

The Processor Implementation: Datapath & Control

“Computer Organization & Design”

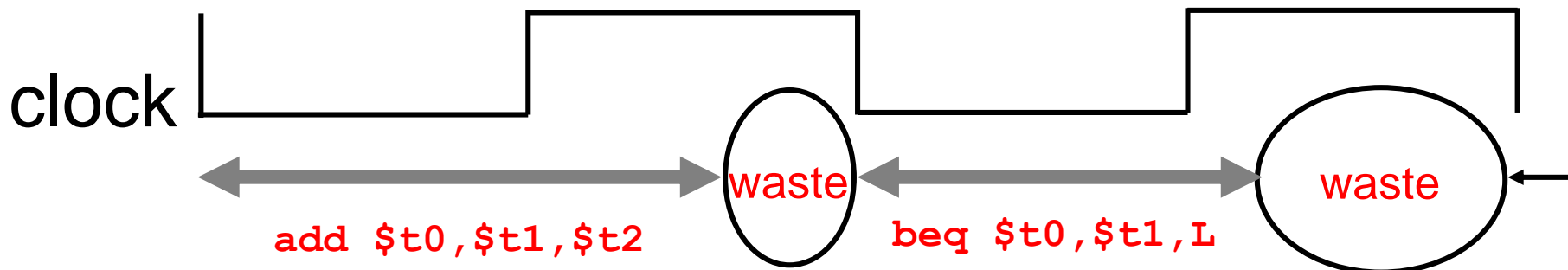
第四章

内容提要

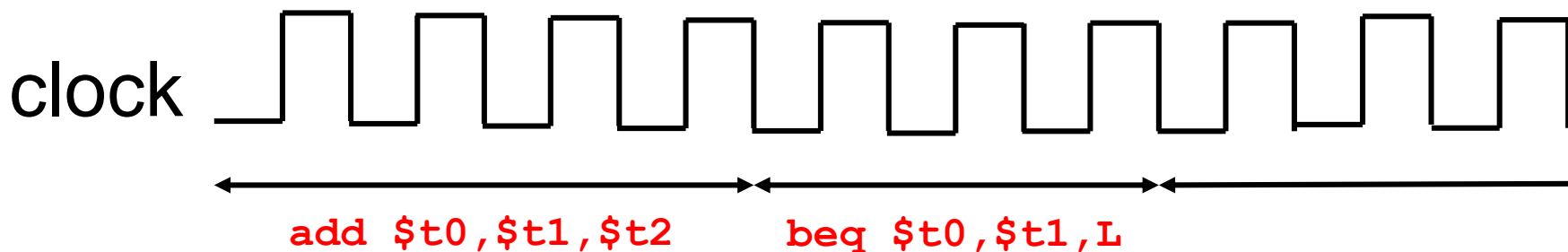
- 多周期设计优势：CISC采用
 - 时钟周期对性能的影响
 - 复用功能部件，减少硬件开销
- 多周期实现
 - 数据通路，控制器（状态机，微程序）
- COD5, RV Edition_2ed, \$4.5
- 唐本

指令周期: single-cycle vs. multicycle

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



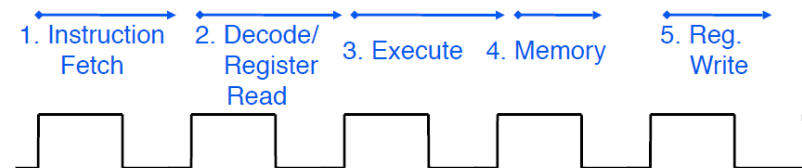
Multicycle Implementation: 不定长指令周期, 定长CC



- **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）

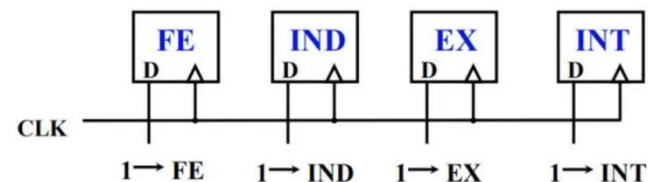
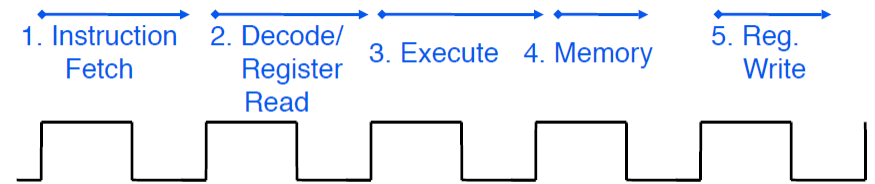
- 写回：lw写回（完成）

- 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

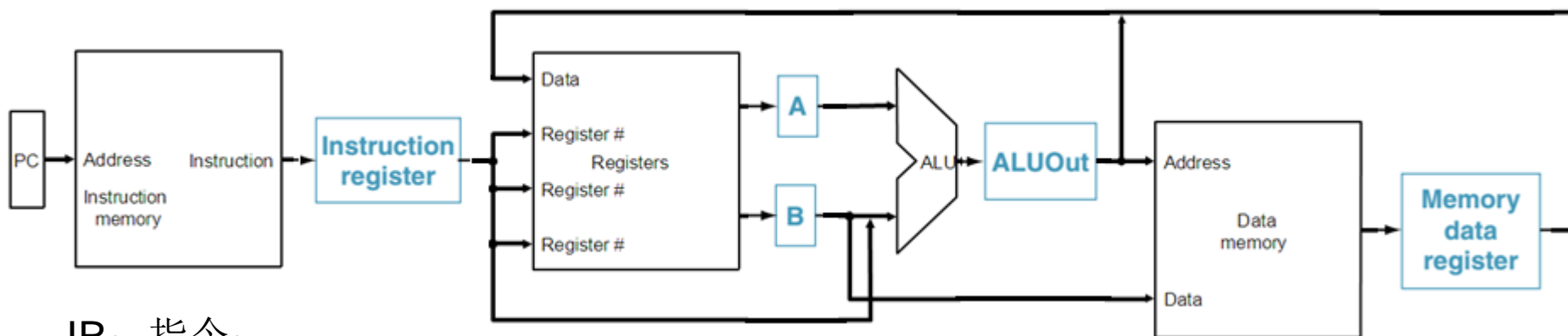
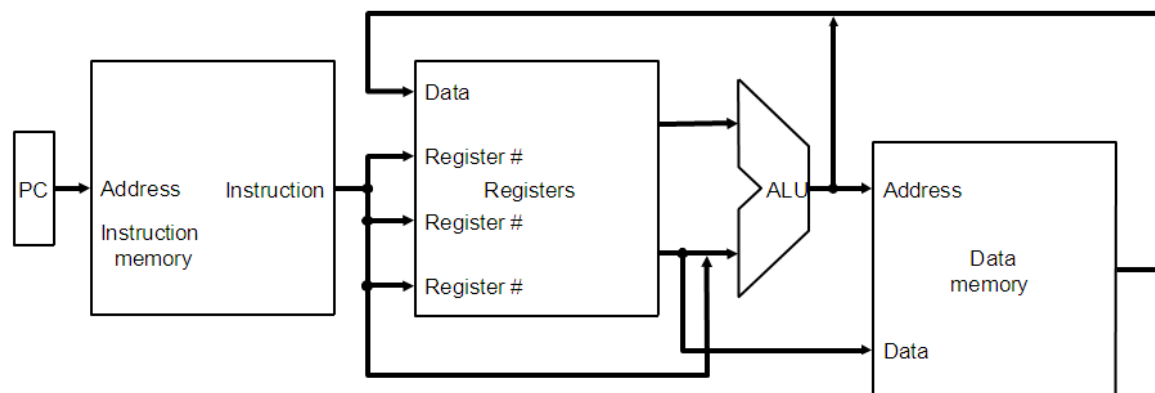
- 周期标志触发器：clk gating

