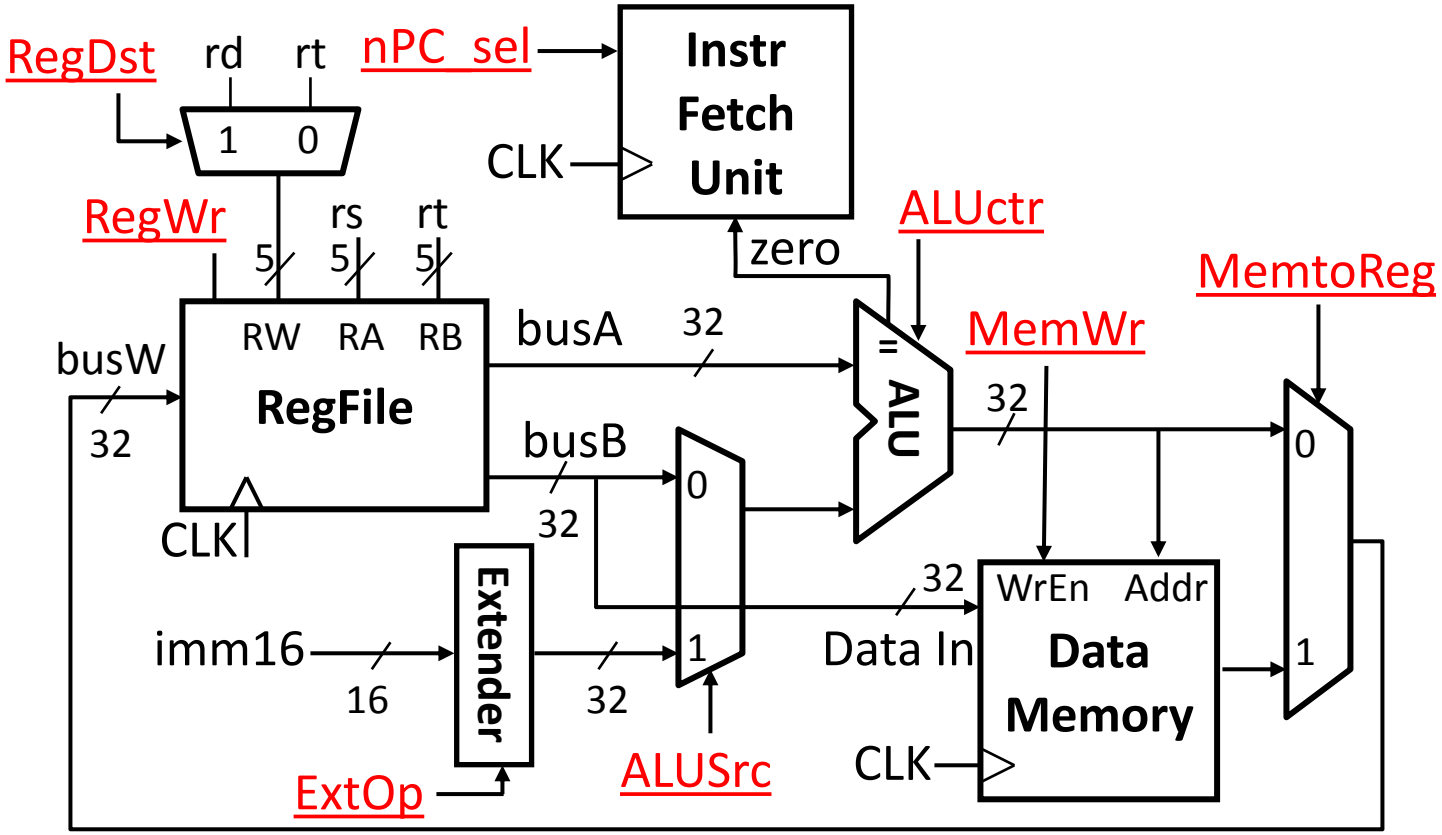
- Route parts of datapath based on ISA needs
  - Add MUXes to select from multiple inputs
  - Add *control signals* for component inputs and MUXes
- Analyze control signals
  - How wide does each one need to be?
  - For each instruction, assign appropriate value for correct routing

