The Processor Implementation: Datapath & Control

“Computer Organization & Design ”

第四章

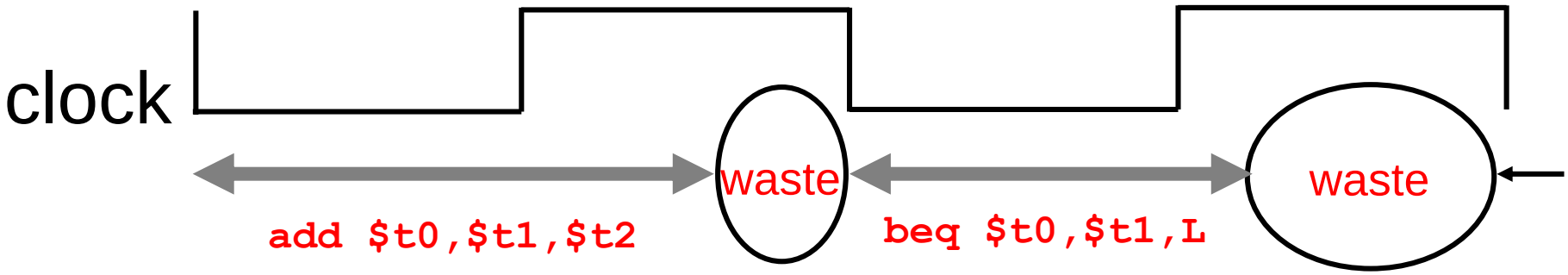
内容提要

- 多周期设计优势：CISC采用
  - 时钟周期对性能的影响
  - 复用功能部件，减少硬件开销
- 多周期实现
  - 数据通路，控制器（状态机，微程序）

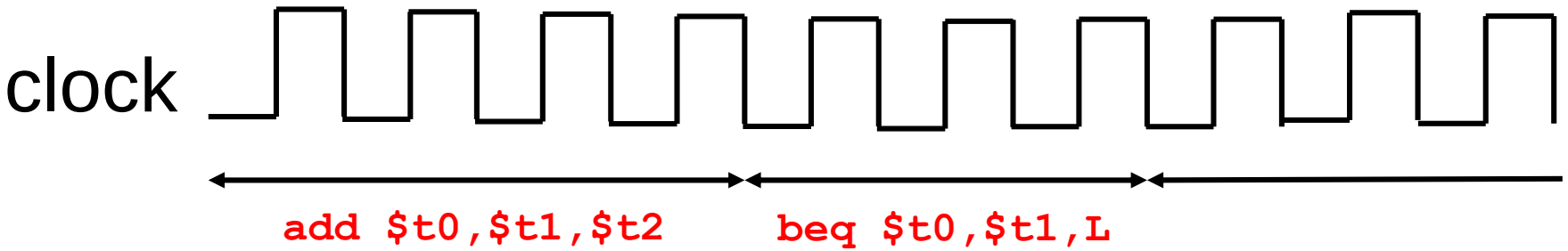
- COD5，RV Edition_2ed，$4.5
- 唐本

指令周期：single-cycle vs. multicycle

Single-cycle Implementation：定长指令周期，定长CC



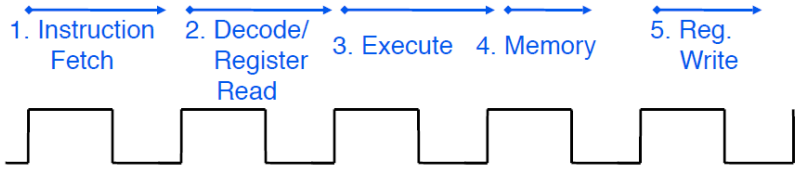
Multicycle Implementation：不定长指令周期，定长CC



- **Multicycle Implementation:**
  less waste = higher performance

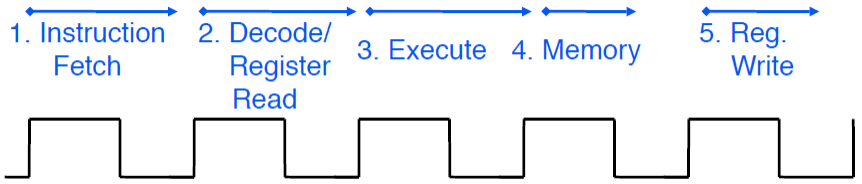
多周期实现(COD5-RV32ed$4.5)

- 将指令执行过程划分成多个阶段
  - 指令周期=n个机器周期
- 每个阶段（机器周期）一个时钟周期
  - 指令周期=n个机器周期=n个时钟周期
  - 在一个周期内的各个部件并行工作
    - 只有控制信号有效的部件作有用功！
- 时钟周期定长：假设一个时钟周期内可以完成
  - 一次寄存器读写（2 reads or one write），or
  - 一次ALU操作，or
  - 一次MEM访问（IF，lw，sw）

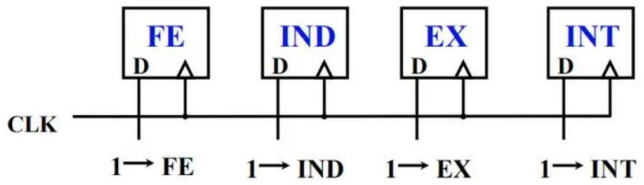


指令执行的阶段划分

- 共5个阶段
  - 取指：取指，PC+1
  - 译码：译码，读opr
    - 计算beq目标地址
  - 执行：R-type计算、访存地址计算，分支比较
    - 分支更新（或不更新）PC（完成）
  - 访存：lw读，sw写（完成），R-type写回（完成）
  - 写回：lw写回（完成）

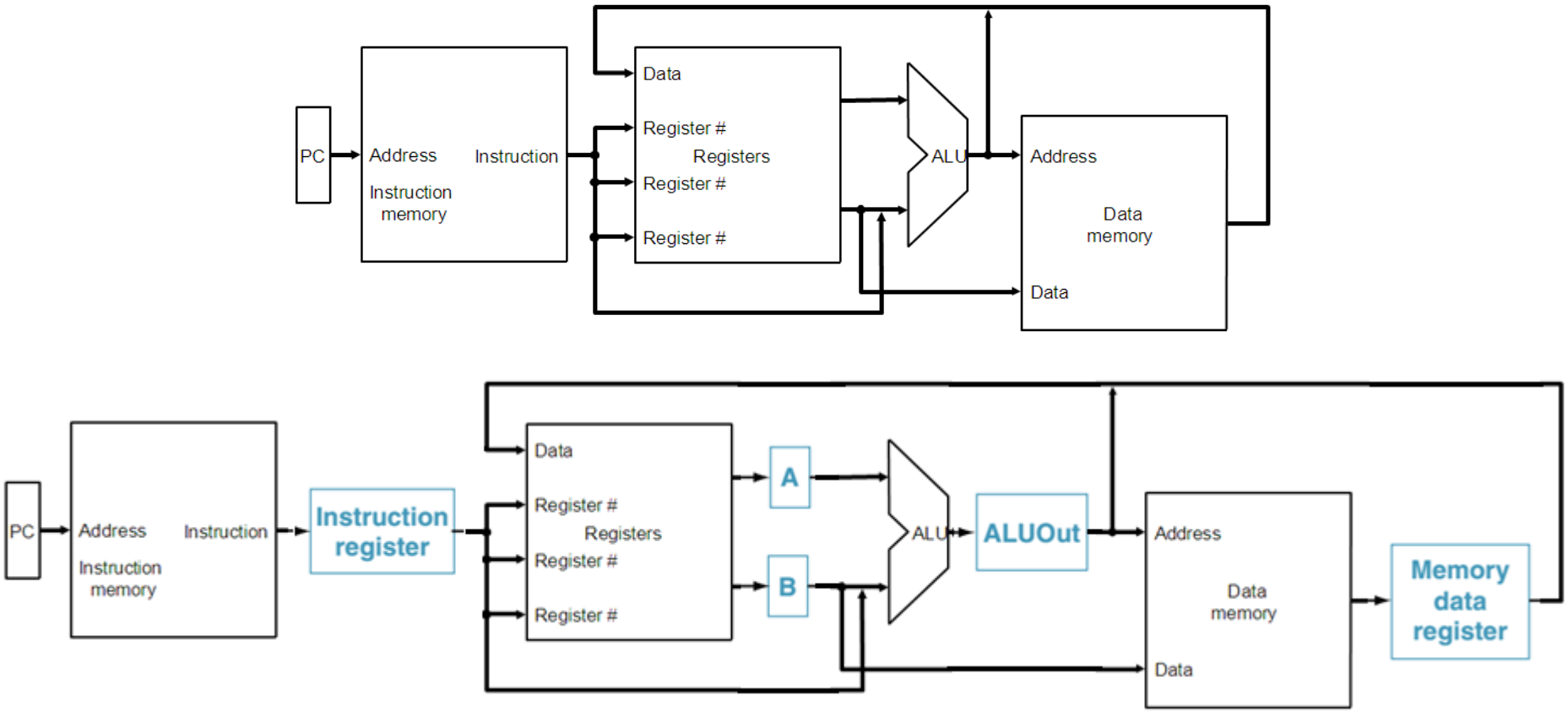


- 指令周期不定长，分别为3、4、5个CC
- 当前时钟周期标识：主控制器据此发出所需控制信号
  - 周期标志触发器：clk gating
  - 状态机：状态ID



唐图8.9

多周期DP：IF-ID-EX-MM-WB暂存



IR：指令暂存；ALUOut：ALU结果暂存

MDR：访存结果暂存。单周期内无法完成“访存+RF写”

复用：功能部件可以在不同周期重复使用

memory复用：IF/MM，*IR=MDR？*

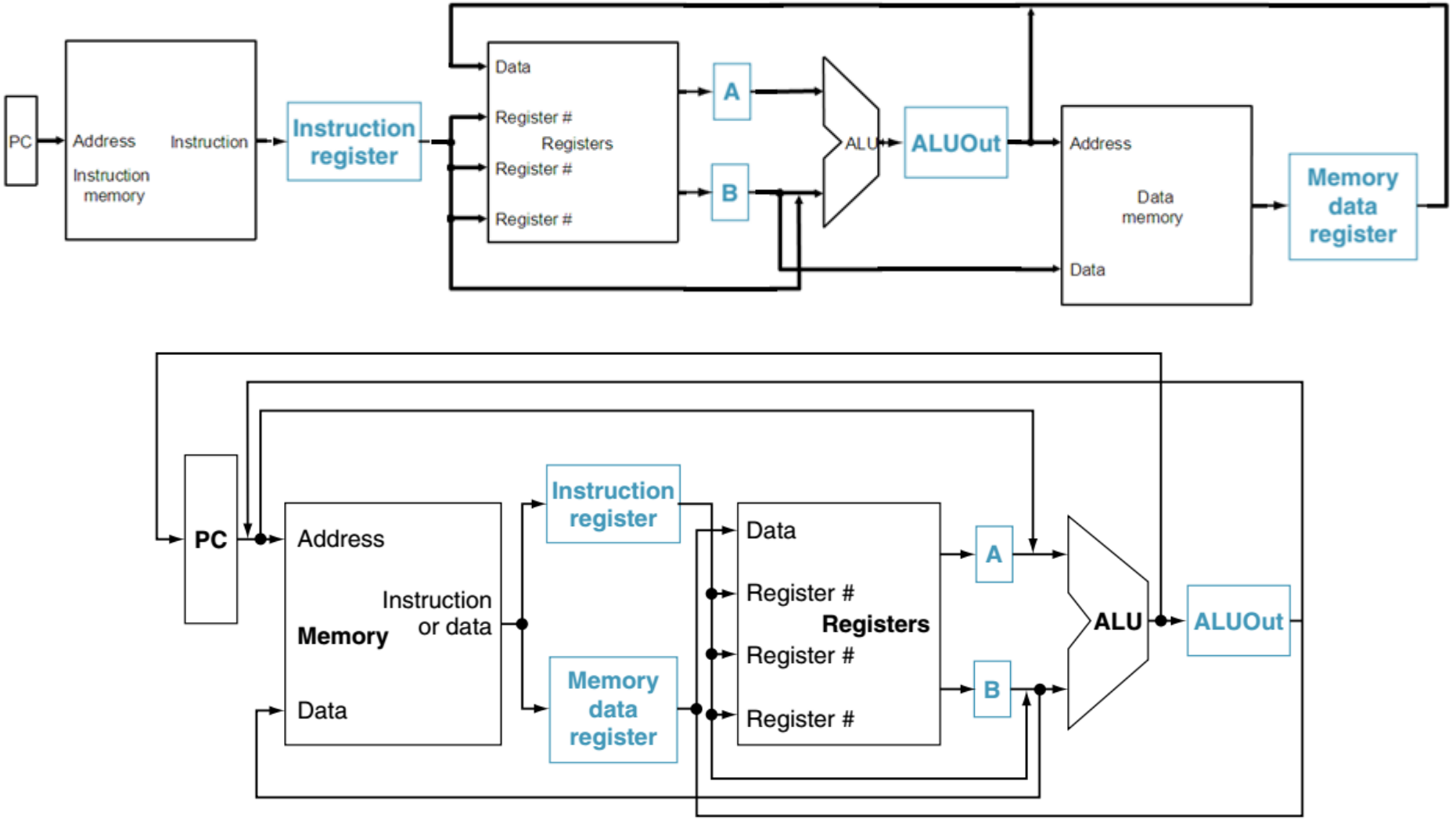
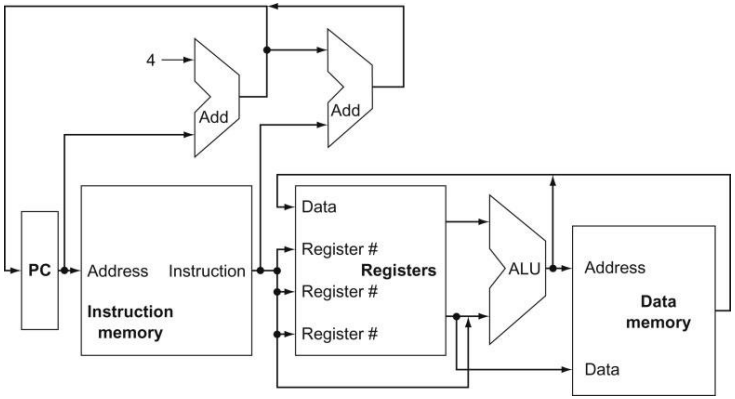