- 状态机：状态ID



唐图8.9

多周期DP: IF-ID-EX-MM-WB 暂存



IR: 指令;

A、B: 源操作数

ALUOut: ALU结果

MDR: 访存结果。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/MA, $IR=MDR$?

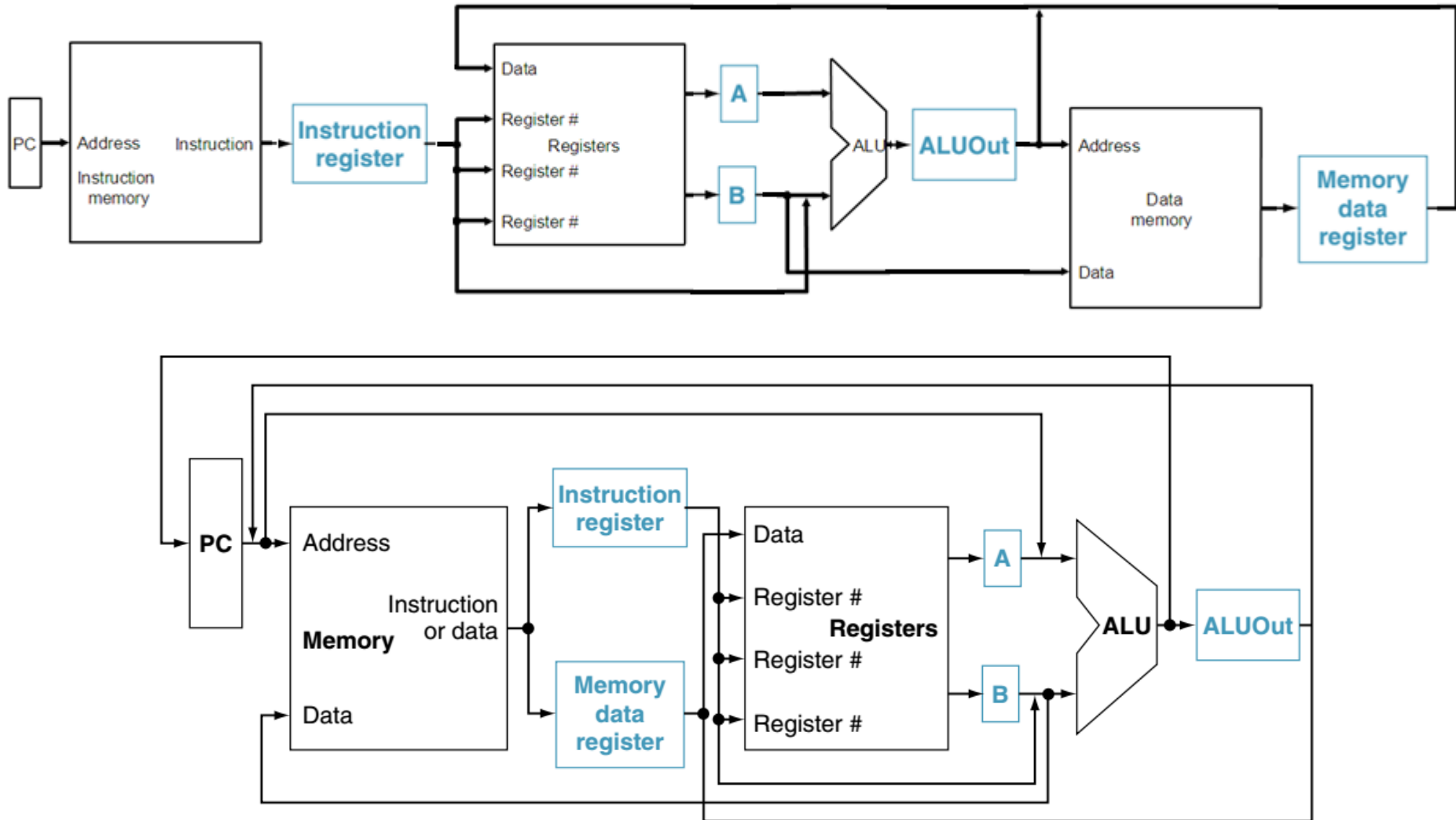
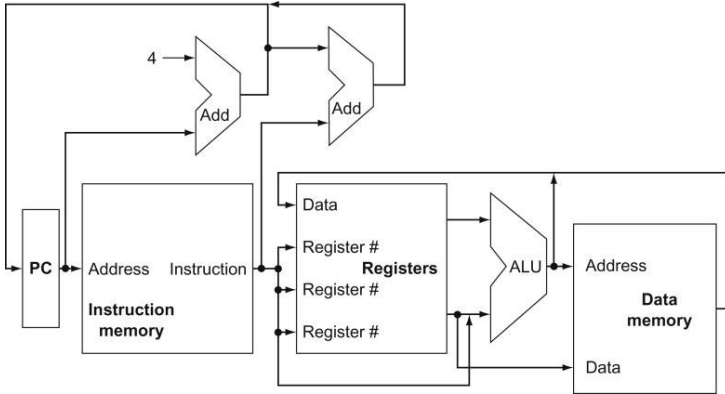