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

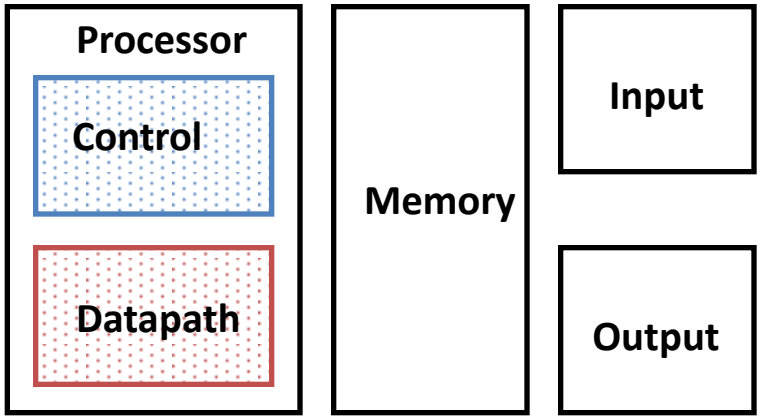


提纲

- 内容主要取材
  - CS617的21讲
- 快速回顾数据通路
- 实现控制
- 时钟方法

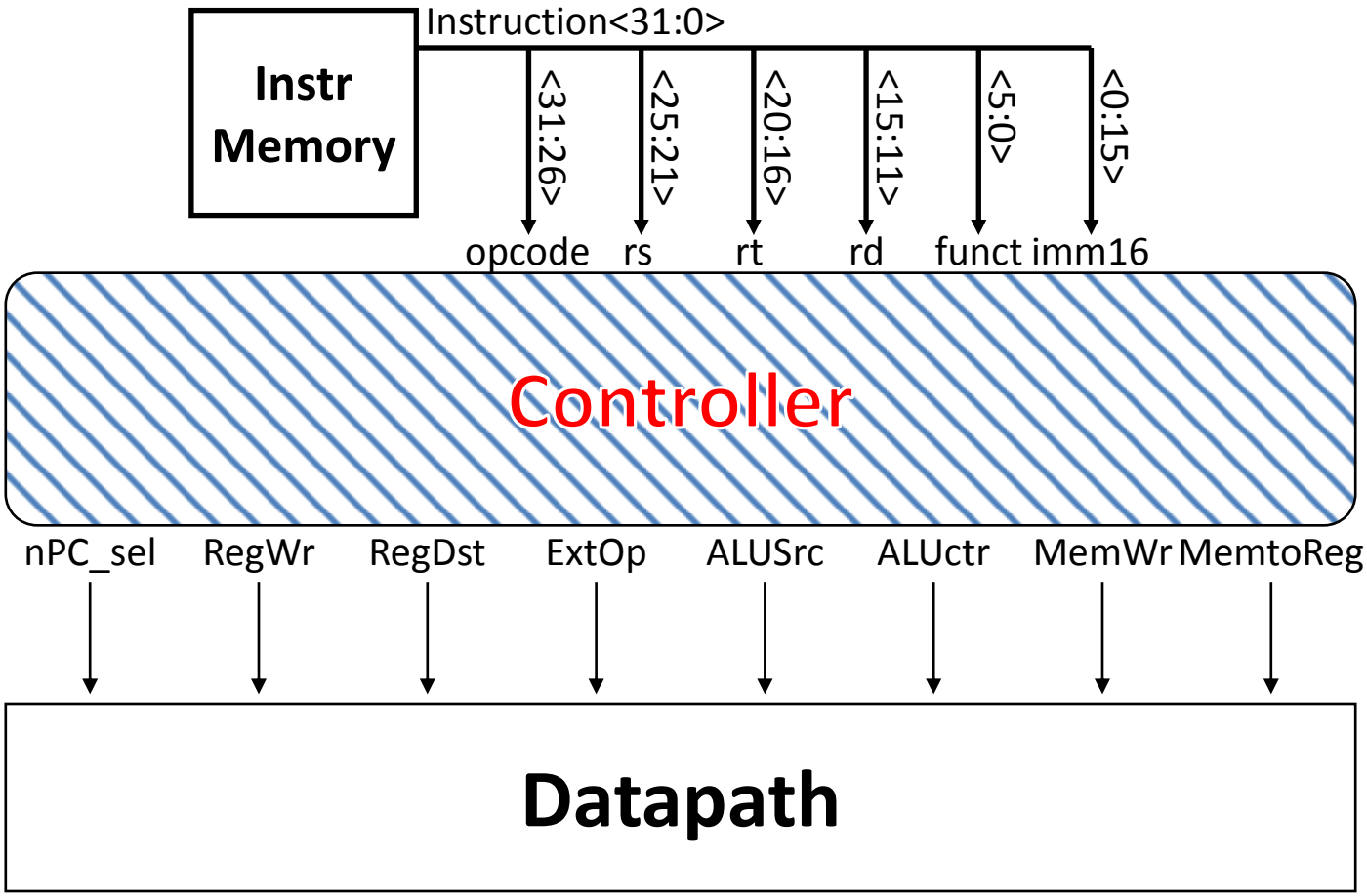
Processor Design Process

- 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

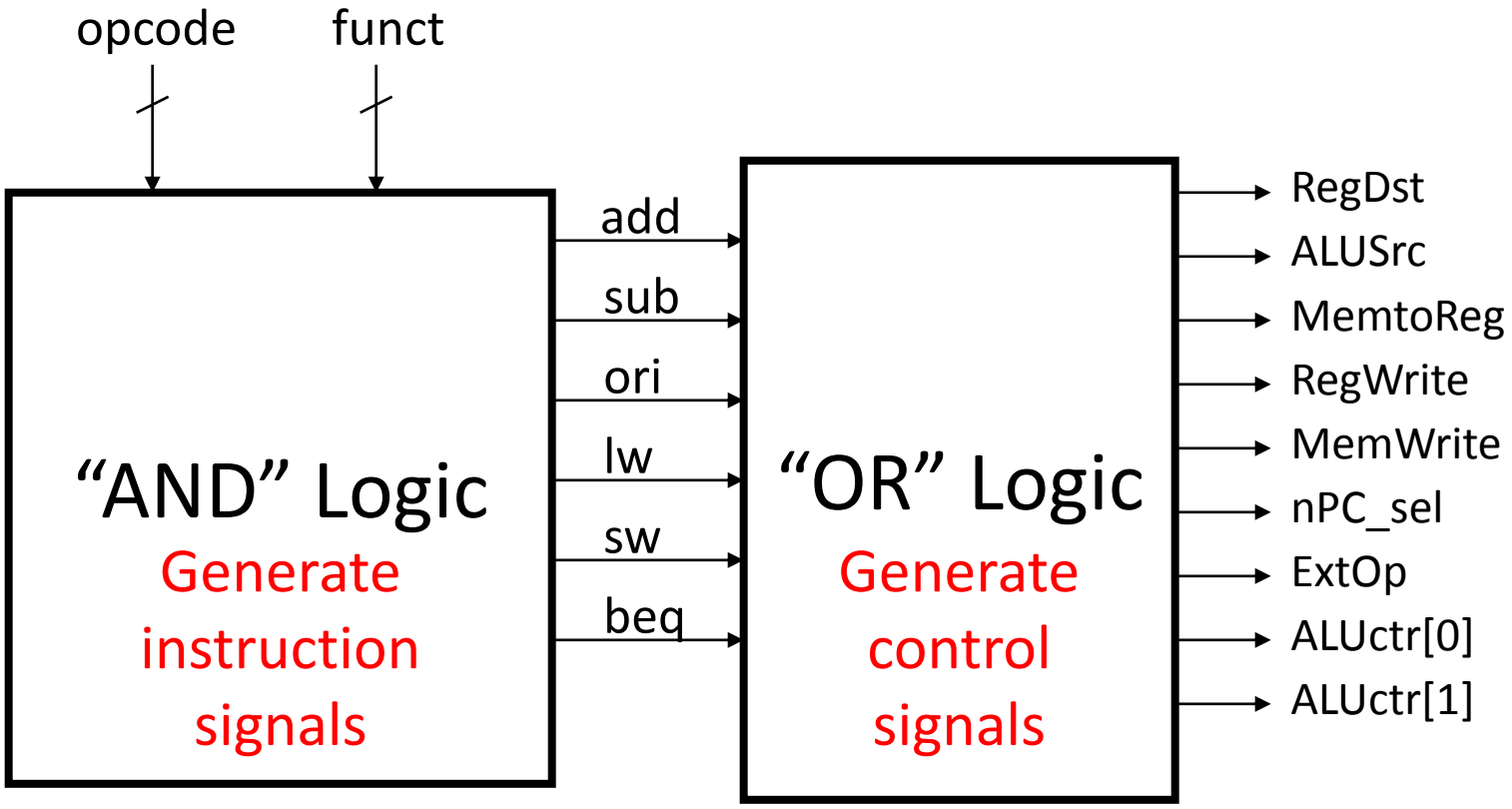