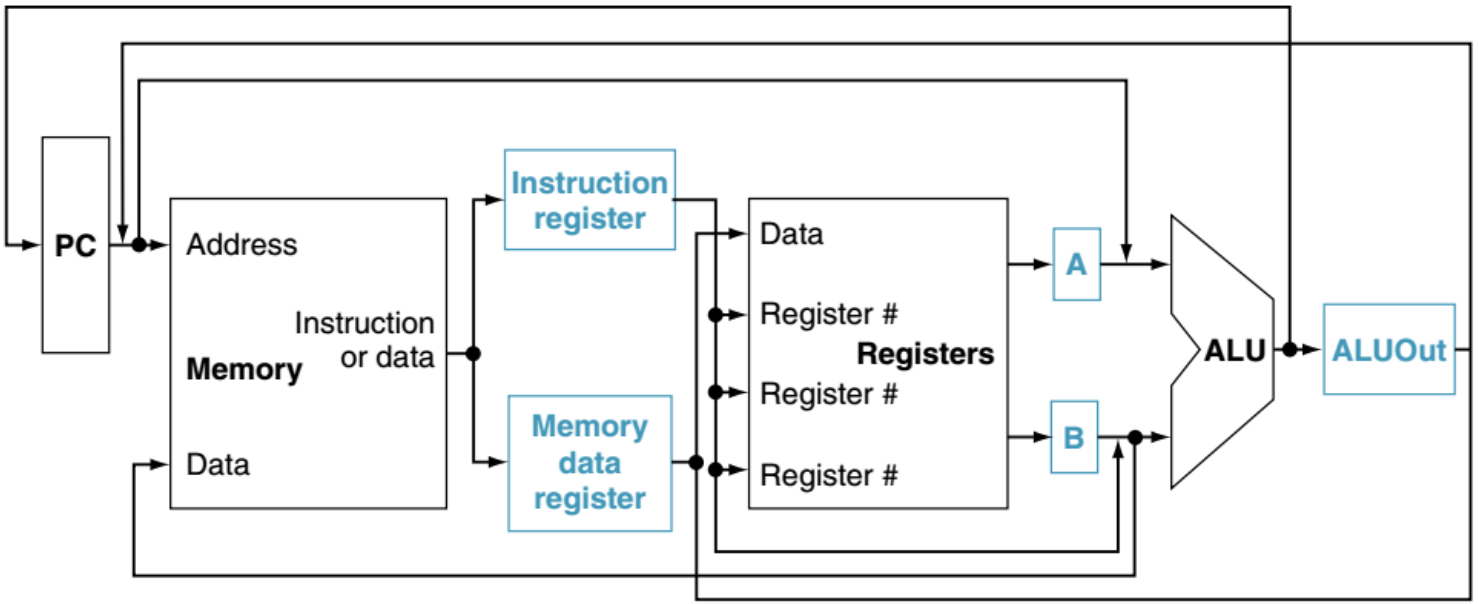
FIGURE e4.5.1，The high-level view

ALU复用：PC+1/EX/beq

FIGURE e4.5.6



Step name	Action for R-type instructions	Action for memory reference instructions	Action for branches
Instruction fetch	IR <= Memory[PC] PC <= PC + 4		
Instruction decode/register fetch	A<= Reg [IR[19:15]] B <= Reg [IR[24:20]] ALUOut <= PC + immediate		
Execution, address computation, branch/jump completion	ALUOut <= A op B	ALUOut <= A + immediate	if (A == B) PC <= ALUOut
Memory access or R-type completion	Reg [IR[11:7]] <= ALUOut	Load: MDR <= Memory[ALUOut] or Store: Memory [ALUOut] <= B	
Memory read completion		Load: Reg[IR[11:7]] <= MDR	



多周期数据通路

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

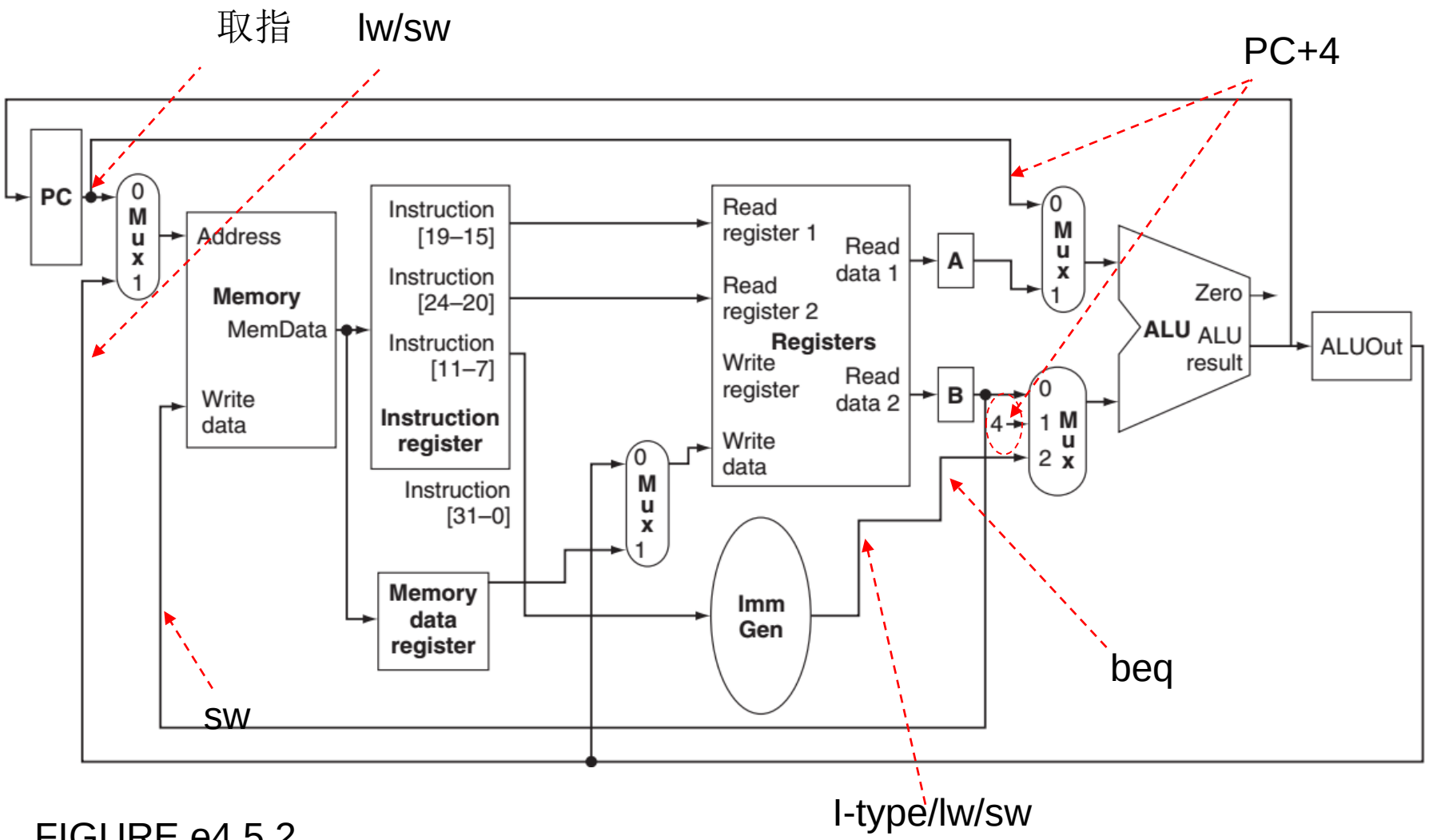


FIGURE e4.5.2

多周期控制信号

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

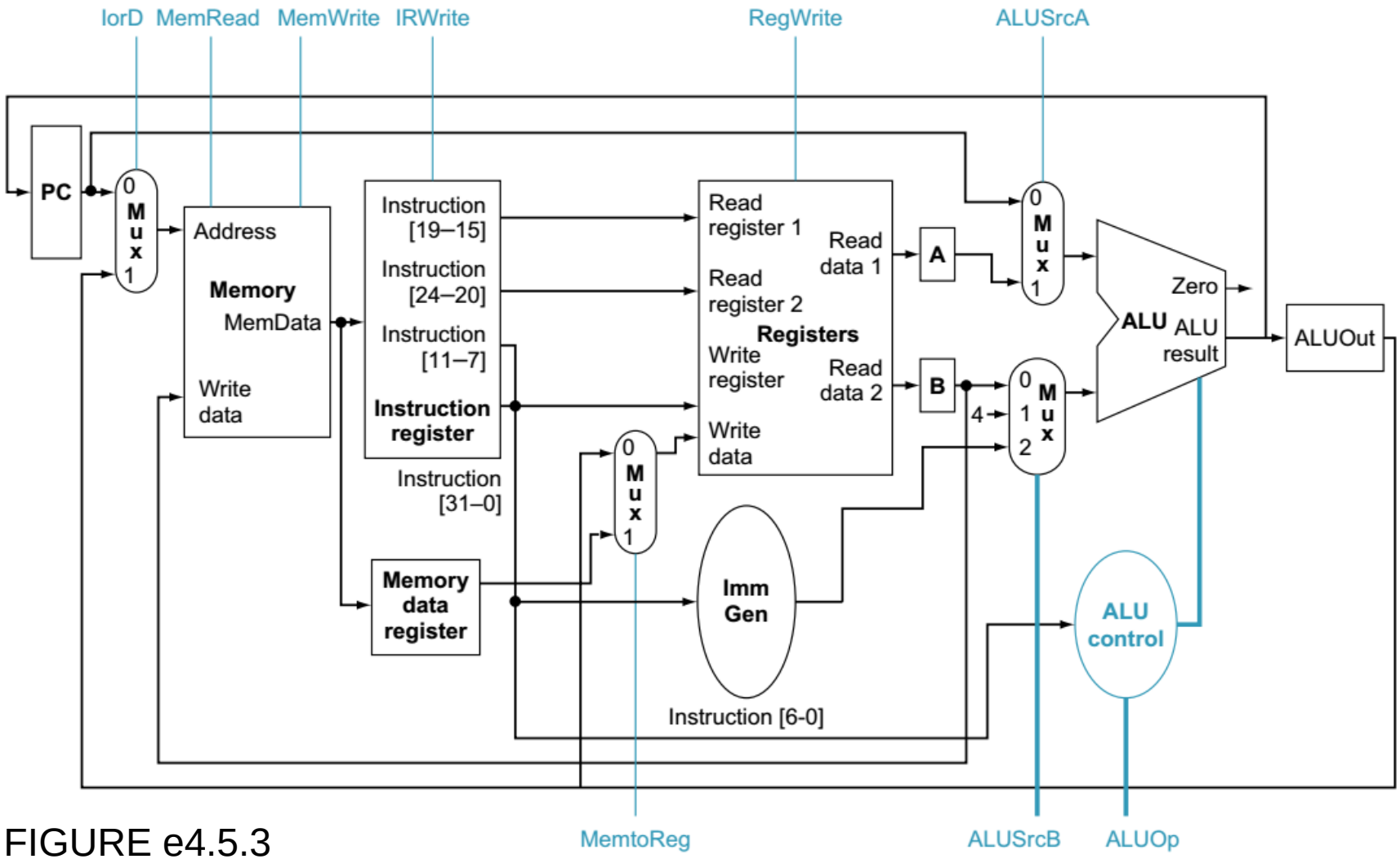
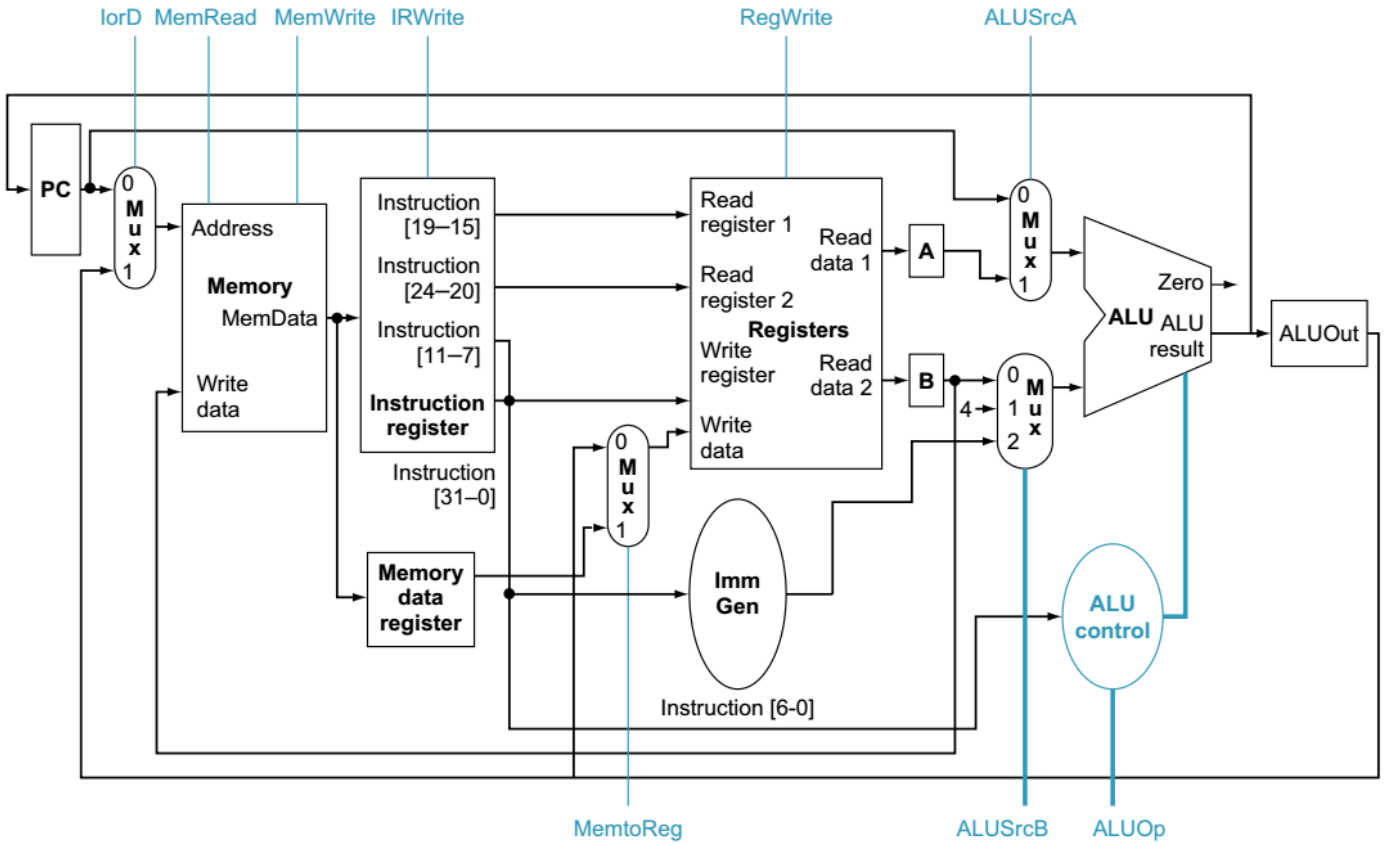


