计算机组成原理

单周期CPU设计
数据通路

请预习课件后完成Blackboard站点 “课堂测验” -> “Lecture8 preliminary _quiz”

教学目标

- 可以支持RV32I指令集的单周期CPU硬件设计
  - Datapath (数据通路)
  - Control (控制单元)

RISC-V32I

RISC-V-Reference-Data.pdf

I type {imm[11:0], rs1, funct3, rd, opcode}

inst	OPCODE	FUNCT3	FUNC7	hex
lb	000_0011	0		03/0/-
lh	000_0011	1		03/1/-
lw	000_0011	10		03/2/-
ld	000_0011	11		03/3/-
lbu	000_0011	100		03/4/-
lhu	000_0011	101		03/5/-
jalr	110_0111	0		67/0/-
addi	001_0011	0		13/0/-
slli	001_0011	1	0	13/1/0
slti	001_0011	10		13/2/-
sltiu	001_0011	11		13/3/-
xori	001_0011	100		13/4/-
srli	001_0011	101	0	13/5/0
srai	001_0011	101	010_0000	13/5/20
ori	001_0011	110		13/6/-
andi	001_0011	111		13/7/-

S type {imm[11:5], rs2, rs1, funct3, imm[4:0], opcode}

inst	OPCODE	FUNCT3	FUNC7	hex
sb	010_0011	0		23/
sh	010_0011	1		
sw	010_0011	10		
sd	010_0011	11		

R type {func7, rs2, rs1, func3, rd, opcode}

inst	OPCODE	FUNCT3	FUNC7	hex
add	011_0011	0	0	33/
sub	011_0011	0	010_0000	
sll	011_0011	1	0	
slt	011_0011	10		
sltu	011_0011	11		
xor	011_0011	100		
srl	011_0011	101	0	
sra	011_0011	101	010_0000	
or	011_0011	110		
and	011_0011	111		

CORE INSTRUCTION FORMATS

	31 27	26 25	24 20	19 15	14 12	11 7	6 0
R	funct7		rs2	rs1	funct3	rd	opcode
I	imm[11:0]			rs1	funct3	rd	opcode
S	imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode
SB	imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode
U	imm[31:12]					rd	opcode
UJ	imm[20|10:1|11|19:12]					rd	opcode

SB type {imm[12|10:5], rs2, rs1, funct3, imm[4:1|11], opcode}

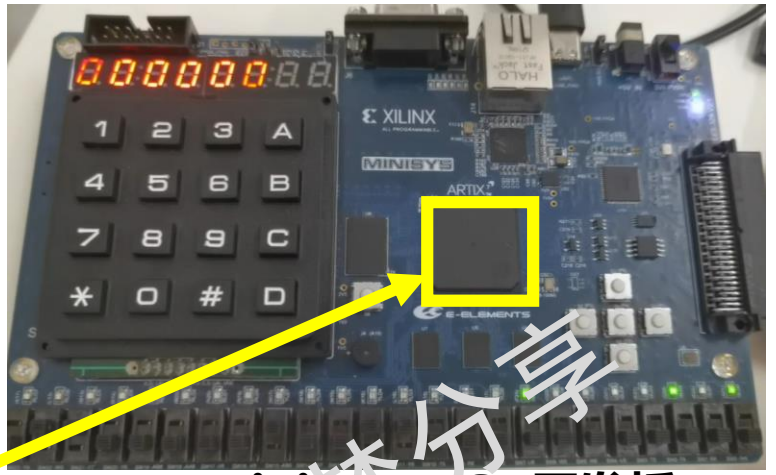
inst	OPCODE	FUNCT3	FUNC7	hex
beq	110_0011	0		63/
bne	110_0011	1		
blt	110_0011	100		
bge	110_0011	101		
bltu	110_0011	110		
bgeu	110_0011	111		

U type {imm[31:12], rd, opcode}

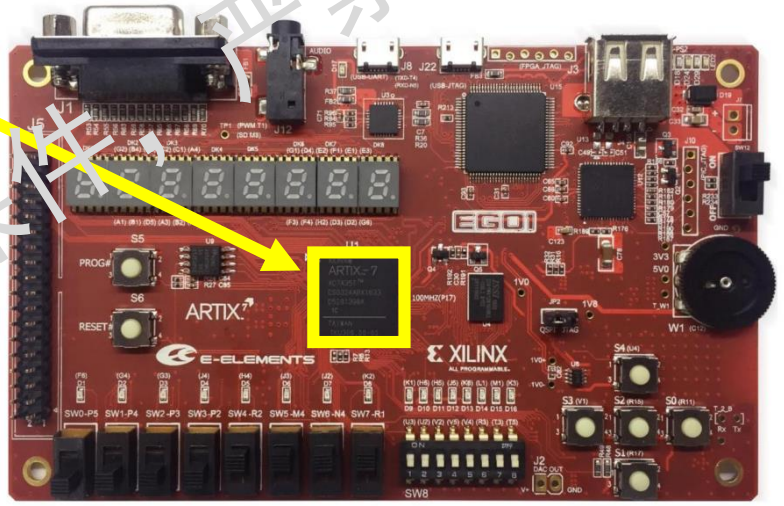
inst	OPCODE	FUNCT3	FUNC7	hex
auipc	001_0111			17/
lui	011_0111			37/

UJ type {imm[20|10:1|11|19:12], rd, opcode}

inst	OPCODE	FUNCT3	FUNC7	hex
jal	110_1111			6f/



Minisys FPGA开发板



EGO1 FPGA开发板

分析指令集

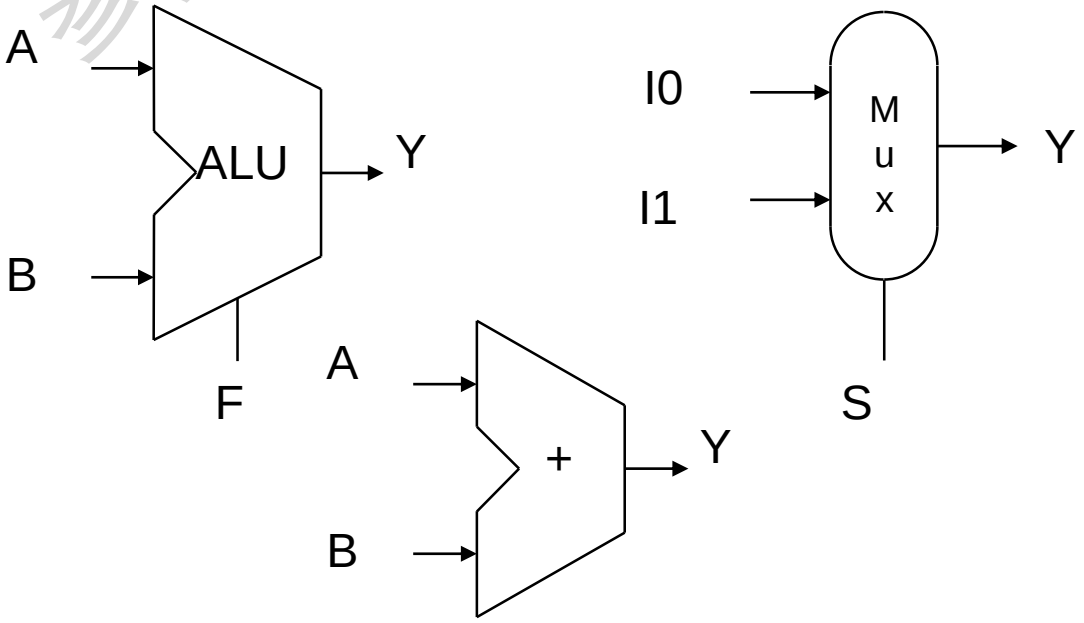
- 我们学过哪些典型基础指令?
- 核心指令子集:
  - 算术/逻辑指令: add, sub, and, or
  - 内存访问指令: lw, sw
  - 分支跳转指令: beq
  - lw执行步骤: 取指->译码->运算->访存->写回

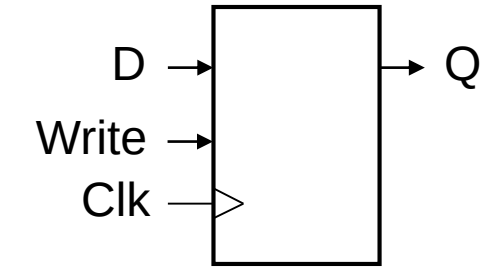
- 指令的通用操作:
  - 用程序计数器(PC)从指令内存取指令
  - 运算(算数/逻辑、内存地址)
  - 访问寄存器
  - 访问内存