- **Idea #2:** Use instruction variables to generate control signals
  - Make each control signal the combination of all instructions that need that signal to be a 1
- Examples:
  - `MemWrite = sw`
  - `RegWrite = add + sub + ori + lw`

Read from row of table

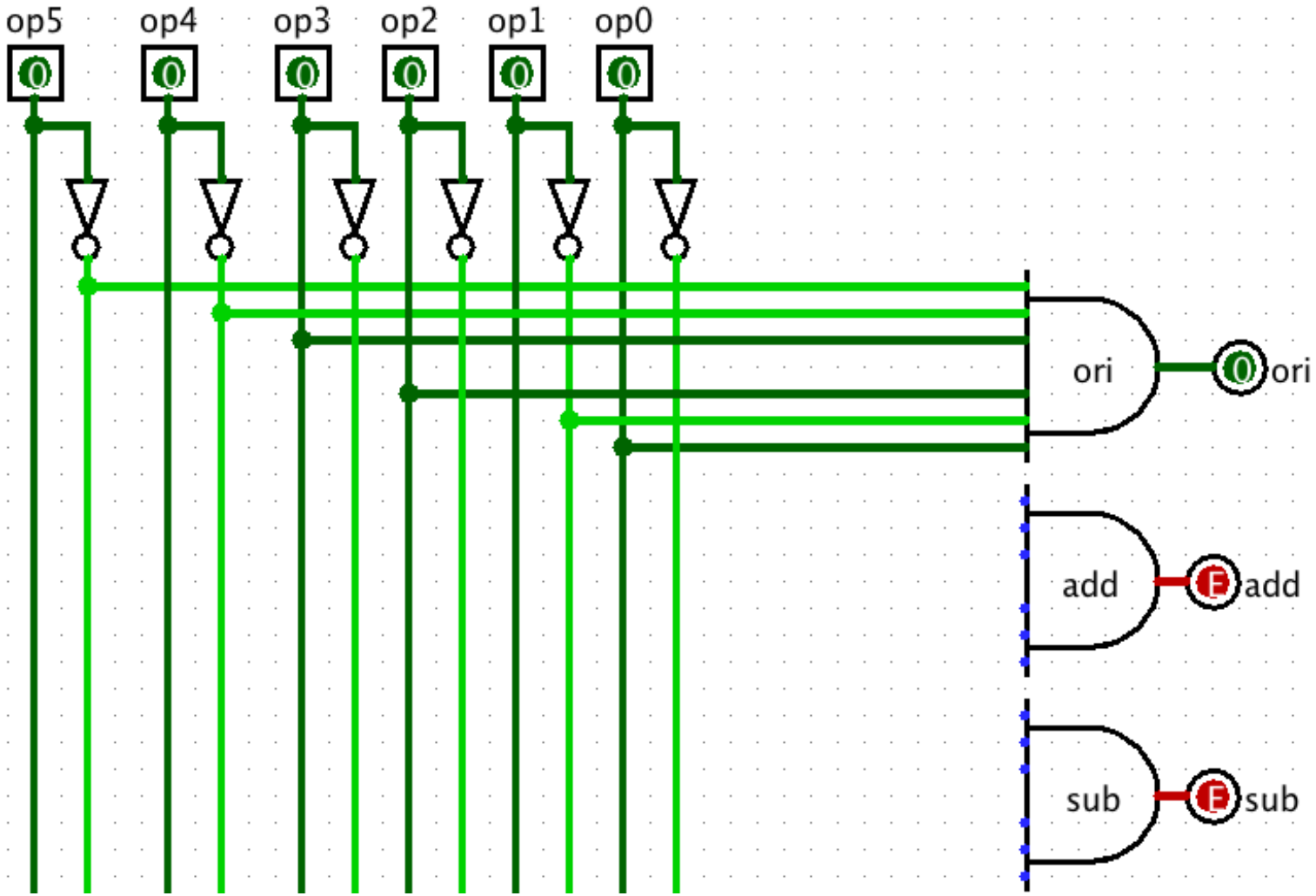
- What about don't cares (X's)?
  - Want simpler expressions; set to 0!