FIGURE e4.5.3

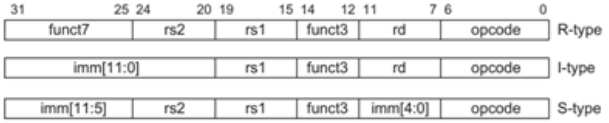
第一阶段：取指

- 根据PC从MEM中取指，IR=MEM[PC]
- 计算NPC，PC=PC+4
- 控制信号：IorD, MemRead, IRWrite；ALUSrcA, ALUSrcB, ALUOp, *PCWrite*

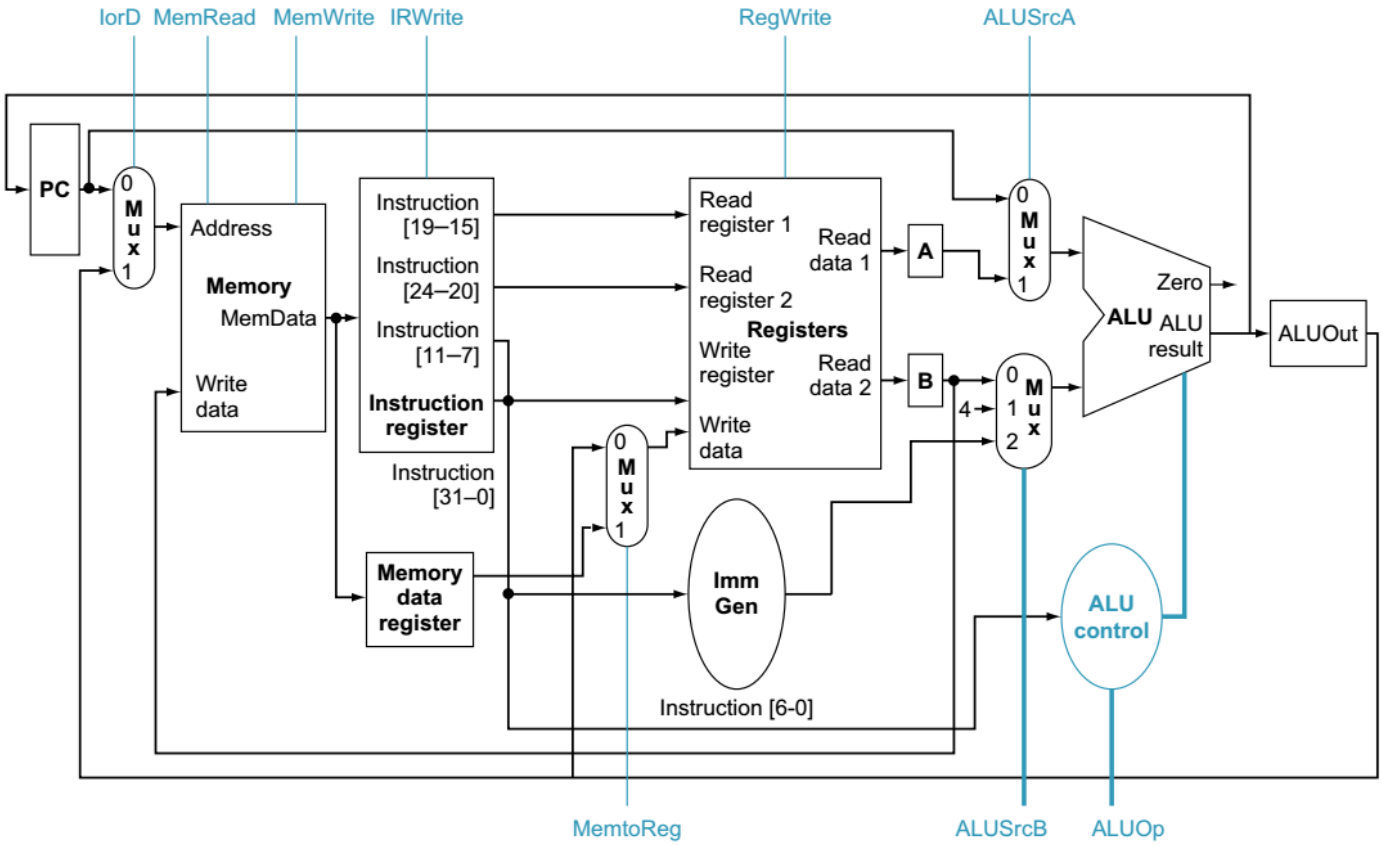
PC“后写”？



第二阶段：译码



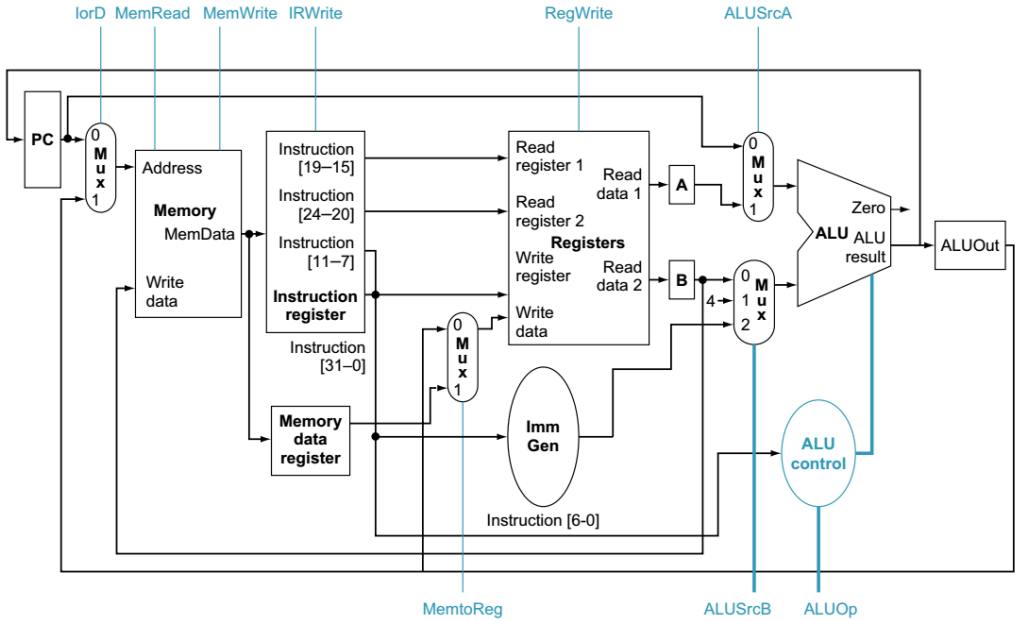
- 指令译码和寄存器读
  - 将rs和rt送往A和B：A=Reg[IR[19-15]], B=Reg[IR[24-20]]
- 计算beq目标地址



第三阶段：R-type执行、访存地址计算

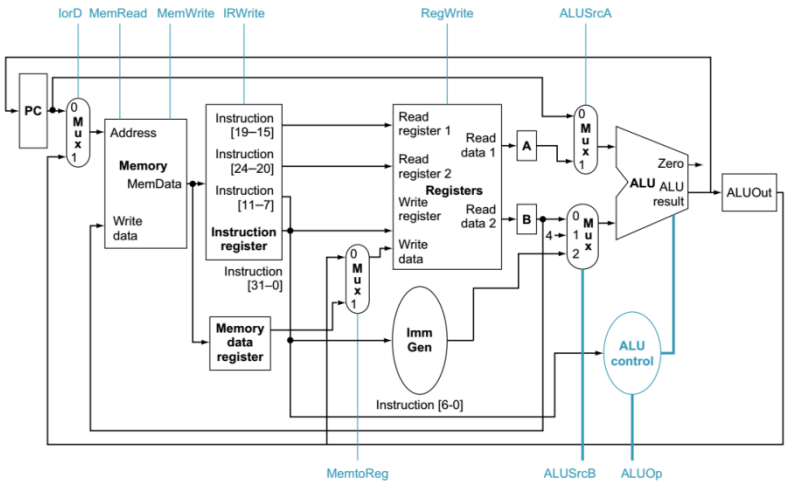
- 依赖于指令类型
  - R-type指令
    - ALUOut= A op B
  - 访存指令：计算访存地址
    - ALUOut=A+(sign-extend(IR[Imm])
  - beq比较
- 需要的控制信号?

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



第四阶段，第五阶段

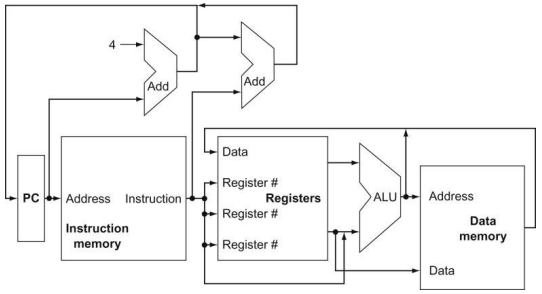
- 第四阶段：R-type和sw完成、lw读阶段
  - R-type完成：结果写回
    - Reg[IR[11-7]] = ALUOut
  - sw完成：写入MEM
    - MEM[ALUOut] = B
  - lw读：
    - MDR = MEM[ALUOut]
- 第五阶段：lw写阶段
  - lw写回：Reg[IR[11-7]] = MDR
- 所需的控制信号？

