

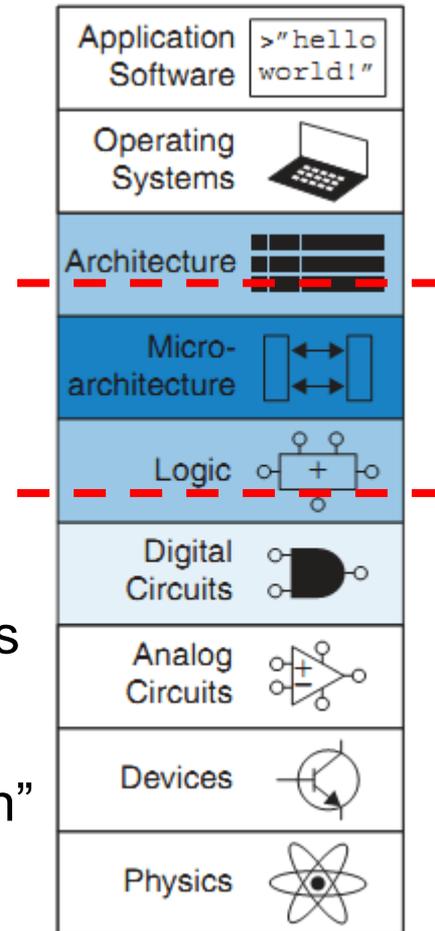
# The Processor Implementation: Datapath & Control

“Computer Organization & Design”

第四章

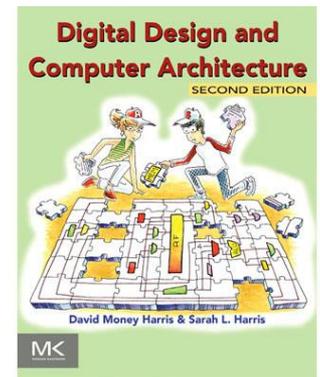
# Instruction-Set Processor Design

- Architecture (ISA) *programmer/compiler view*
  - “**functional** appearance to its immediate user/system programmer”
  - Opcodes, addressing modes, architected registers, IEEE floating point
  - 机器语言
- Implementation ( $\mu$ Arch) *processor designer view*
  - “**logical structure** or **organization** that performs the architecture”
  - functional units, pipelining, caches, physical registers
- Realization (chip) *chip/system designer view*
  - “**physical structure** that embodies the implementation”
  - Gates, cells, transistors, wires



# 内容提要

- CPU uArch实现概要： \$4.1， \$4.2
  - 设计步骤
    - 4步法：数据通路，控制器，定时，综合
    - 定时：指令周期，时钟周期，机器周期（总线周期，访存周期）
- 单周期设计：早期RISC采用， \$4.3， \$4.4
  - 输入：RV32I ISA
  - 输出：功能部件，数据通路，控制器（真值表），Clocking
- 性能分析
- COD5-RV
  - \$4.1：引言
  - \$4.2：逻辑设计的一般方法（设计约定）
  - \$4.3：数据通路
  - \$4.4：单周期实现（ALU控制，主控制器）



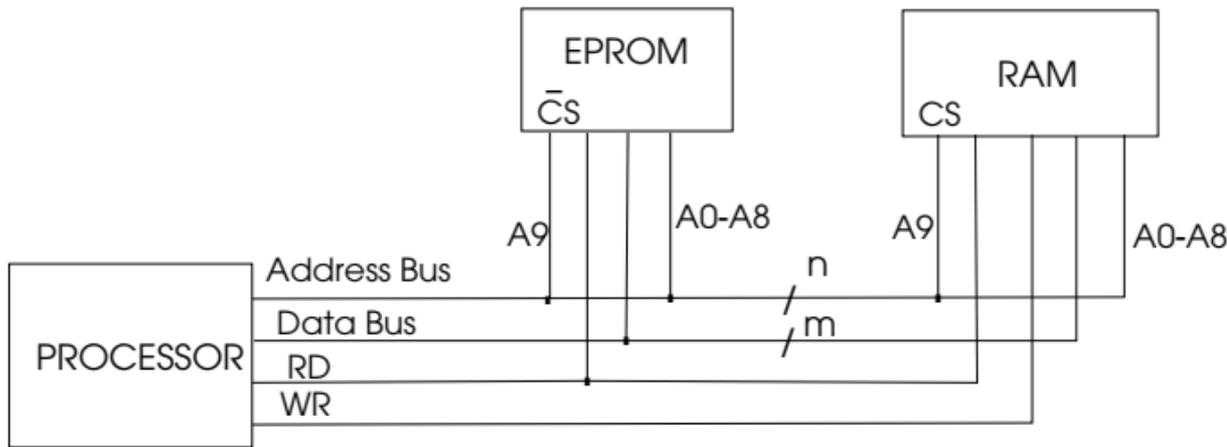
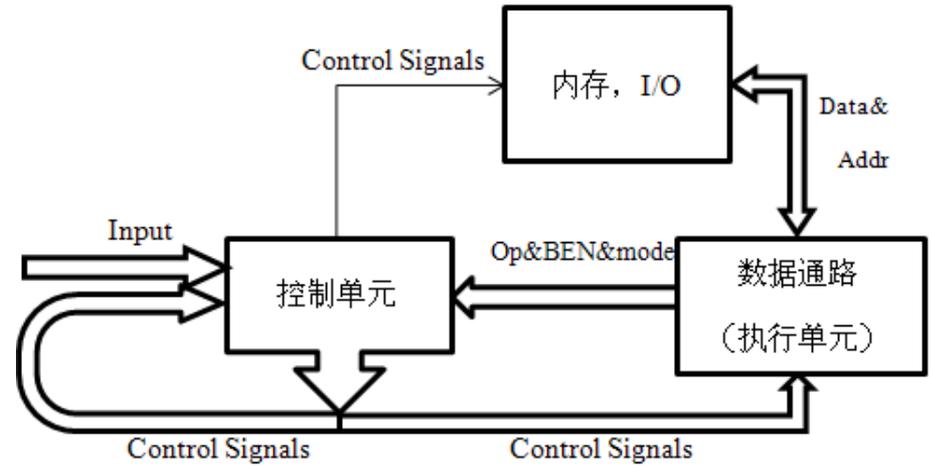
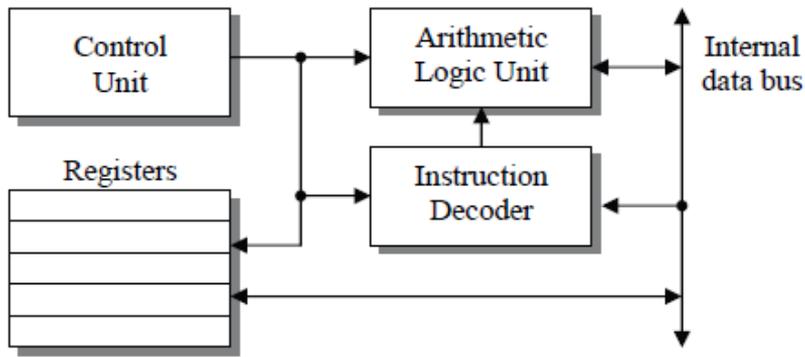
David M. Harris, Sarah L. Harris,  
Digital Design and Computer  
Architecture, 2012

# ISA Processor Design: RV的核心子集

Example instruction	Instruction name	Meaning
add x1,x2,x3	Add	$\text{Regs}[x1] \leftarrow \text{Regs}[x2] + \text{Regs}[x3]$
addi x1,x2,3	Add immediate unsigned	$\text{Regs}[x1] \leftarrow \text{Regs}[x2] + 3$
lw x1,60(x2)	Load word	$\text{Regs}[x1] \leftarrow_{64} \text{Mem}[60 + \text{Regs}[x2]]_0^{32} \text{##}$ $\text{Mem}[60 + \text{Regs}[x2]]$
sw x3,500(x4)	Store word	$\text{Mem}[500 + \text{Regs}[x4]] \leftarrow_{32} \text{Regs}[x3]_{32..63}$
beq x3,x4,offset	Branch equal zero	if ( $\text{Regs}[x3] == \text{Regs}[x4]$ ) $\text{PC} \leftarrow \text{PC} + (\text{offset} \ll 1)$
jal x1,offset	Jump and link	$\text{Regs}[x1] \leftarrow \text{PC} + 4$ ; $\text{PC} \leftarrow \text{PC} + (\text{offset} \ll 1)$
jalr x1,x2,offset	Jump and link register	$\text{Regs}[x1] \leftarrow \text{PC} + 4$ ; $\text{PC} \leftarrow \text{Regs}[x2] + \text{offset}$

Name (Field size)	Field						Comments
	7 bits	5 bits	5 bits	3 bits	5 bits	7 bits	
R-type	funct7	rs2	rs1	funct3	rd	opcode	Arithmetic instruction format
I-type	immediate[11:0]		rs1	funct3	rd	opcode	Loads & immediate arithmetic
S-type	immed[11:5]	rs2	rs1	funct3	immed[4:0]	opcode	Stores
SB-type	immed[12,10:5]	rs2	rs1	funct3	immed[4:1,11]	opcode	Conditional branch format
UJ-type	immediate[20,10:1,11,19:12]				rd	opcode	Unconditional jump format
U-type	immediate[31:12]				rd	opcode	Upper immediate format

# 微结构：数据通路，控制器

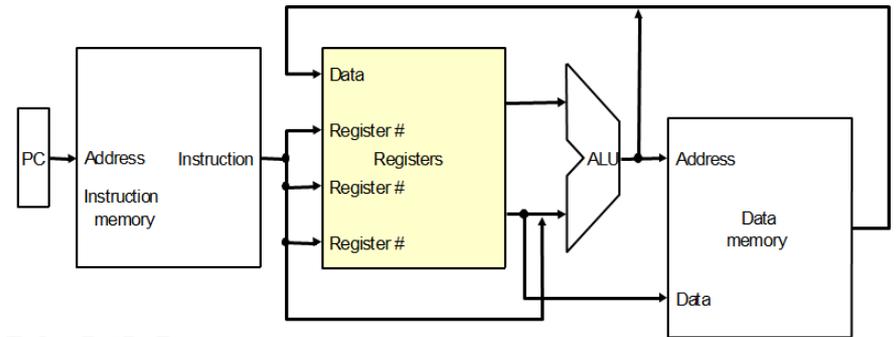


内存、I/O:  
可视为datapath的组成  
部分

# 数据通路 Abstract / Simplified View

- RV Instruction Execution Phases

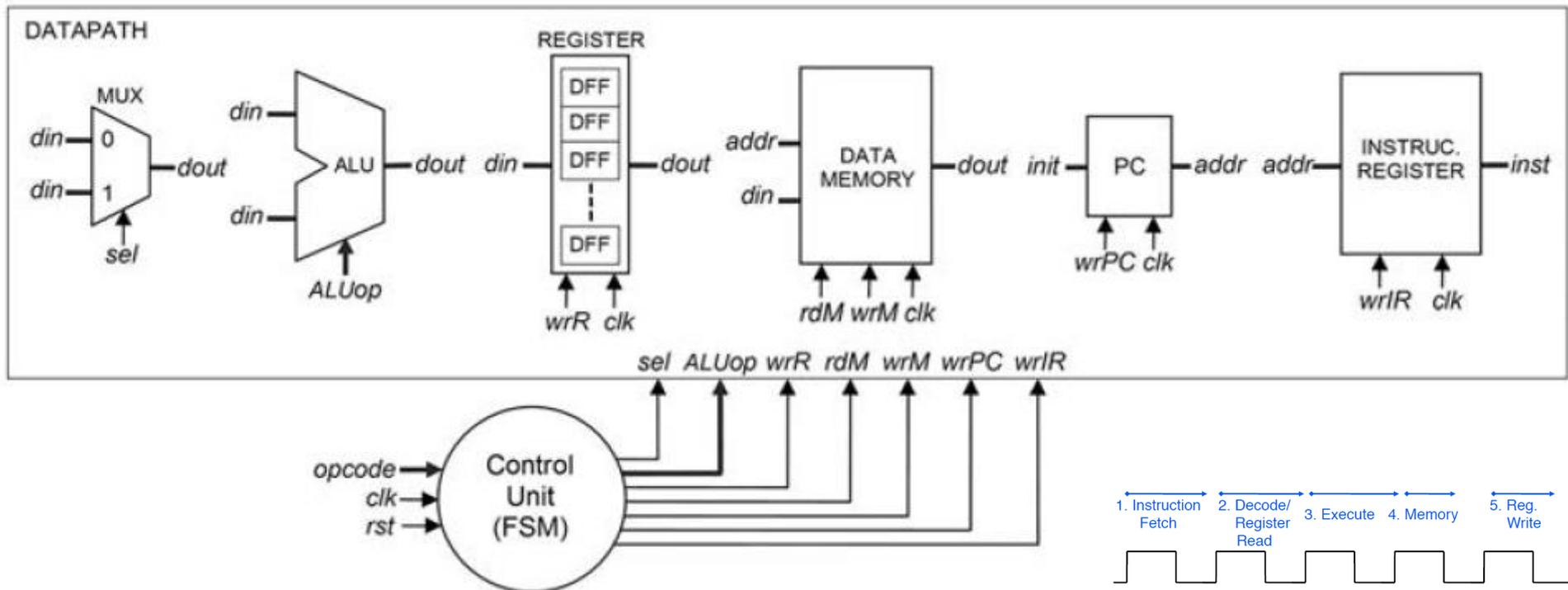
- Instruction Fetch
- Instruction Decode
- Register Fetch
- ALU Operations
- *Optional* Memory Operations
- *Optional* Register Write back
- Calculate Next Instruction Address



- Two types of functional units

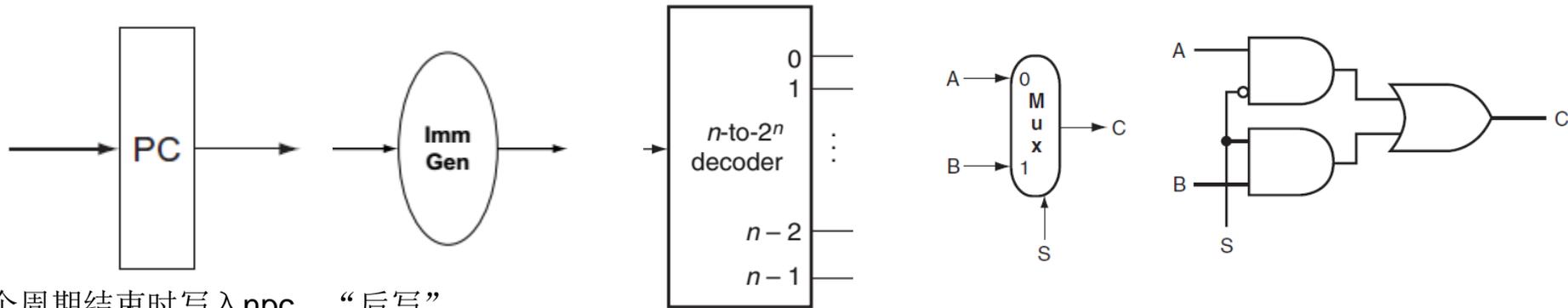
- elements that operate on data values (combinational)
- elements that contain state (sequential)