- 必要组件:
  - 存储器: 存储指令和数据
  - 寄存器
  - ALU
  - 控制逻辑等

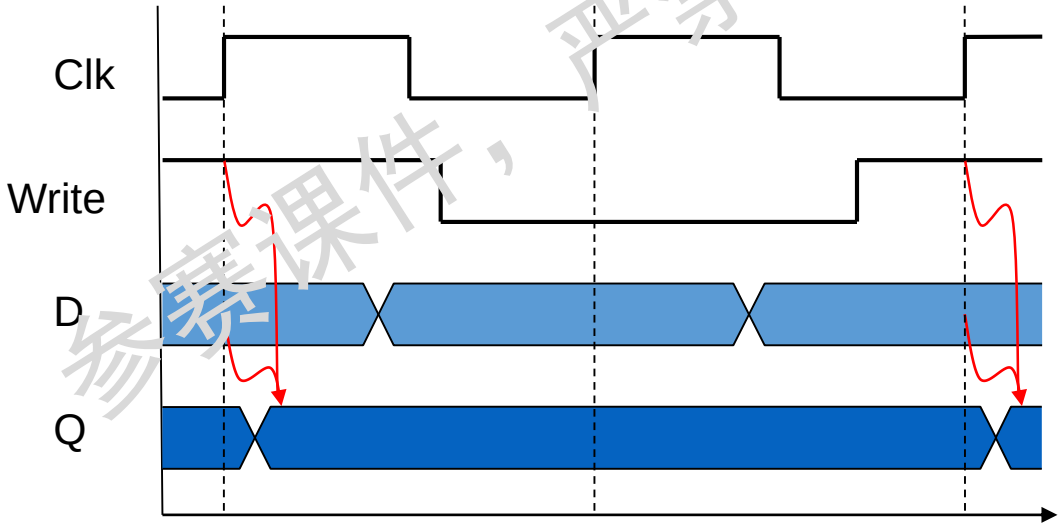
数据通路组件

- 组合元件：ALU、多路选择器等
- 时序元件(状态元件)：DFF、寄存器等
- 组合逻辑和时序逻辑的区别?



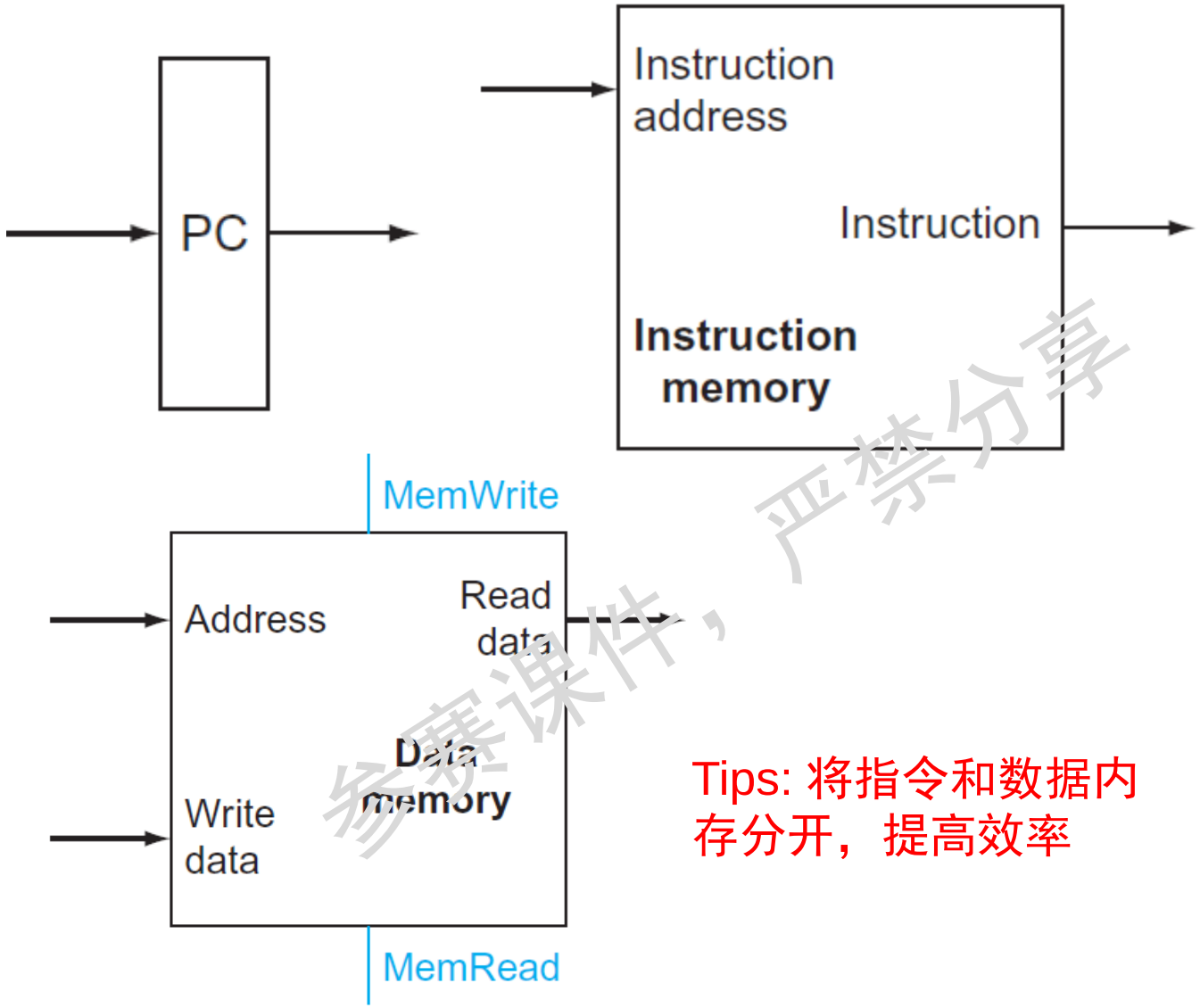


- 带写控制的时序元件（如数据内存/寄存器）：
  在写控制信号为1时，在时钟边沿(0→1)触发更新



数据通路时序元件

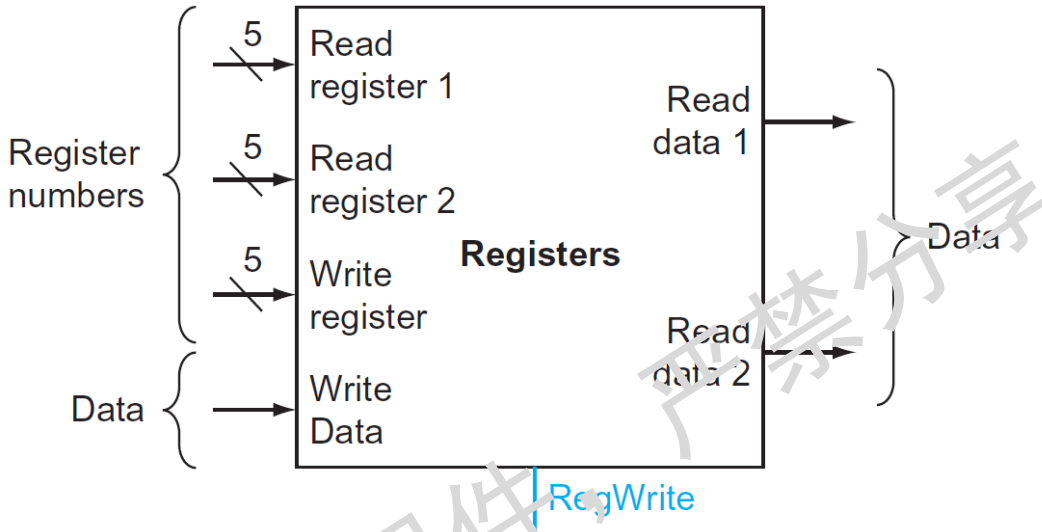
- 程序计数器（PC）：
  - 32位寄存器
  - 输入/输出：指令地址（32位）
- 指令内存 (IMEM)：
  - 输入：指令地址（32位）
  - 输出：指令（32位）
- 数据内存 (DMEM)：
  - 输入：
    - 地址（32位）
    - 写入数据（32位）
    - MemWrite（1位）
    - MemRead（1位）
  - 输出：
    - 读取数据（32位）