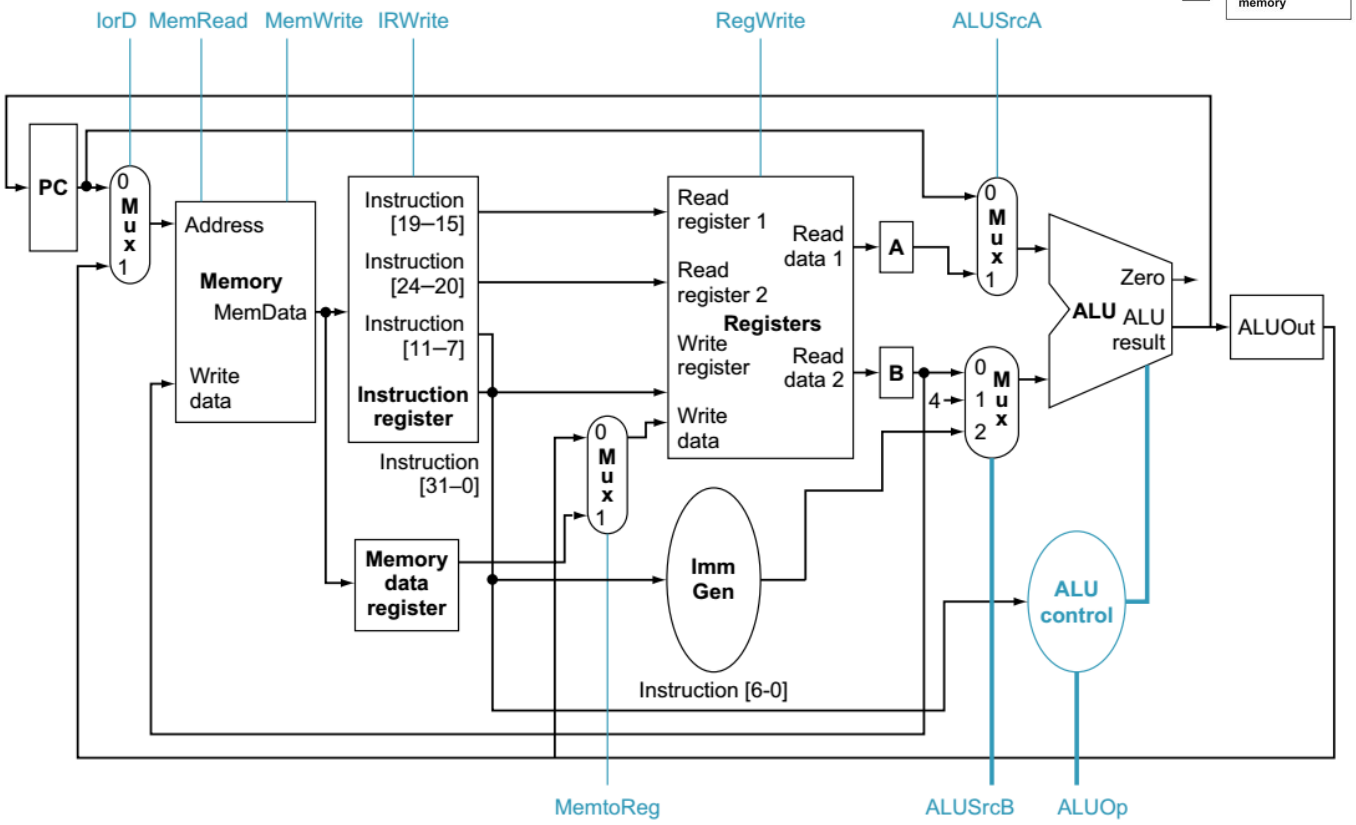


beq指令数据通路

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

- beq在执行周期完成，两个问题
  - 比较 || 计算目标地址：ALU冲突
  - PC写控制

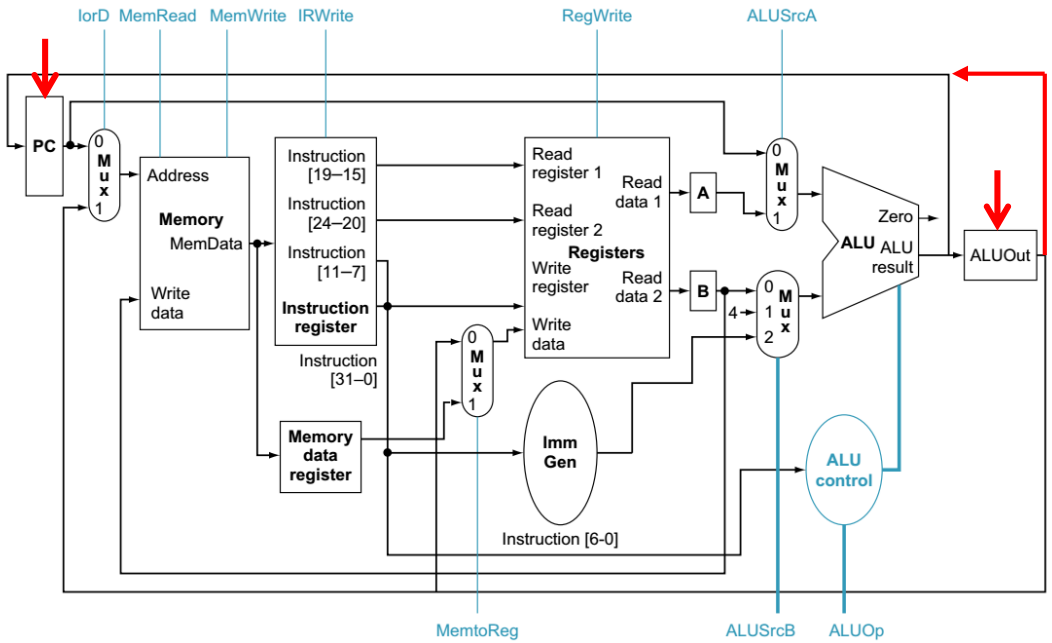




beq执行过程

- 译码周期：
  - 计算beq目标地址：ALUOut = PC+offset
    - 此时尚不确定是否分支，读寄存器和计算分支地址可能无效，但无害
    - 控制信号：ALUSrcA, ALUSrcB, ALUOp，置PCWriteCond=1
- 执行周期：beq比较，if (A == B) nPC=ALUOut；
  - ALUOut->PC的通路?见总图
  - PC写控制？见总图
  - ALUOut写控制？
- beq何时刷新PC？
  - 比较与写PC同时并行

ALUOut写控制：*译码周期为beq目标地址，执行周期为beq比较结果，需避免覆盖，禁止写ALUOut ？！*



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

多周期总图

IF：PCsource+PCWrite

EX：PCsource+PCWriteCond

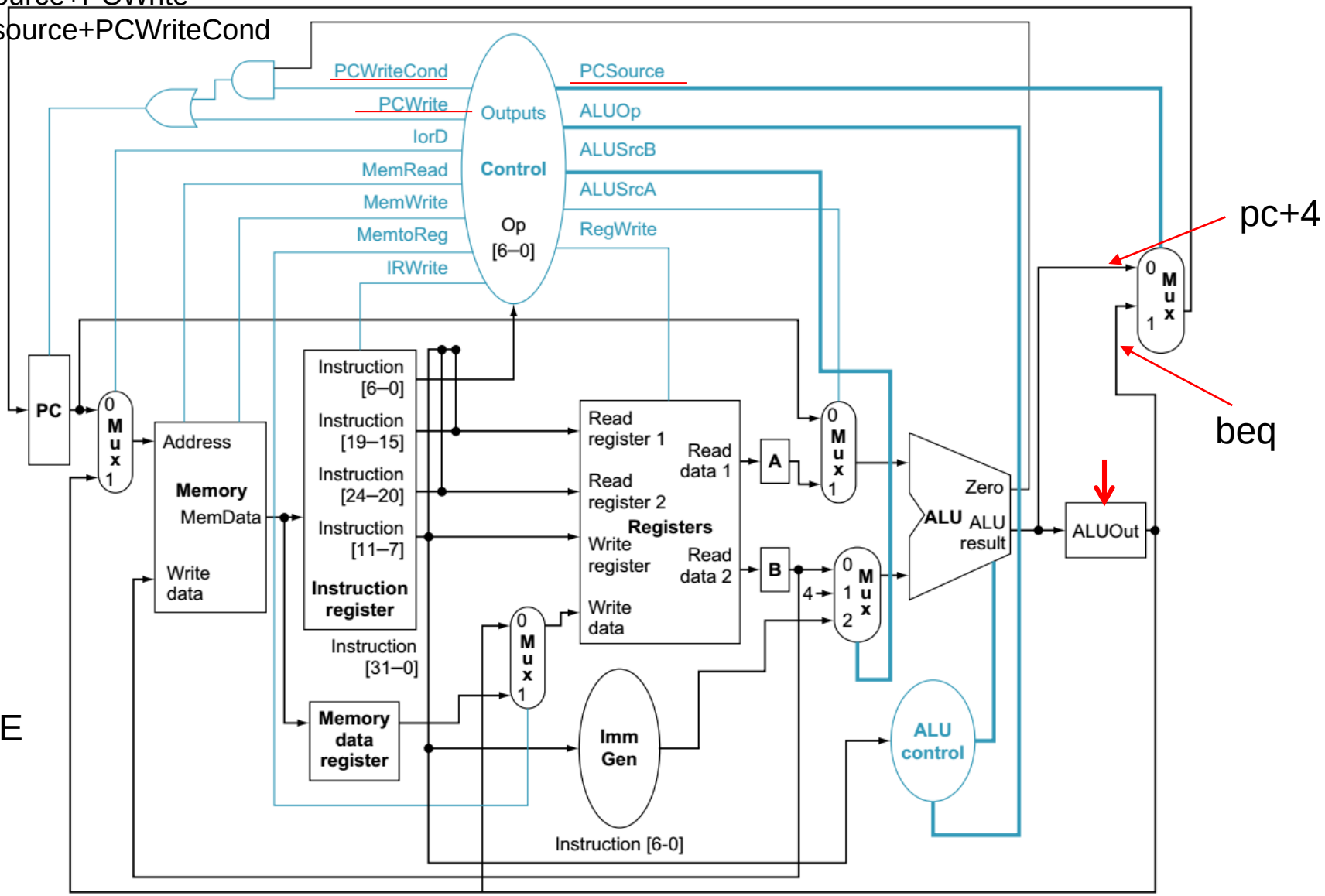


FIGURE
e4.5.4

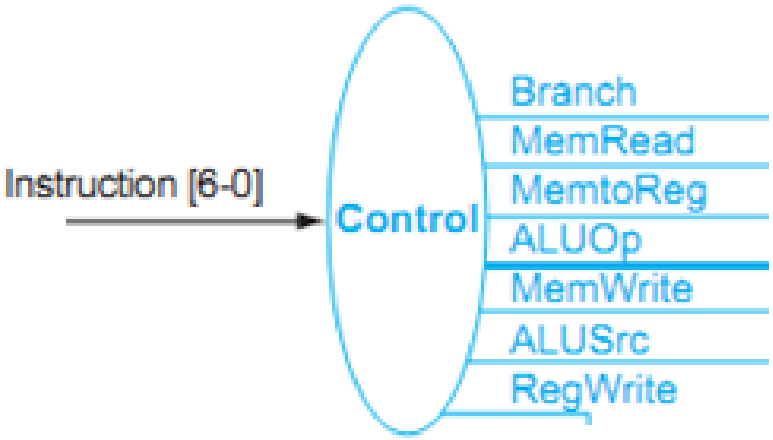
Multicycle指令时序

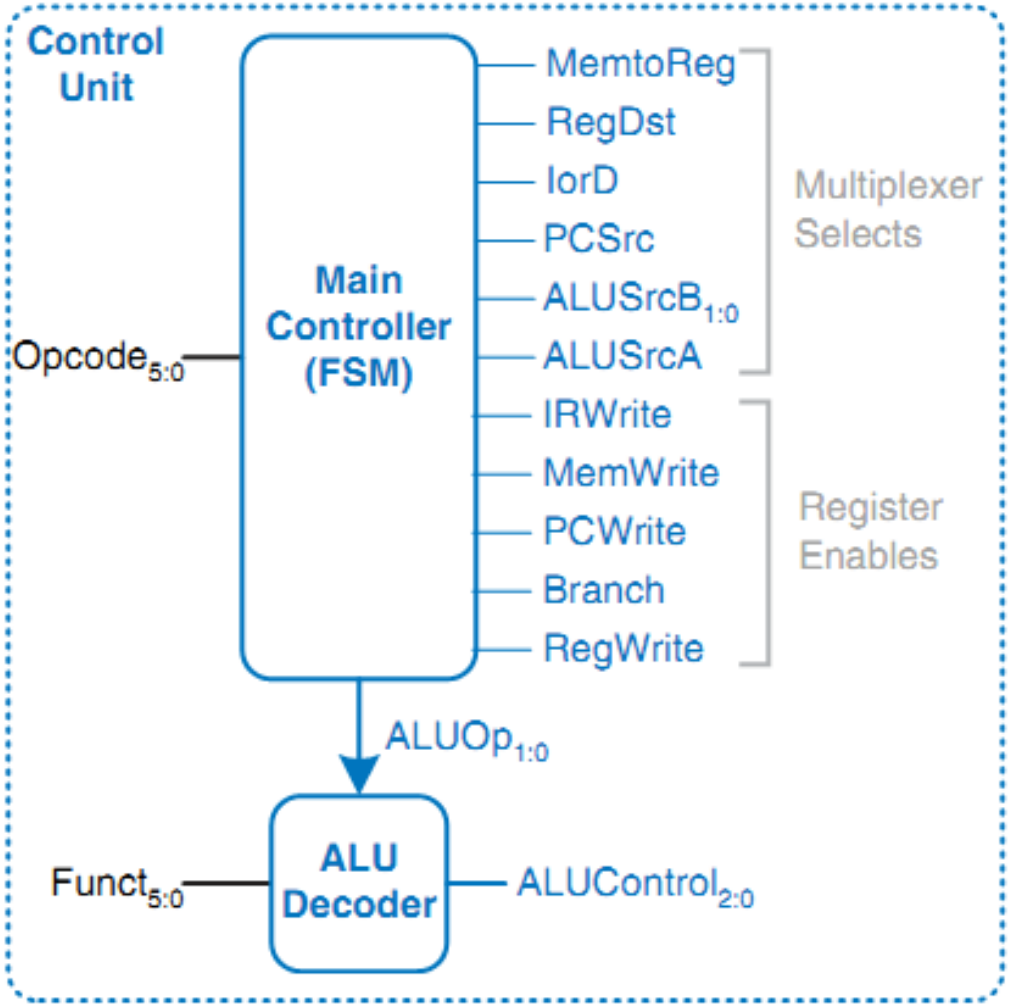
FIGURE e4.5.6

Step name	Action for R-type instructions	Action for memory reference instructions	Action for branches
Instruction fetch	IR <= Memory[PC] PC <= PC + 4		
Instruction decode/register fetch	A<= Reg [IR[19:15]] B <= Reg [IR[24:20]] ALUOut <= PC + immediate		
Execution, address computation, branch/jump completion	ALUOut <= A op B	ALUOut <= A + immediate	if (A == B) PC <= ALUOut
Memory access or R-type completion	Reg [IR[11:7]] <= ALUOut	Load: MDR <= Memory[ALUOut] or Store: Memory [ALUOut] <= B	
Memory read completion		Load: Reg[IR[11:7]] <= MDR	