# Controller : Abstract/Simplified View, 附录C



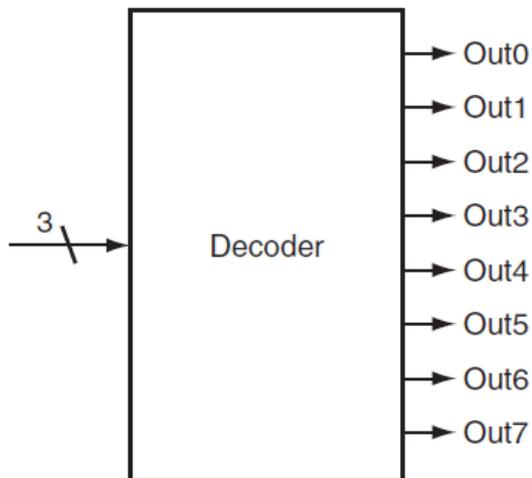
- 功能：译码，并在正确的时间产生正确的控制信号
  - Clk, Reset?
- 由两部分构成：生成控制信号和下一状态 + 时序控制
  - 真值表，FSM (ROM, PLA, Sequencer)，微程序
  - 组合逻辑 (计算)，顺序逻辑 (状态保存，时序控制)

# PC, 立即数生成/符号扩展, 译码器, MUX



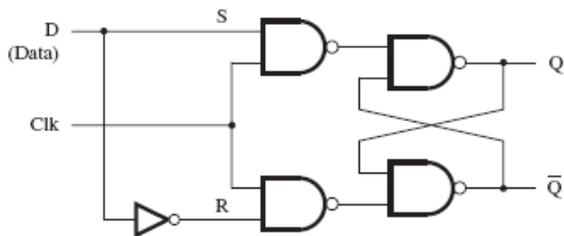
每个周期结束时写入npc, “后写”  
无需写控制信号 (简化, 图4-5)

## 3-8译码

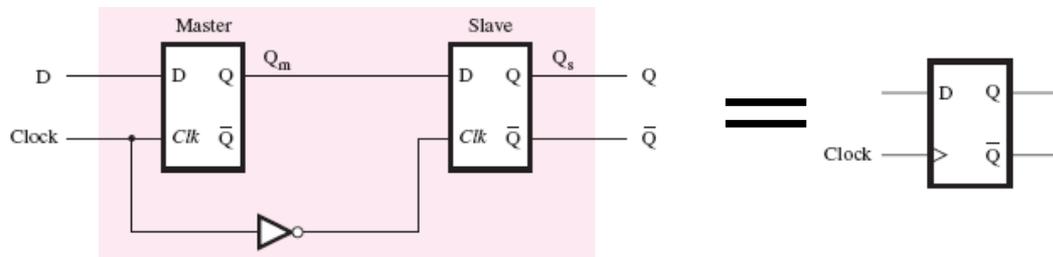


Inputs			Outputs							
12	11	10	Out7	Out6	Out5	Out4	Out3	Out2	Out1	Out0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	1	0	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

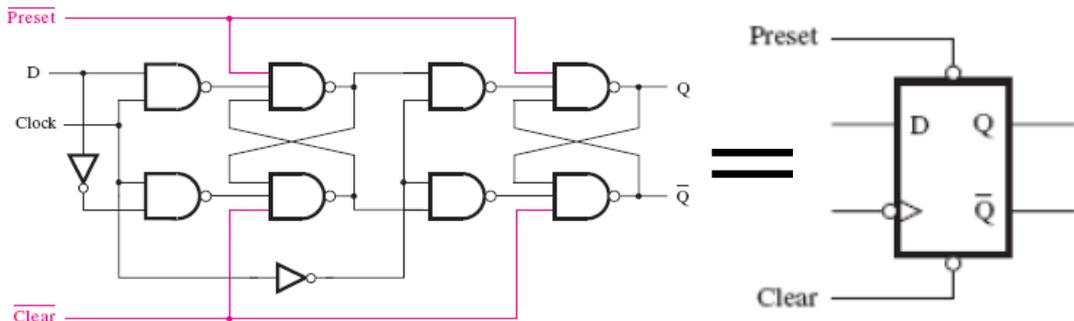
# \$A.8.1: D触发器, 寄存器



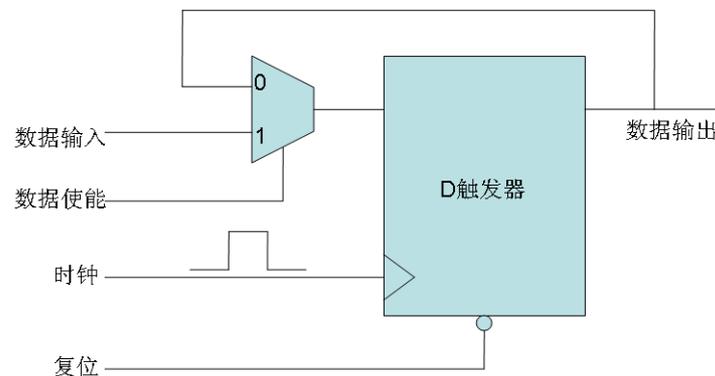
**D 锁存器: clk电平控制**



**Flip-Flops: 一位 D 触发器, clk边沿控制**

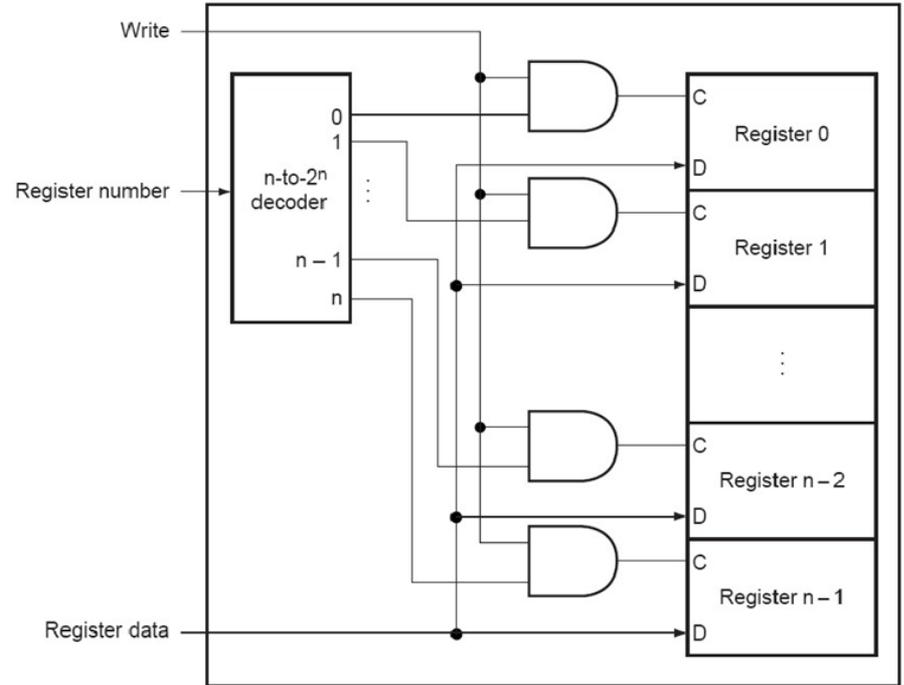
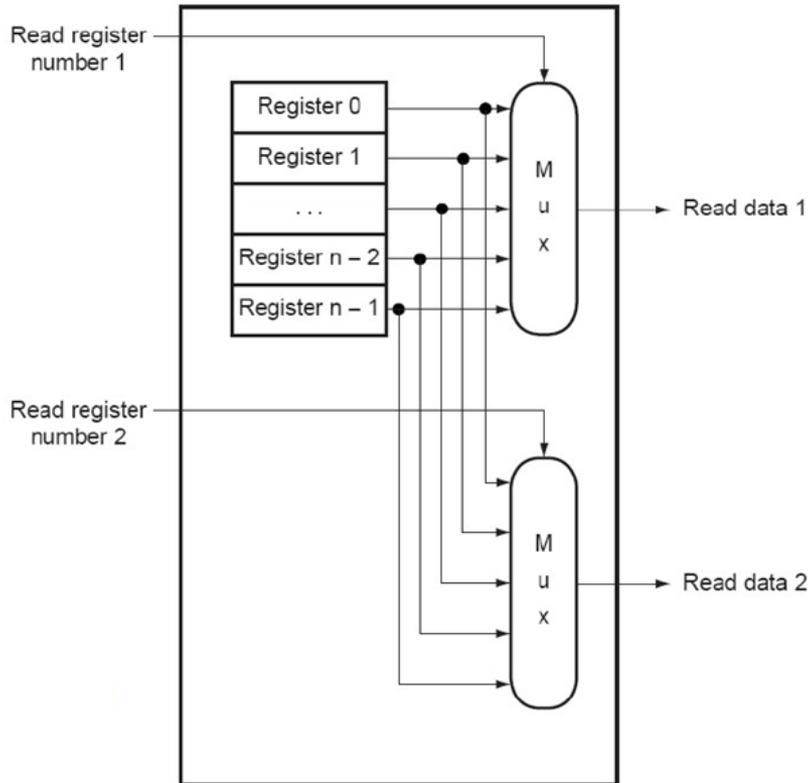


**带清零Clear和预置Preset的D 触发器**

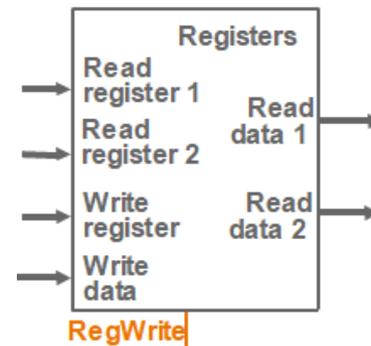


**一位寄存器, 数据使能=write, clk**

# n位RegFile: 图A-8-7, A-8-8, A-8-9

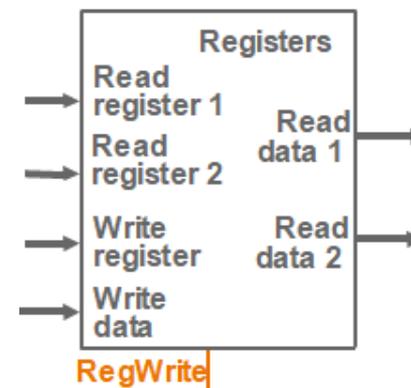
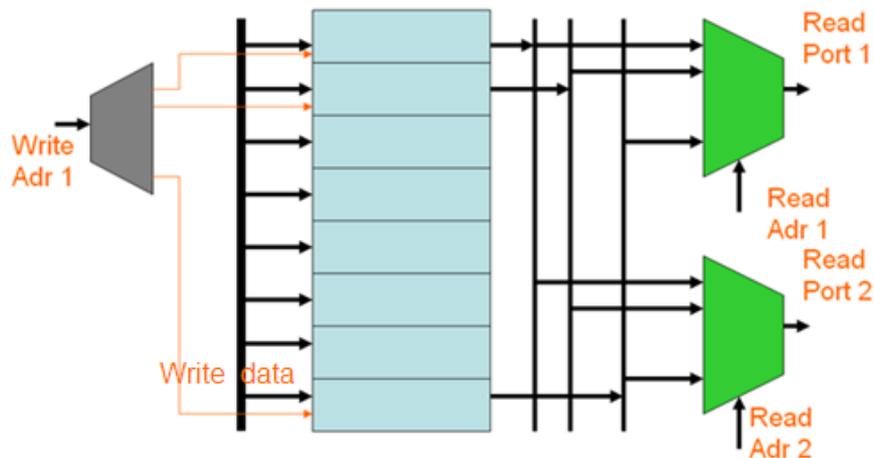


clk、Clear、Preset、write?  
 读操作，写操作  
 “两读一写”，“异步读，同步写”



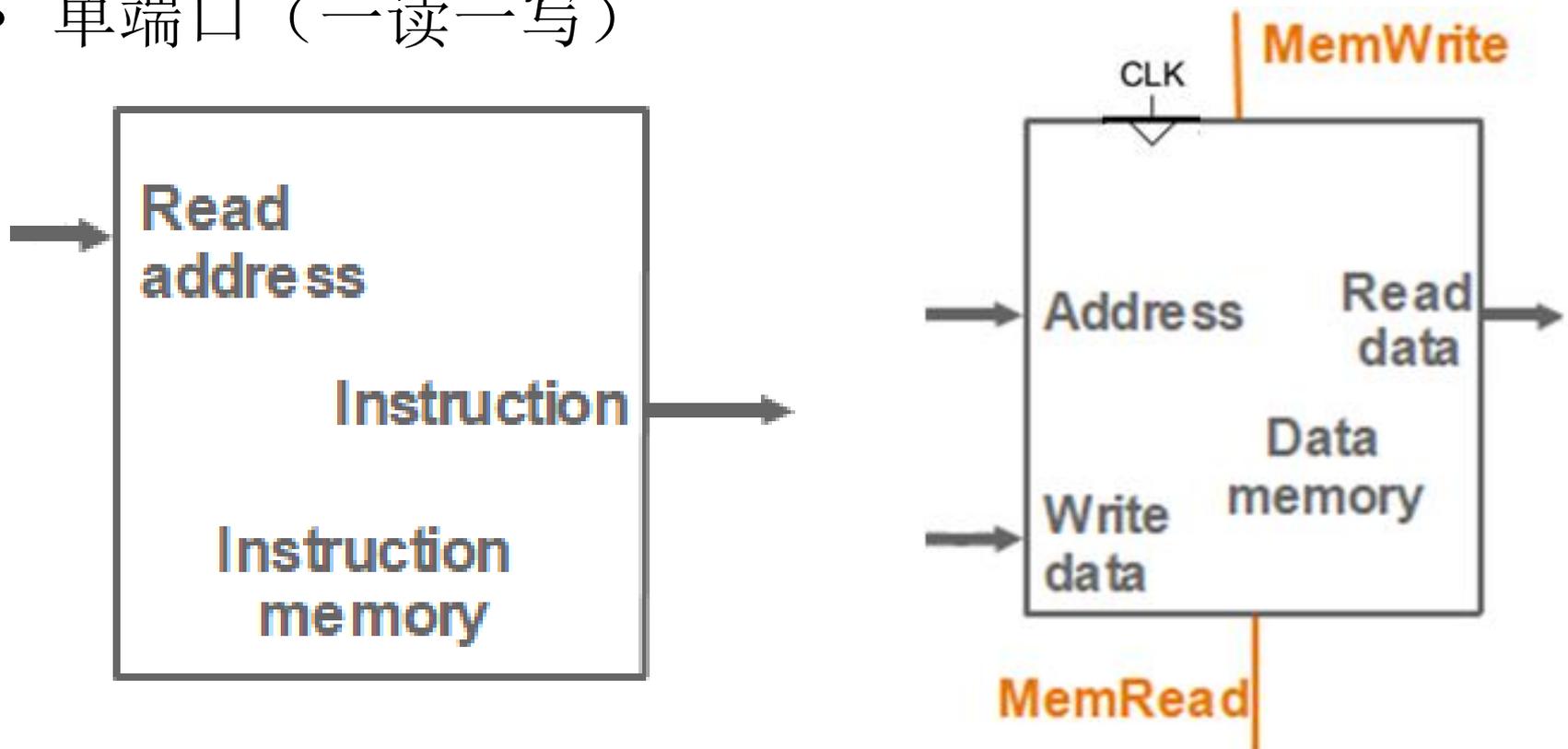
# RegFile读写操作约定：后写，先写

- 不同寄存器读写：在一个周期内，RF支持“两读一写”
  - 写控制RegWrite与clk同步，边沿触发
- 同一寄存器读写：一个周期内同时对同一寄存器读和写
  - 一条指令一个周期内读写同一寄存器：`add $t0, $s2, $t0`
    - 后写（late write）：前半周期读，后半周期写。——单周期用！
      - 先读后写：一个周期内完成读写，但读出的是上一周期写入的值。图4-7
  - 两条指令一个周期内读写同一寄存器：结构冲突
    - 先写（early write）：前半周期写，后半周期读。——流水线用！
      - 先写后读：读出的是前半周期末写入的值。图4-32，图4-50