Controller Implementation

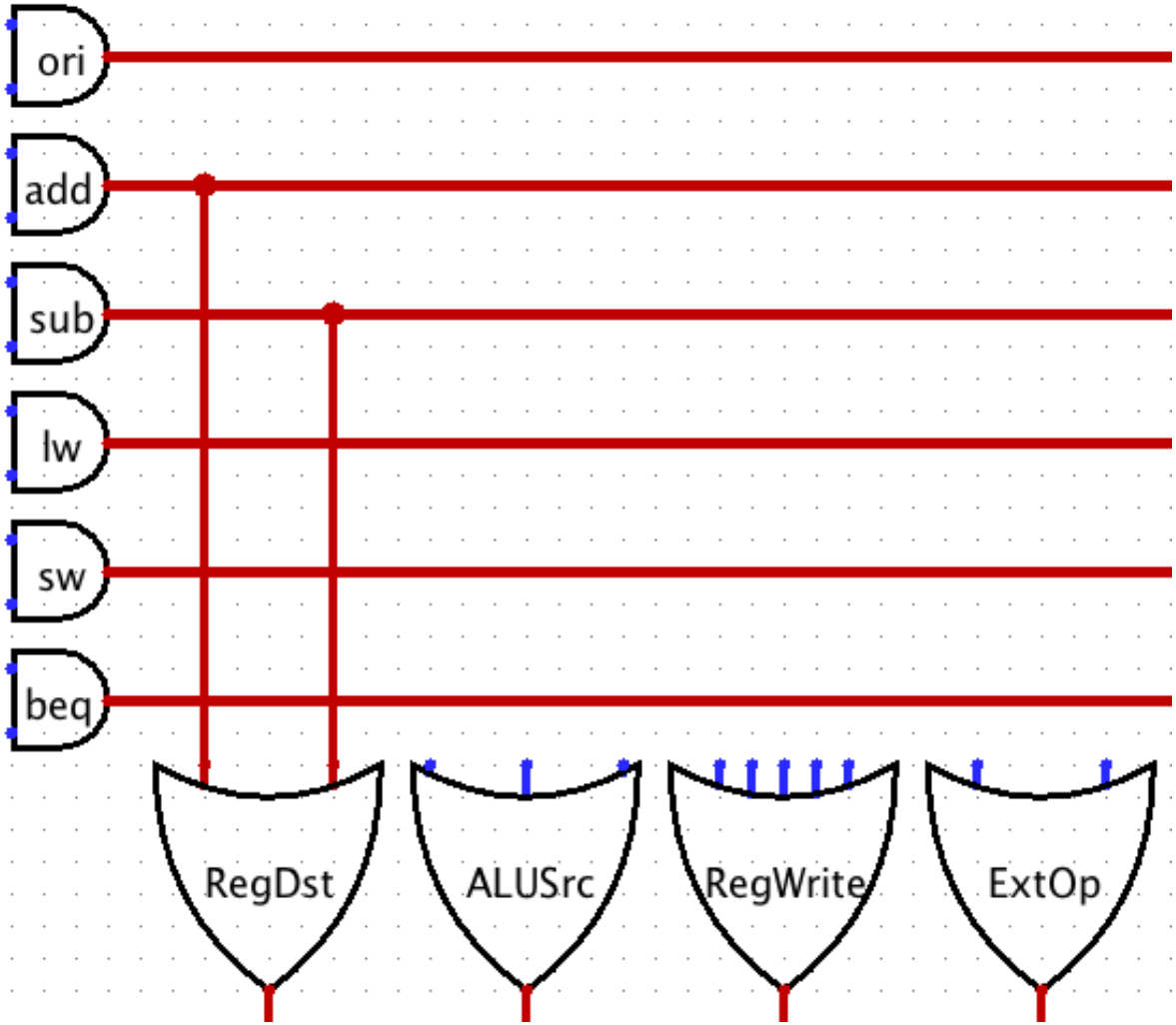
- Use these two ideas to design controller:



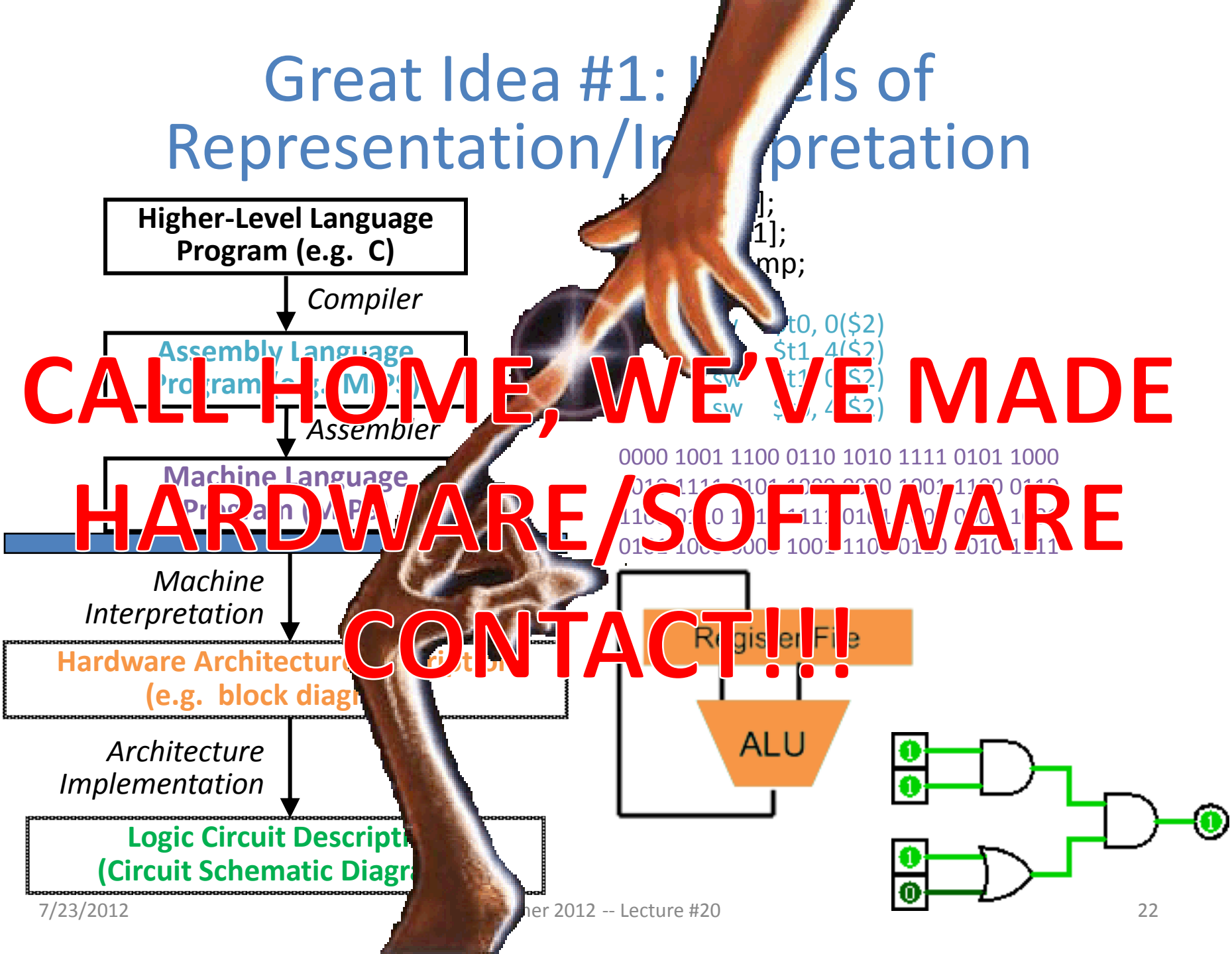
AND Control Logic in Logisim



OR Control Logic in Logisim



Great Idea #1: Levels of Representation/Interpretation



提纲

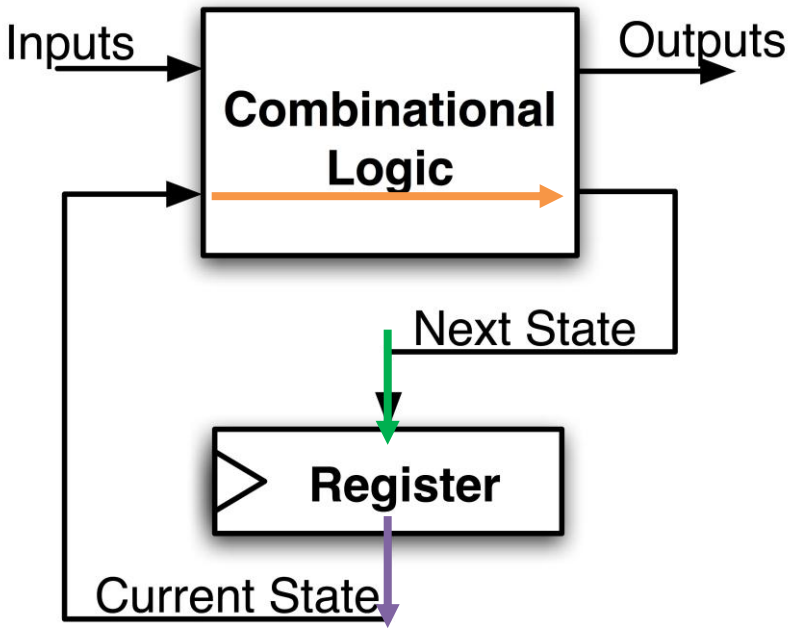
- 内容主要取材
  - CS617的21讲
- 快速回顾数据通路
- 实现控制
- 时钟方法