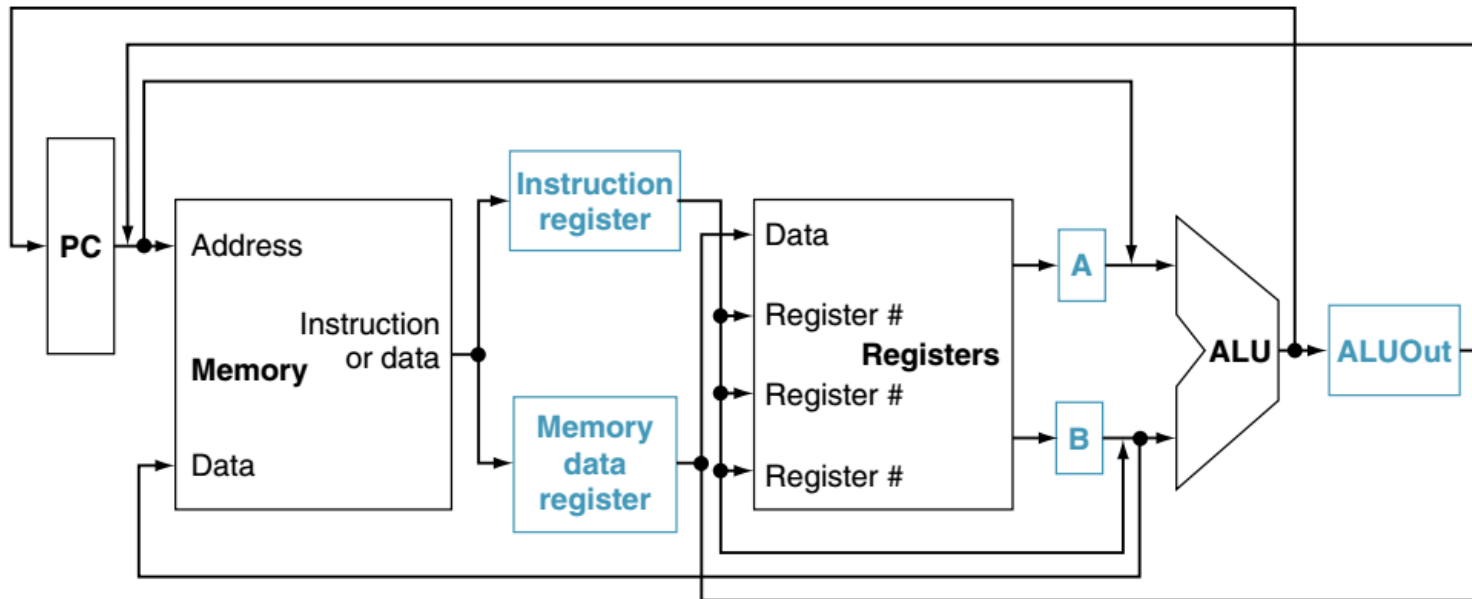
FIGURE e4.5.1, The high-level view

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

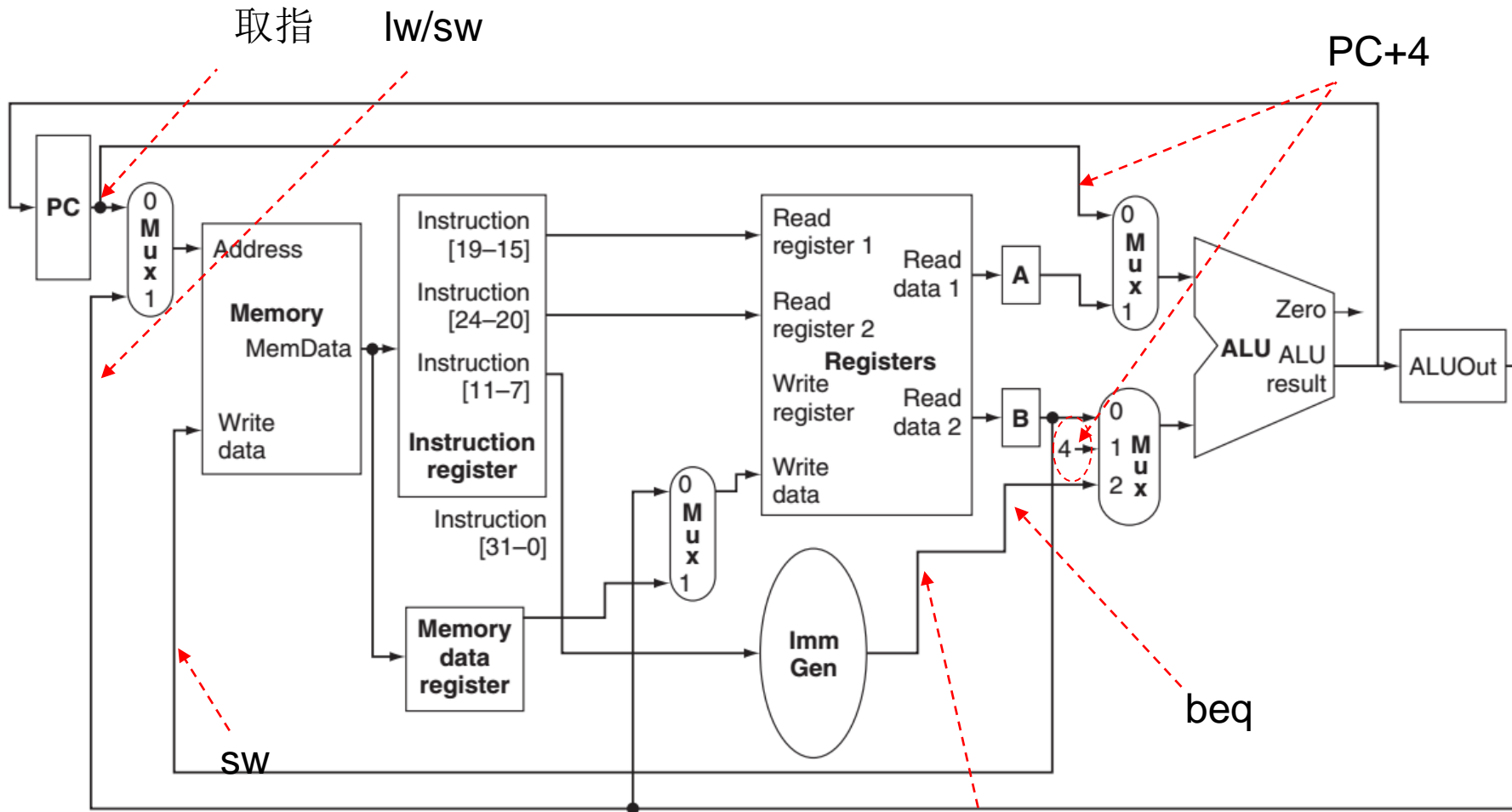


FIGURE e4.5.2

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