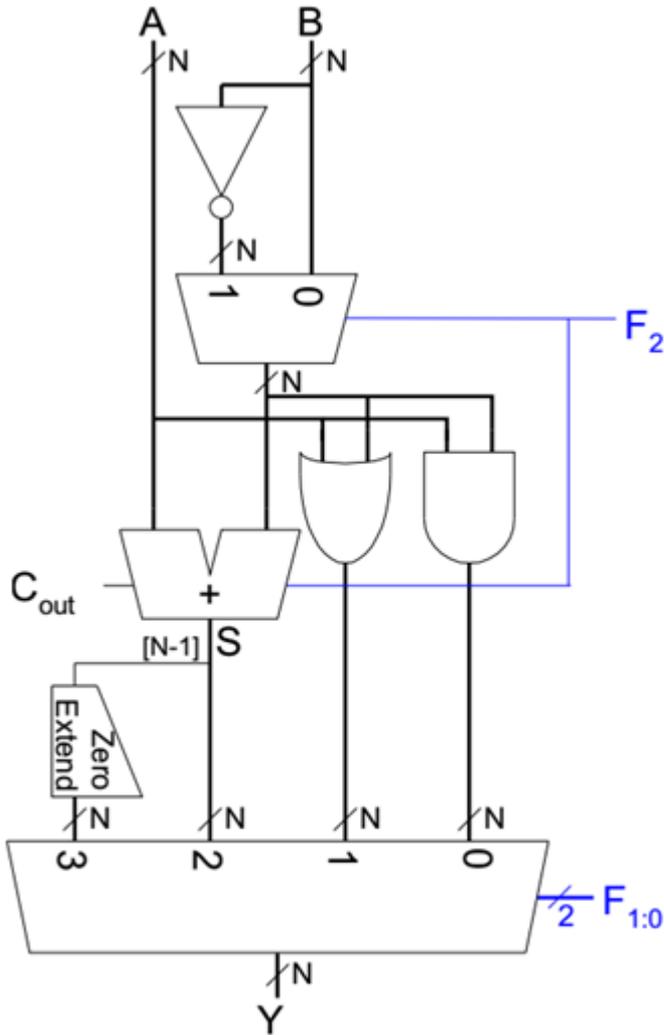


# 访存（mem access, 图4-5, 图4-7）

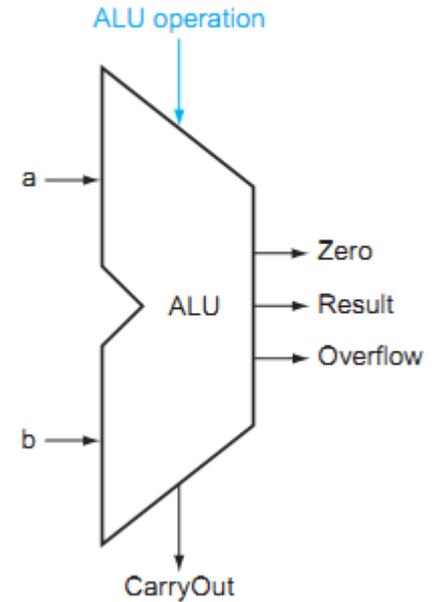
- 读写控制：地址、数据、命令
  - 时钟同步假设：异步读，同步写，1个周期内
  - 读写不能同时：任一时刻只能完成读写之一
- 单端口（一读一写）



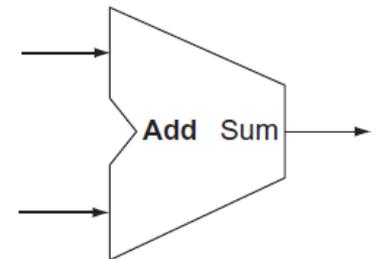
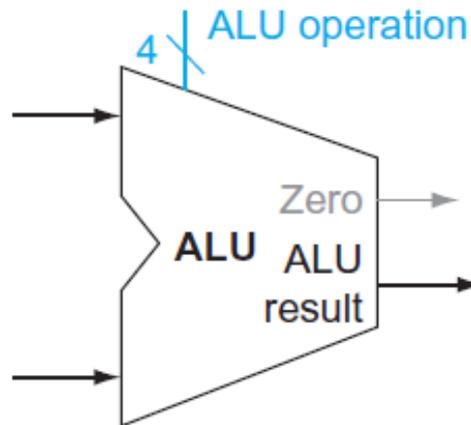
# ALU Interface and Impl: 见A.5



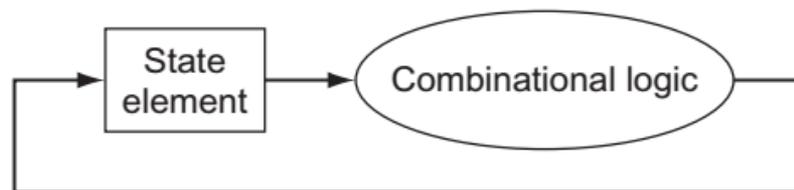
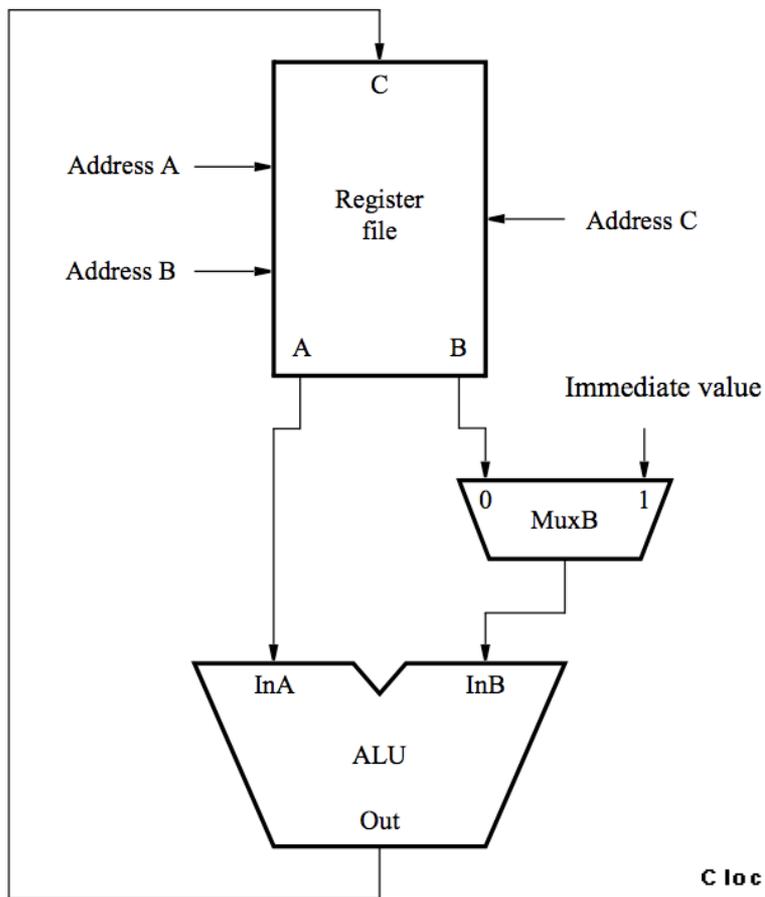
F2	F1	F0	Func.
0	0	0	A & B
0	0	1	A   B
0	1	0	A + B
0	1	1	
1	0	0	A & $\bar{B}$
1	0	1	A   $\bar{B}$
1	1	0	A - B
1	1	1	A < B (slt)



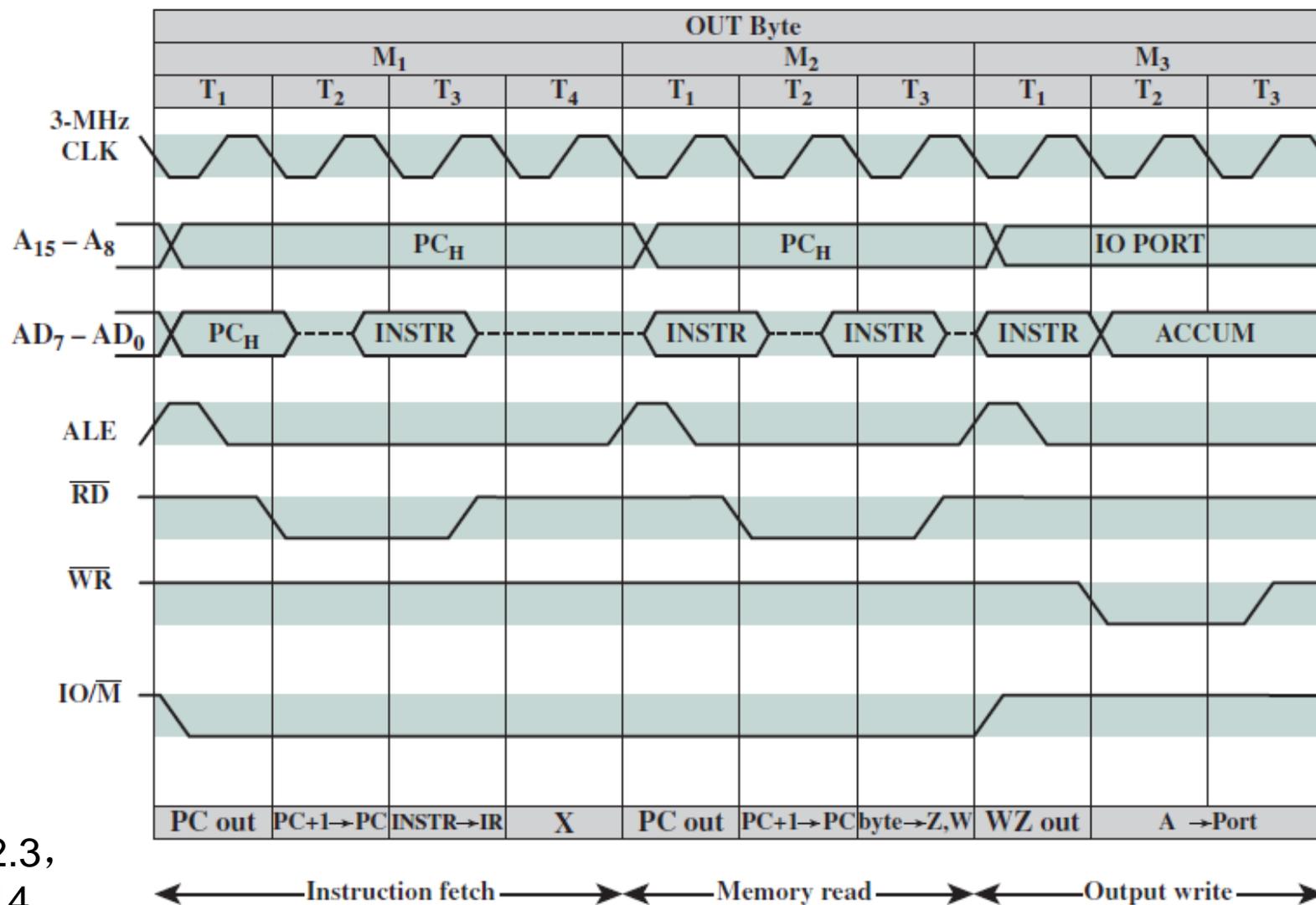
图A-5-14



# 同步电路、clk cycle、控制信号



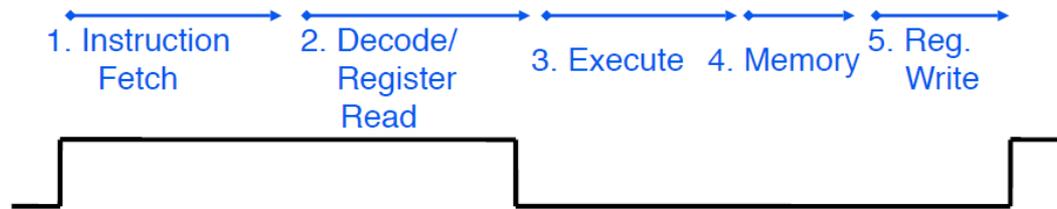
# 时钟周期、机器周期、指令周期



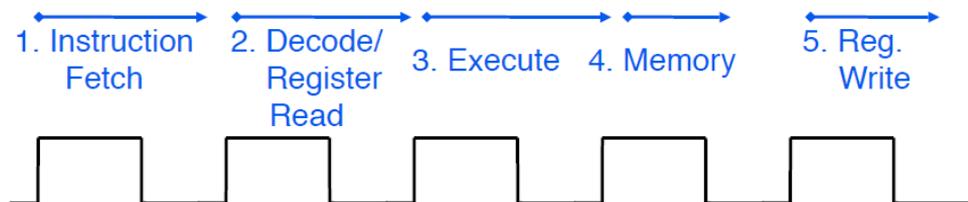
唐9.2.3,  
图9.14

# 指令周期的定时模式

- 单周期实现：1指令周期 = 1 时钟周期
  - All stages of an instruction are completed within one *long* clock cycle.
  - 所需控制信号同时生成