Tips: 将指令和数据内存分开，提高效率

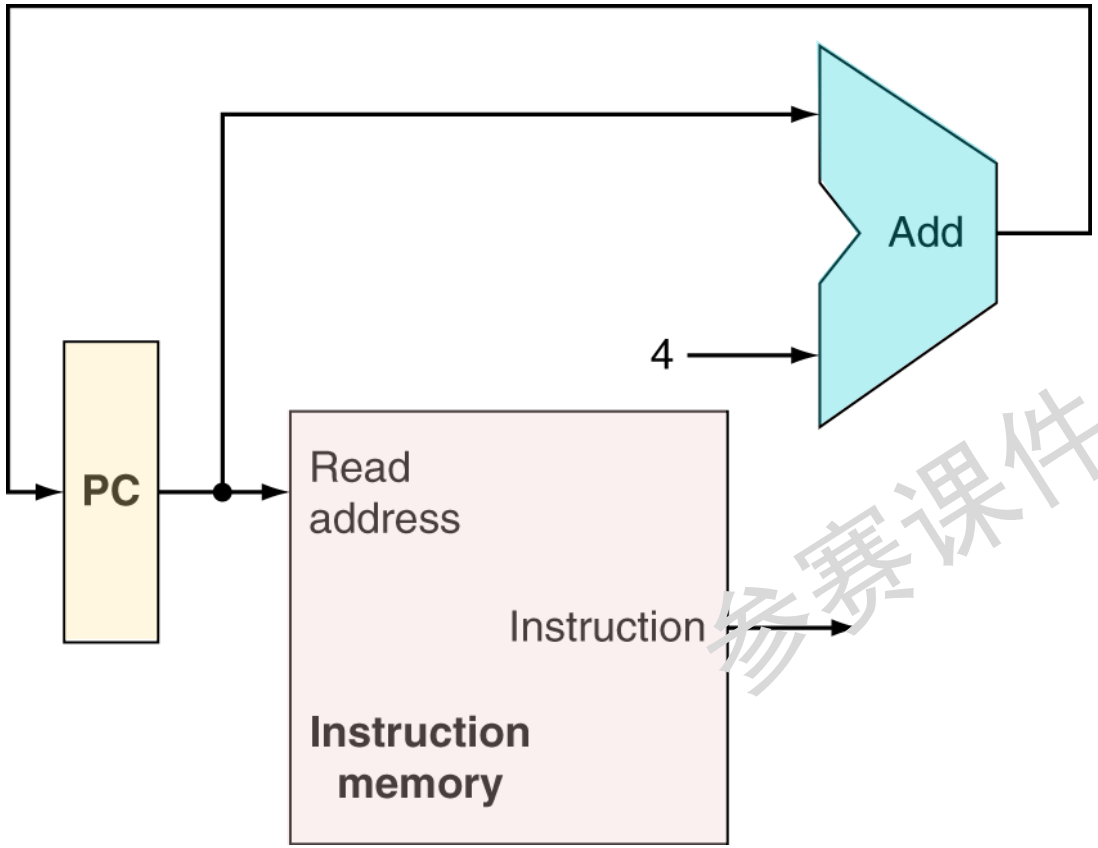
数据通路时序元件

- 寄存器堆 (REGs)：
  - 包含一系列寄存器的状态单元，可通过寄存器编号进行读写
  - 组成：32个寄存器（32×32位数据）
- 输入：
  - 三个寄存器编号rs1,rs2,rd（5位×3）
  - 写入数据（32位）
  - RegWrite（1位）
- 输出：
  - 读取数据rdata1,rdata2（32位×2）

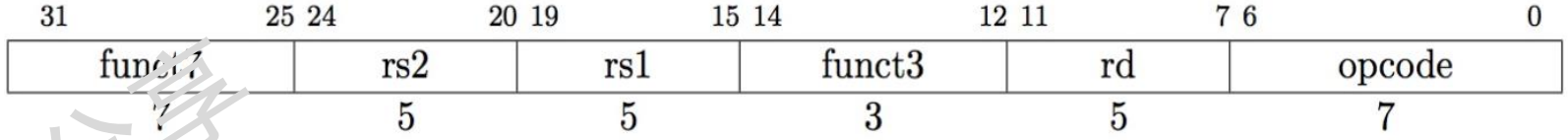


取指令操作

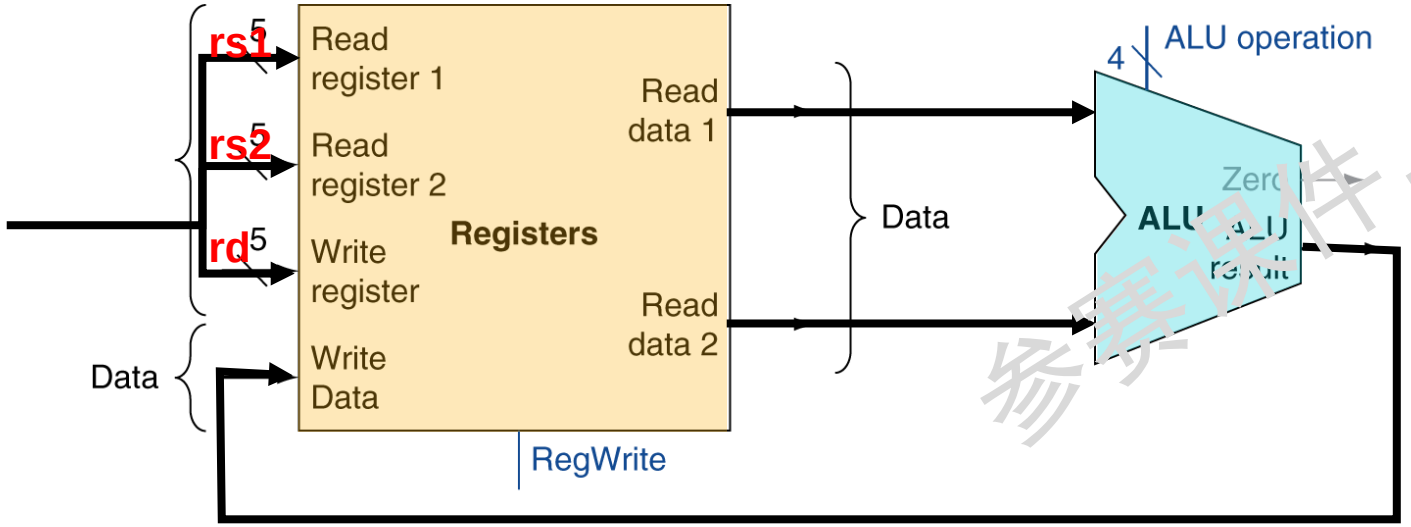
- Fetch(取指)操作：从指令内存取出指令，同时计算PC+4为下一条指令做准备
- 组件：PC、指令内存、加法器
- PC特性：32位寄存器，每时钟上升沿自动更新，无需写控制