- J-type？

Control unit internal structure





- opcode译码
- MIPS信号名与RV32不完全相同

Digital Design and Computer Architecture，2012

多周期FSM

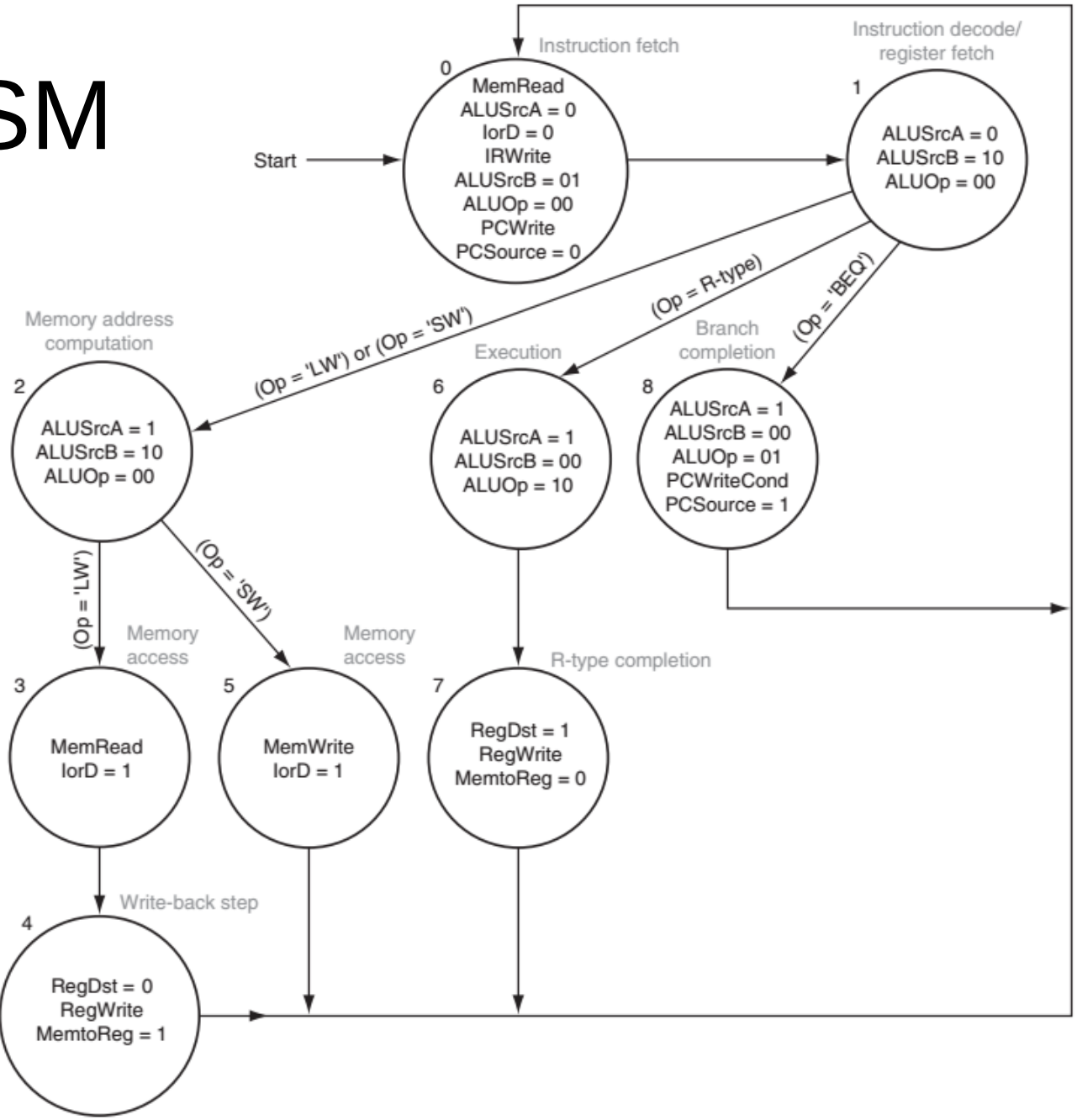
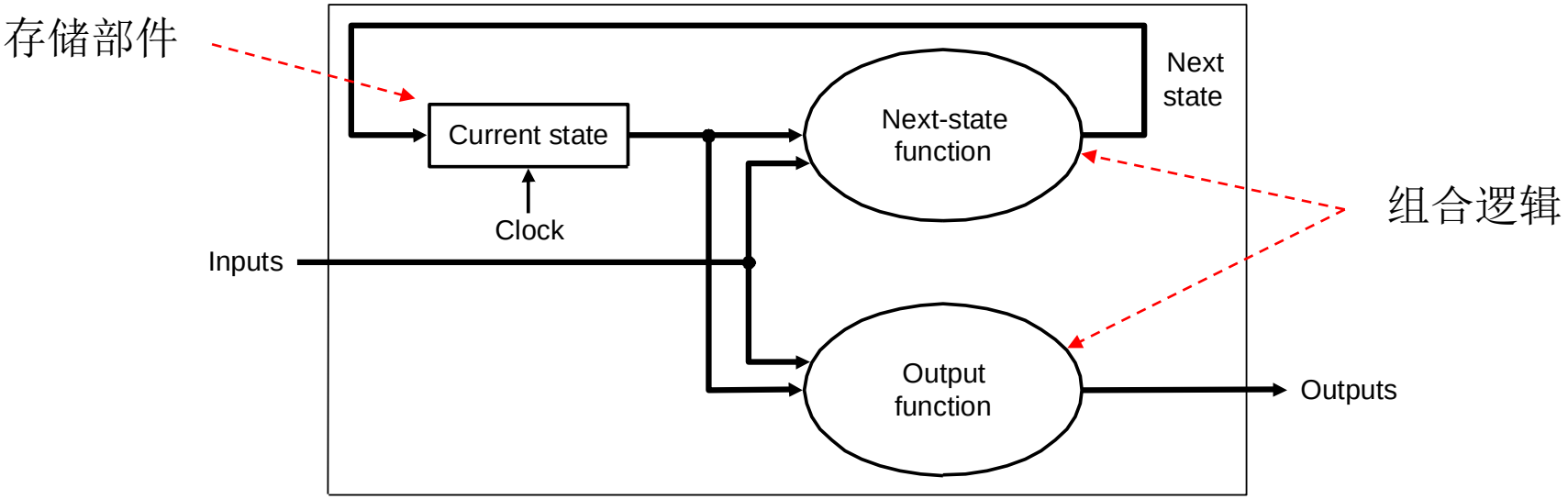


FIGURE e4.5.12
The finite-state diagram for multicycle control.

FIGURE C.3.1

FSM控制部件实现

- Moore型（Edward Moore），Mealy型（George Mealy）
  - Moore型速度快（输出与输入无关，可以在周期开始处就发出控制信号）。一步延迟（one-step-delay）。输出与时钟完全同步。
  - Mealy型电路较小。
  - 两种状态机可以相互转换。
- EDA工具可以根据FSM自动综合生成控制器



FSM控制器实现：PLA

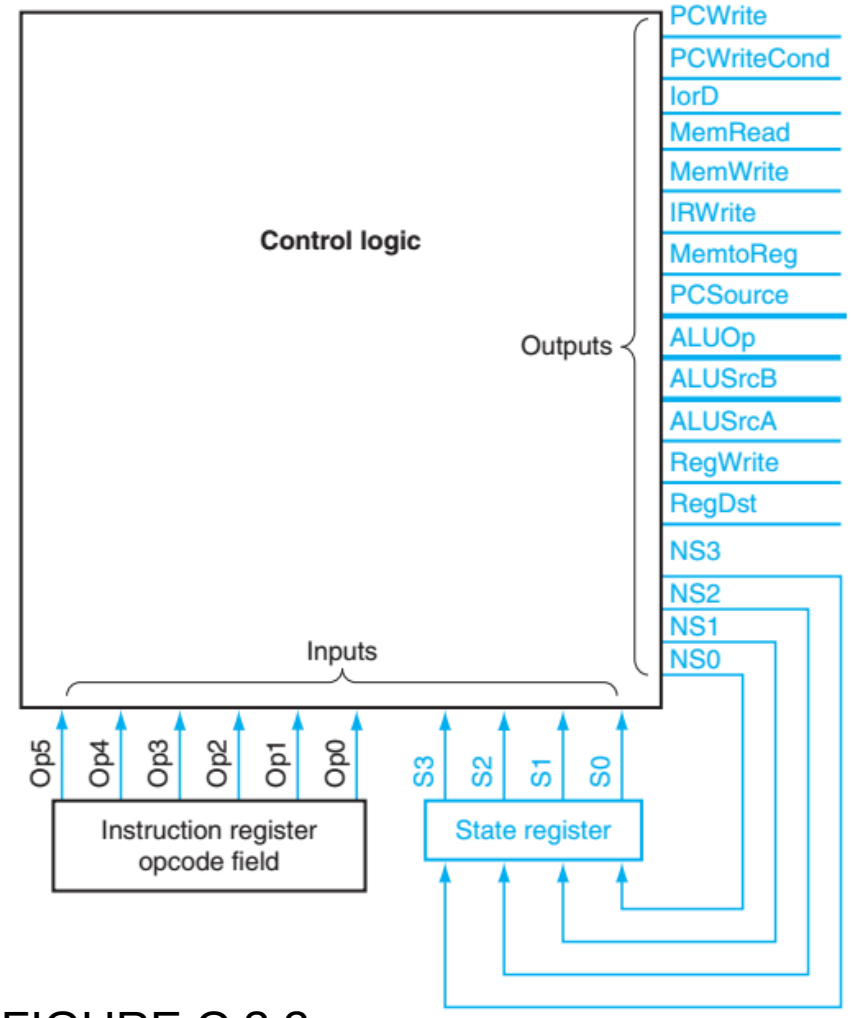


FIGURE C.3.2

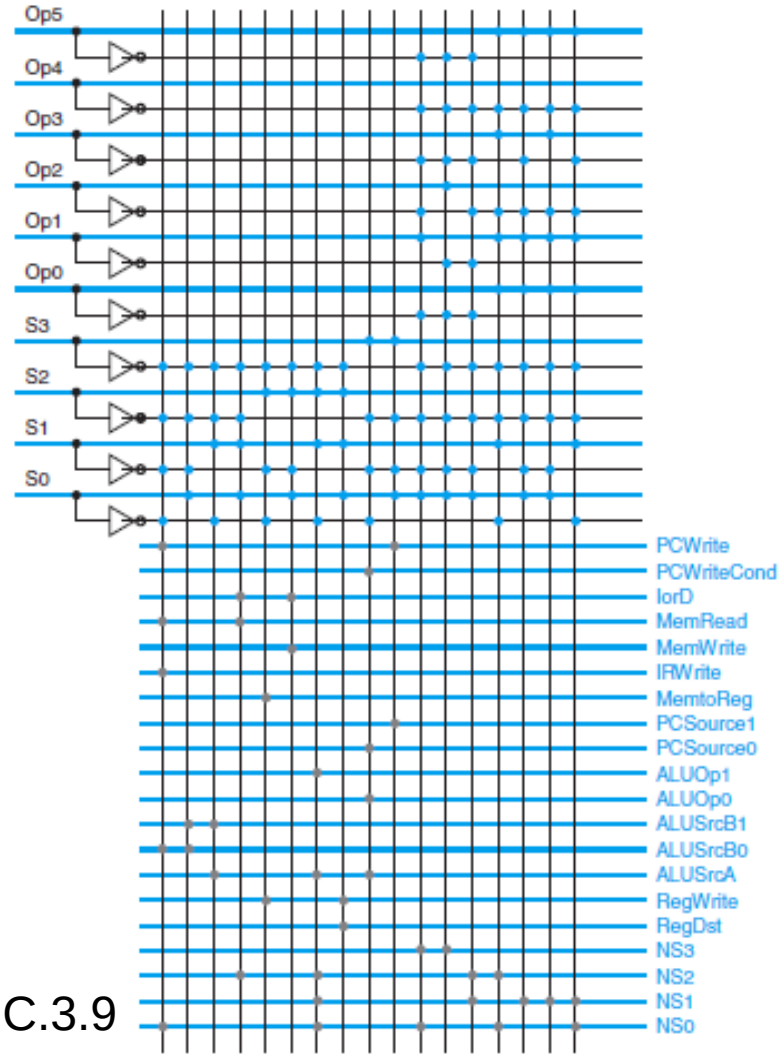
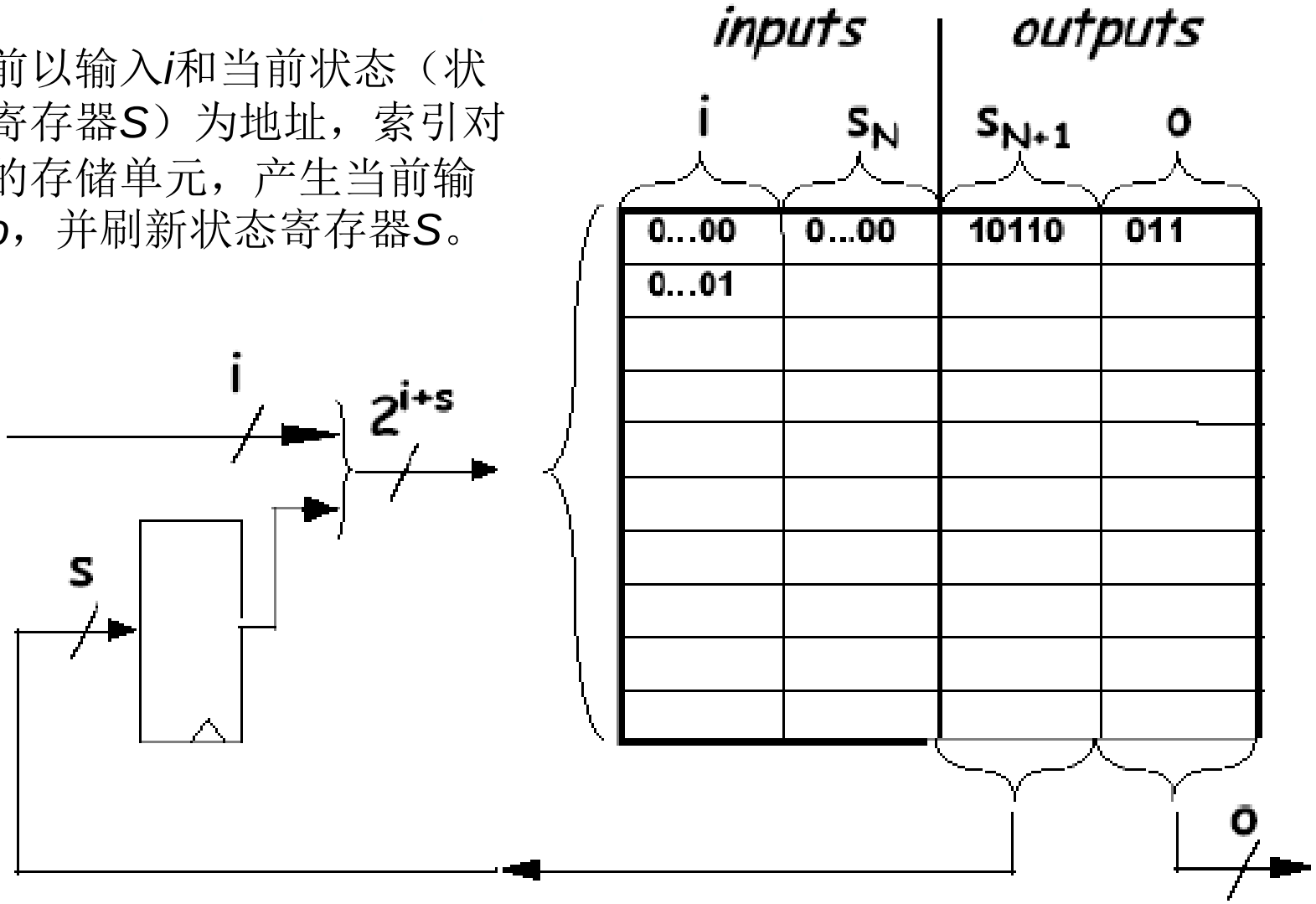