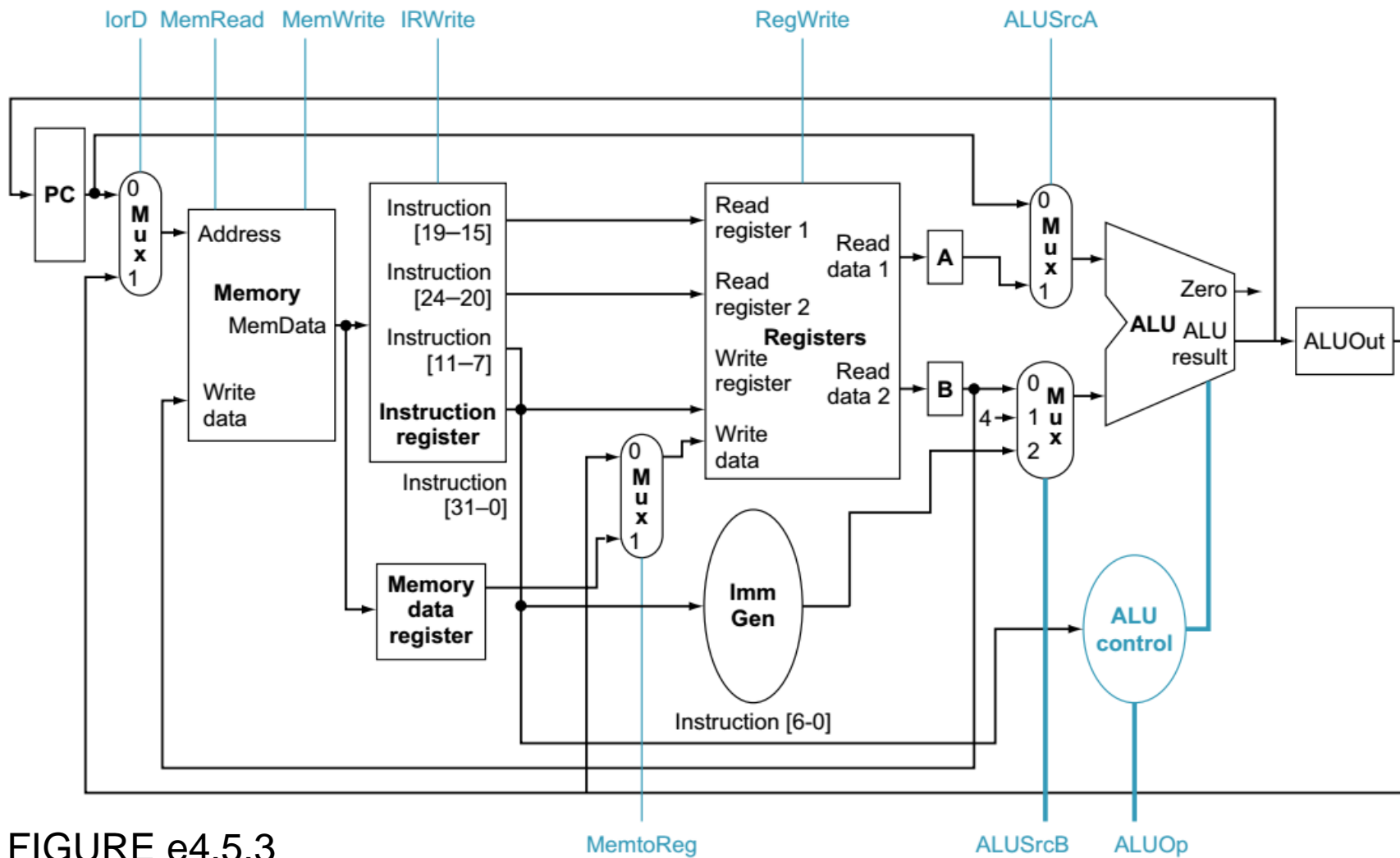
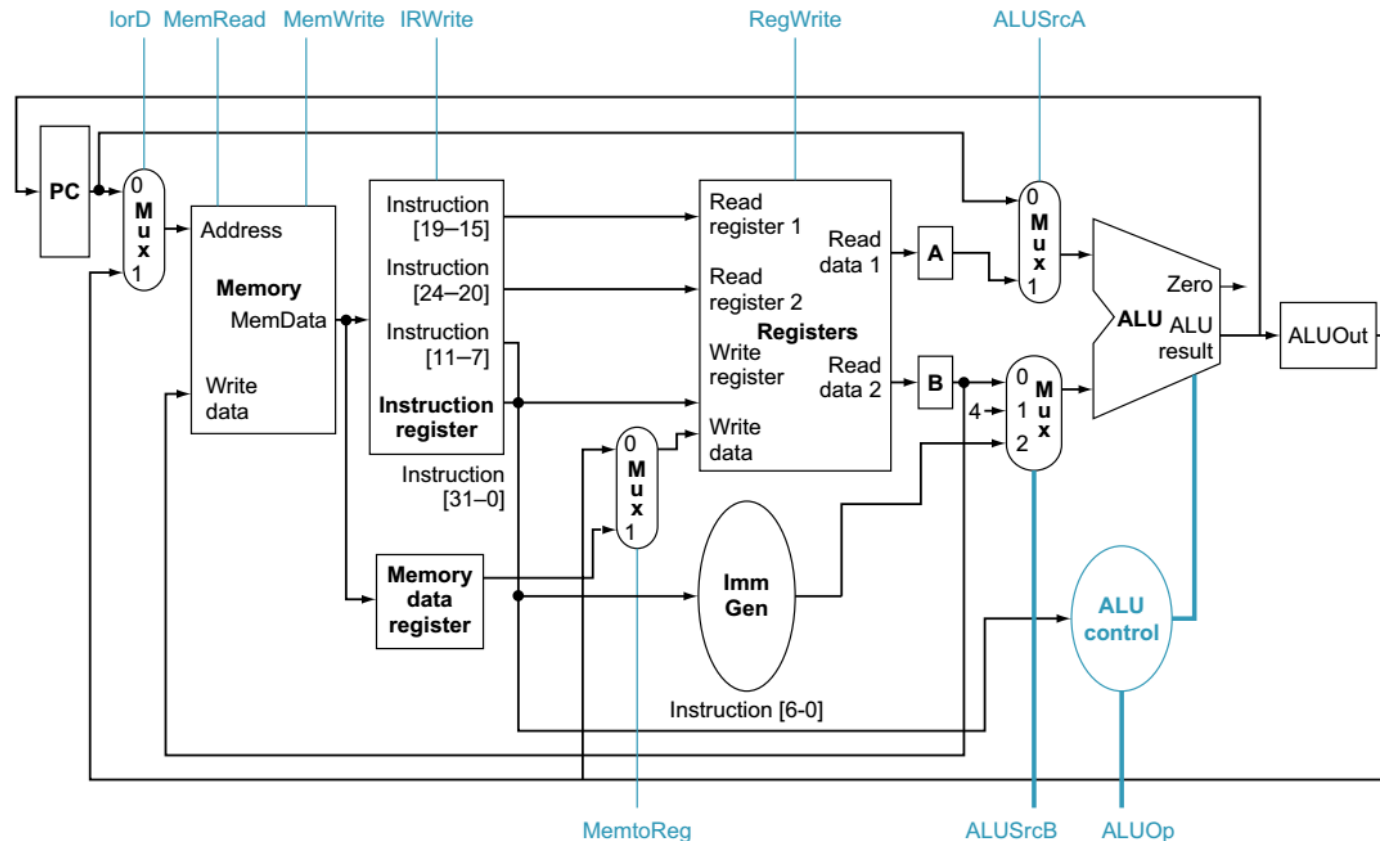


FIGURE e4.5.3

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, MemWrite, IRWrite; ALUSrcA, ALUSrcB, ALUOp, *PCWrite*