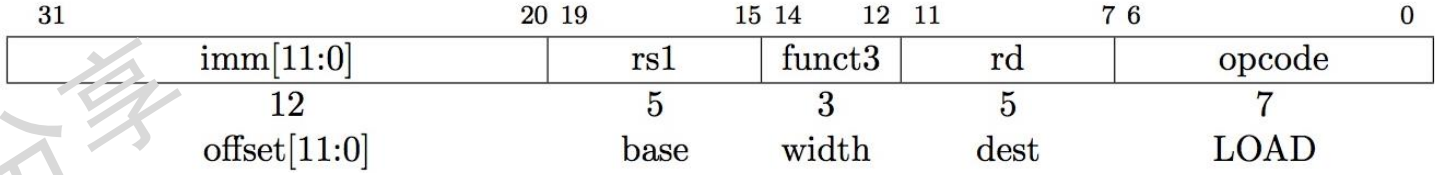
加法指令操作



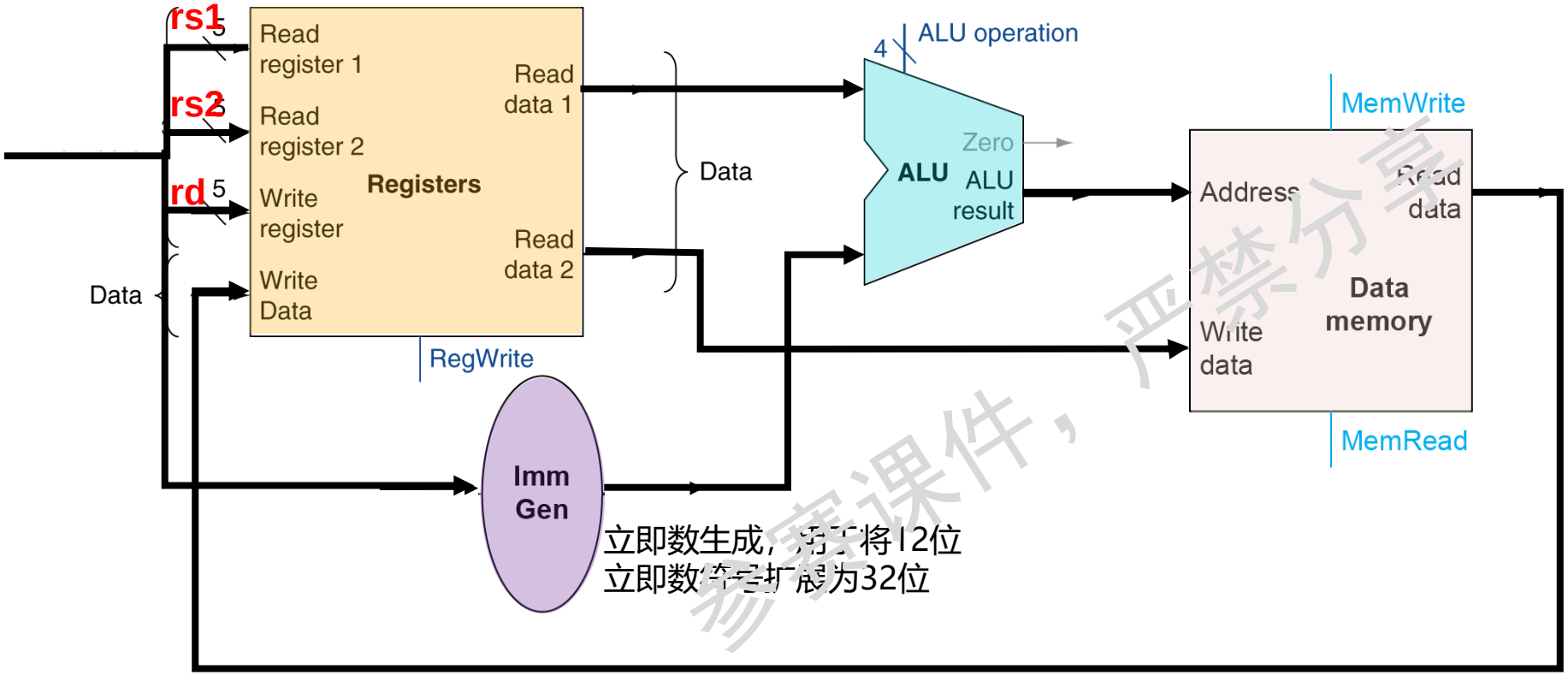
- R型指令：如add rd, rs1, rs2
  - 通过ALU将Registers读取的源操作数相加
  - 写寄存器
- 控制信号：
  - RegWrite：将结果写入目标寄存器
  - ALUOp：控制ALU执行加法(或其他运算)



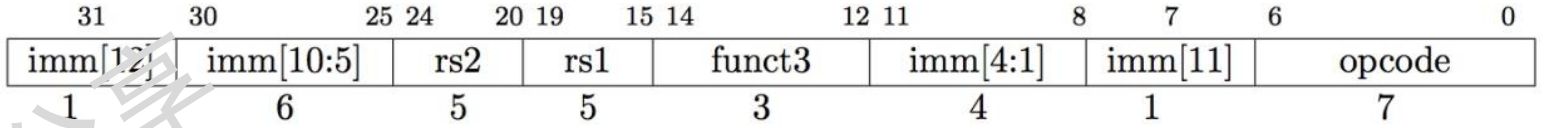
访存指令操作



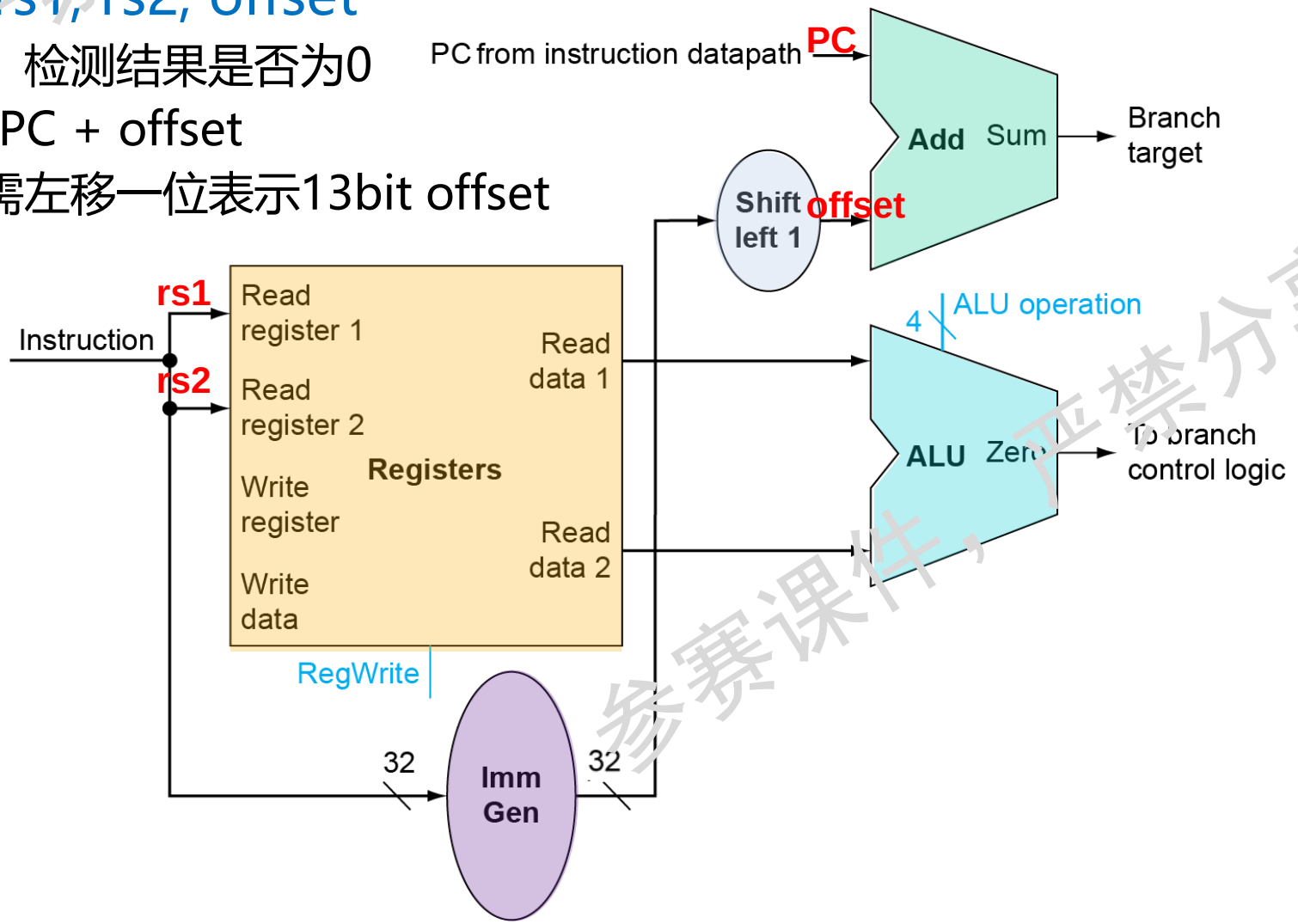
- 访存指令：如lw rd, offset(rs1), sw rs2, offset(rs1)
  - 地址计算：使用ALU将Registers读取的源操作数与offset相加
  - 访问内存
  - 写寄存器
- 控制信号：
  - MemRead
  - MemWrite



分支指令操作



- 分支跳转：如beq rs1, rs2, offset
  - ALU计算 rs1 - rs2，检测结果是否为0
  - 若相等，PC更新为 PC + offset
  - 12 bit immediate需左移一位表示13bit offset