Register Timing Terms (Review)

- *Setup Time:* how long the input must be stable *before* the CLK trigger for proper input read
- *Hold Time:* how long the input must be stable *after* the CLK trigger for proper input read
- *"CLK-to-Q" Delay:* how long it takes the output to change, measured from the CLK trigger

Maximum Clock Frequency

- What is the max frequency of this circuit?
  - Limited by how much time needed to get correct Next State to Register

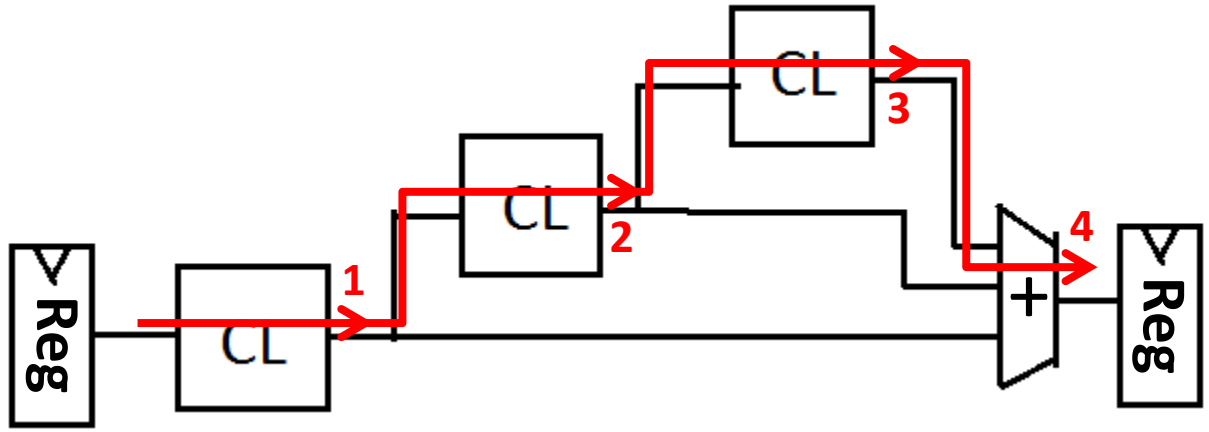


Max Delay = Setup Time
+ CLK-to-Q Delay
+ CL Delay

Max Freq = 1/Max Delay

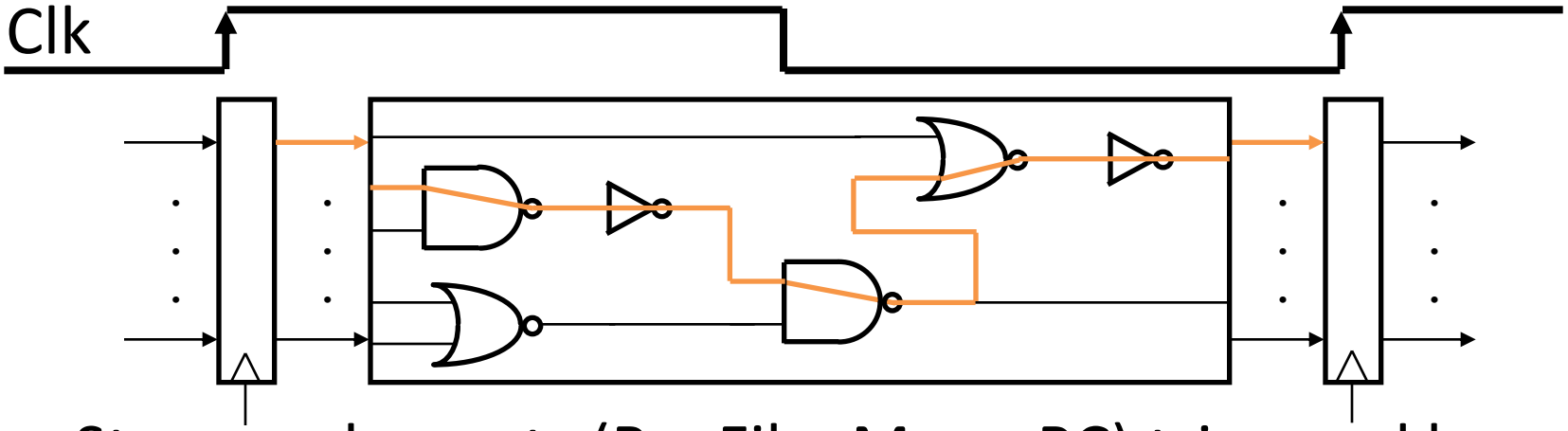
The Critical Path

- The *critical path* is the longest delay between *any* two registers in a circuit
- The clock period must be *longer* than this critical path, or the signal will not propagate properly to that next register