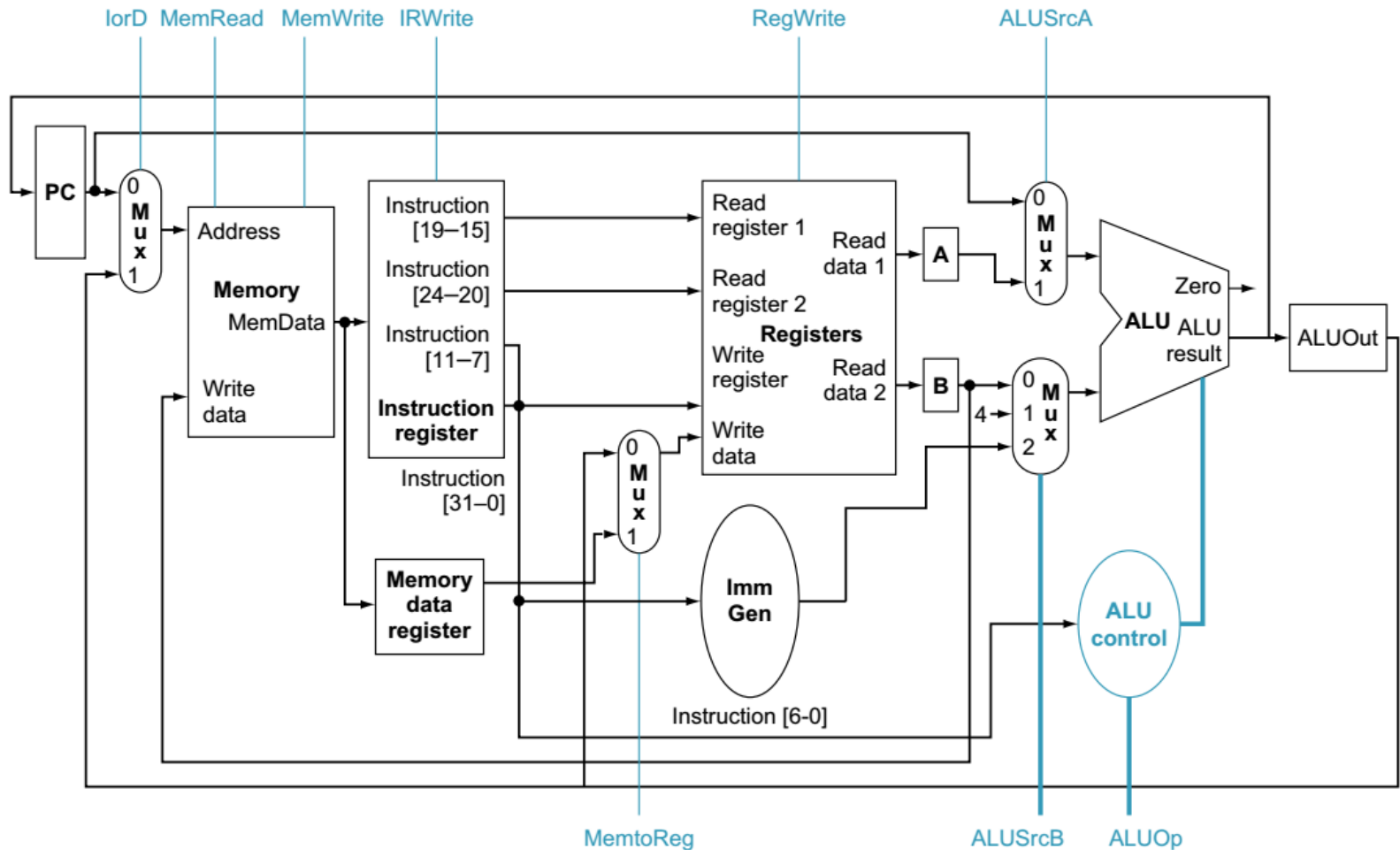
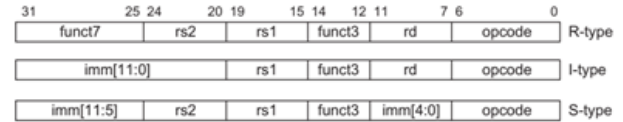
PC “后写”？

FIGURE e4.5.3

第二阶段：译码

- 指令译码和寄存器读

- 将rs和rt送往A和B: $A = \text{Reg}[\text{IR}[19-15]]$, $B = \text{Reg}[\text{IR}[24-20]]$



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

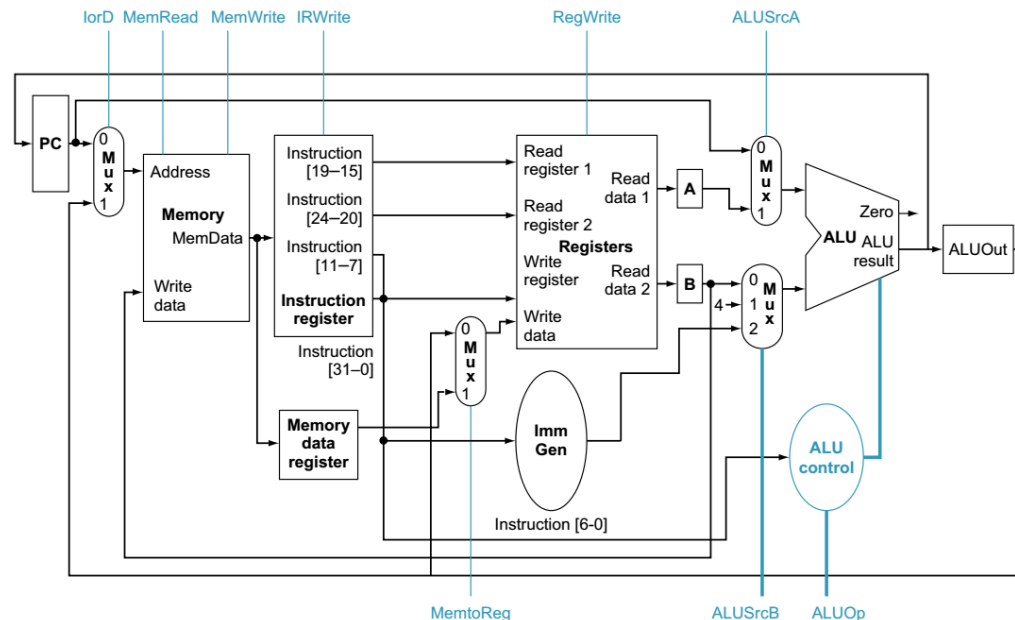
- $ALUOut = A \text{ op } B$

- 访存指令：计算访存地址

- $ALUOut = A + (\text{sign-extend}(\text{IR}[\text{Imm}]))$

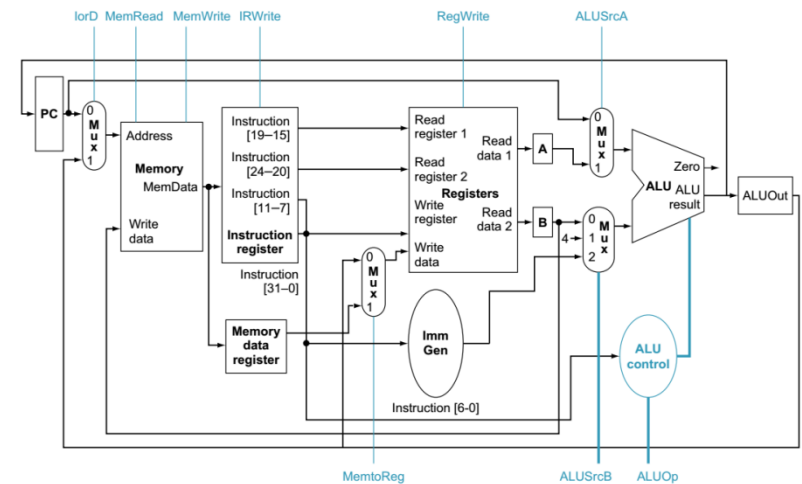
- 需要的控制信号？

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
 - $\text{Reg}[\text{IR}[11-7]] = \text{ALUOut}$
 - sw完成：写入MEM
 - $\text{MEM}[\text{ALUOut}] = \text{B}$
 - lw读：
 - $\text{MDR} = \text{MEM}[\text{ALUOut}]$
- 第五阶段：lw写阶段
 - lw写回： $\text{Reg}[\text{IR}[11-7]] = \text{MDR}$
- 所需的控制信号？