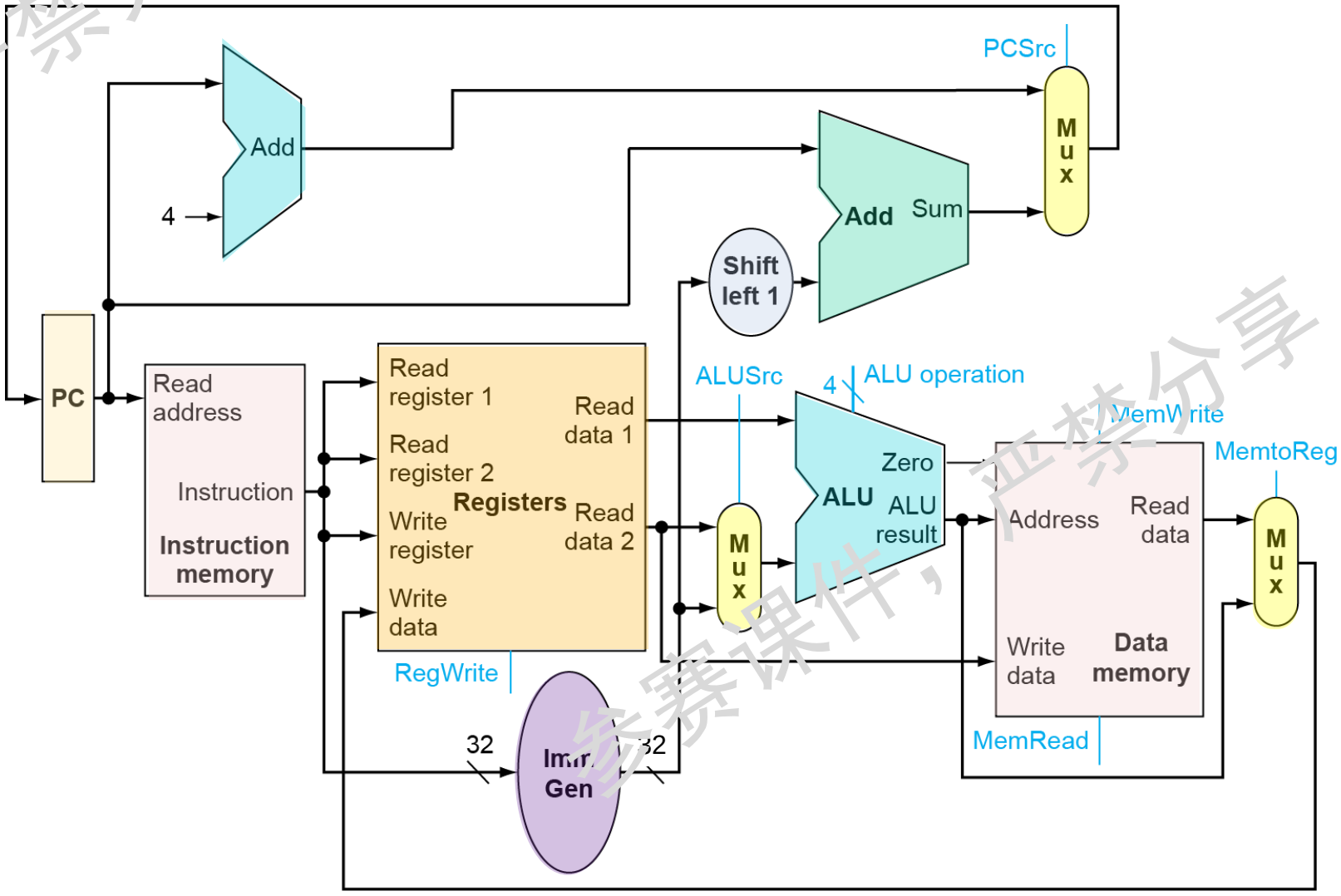


完整数据通路

1. **PC**的作用？如何更新？
2. **从指令Memory**获取指令时需要什么输入？
3. **ALU**的输入来自哪里？
4. **Registers**(寄存器堆)的输入有哪些？
5. **写回Registers的数据**来自哪里？

总结

- 完整的数据通路
- 思考题
  - j型指令的数据通路?