Fig C.3.9

rom-based finite state machines

当前以输入*i*和当前状态（状态寄存器*S*）为地址，索引对应的存储单元，产生当前输出*o*，并刷新状态寄存器*S*。



顺序器实现

- Sequencer
  - 基于计数器
- 下一状态
  - 顺序：+1
    - Adder
  - 分支
    - addr sel logic

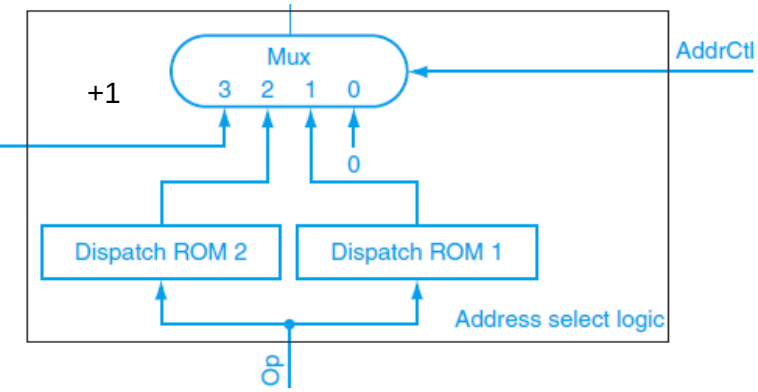


Fig C.4.2

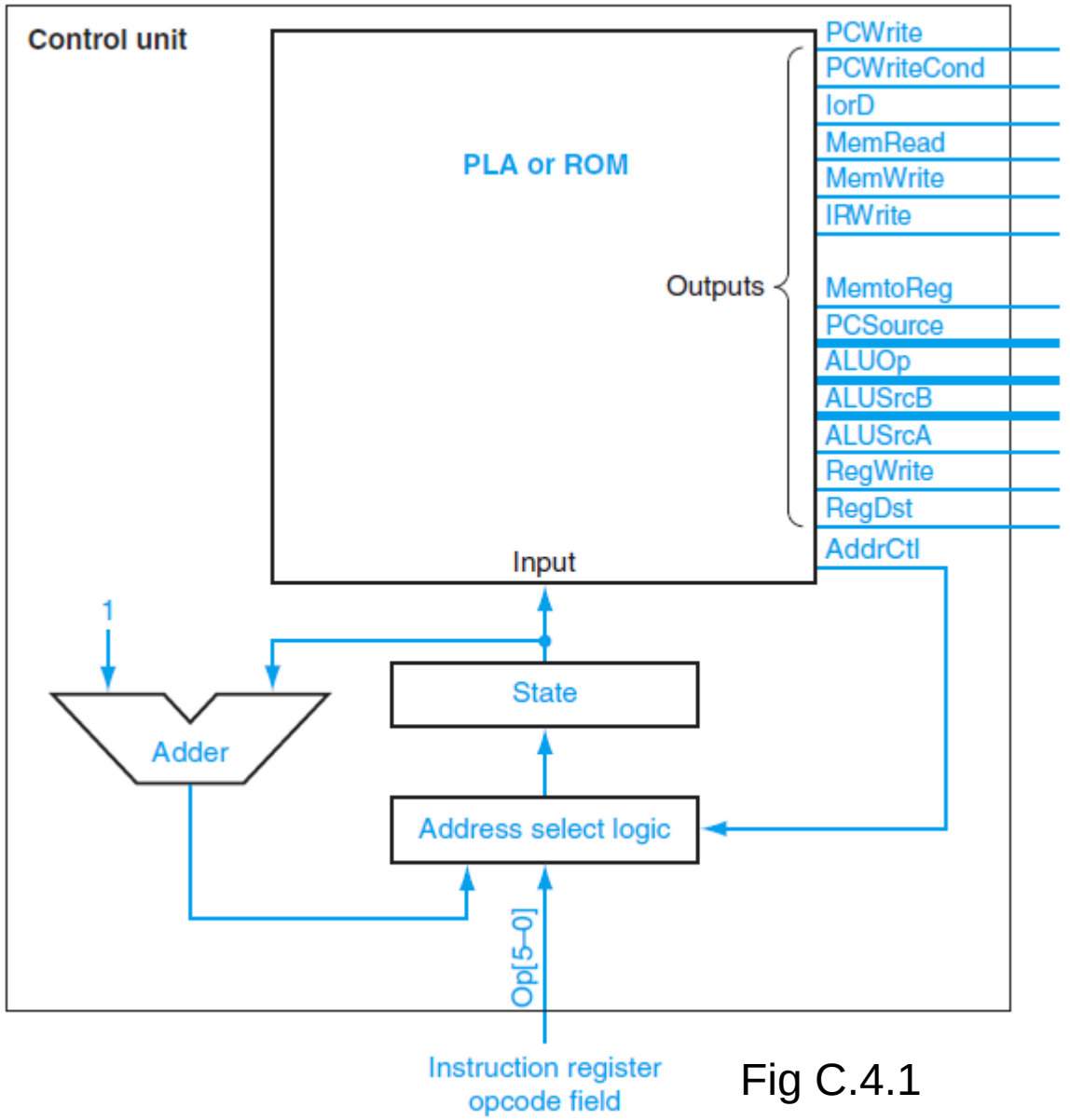


Fig C.4.1

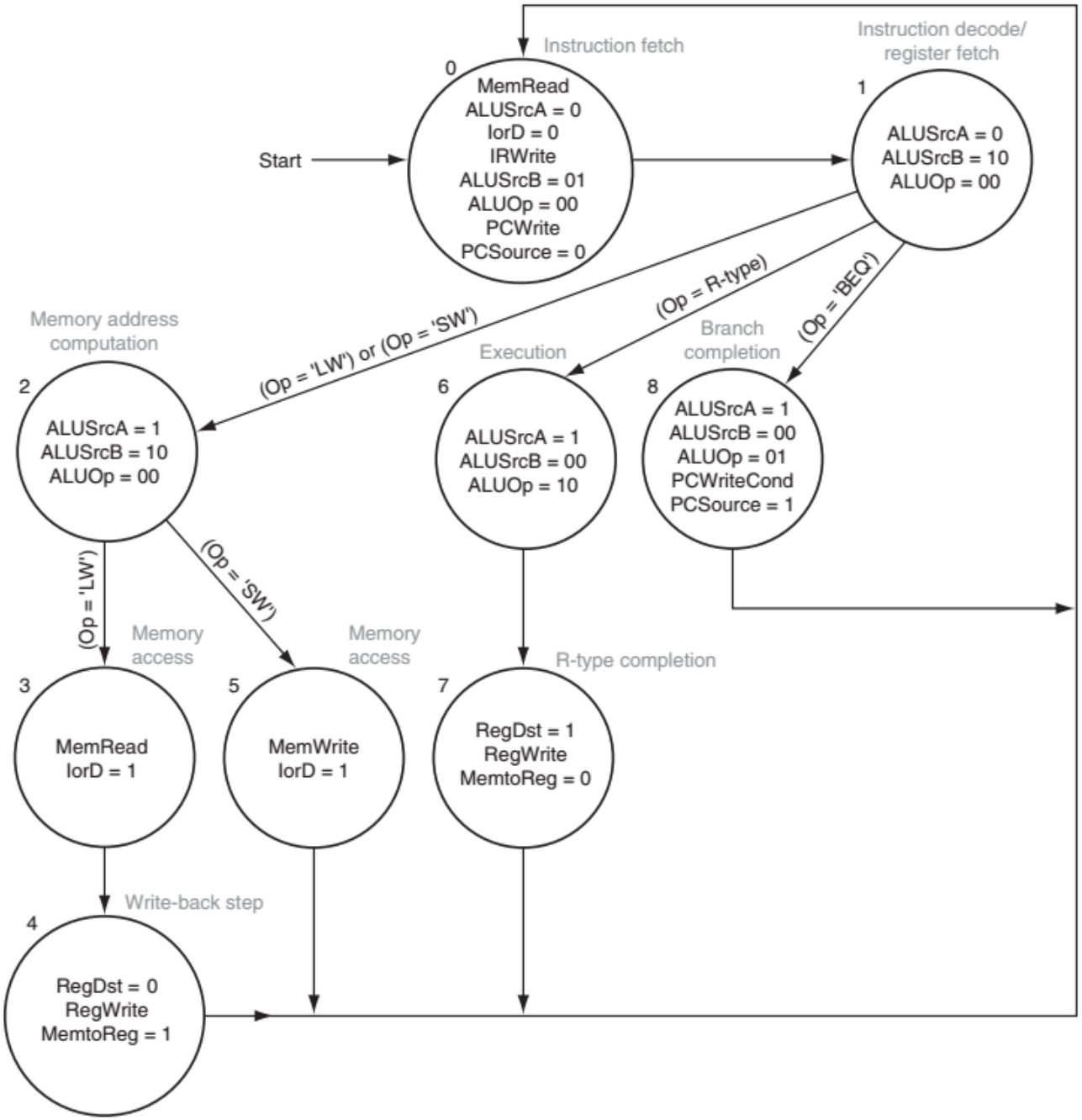
J-type？

FIGURE e4.5.12
The finite-state diagram for multicycle control.

FIGURE C.3.1

Jump指令？
nop指令？
lui指令？

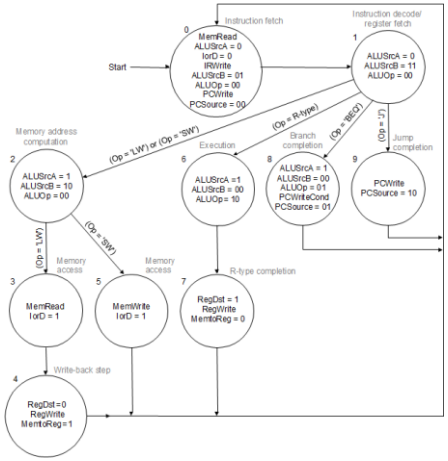
异常？



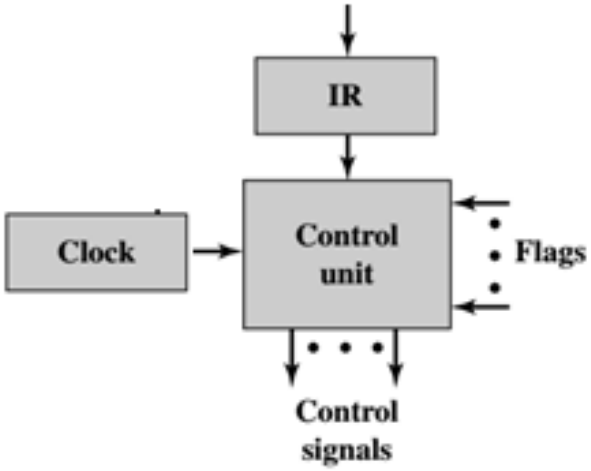
多周期微结构中，CPI = ？

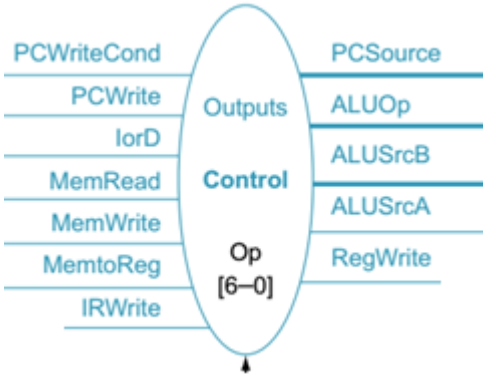
- CPI is always an average over a large number of instructions.
  - CPI=CPU clock cycles / Instruction count