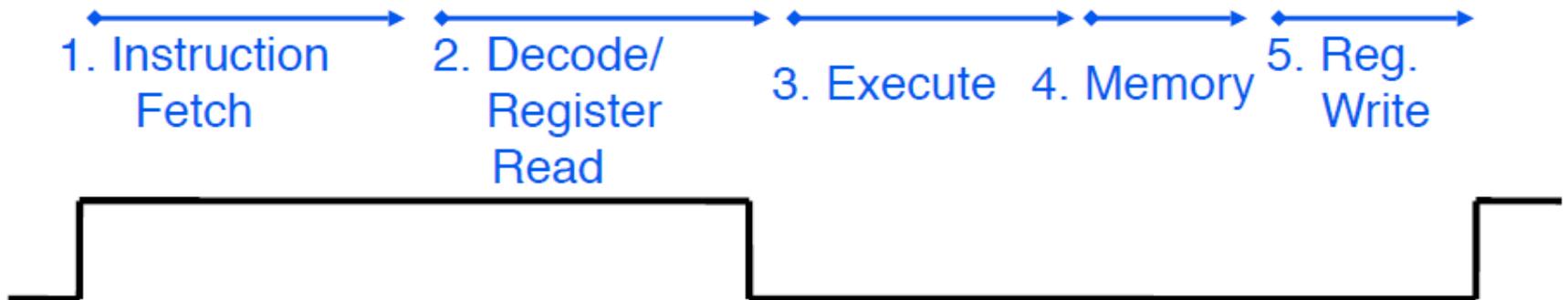
- 多周期实现：指令周期 = n 时钟周期
  - Only one stage of instruction per clock cycle
  - 按时钟周期 (= 机器周期) 生成当前周期所需控制信号



# 单周期数据通路实现

# 定长单周期

- 指令周期：取指、译码、执行、访存、写回
  - **All stages** of an instruction are completed within **one** long clock cycle.
  - 采用时钟边沿触发方式
    - 所有指令在时钟的一个边开始执行，在下一个边结束
    - 控制信号在一个clk内有效



# RV的指令数据通路Overview

- 不同指令执行的多数工作都是相同的
- 公共操作：与**指令类型**无关
  - 取指：将PC送MEM， $nPC = PC + 1$
  - 取数：根据指令字中的地址域读寄存器
    - 非ld/sd：读两个寄存器
    - ld读一个寄存器；sd读两个寄存器
- 特定操作：**同类**指令非常类似
  - 不同类型指令都要使用ALU
    - 算逻指令使用ALU完成计算
    - 访存指令使用ALU计算地址(add)
    - 分支指令使用ALU进行条件比较(add)
  - 其后，各个指令的工作就不同了
    - 算逻指令将ALU结果写回寄存器
    - 访存指令对存储器进行读写
    - 分支指令将基于比较结果修改下一条指令的地址

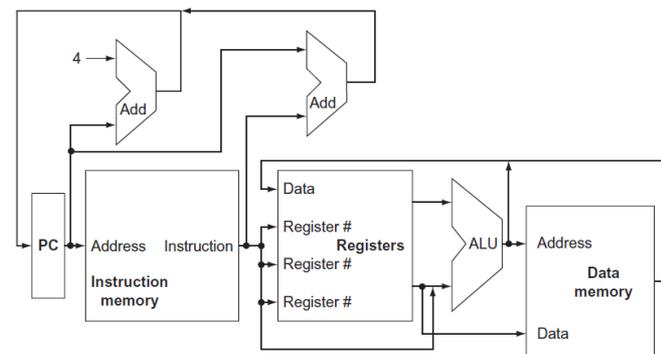
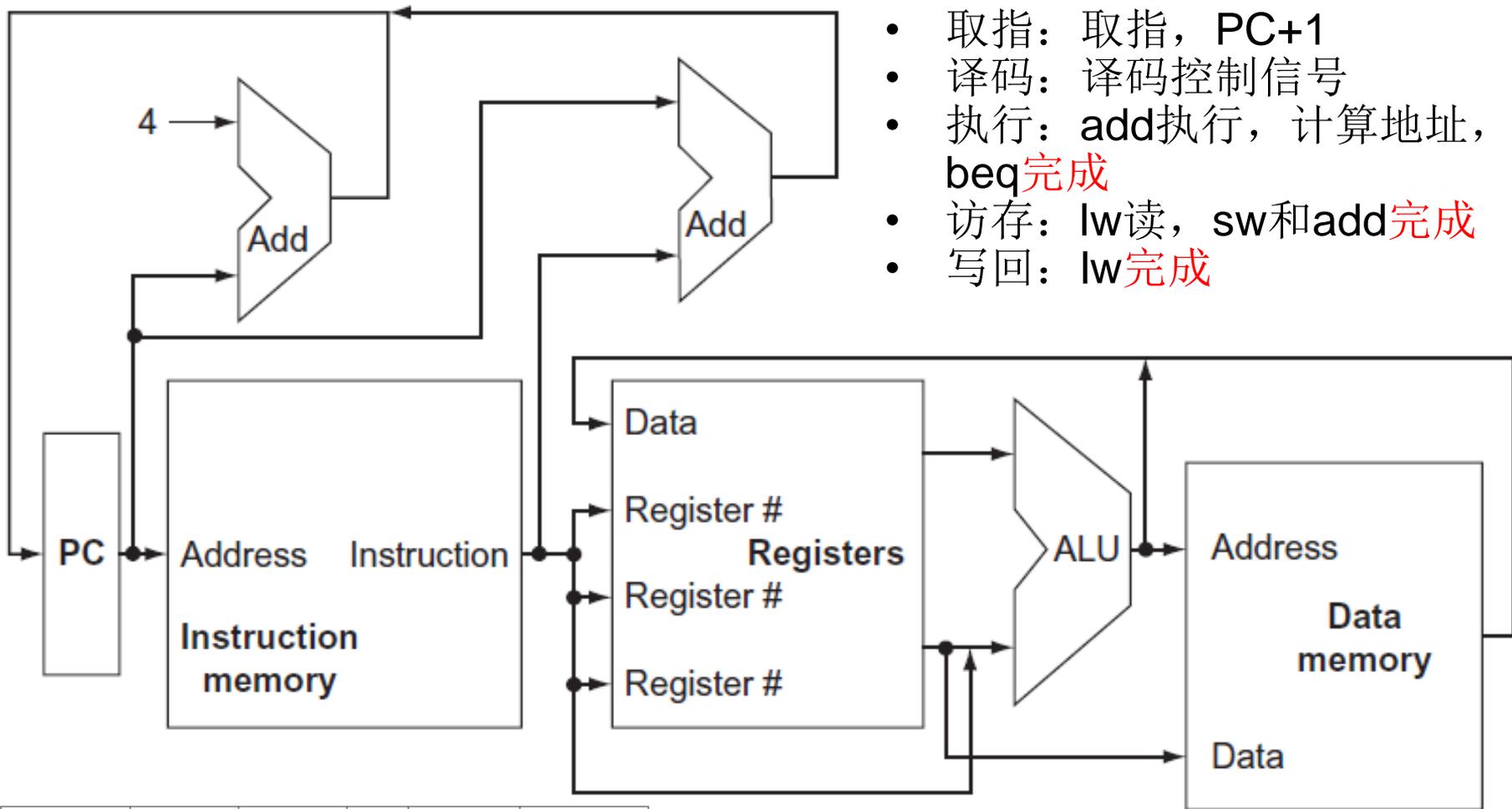


图4-1

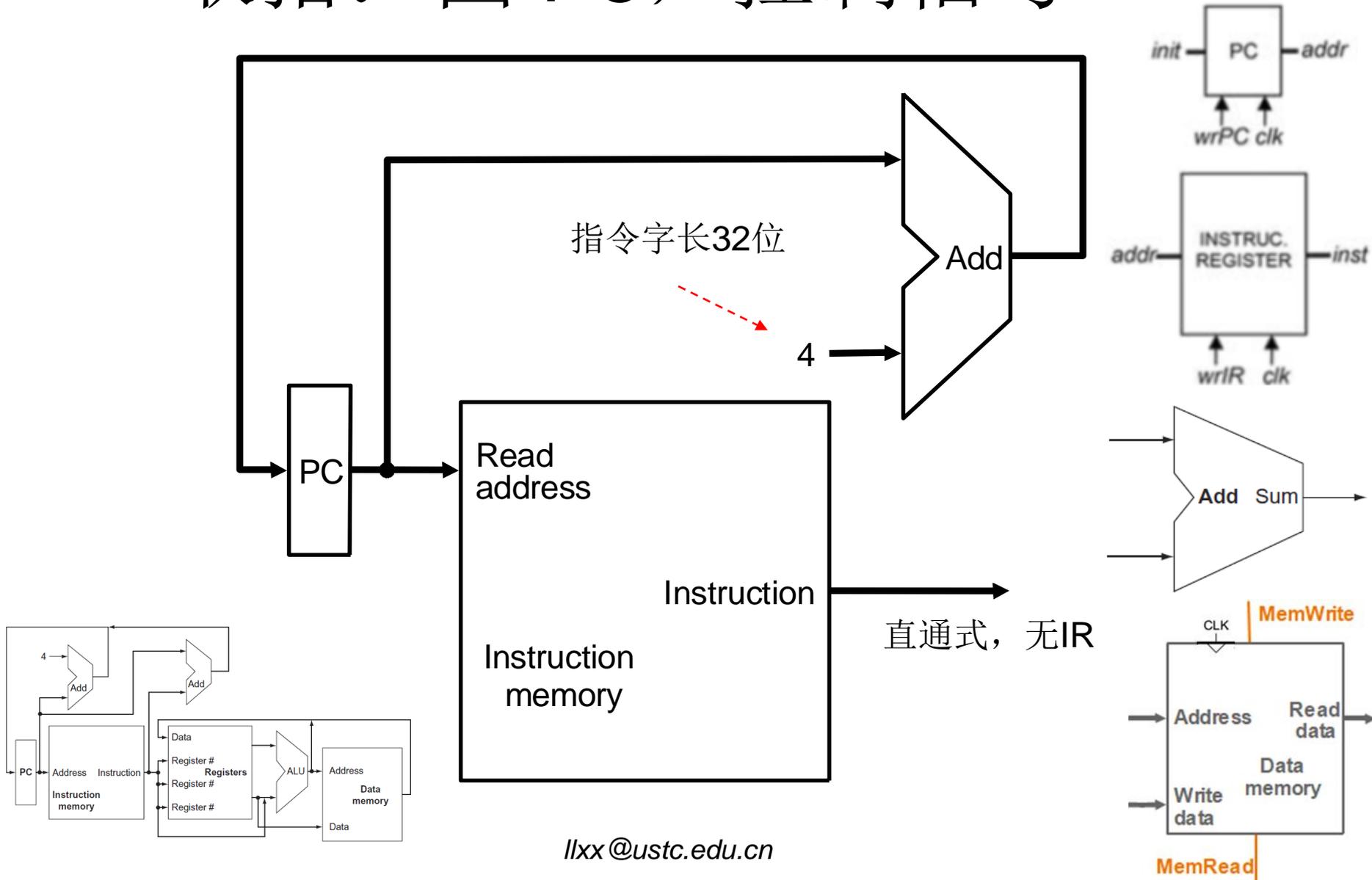
# 单周期数据通路：七大部件，图4-1



- 取指：取指，PC+1
- 译码：译码控制信号
- 执行：add执行，计算地址，**beq完成**
- 访存：lw读，sw和add**完成**
- 写回：lw**完成**

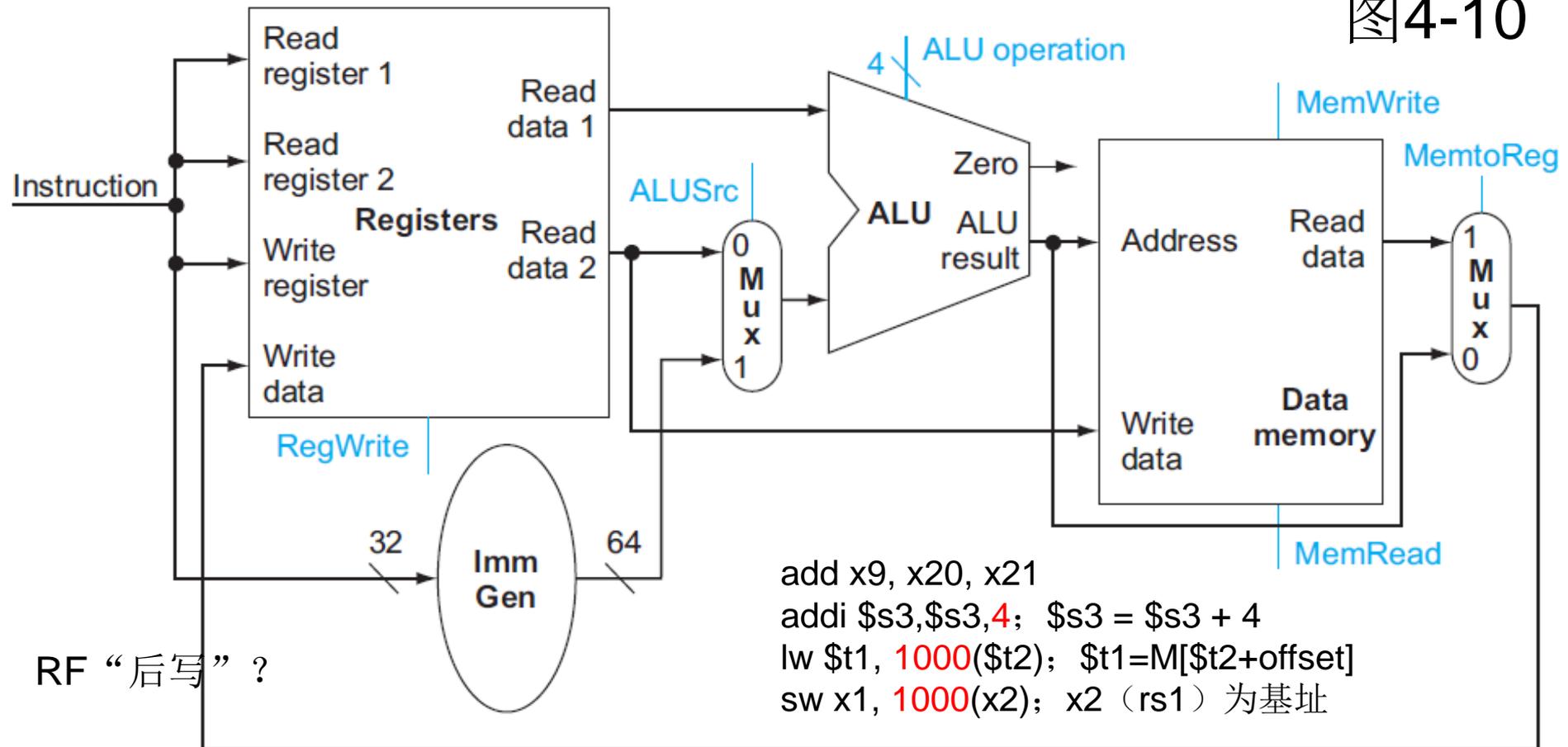
R-type	funct7	rs2	rs1	funct3	rd	opcode
I-type	immediate[11:0]		rs1	funct3	rd	opcode
S-type	immed[11:5]	rs2	rs1	funct3	immed[4:0]	opcode
B-type	immed[12,10:5]	rs2	rs1	funct3	immed[4:1,11]	opcode