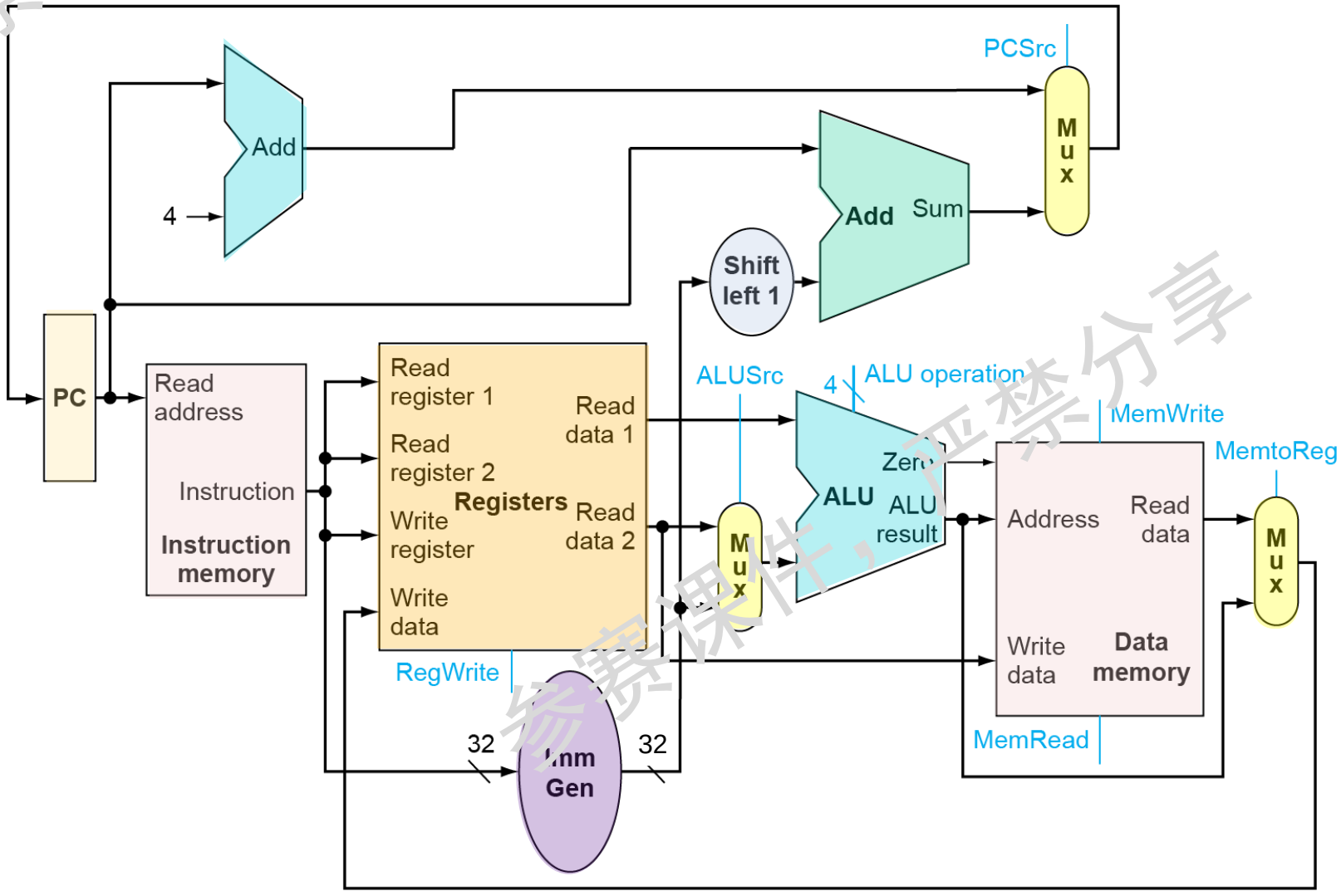


计算机组成原理

单周期CPU设计
控制逻辑

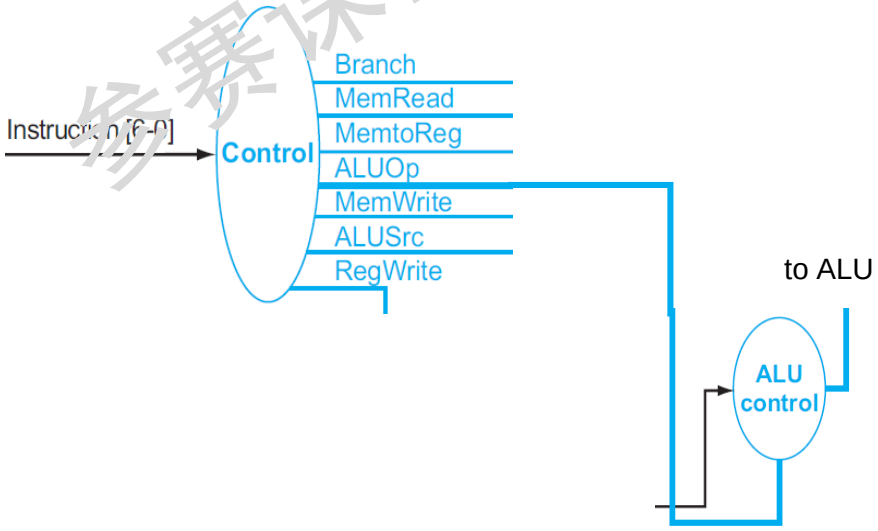
课题引入

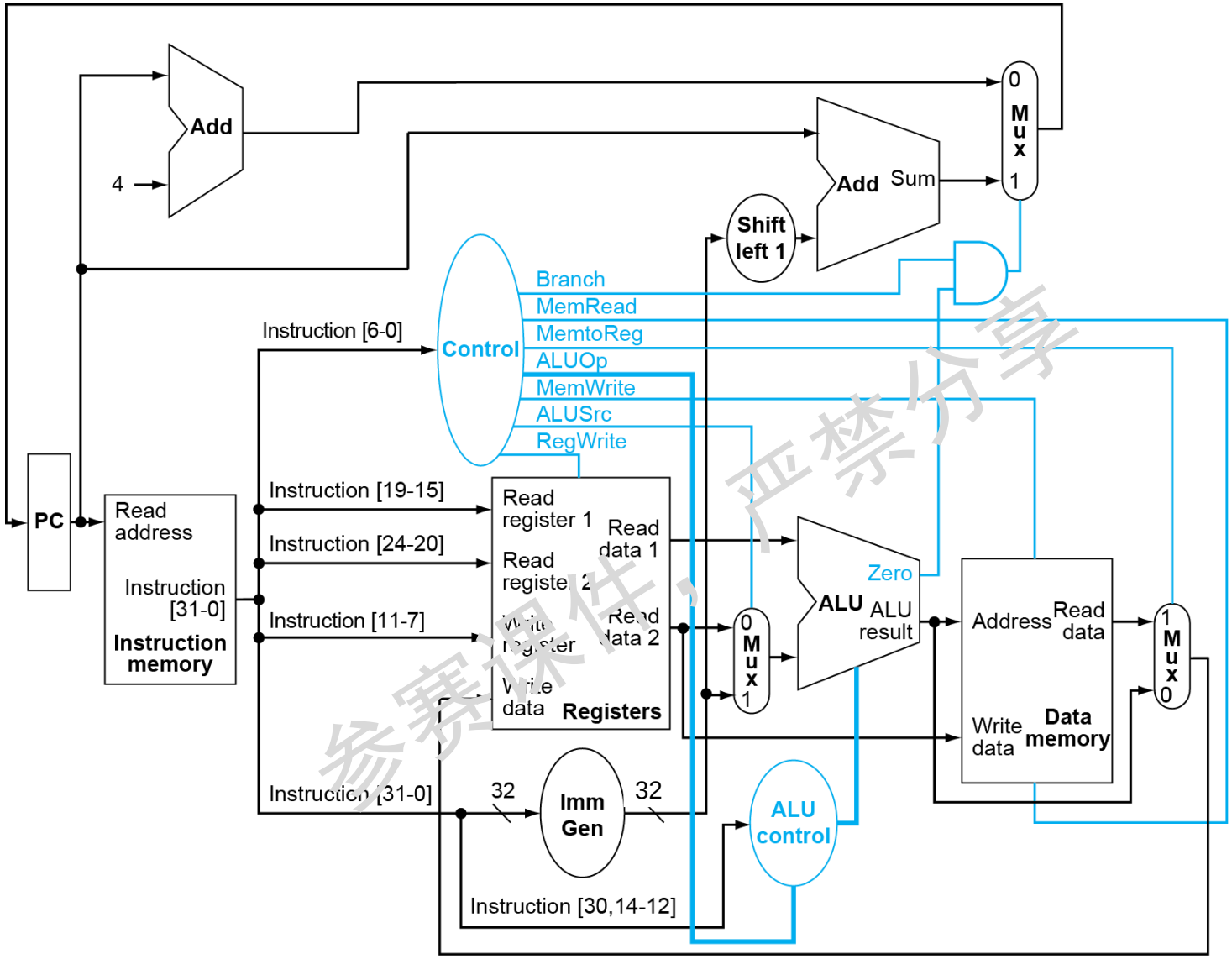
- RISC-V单周期CPU设计：实现指令集架构中的指令
  - **Datapath (数据通路)**
- **教学目标**
  - **Control (控制单元)**