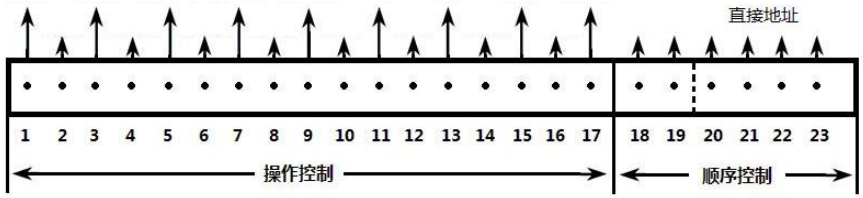
- 设
  - SPECINT2000中，load为25%（取byte占1%，取word为24%），store为10%（存byte为1%，存word为9%），ALU占52%，branch为11%（6%beq，5%bne），jump为2%（1%jal+1%jr），则
  - 指令周期定长时：CPI=5.0
  - 指令周期不定长时：CPI=4.12
    - = 0.25×5+0.10×4+0.52×4+0.11×3+0.02×3 = 4.12
  - *单周期：CPI=1.0*
  - *CPI越小越好？*
  - *指令阶段越细，周期越多（越小）越好？*
  - *IPC=?*

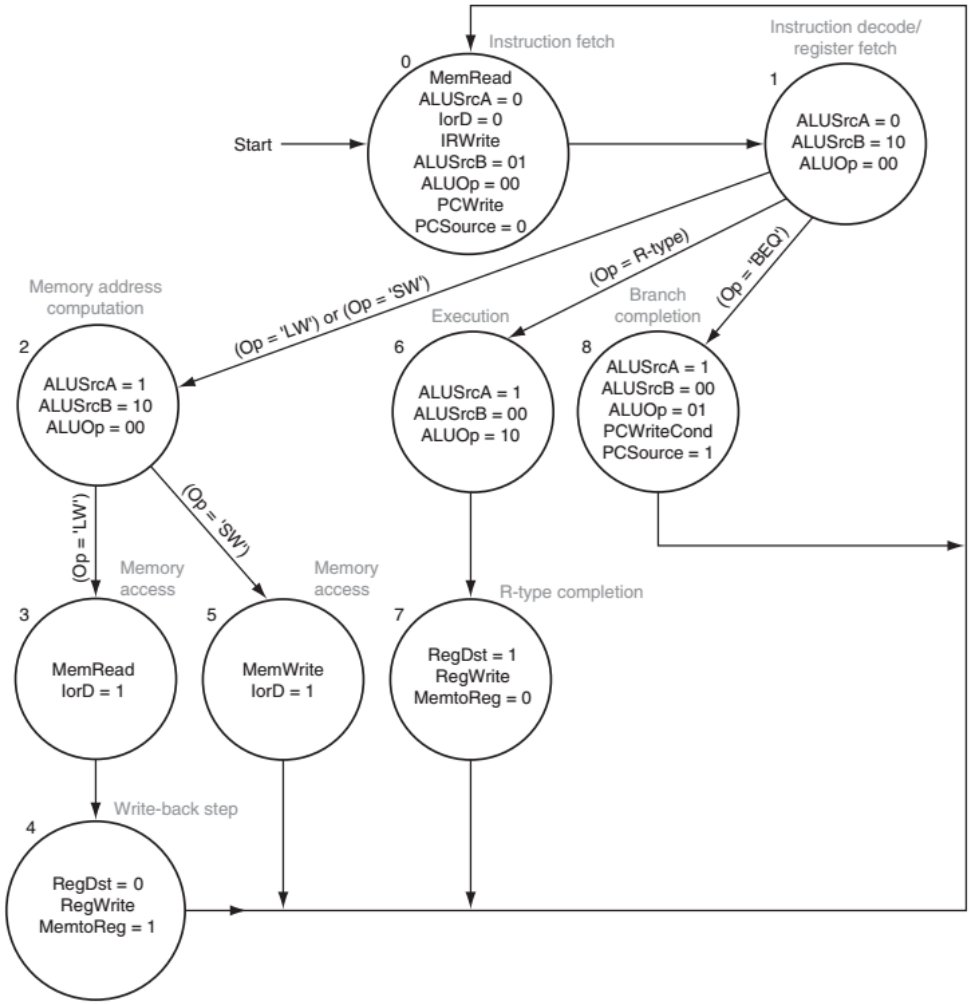


微程序控制原理：微指令，微程序



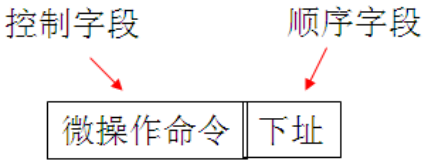






微程序控制器，$C.5

- Microcode
  - 控存



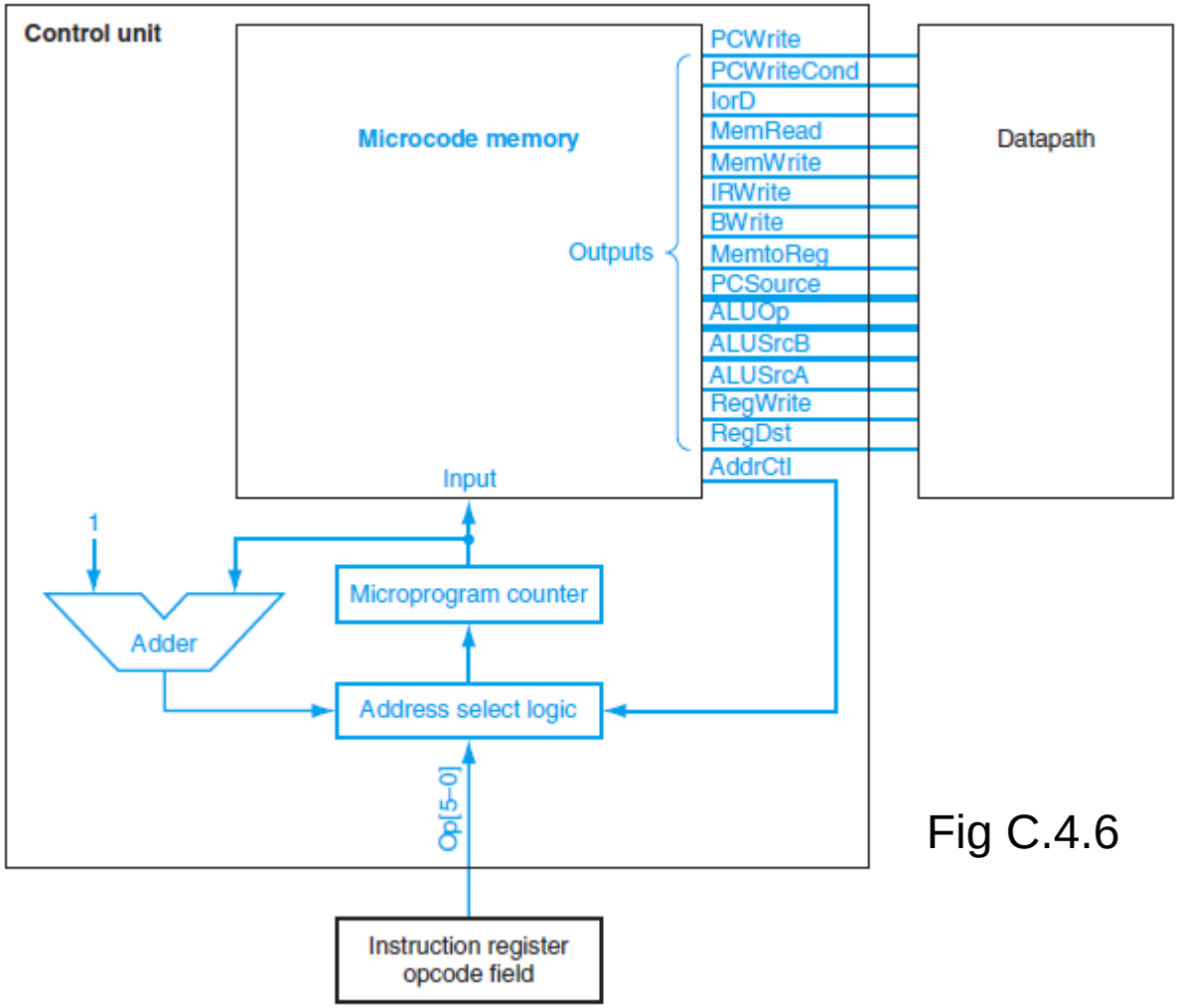


Fig C.4.6

微程序控制
Microprogrammed Control

英国剑桥大学Maurice. V. Wilkes教授

1946～1949年Wilkes在剑桥实现第一台存储程序式计算机EDSAC。获第二届图灵奖，1967

于1951年提出微程序控制的概念和原理。

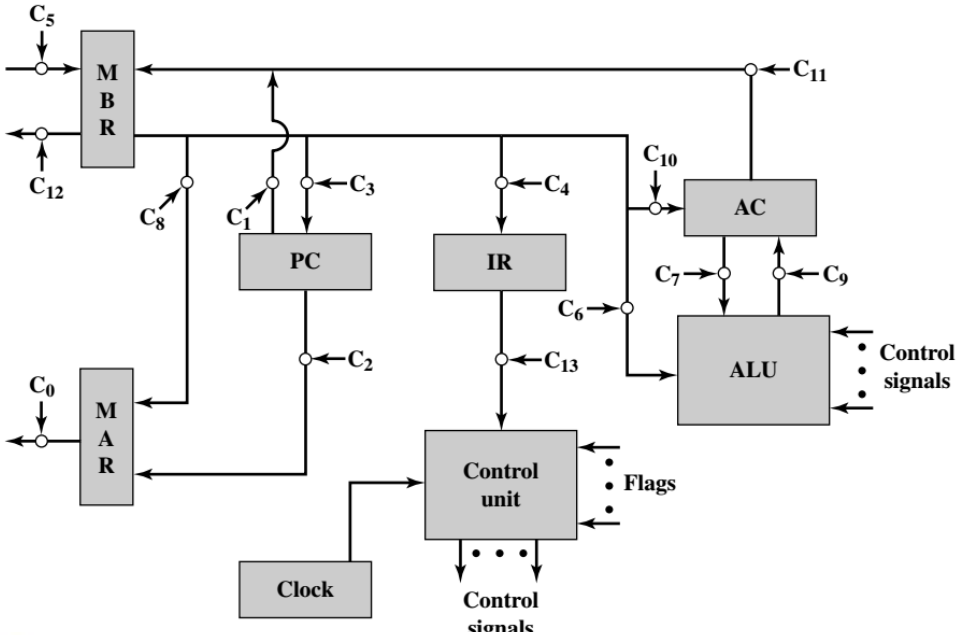
Foreshadowed by Babbage's "Barrel" and mechanisms in earlier programmable calculators

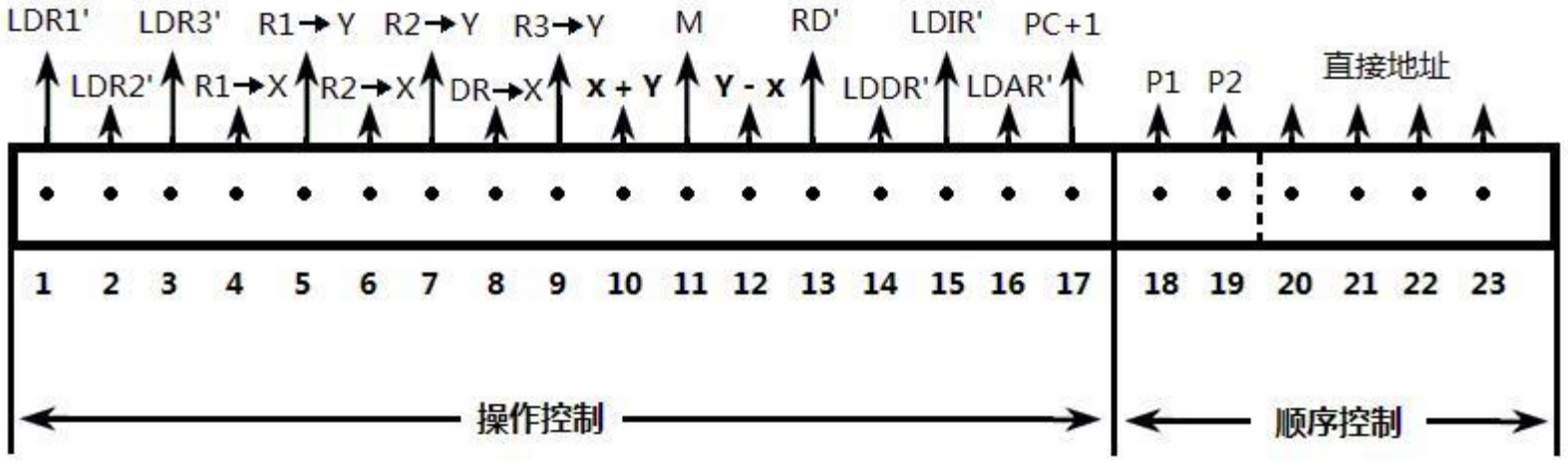
控存性能是瓶颈，直至60's半导体存储器出现。1964年IBM System/360首次采用了此技术。

1）组合逻辑控制器设计复杂；2）可重构计算。

Overview：微指令、微命令

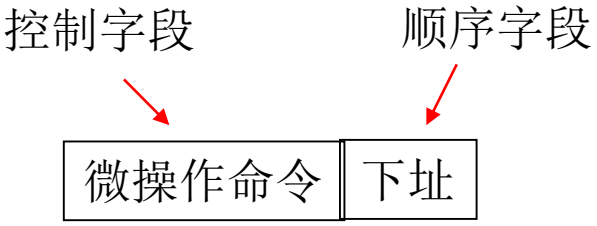
- 取指周期
  - T_0: PC->MAR, 1->R
  - T_1: M(MAR)->MDR, PC+1->PC
  - T_2: MDR->IR, OP(IR)->ID
- 间址周期
  - T_0: Ad(IR)->MAR, 1->R
  - T_1: M(MAR)->MDR
  - T_2: MDR->Ad(IR)