# 取指：图4-6，控制信号？



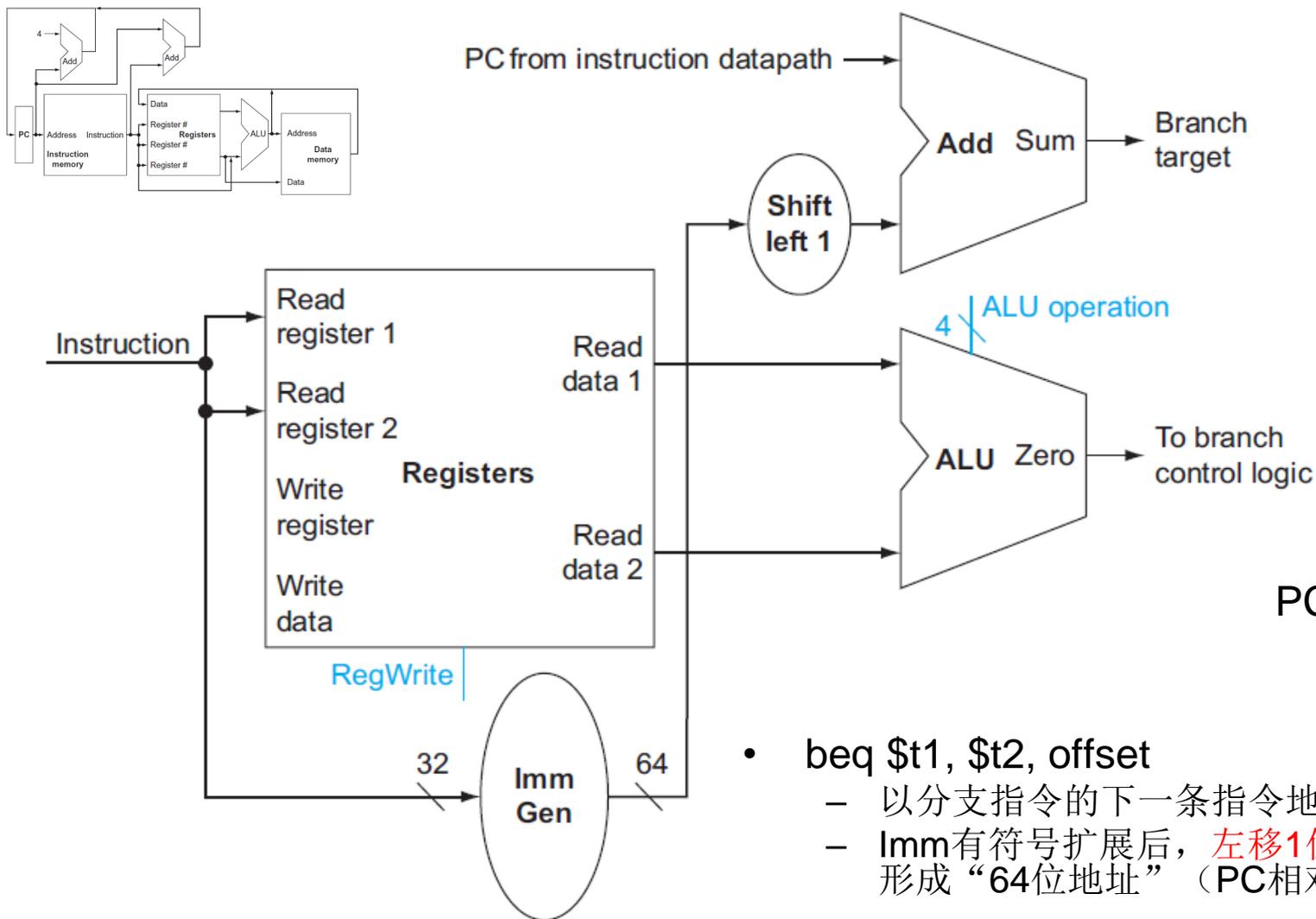
# R/I/S-type的ID/FO/EX/MA/WB

图4-10



funct7	rs2	rs1	funct3	rd	opcode	R-type
immediate[11:0]		rs1	funct3	rd	opcode	I-type
immed[11:5]	rs2	rs1	funct3	immed[4:0]	opcode	S-type

# B-type: 条件转移beq, 图4-9

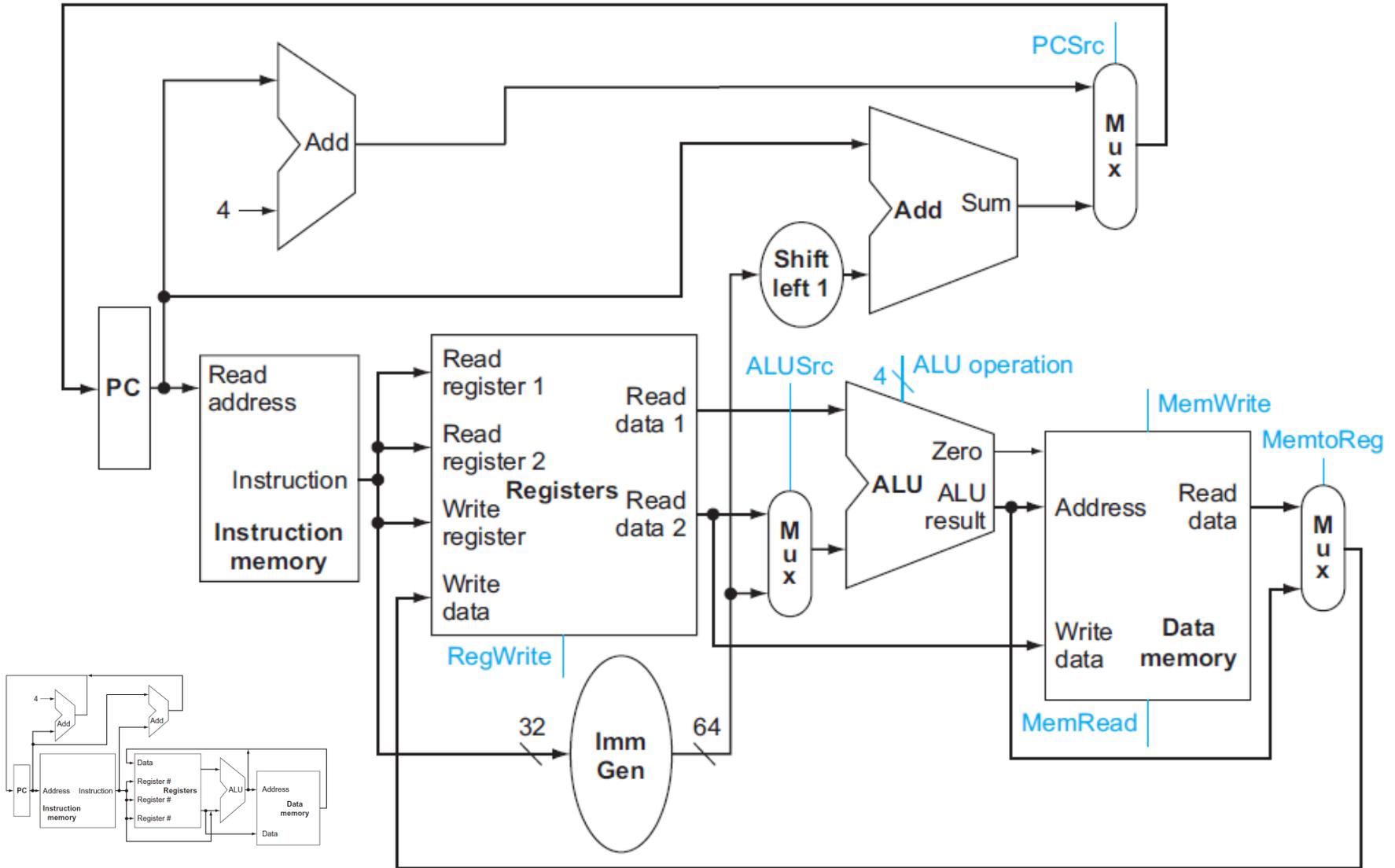


PC “后写” ?

- beq \$t1, \$t2, offset
  - 以分支指令的下一条指令地址nPC为基址
  - Imm有符号扩展后, 左移1位 (半字对齐), 形成“64位地址” (PC相对寻址)

immed[12,10:5]	rs2	rs1	funct3	immed[4:1,11]	opcode
----------------	-----	-----	--------	---------------	--------