添加控制逻辑的完整数据通路

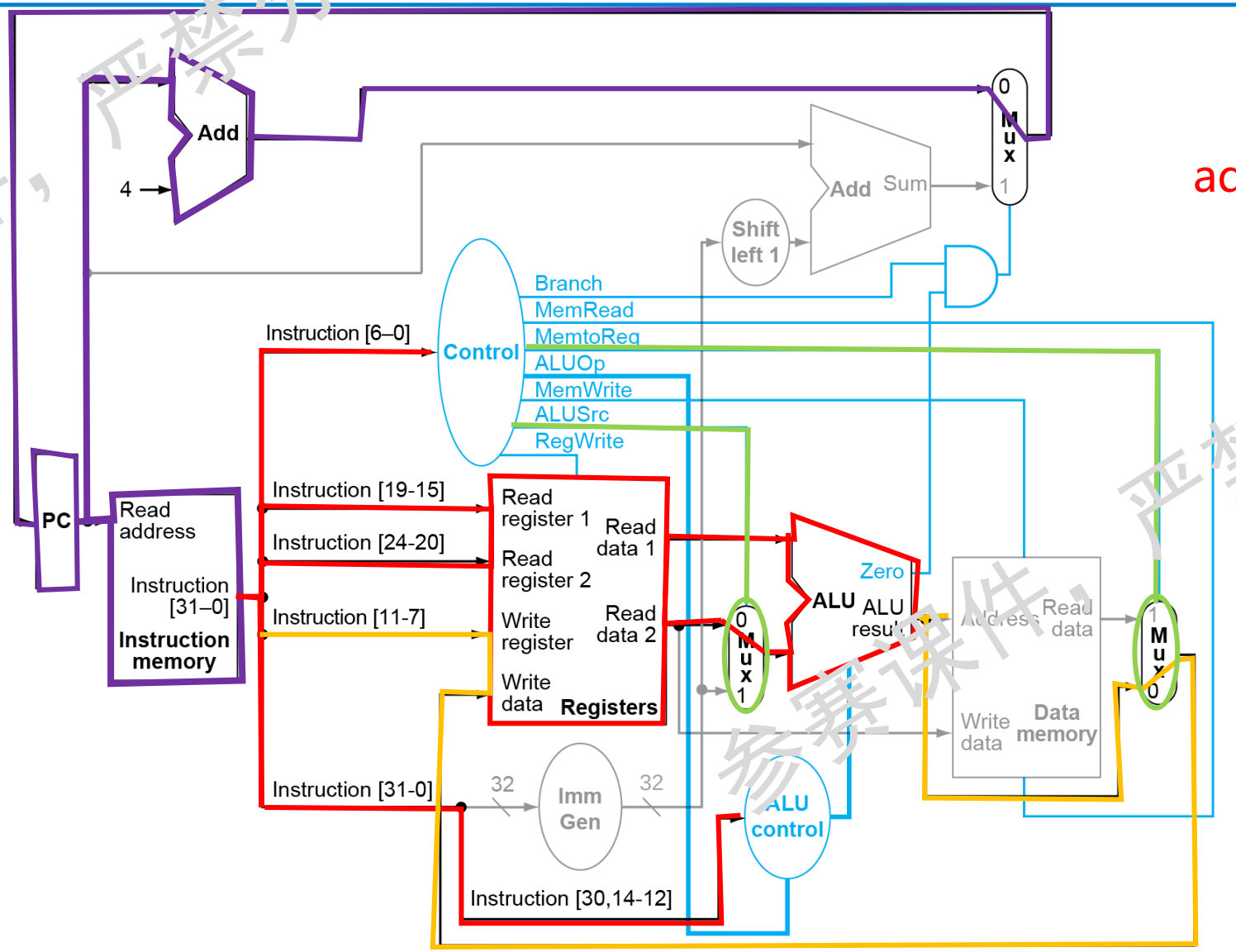
- 控制单元
  - Main Control
  - ALU Control





R型指令

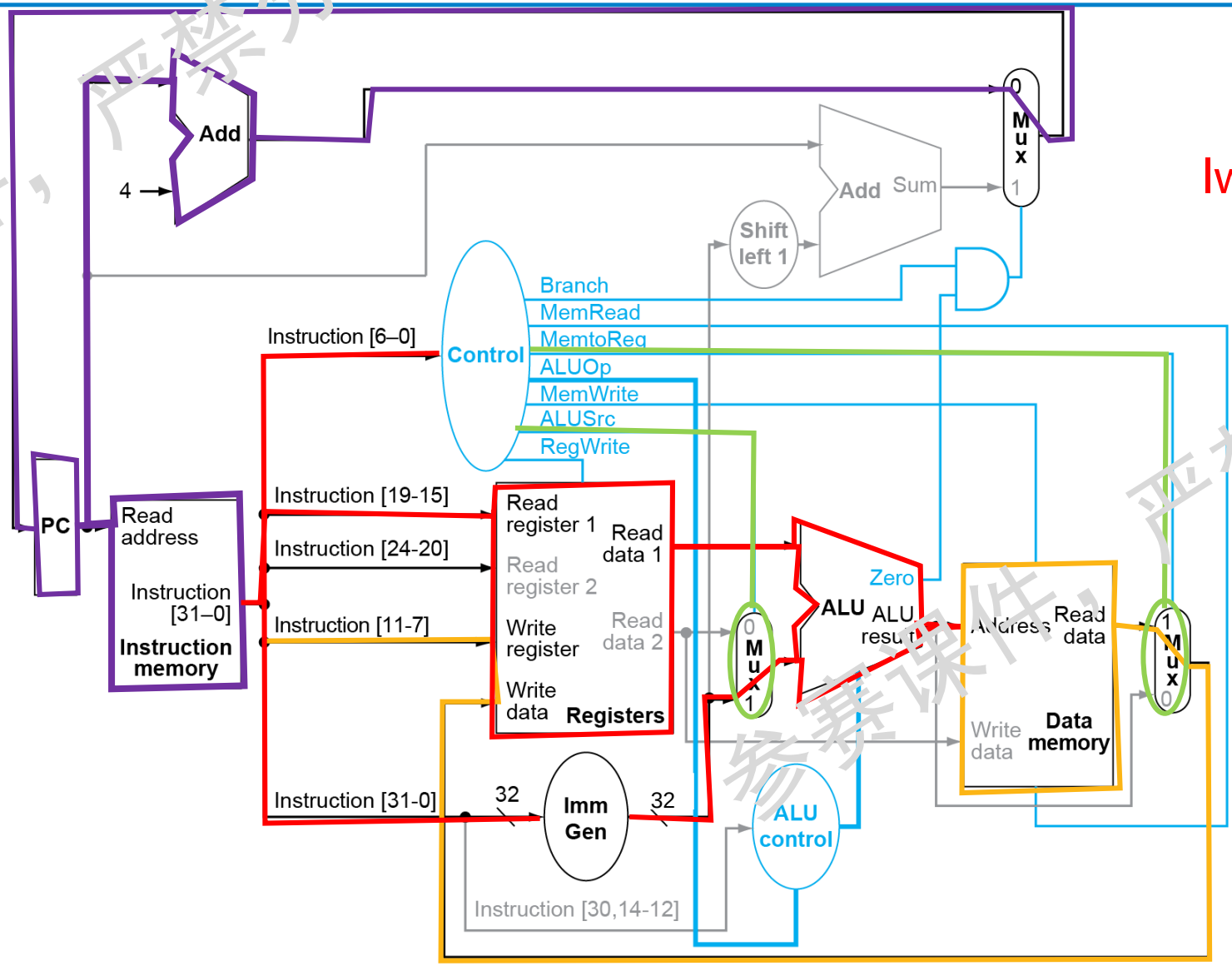
- ALUSrc=0
- MemtoReg=0
- RegWrite=1
- MemRead=0
- MemWrite=0
- Branch=0



add rd, rs1, rs2

lw指令

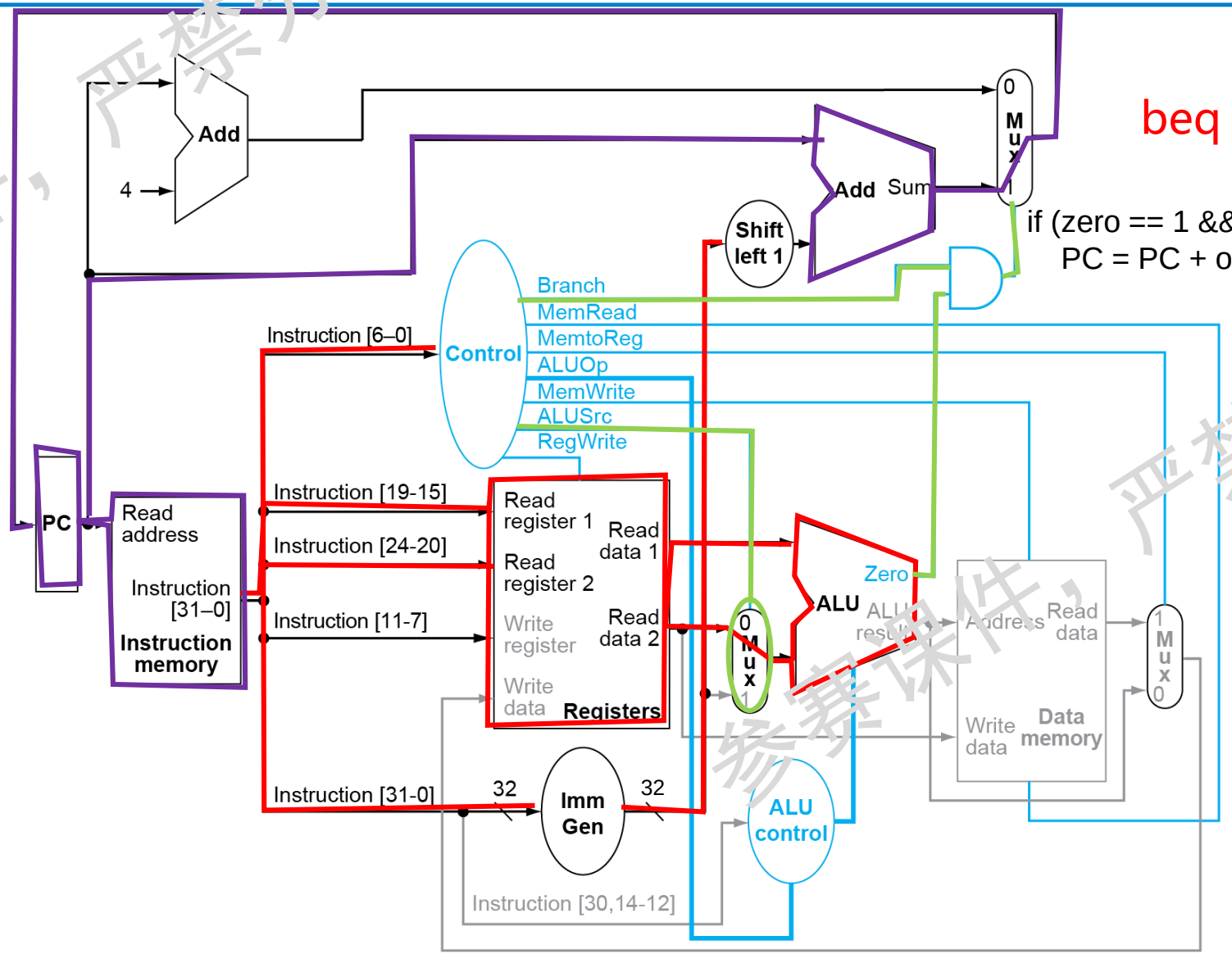
- ALUSrc=
- MemtoReg=
- RegWrite=
- MemRead=
- MemWrite=
- Branch=



lw rd, offset(rs1)

beq指令

- ALUSrc=
- MemtoReg=
- RegWrite=
- MemRead=
- MemWrite=
- Branch=



beq rs1, rs2, offset

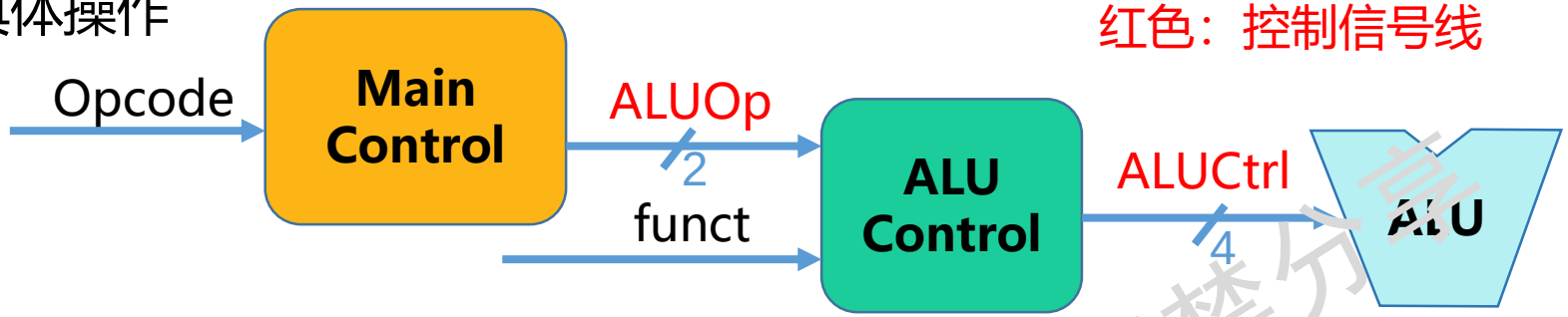
if (zero == 1 && Branch == 1)
PC = PC + offset