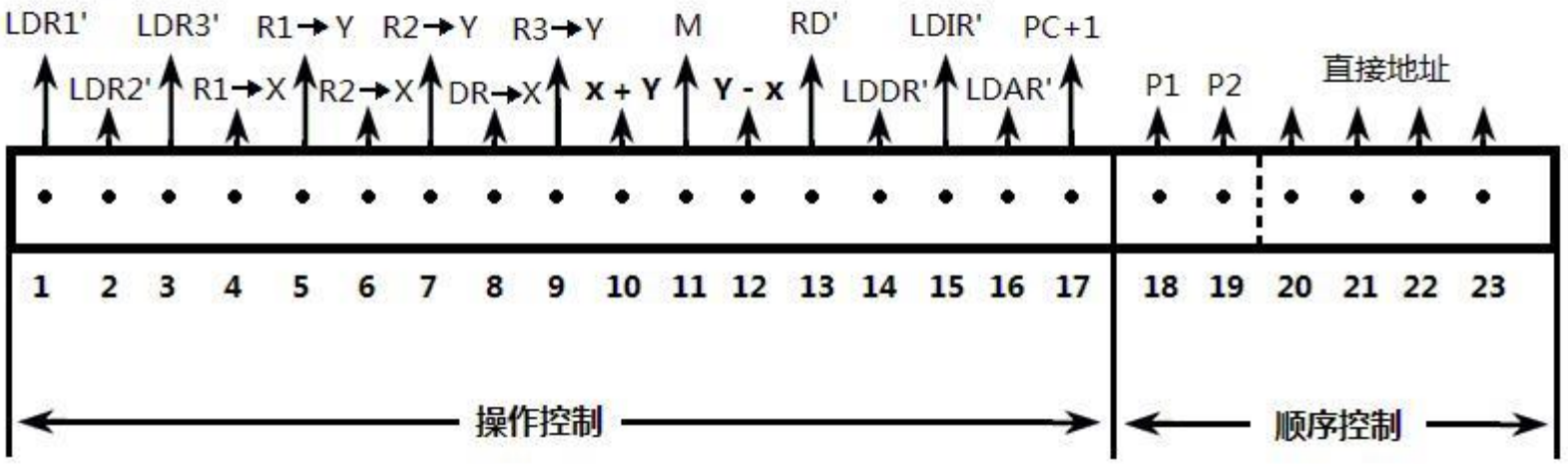




微程序

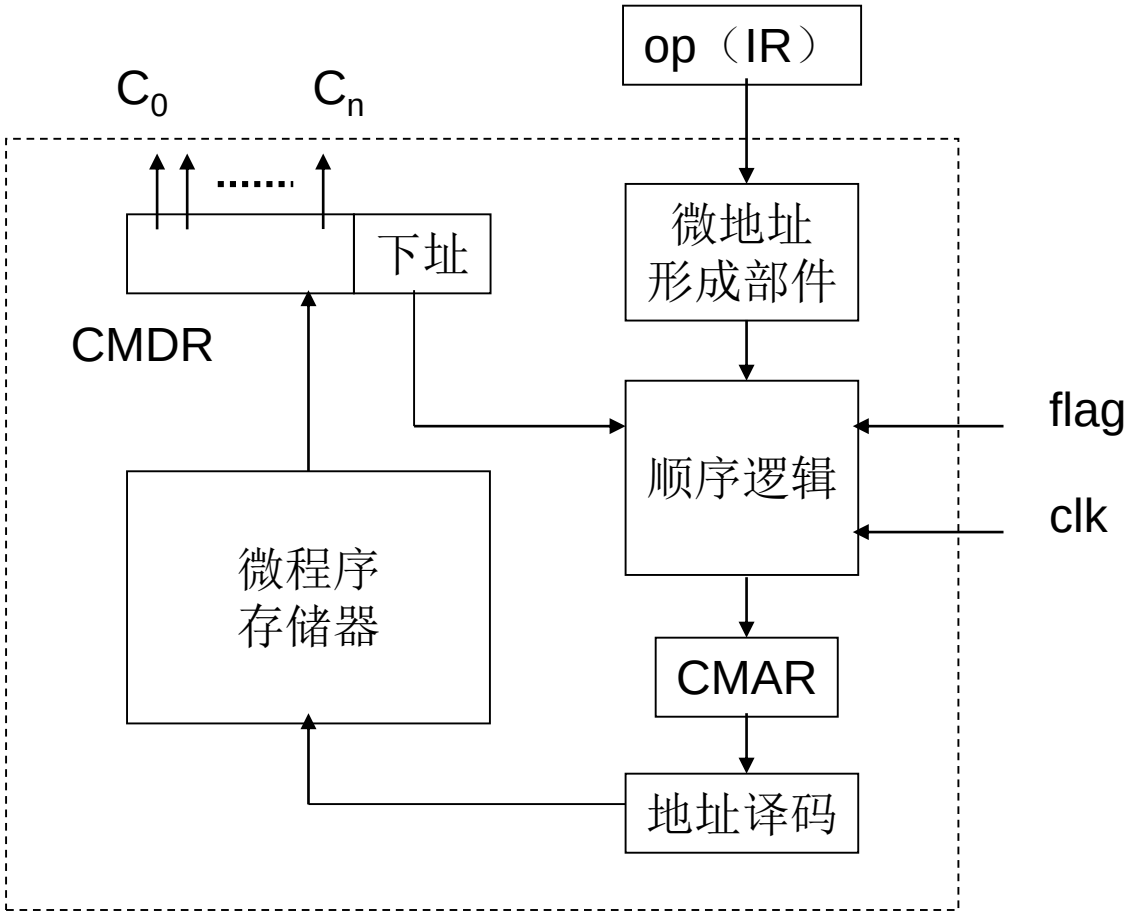
- 用一个微程序实现一条机器指令：取指，译码，执行。。。
  - 微程序由多条微指令组成
    - 与节拍对应
  - 一条微指令对应一个或多个微操作命令，称“微命令”，即实现微操作的控制信号。
    - 微指令的编码方式
  - 微程序存储在“控存”中
- 微指令的格式：水平微指令





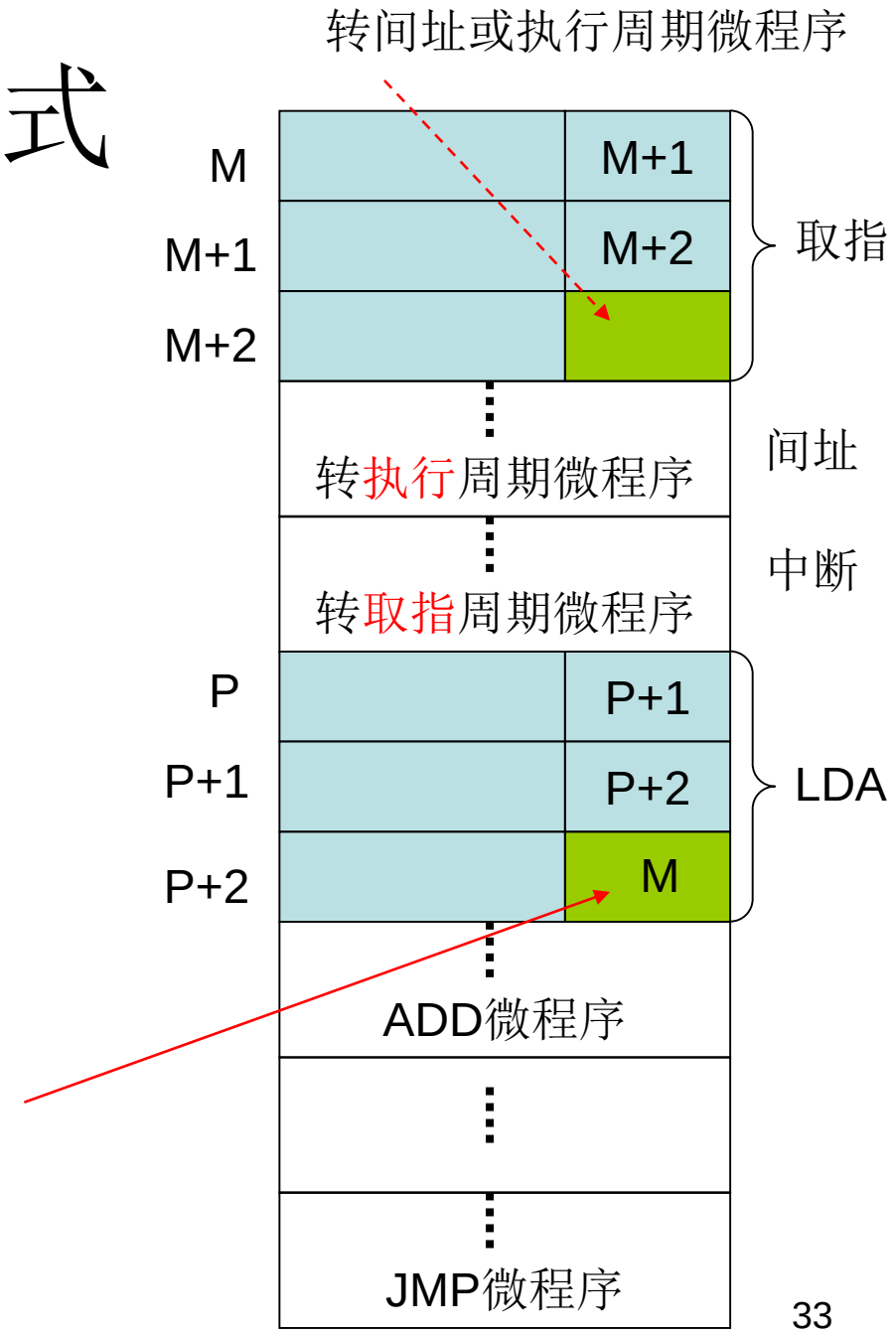
微程序控制部件的结构

- 微地址形成部件
  - 形成微程序的首址
- 顺序逻辑
  - 形成下一条微指令的地址（计数器）
- 微程序存储器（控存）
  - 存储微程序
- C_0~C_n：控制信号
  - CPU内部及系统总线



微程序的存储形式

- “取指”、“间址”、“中断”微程序等三个微程序所有指令共用
- “执行”各个指令不同
- 微程序个数为3＋X

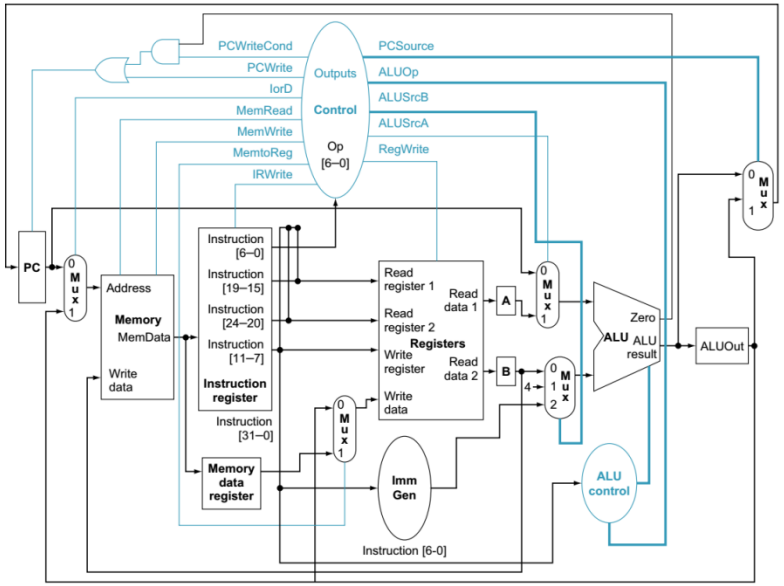


例：计算下址字段位数？

- 某计算机采用微程序控制器，共有32条指令。取指微程序包含两条微指令，其他指令对应的微程序平均包含4条微指令。设采用下址字段确定下一条微指令的地址，则下址字段位数至少多少位？
  - 总共需要存储32×4+2=130条微指令
  - 128<130<256，因此需要8位寻址
  - 故下址字段至少8位。

多周期技术思考

- 本书有哪些逻辑设计惯例（约定）？
  - A、B、MDR、ALUOut等寄存器无写控制，PC、RF、IR有？
  - 为何需要MDR（如果MEM=>RF，则ld可少一周期）？
  - 为何单周期中，PC无写控制，多周期有？
- 为何采用多周期？应该几个周期？
- 指令周期定长？
- 每个周期需要哪些控制信号？
- 每个周期有哪些部件空闲（做无用功）？
- PCWrite与PCWriteCond何时有效？
- 关于beq
  - 执行分支指令时，PC执行了几次写操作？
  - 为何在ID计算目的地址？
  - 在EXE计算地址，MEM比较完成，ok？状态机？CPI？
  - 先比较，后计算地址，ok？
- 哪些寄存器程序员不可见？
- Memory-memory add: M[rd] = M[rs1] + M[rs2]



小结

- 多周期：一个周期完成指令的一步（微操作）
  - 与单周期的区别：功能部件复用，中间结果暂存
    - 可以实现不定长指令周期，提高性能
    - 能否采用单周期的数据通路？
  - 按当前周期（标识？）产生响应的控制信号
    - 何时刷新PC：指令周期中PC保持不变如何实现？
- 作业：

1. 每一类指令的指令周期各含多少个时钟周期？
2. 分别分析R/I/S/B-type指令的多周期设计方案中每个周期所用到的功能部件。
3. 比较不同FSM控制器实现方式的特点。
4. 解释水平微指令和垂直微指令的特点。


Thank you