# R-/I-/S-/B-type综合：图4-11

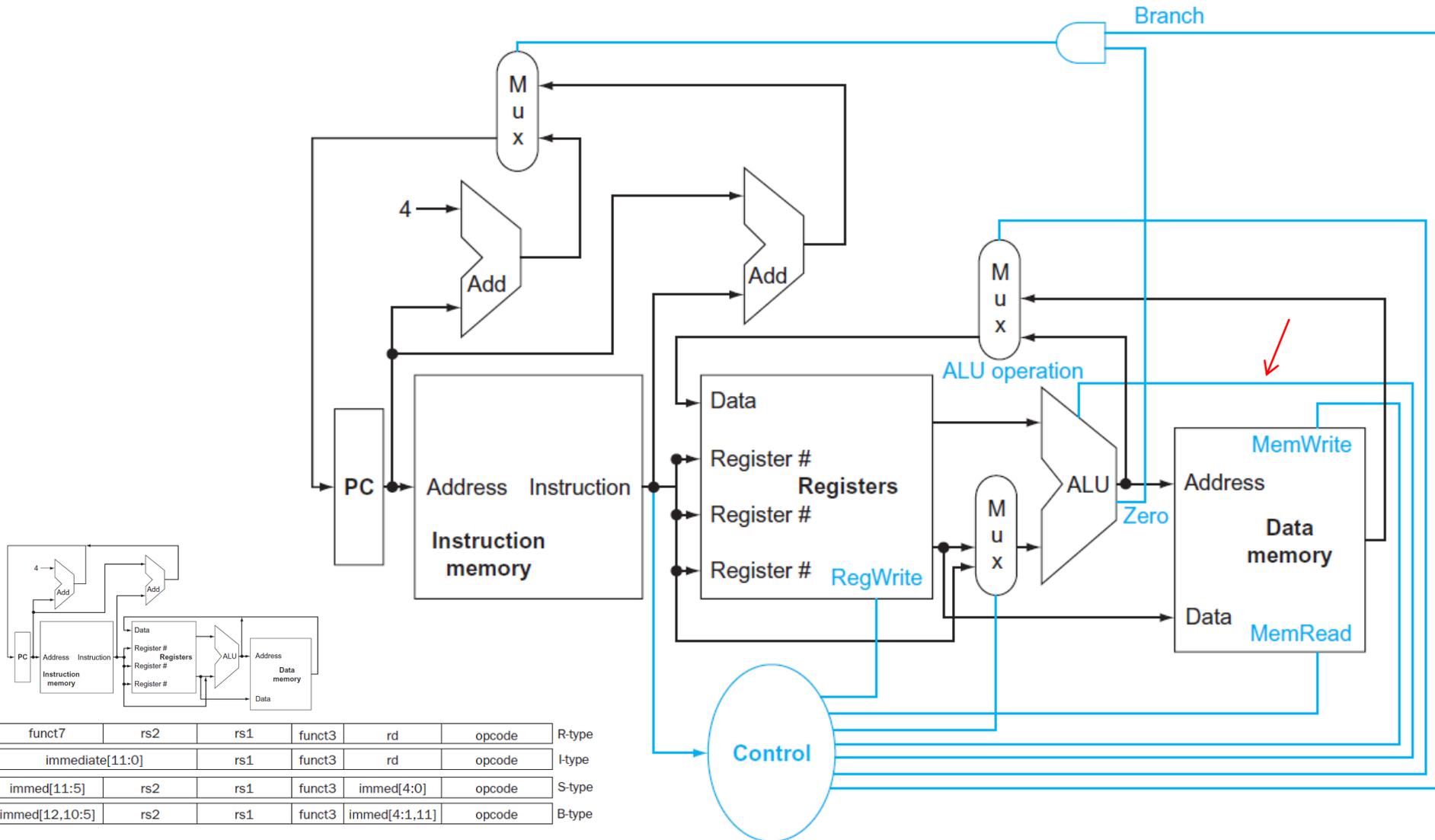


# 单周期控制器实现

ALU控制器

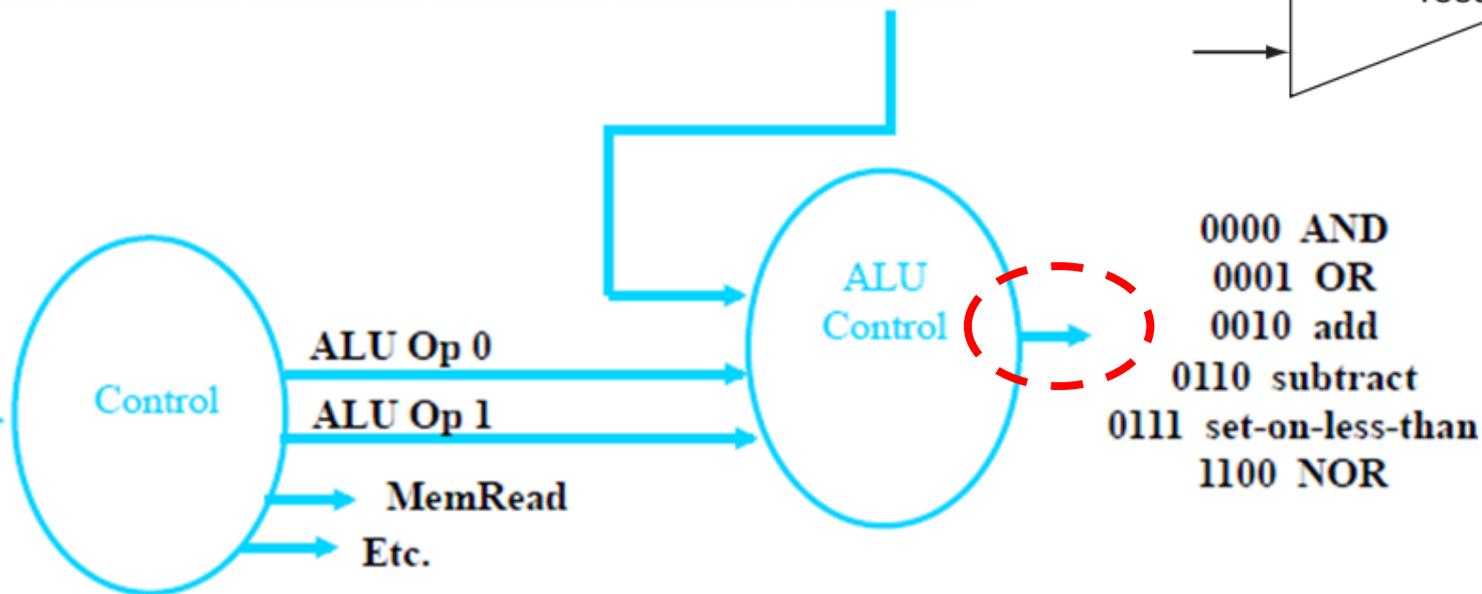
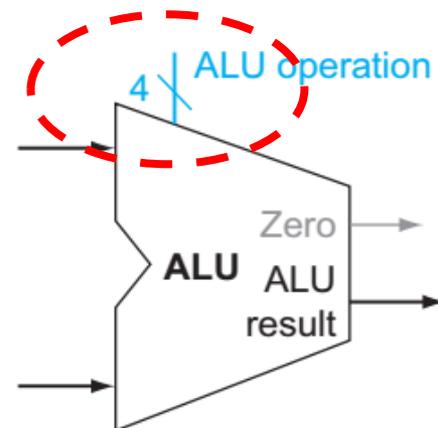
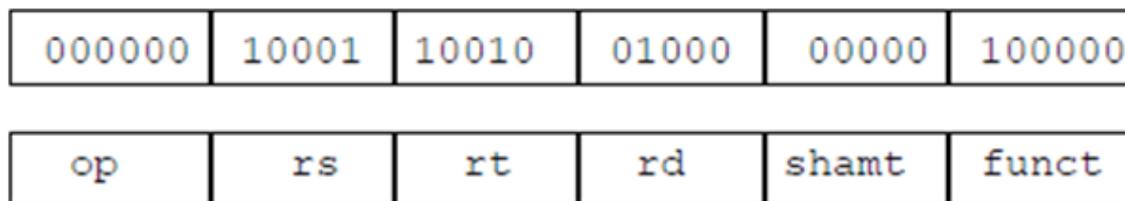
主控制器

# RV数据通路与控制：图4-2

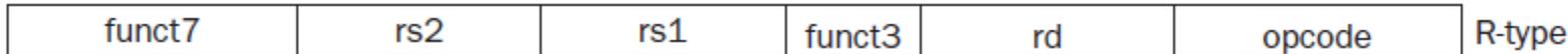


# ALU操作控制：两级译码， §4.4.1

Example: add \$8, \$17, \$18



§4.4.1: “多级控制可以减小主控单元的规模，减小延迟”

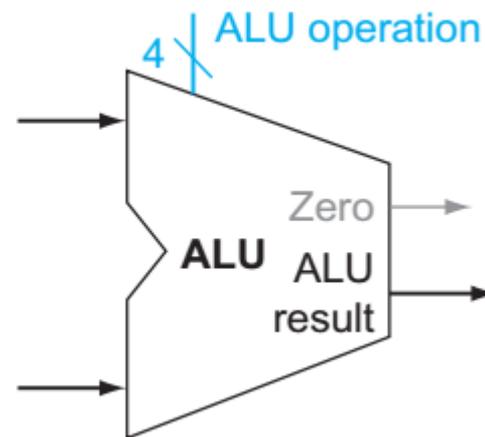


# ALU控制器，图4-12

Instruction opcode	ALUOp	Operation	Funct7 field	Funct3 field	Desired ALU action	ALU control input
ld	00	load doubleword	XXXXXXX	XXX	add	0010
sd	00	store doubleword	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

funct7	rs2	rs1	funct3	rd	opcode	R-type
immediate[11:0]		rs1	funct3	rd	opcode	I-type
immed[11:5]	rs2	rs1	funct3	immed[4:0]	opcode	S-type
immed[12,10:5]	rs2	rs1	funct3	immed[4:1,11]	opcode	B-type

由2位ALUOp（指令op译码产生）和funct7+funct3生成 **ALU\_ctrl\_input**（4位，即ALU operation）



# ALU control block: 组合逻辑控制器

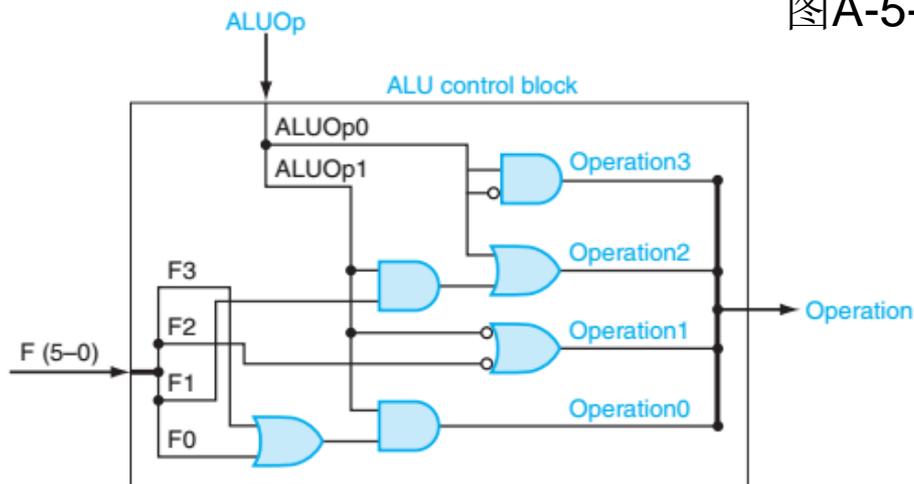
ALUOp		Funct7 field							Funct3 field			Operation
ALUOp1	ALUOp0	I[31]	I[30]	I[29]	I[28]	I[27]	I[26]	I[25]	I[14]	I[13]	I[12]	
0	0	X	X	X	X	X	X	X	X	X	X	0010
X	1	X	X	X	X	X	X	X	X	X	X	0110
1	X	0	0	0	0	0	0	0	0	0	0	0010
1	X	0	1	0	0	0	0	0	0	0	0	0110
1	X	0	0	0	0	0	0	0	0	1	1	0000
1	X	0	0	0	0	0	0	0	0	1	0	0001

图4-13, ALU Operation由ALUOp和Funct译码生成

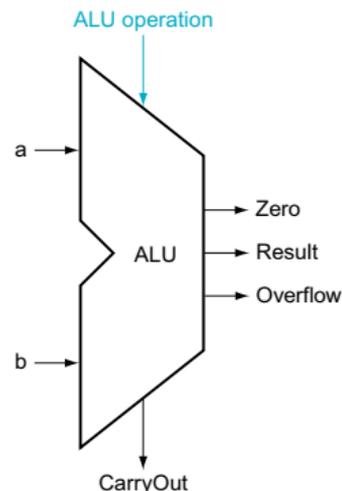
注意: Funct7只有sub的[30]为1, 因此只需[30, 14, 13, 12]位

ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract

图A-5-13



RV, FIGURE C.2.3 (unstructured), 注意: 此为MIPS示例, 与图4-13不对应!



图A-5-14

# ALU控制器：ALUOp与ALU operation

图4-15

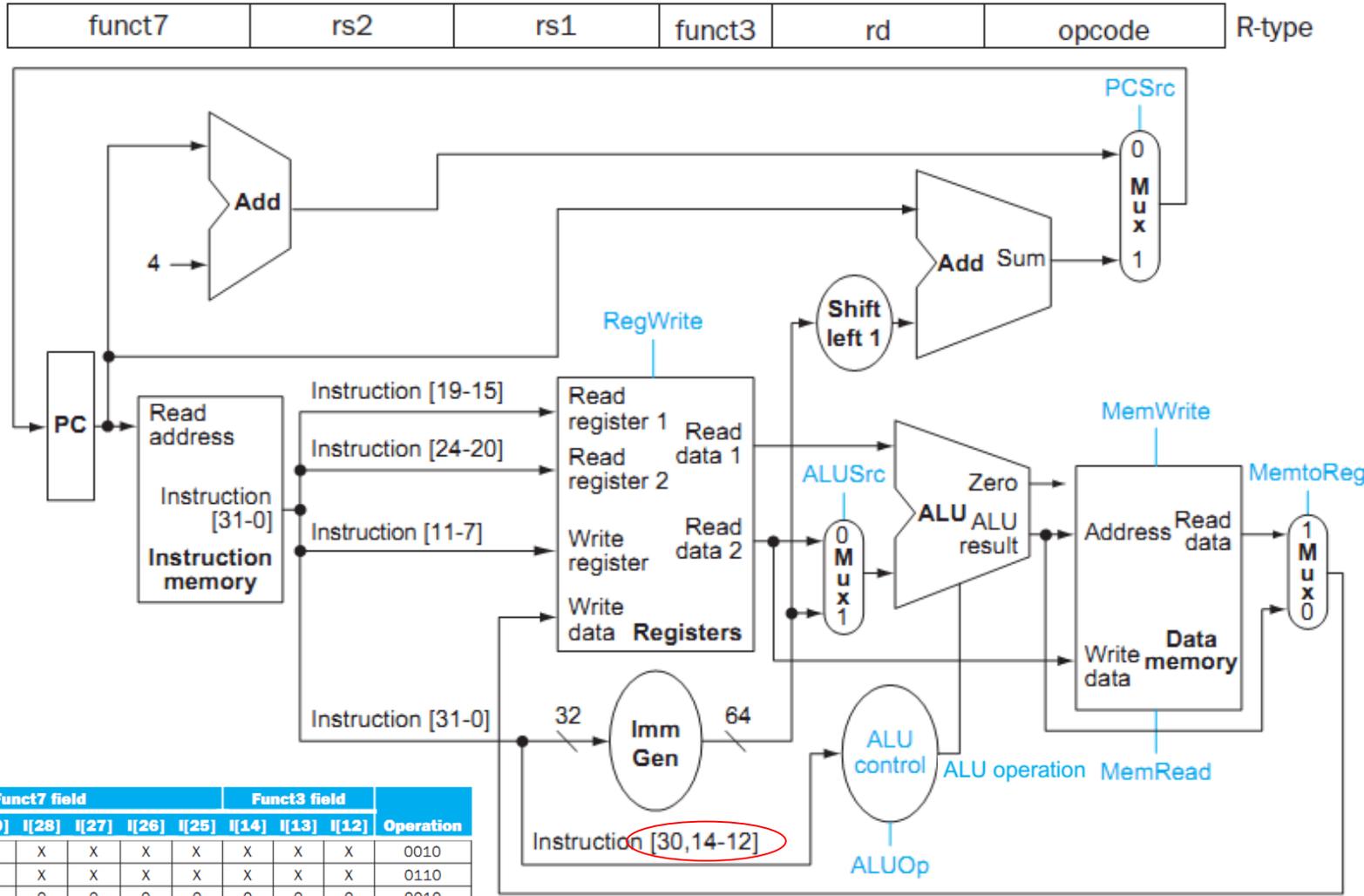


图4-13

ALUOp		Funct7 field					Funct3 field					Operation
ALUOp1	ALUOp0	I[31]	I[30]	I[29]	I[28]	I[27]	I[26]	I[25]	I[14]	I[13]	I[12]	
0	0	X	X	X	X	X	X	X	X	X	X	0010
X	1	X	X	X	X	X	X	X	X	X	X	0110
1	X	0	0	0	0	0	0	0	0	0	0	0010
1	X	0	1	0	0	0	0	0	0	0	0	0110
1	X	0	0	0	0	0	0	0	1	1	1	0000
1	X	0	0	0	0	0	0	0	1	1	0	0001

# 主控制器：控制信号生成

- 控制信号7个： op域译码产生
  - RegWrite: 寄存器写操作控制
  - ALUSrc: ALU的第二个opr来源，二选一
    - RegFile (R-type) 与 Imm (I-/S-/B-type)
  - ALUOp: 2位
  - MemRead: 存储器读控制, lw指令
  - MemWrite: 存储器写控制, sw指令
  - MemtoReg: 目的寄存器数据来源
    - R-type指令与load指令二选一
  - Branch: 分支指令标识 (beq)
    - PCSrc: nPC来源控制, = Branch & Zero
    - If taken, then PC = PC+offset

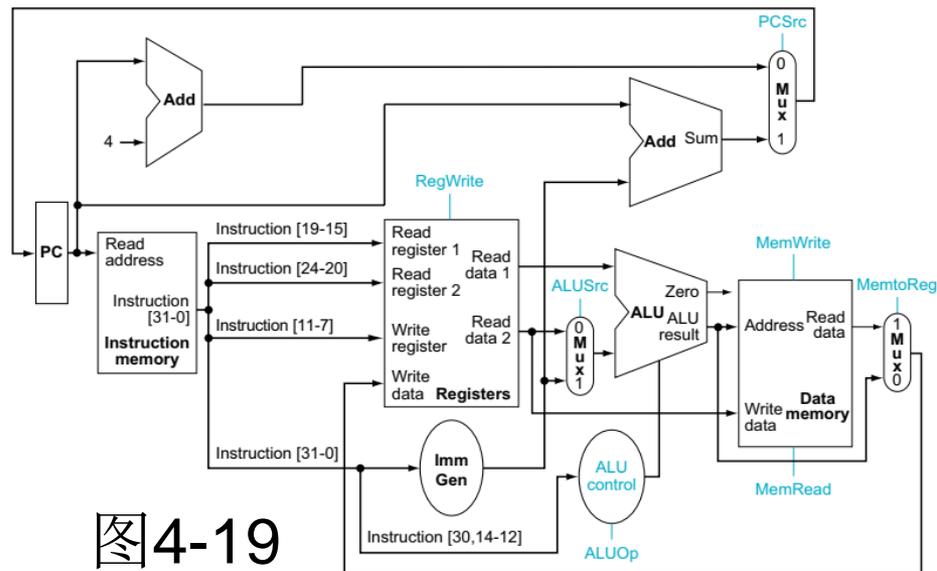
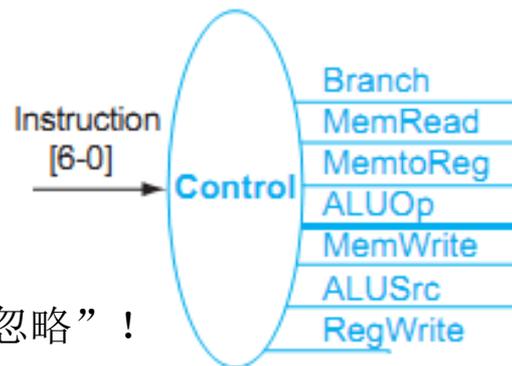