主控制信号总结

Input or output	Signal name	R-format	lw	sw	beq
Inputs	I[6]	0	0	0	1
	I[5]	1	0	1	1
	I[4]	1	0	0	0
	I[3]	0	0	0	0
	I[2]	0	0	0	0
	I[1]	1	1	1	1
	I[0]	1	1	1	1
Outputs	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1

ALU控制

- ALUOp和ALUControl分级控制
  - 主控制单元只需关注指令类型（如算术、分支、访存等）
  - 而ALU控制单元专注于具体操作
  - 提高扩展性

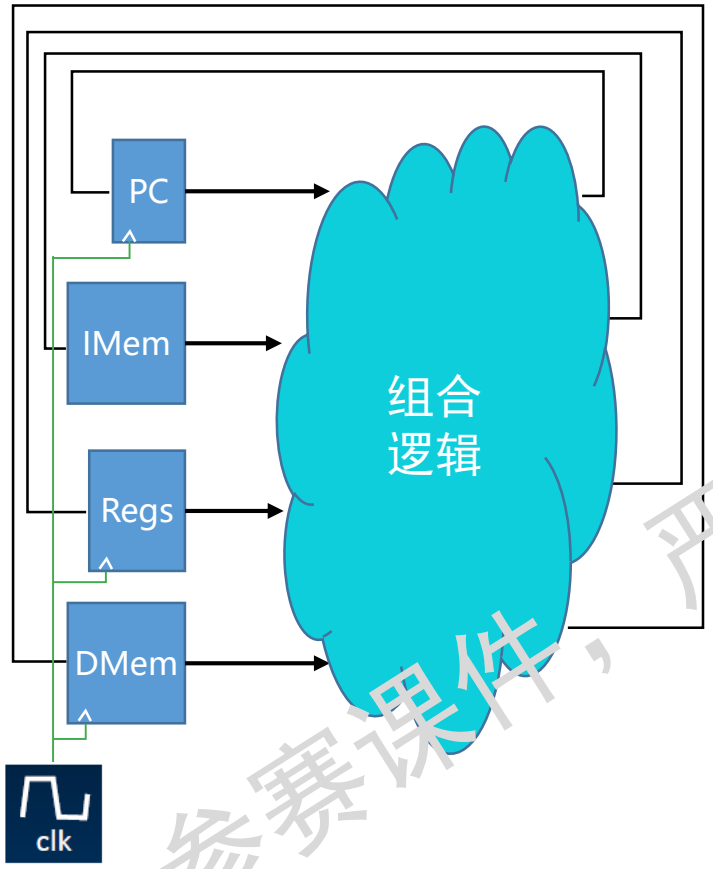


Instruction opcode	ALUOp	Operation	Funct7 field	Funct3 field	Desired ALU action	ALU control input
lw	00	load word	XXXXXXX	XXX	add	0010
sw	00	store word	XXXXXXX	XXX	add	0010
beq	01	branch if equal	XXXXXXX	XXX	subtract	0110
R-type	10	add	0000000	000	add	0010
R-type	10	sub	0100000	000	subtract	0110
R-type	10	and	0000000	111	AND	0000
R-type	10	or	0000000	110	OR	0001

inst[30] inst[14-12]

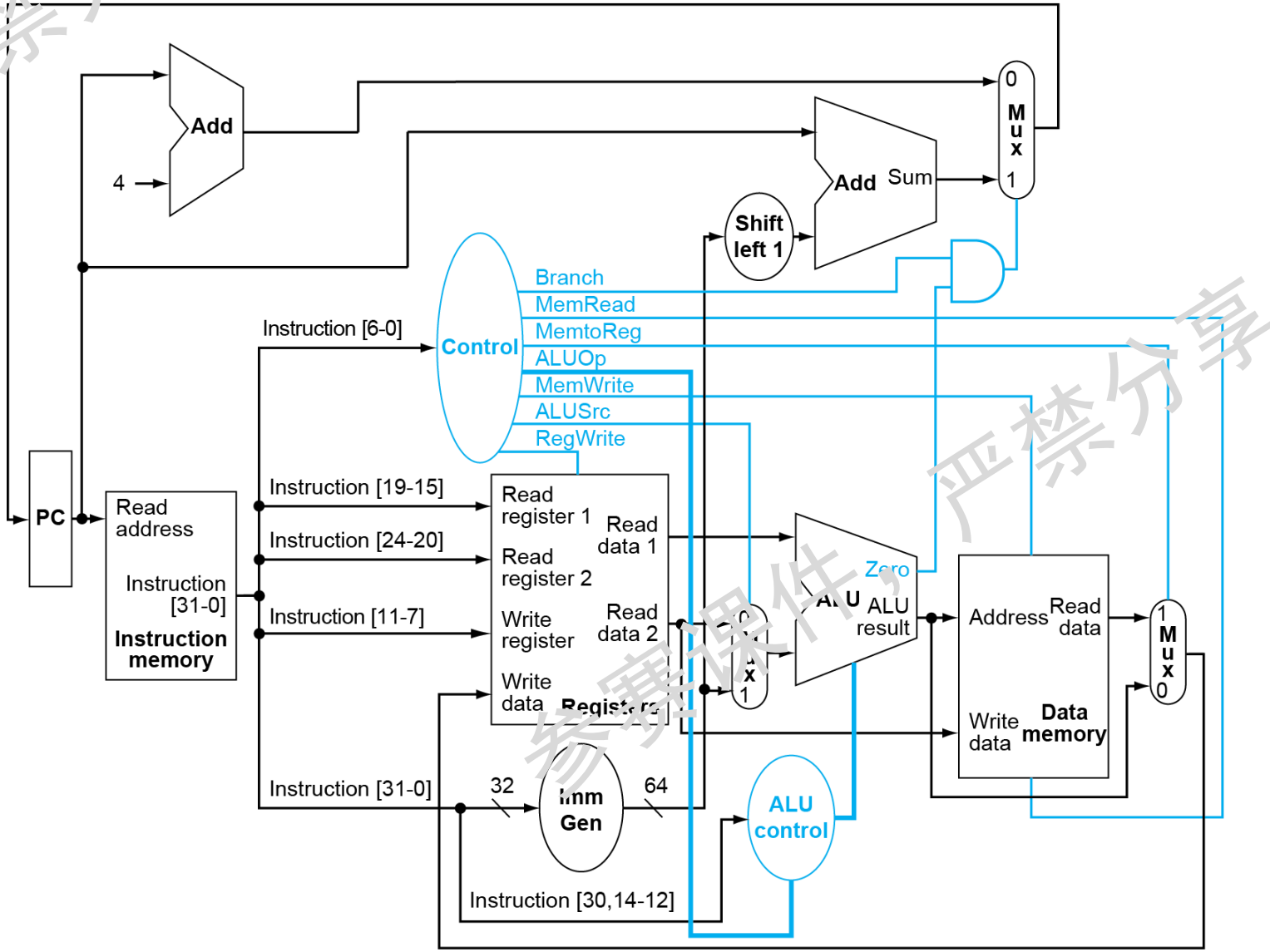
时钟策略

- 单周期CPU
  - 组合逻辑在时钟周期内完成数据转换
  - 其工作时机位于两个时钟边沿之间
  - 输入来自状态元件，输出传递至状态元件
  - 最长延迟决定时钟周期



拓展新指令设计

- 设计一个新的单周期指令，分析指令功能，并画出数据通路并列出对应控制信号



思考与课后拓展

- 思考
  - 作为计算机专业的学生，最基本的专业素质是:
    - 掌握计算机系统基本原理
    - 学会设计计算机系统
      - 系统软件
      - 硬件系统
      - 软硬件协同
    - 构建自主生态大有作为
  - 中国 ”芯” 的路还很长，很艰巨，需要大家树立起足够的信心和勇气
- 课后拓展
  - 比较RISC-V、ARM、LoongArch三种精简指令集的共同点和不同点