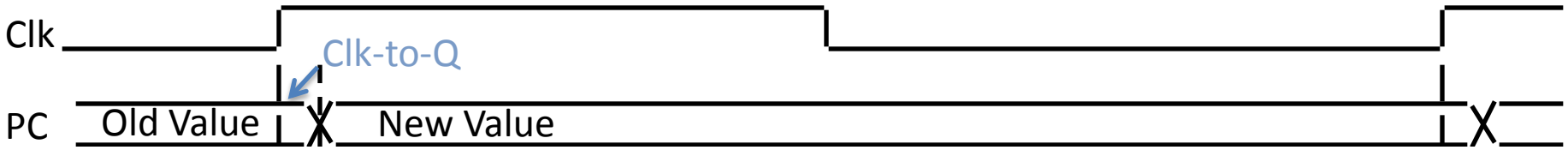
Critical Path =
CL Delay 1
+ CL Delay 2
+ CL Delay 3
+ Adder Delay

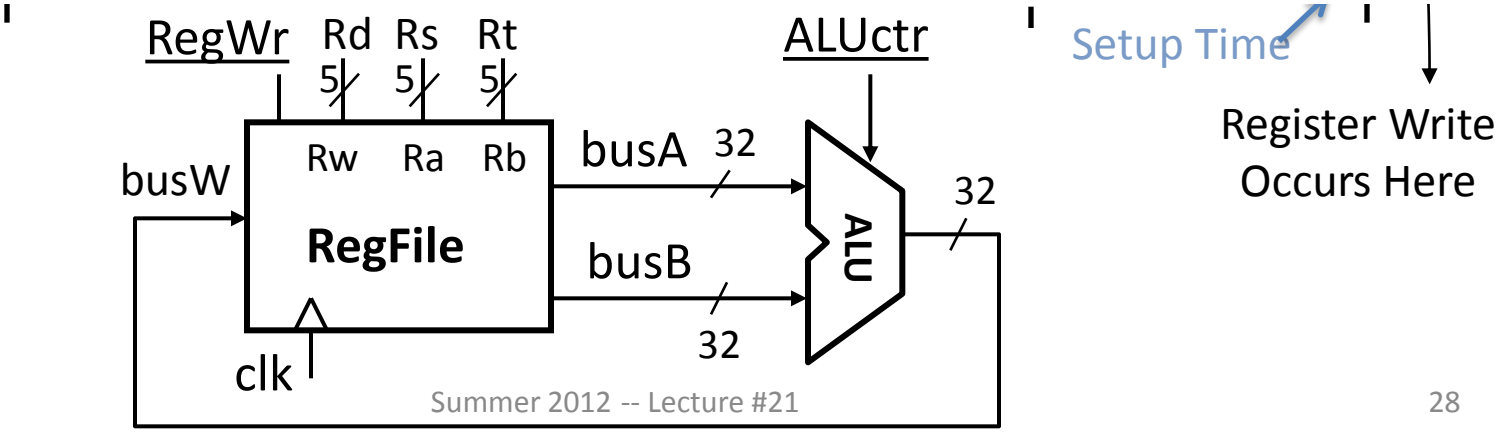
Clocking Methodology



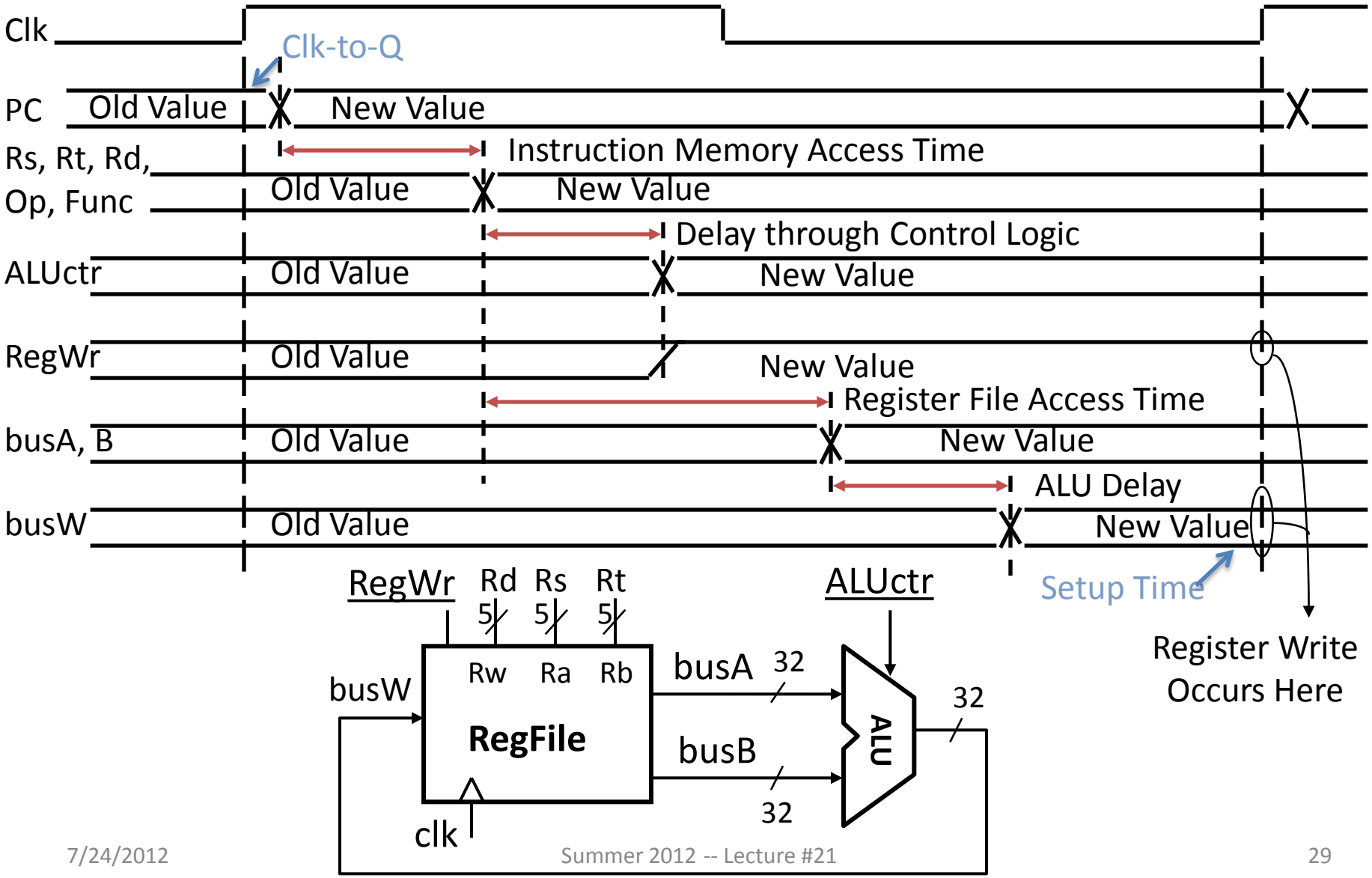
- Storage elements (RegFile, Mem, PC) triggered by same clock
- *Critical path* determines length of clock period
  - This includes CLK-to-Q delay and setup delay
- So far we have built a *single cycle CPU* – entire instructions are executed in 1 clock cycle
  - Up next: pipelining to execute instructions in 5 clock cycles