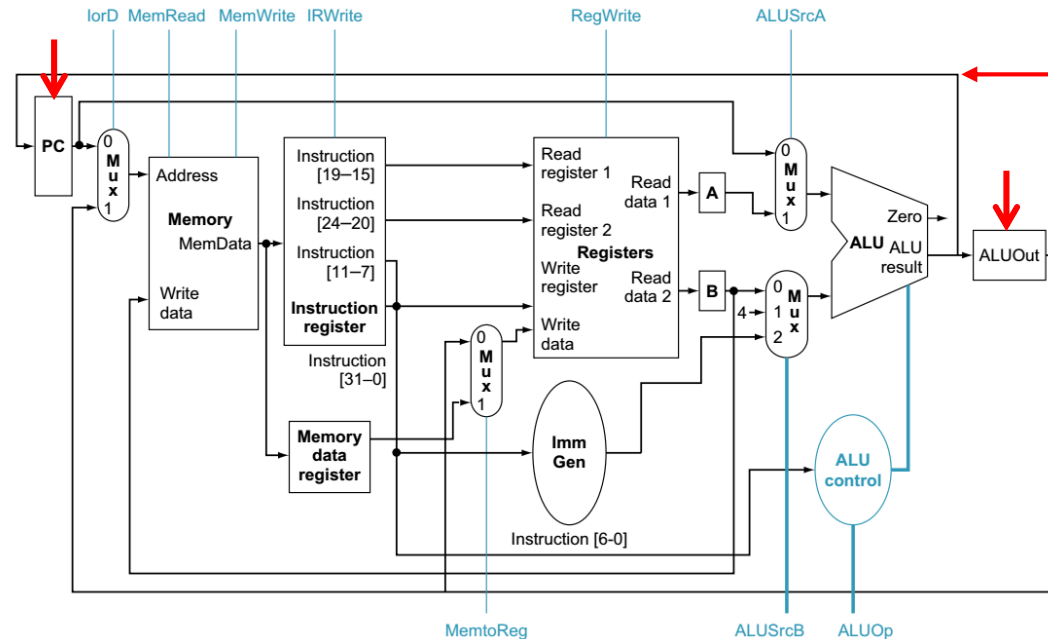


beq执行过程

- 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
 - ALUOut写控制？
- 执行周期：beq比较，if (A == B) nPC=ALUOut；
 - ALUOut->PC的通路？见下页
 - PC写控制？见下页
 - ALUOut写控制？
 - 如果非beq，则允许写ALUOut！
- 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，执行周期禁止写ALUOut？！