图4-19

- 无“PCWrite”？
  - 图4-19题：“每个周期都要写PC（nPC or beq目的），因此忽略”！
- 控制信号何时有效？持续时间？



# 综合：数据通路+主控制器

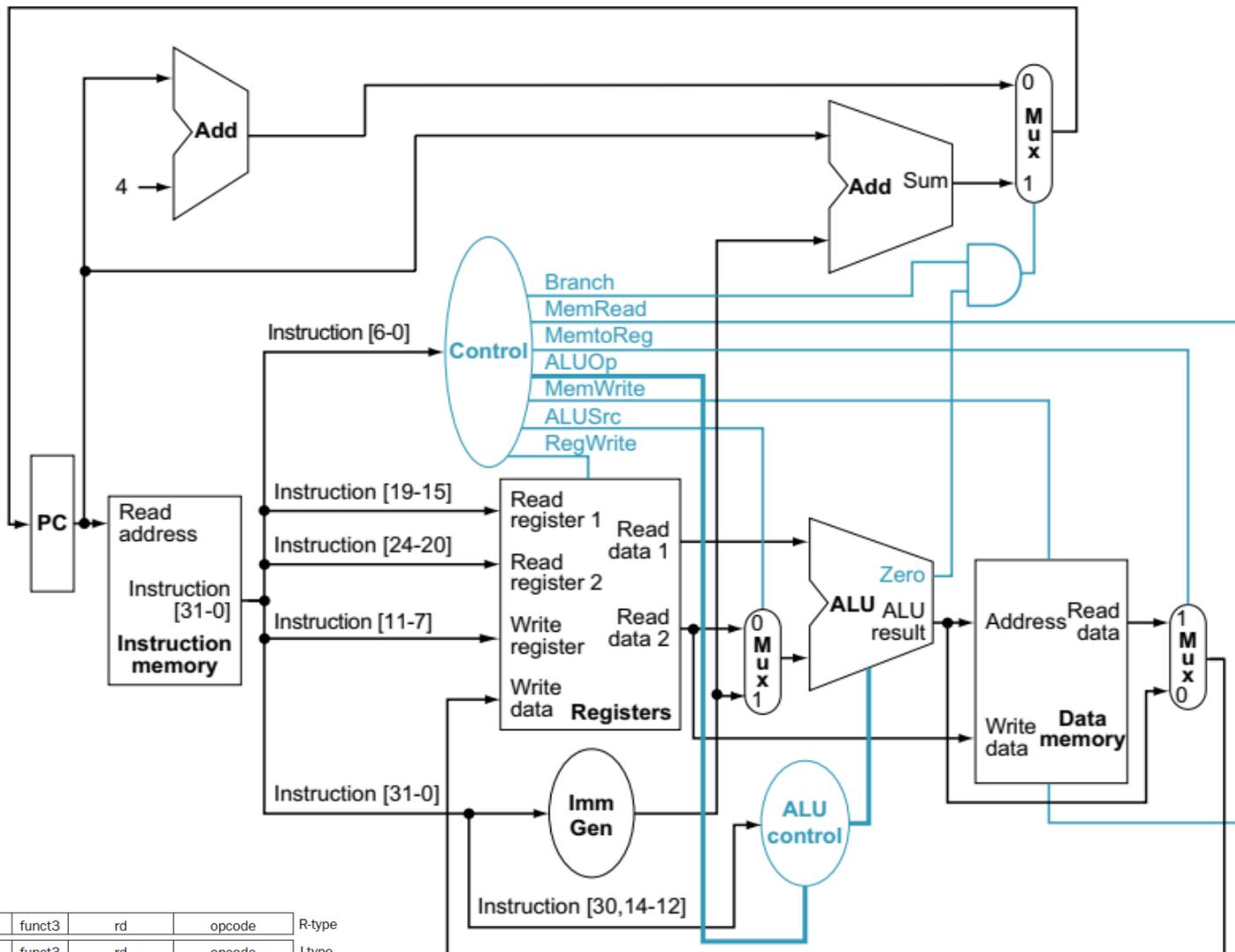


图4-21

funct7	rs2	rs1	funct3	rd	opcode	R-type
immediate[11:0]		rs1	funct3	rd	opcode	I-type
immed[11:5]	rs2	rs1	funct3	immed[4:0]	opcode	S-type
immed[12,10:5]	rs2	rs1	funct3	immed[4:1,11]	opcode	B-type

# 主控制器真值表

Input or output	Signal name	R-format	ld	sd	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

funct7	rs2	rs1	funct3	rd	opcode	R-type
immediate[11:0]		rs1	funct3	rd	opcode	I-type
immed[11:5]	rs2	rs1	funct3	immed[4:0]	opcode	S-type
immed[12,10:5]	rs2	rs1	funct3	immed[4:1,11]	opcode	B-type

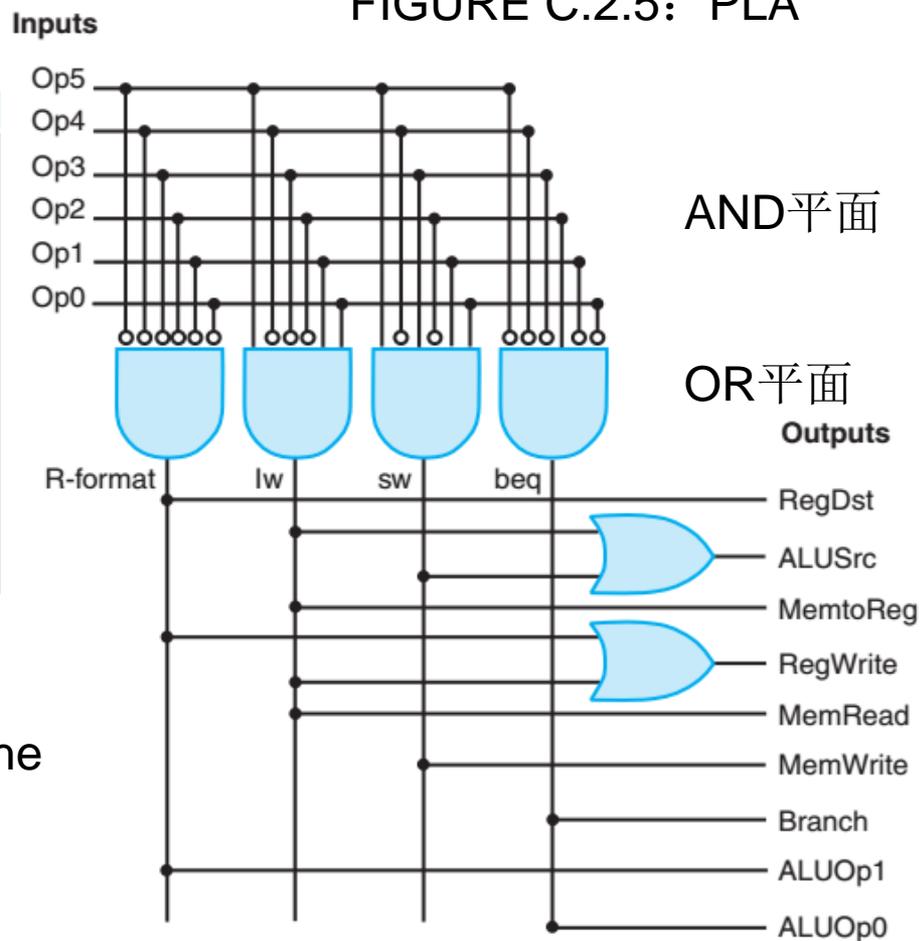
图4-26, 布尔表达式? FSM?

# 主控制器实现：组合逻辑，one-clock impl

FIGURE C.2.4: MIPS

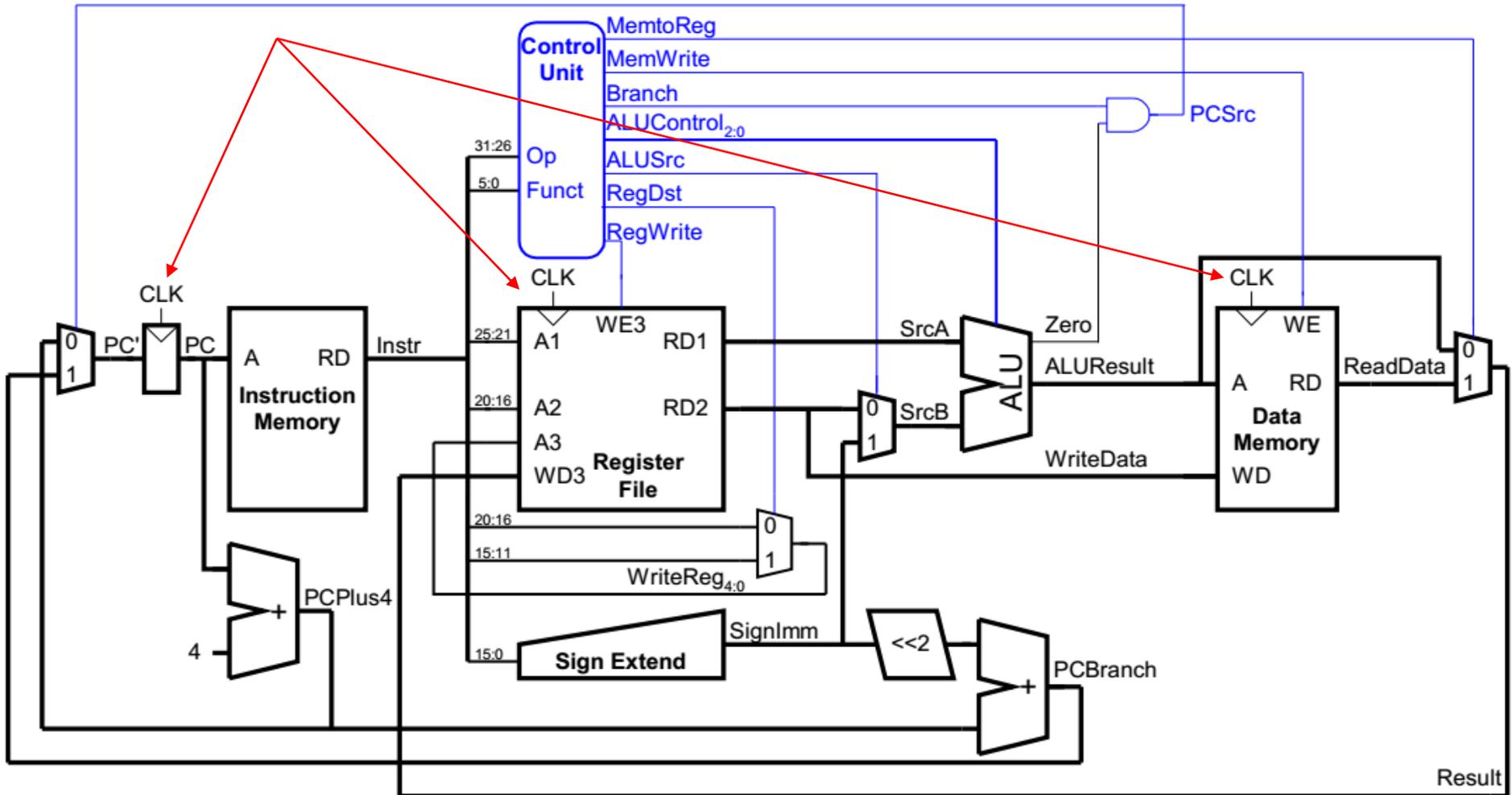
Control	Signal name	R-format	lw	sw	beq
Inputs	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Outputs	RegDst	1	0	X	X
	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
ALUOp0	0	0	0	1	

FIGURE C.2.5: PLA



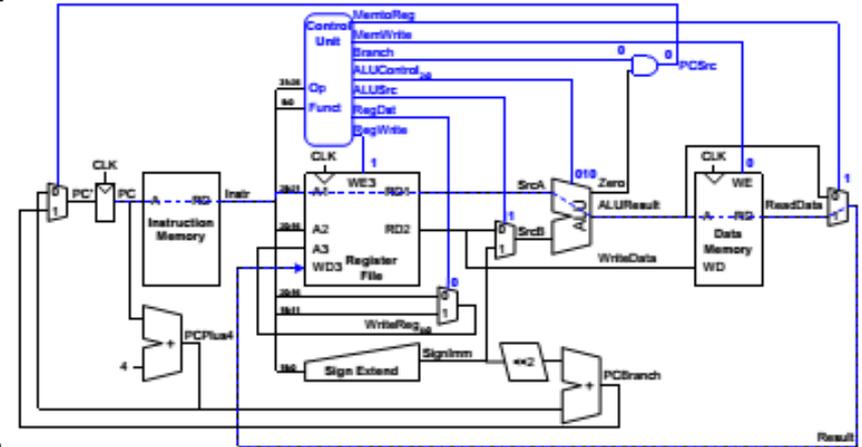
PLA: The **structured** implementation of the control function !