Register-Register Timing: One Complete Cycle for `addu`





Register-Register Timing: One Complete Cycle for `addu`



Single Cycle Performance

- Assume time for actions are 100ps for register read or write; 200ps for other events
- Minimum clock period is?

Instr	Instr fetch	Register read	ALU op	Memory access	Register write	Total time
lw	200ps	100 ps	200ps	200ps	100 ps	800ps
sw	200ps	100 ps	200ps	200ps		700ps
R-format	200ps	100 ps	200ps		100 ps	600ps
beq	200ps	100 ps	200ps			500ps

- What can we do to improve clock rate?
- Will this improve performance as well?
  - Want increased clock rate to mean faster programs

作业1

《计算机组成与设计》	Logicsim	WORD
4.3		✓
4.4		✓
4.5	✓(4.5.1)	✓
4.7		✓
4.10		✓

作业2

- Logicsim
  - 以上次作用3的设计2为基础，实现完整的单周期CPU
  - 构造一个至少20条以上指令的测试程序
    - 每个类型的指令必须出现1次以上
- Word
  - 简要论述设计要点，并对测试用例及测试结果进行简单分析

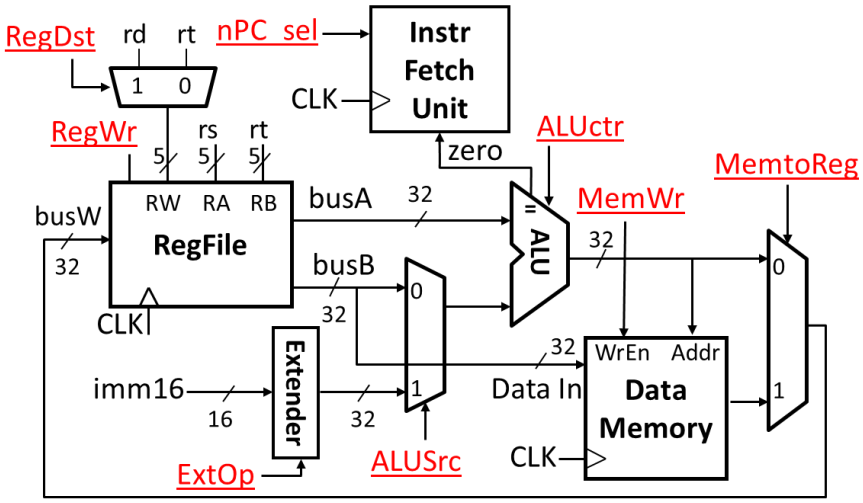
MIPS-lite控制信号

	ADDU	SUBU	ORI	LW	SW	BEQ
nPC_sel	"+4"					
RegDst	rd					
ALUSrc	RegB					
ALUctl	"ADD"					
ExtOp						
MemtoReg						
RegWr	1					
MemWr						

	ADDU	SUBU	ORI	LW
nPC_sel				

MIPS-lite控制信号

	ADDU	SUBU	ORI	LW	SW	BEQ
nPC_sel	"+4"					
RegDst	rd					
ALUSrc	RegB					
ALUctl	"ADD"					
ExtOp						
MemtoReg						
RegWr	1					
MemWr						



MIPS-lite Control Signals (1/2)

Instr	Control Signals
addu	ALUsrc=RegB, ALUctr="ADD", RegDst=rd, RegWr, nPC_sel="+4"
subu	RegWr,
ori	RegWr,
lw	RegWr,
sw	MemWr,
beq	"