FIGURE e4.5.3



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

多周期总图：每个周期哪些部件在idle？

IF: PCsource+PCWrite

EX: PCsource+PCWriteCond

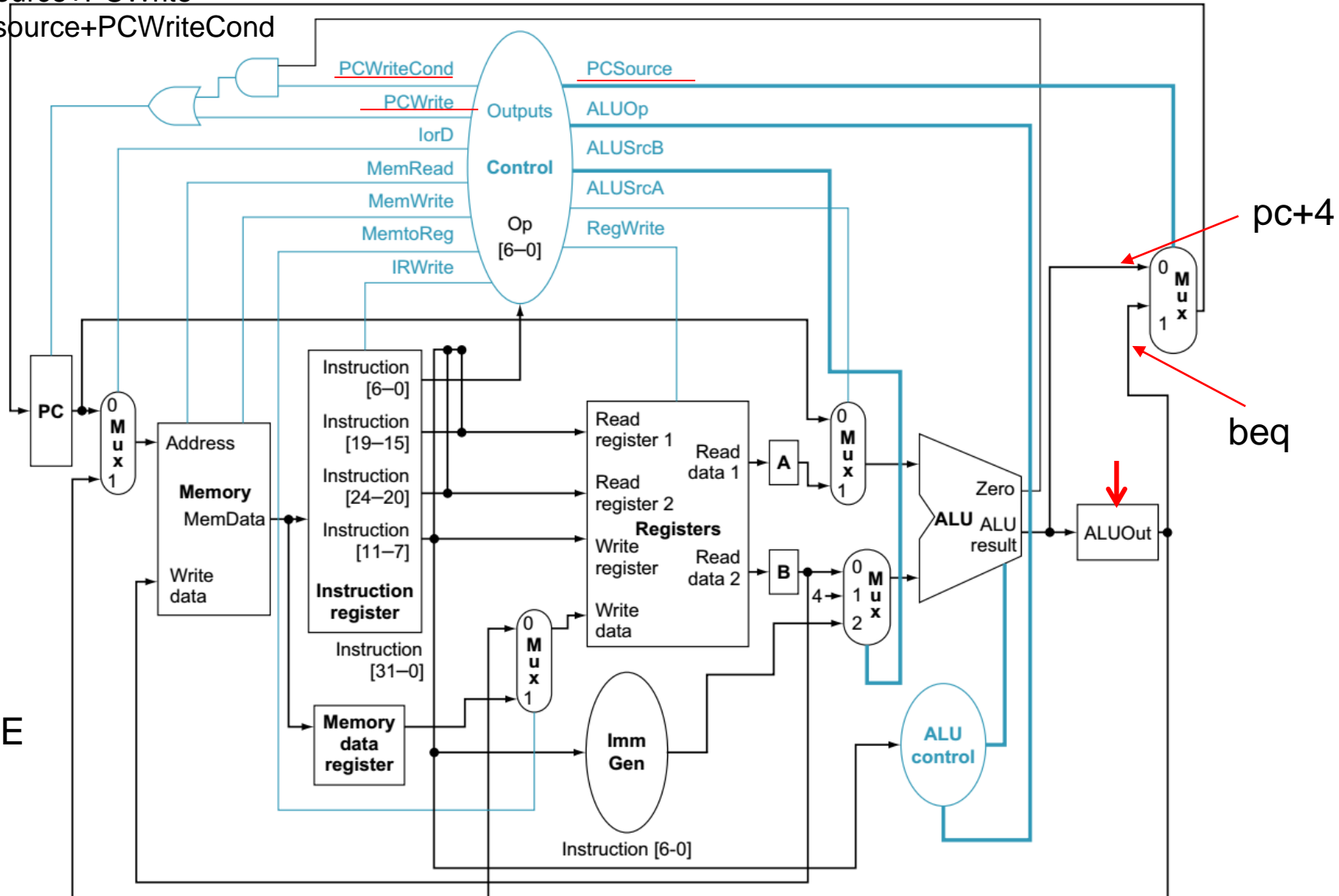


FIGURE e4.5.4

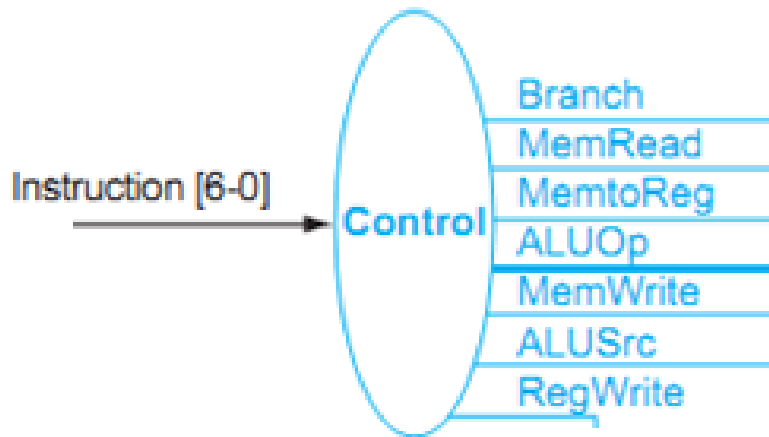
Multicycle指令时序

FIGURE e4.5.6

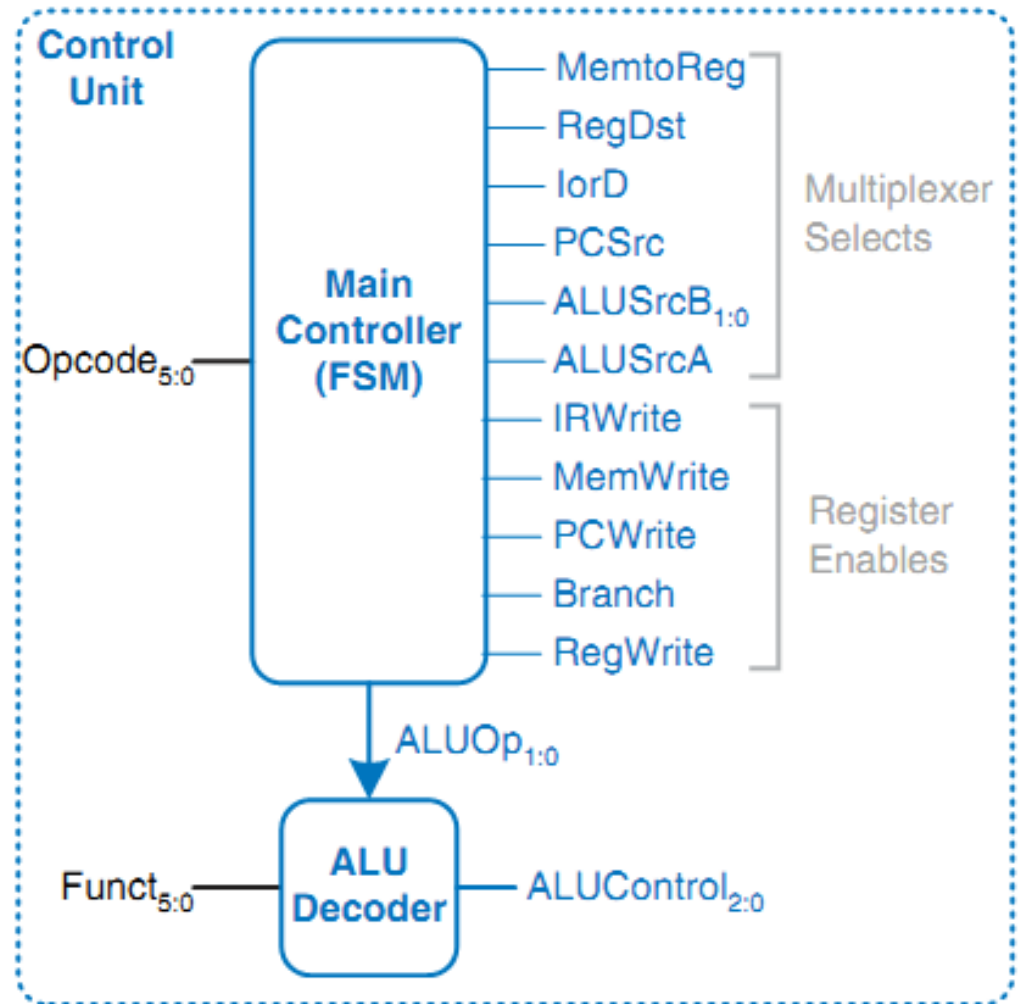
Step name	Action for R-type instructions	Action for memory reference instructions	Action for branches
Instruction fetch	$IR \leftarrow \text{Memory}[PC]$ $PC \leftarrow PC + 4$		
Instruction decode/register fetch	$A \leftarrow \text{Reg}[IR[19:15]]$ $B \leftarrow \text{Reg}[IR[24:20]]$ $ALUOut \leftarrow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

- J-type?