# Single-Cycle Processor: Clocking



# The Critical Path Dictates the Clock Period

Element	Delay
Register clk-to-Q	$t_{pcq-PC}$ 30 ps
Register setup	$t_{setup}$ 20
Multiplexer	$t_{mux}$ 25
ALU	$t_{ALU}$ 200
Memory Read	$t_{mem}$ 250
Register file read	$t_{RFread}$ 150
Register file setup	$t_{RFsetup}$ 20



$$\begin{aligned}
 T_C &= t_{pcq-PC} + t_{mem-I} + t_{RFread} + t_{ALU} + t_{mem-D} + t_{mux} + t_{RFsetup} \\
 &= (30 + 250 + 150 + 200 + 250 + 25 + 20) \text{ ps} \\
 &= 925 \text{ ps} \\
 &= 1.08 \text{ GHz}
 \end{aligned}$$

关键路径：数据通路最长者？  
访存与ALU时延接近？

# 单周期：指令周期定长or不定长？

- 设程序中load有24%，store有12%，R-type有44%，beq有18%，jump有2%。试比较时钟定长单周期实现和不定长单周期实现的性能。

– 程序执行时间 = 指令数 × CPI × 时钟宽度

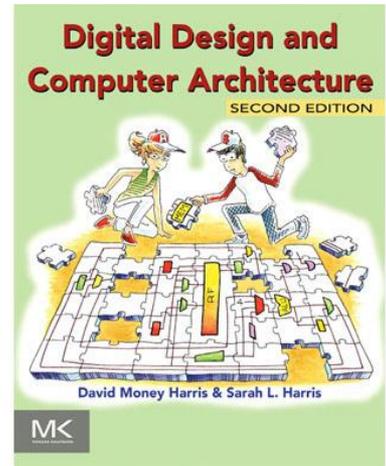
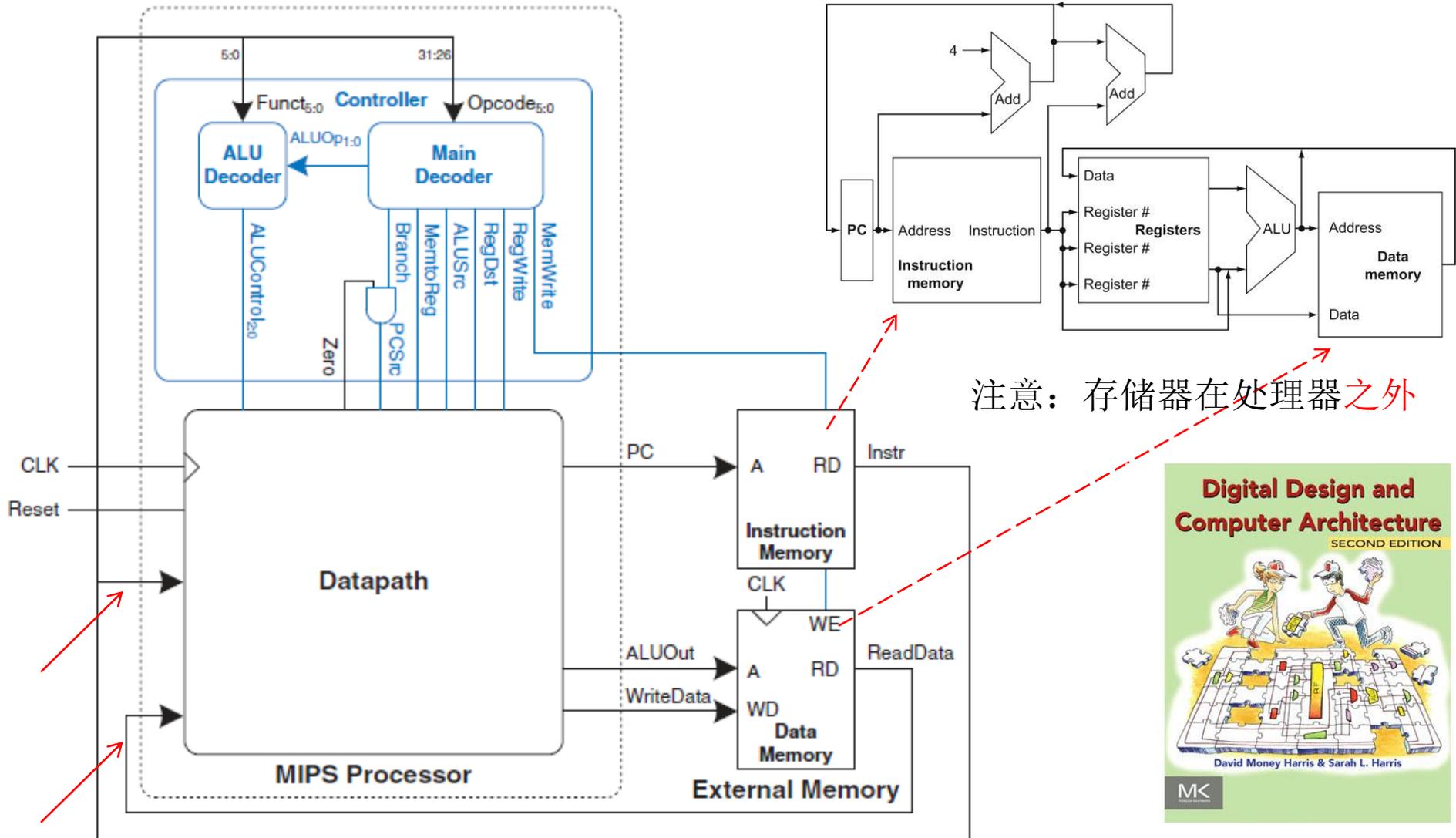
– 定长单周期的时钟宽度为8ns

– 不定长单周期的时钟宽度可以是2ns~8ns。

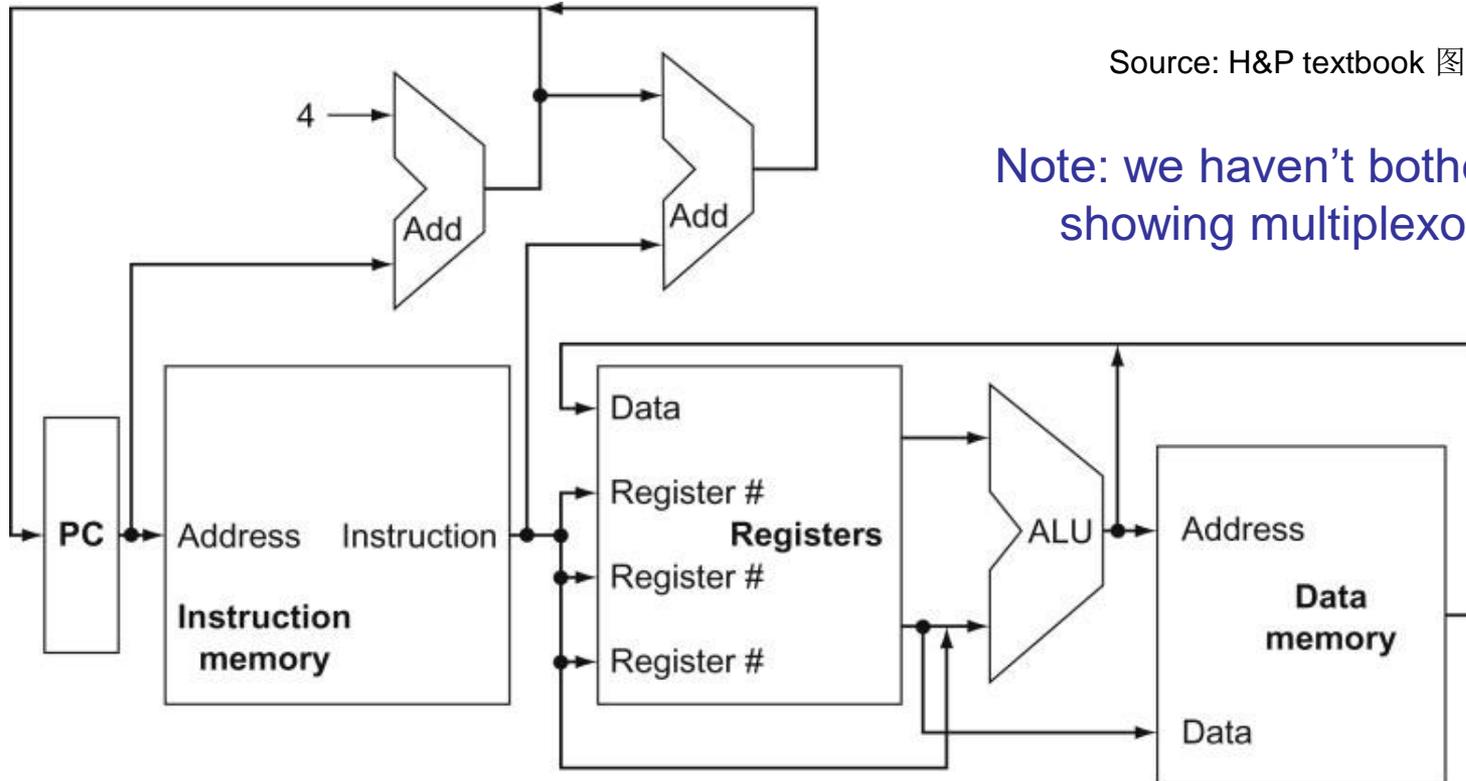
- 程序平均指令执行时间 =  $8 \times 24\% + 7 \times 12\% + 6 \times 44\% + 5 \times 18\% + 2 \times 2\% = 6.3\text{ns}$

– 因此，变长实现较定长实现快  $8/6.3 = 1.27$  倍

# single-cycle processor interfaced to external memory



# View from 30,000 Feet



Source: H&P textbook 图4-1

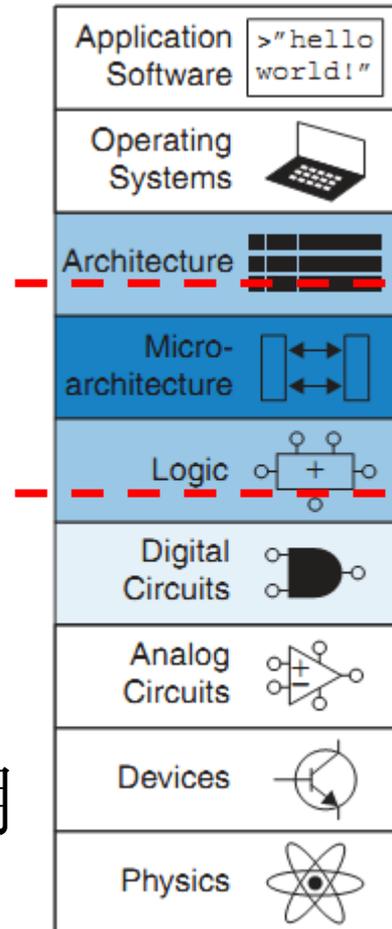
Note: we haven't bothered showing multiplexors

- What is the role of the Add units?
- Explain the inputs to the data memory unit
- Explain the inputs to the ALU
- Explain the inputs to the register unit

- Clock cycle的宽度?
- 能否采用合体MEM?
- 能否不用加法器?
- 寻址方式?
- CPI = ?

# μArch设计方法：4步法

- 数据通路：确定ISA各指令的数据通路
  - 微操作分解
    - 取指、执行
    - 取指、译码、执行
    - 取指、译码、间址、取数、执行、访存、写回
    - 取指、译码、执行、访存、写回
  - 完成各个微操作所需的功能部件
- 控制器：罗列各部件的控制信号
- 定时：指令周期，机器周期，时钟周期
- 综合：产生总体数据通路和控制器



# 实现jalr (I-type) 和jal (J-type) ?

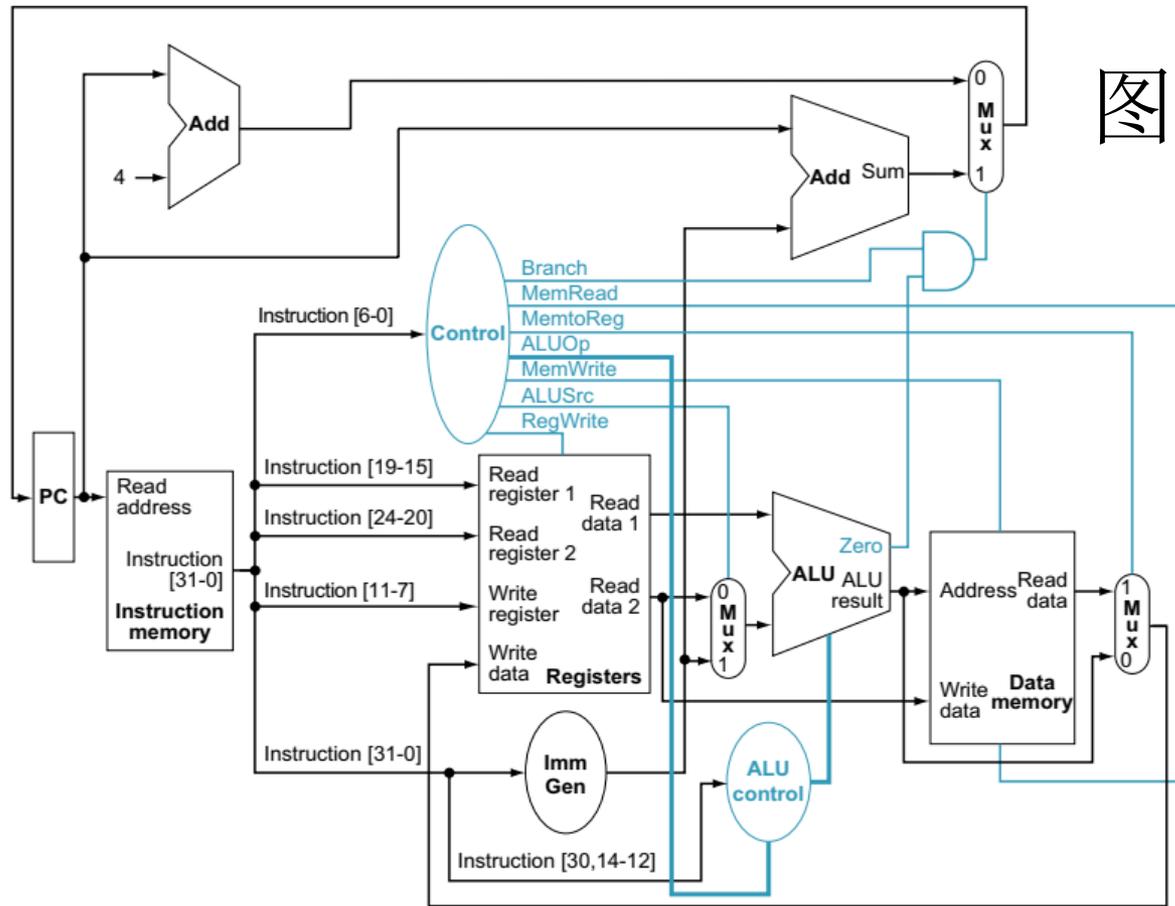


图4-17

微操作?  
寻址方式?  
数据通路?  
控制器?

Unconditional branch	Jump and link	<code>jal x1, 100</code>	<code>x1 = PC+4; go to PC+100</code>	PC-relative procedure call
	Jump and link register	<code>jalr x1, 100(x5)</code>	<code>x1 = PC+4; go to x5+100</code>	Procedure return; indirect call

<code>imm[20]</code>	<code>imm[10:1]</code>	<code>imm[11]</code>	<code>imm[19:12]</code>	<code>rd</code>	<code>opcode</code>	J-type
	<code>imm[11:0]</code>	<code>rs1</code>	<code>funct3</code>	<code>rd</code>	<code>opcode</code>	I-type

# 思考，作业

- 本书有哪些逻辑设计惯例（约定）？
  - 功能部件，时钟方法（clocking methodology）
- 单周期：一个周期内完成指令的所有微操作
  - 寻址方式是如何实现的？
  - 周期宽度如何确定？
  - 能否“在一个clk内完成”？
  - 能否将两个adder合二为一？
  - 能否将两个memory合二为一？
- 控制逻辑有哪些实现方式？
- 作业
  - 4.1, 4.7, C.2
  - 参考图4-22，画出主控制器的PLA实现图

Thank You