Control unit internal structure



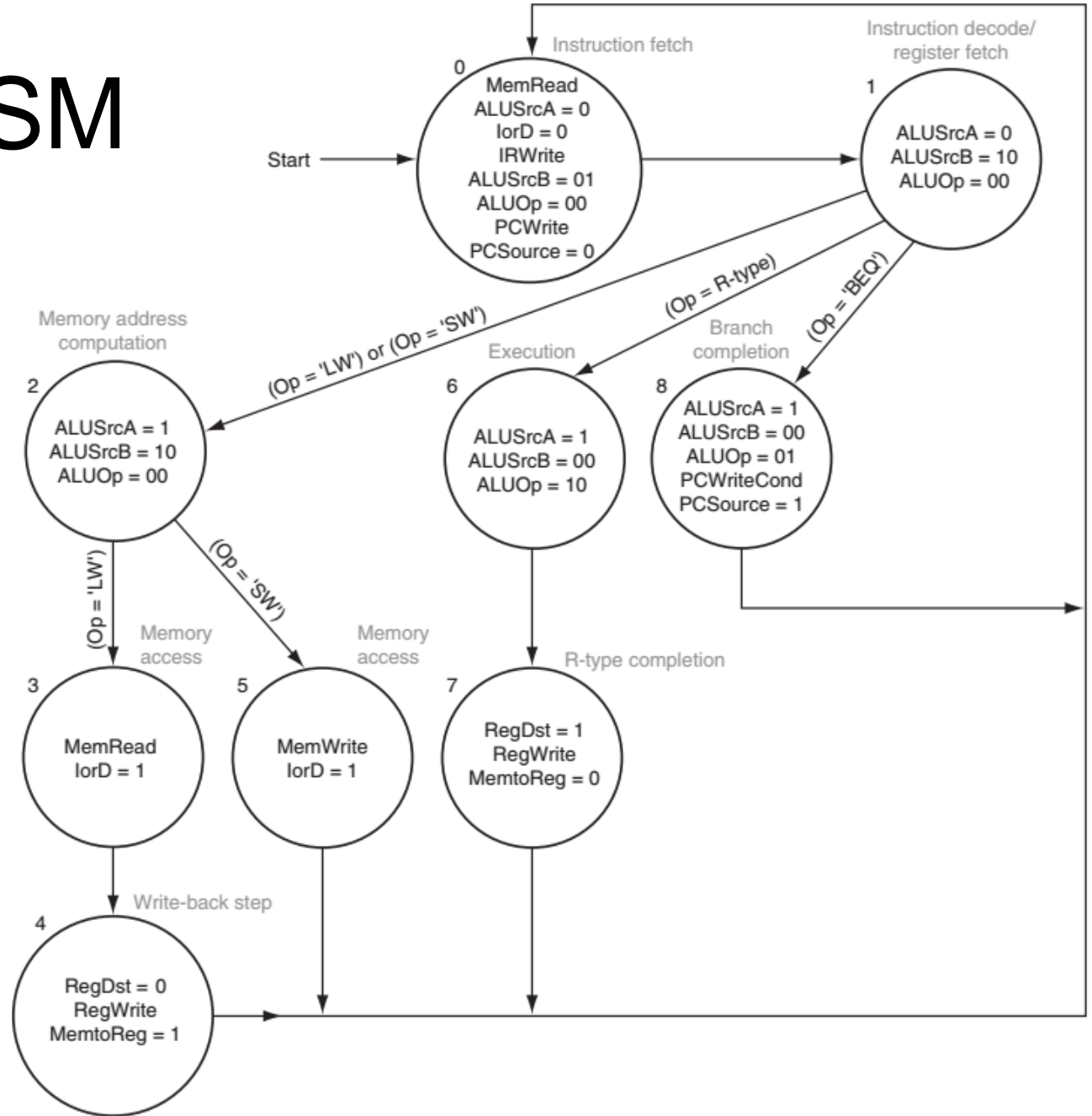
- opcode译码
- RV32I信号名与MIPS不完全相同



多周期FSM

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

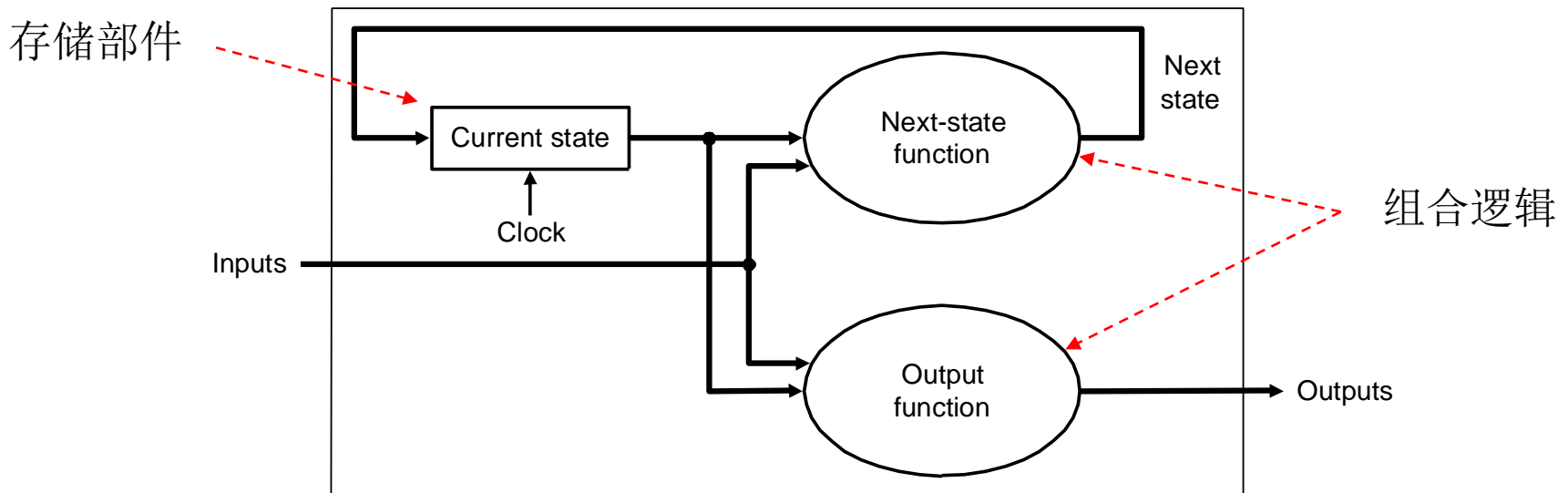
FIGURE C.3.1



beq的执行态缺PCWrite?

FSM控制部件实现, §5.9.2

- 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：组合逻辑方式

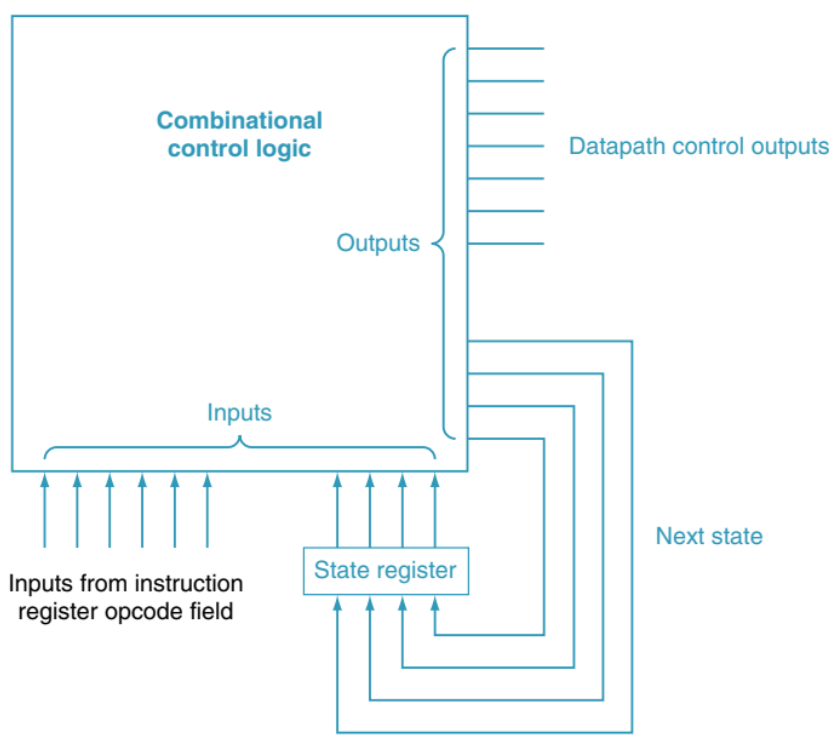
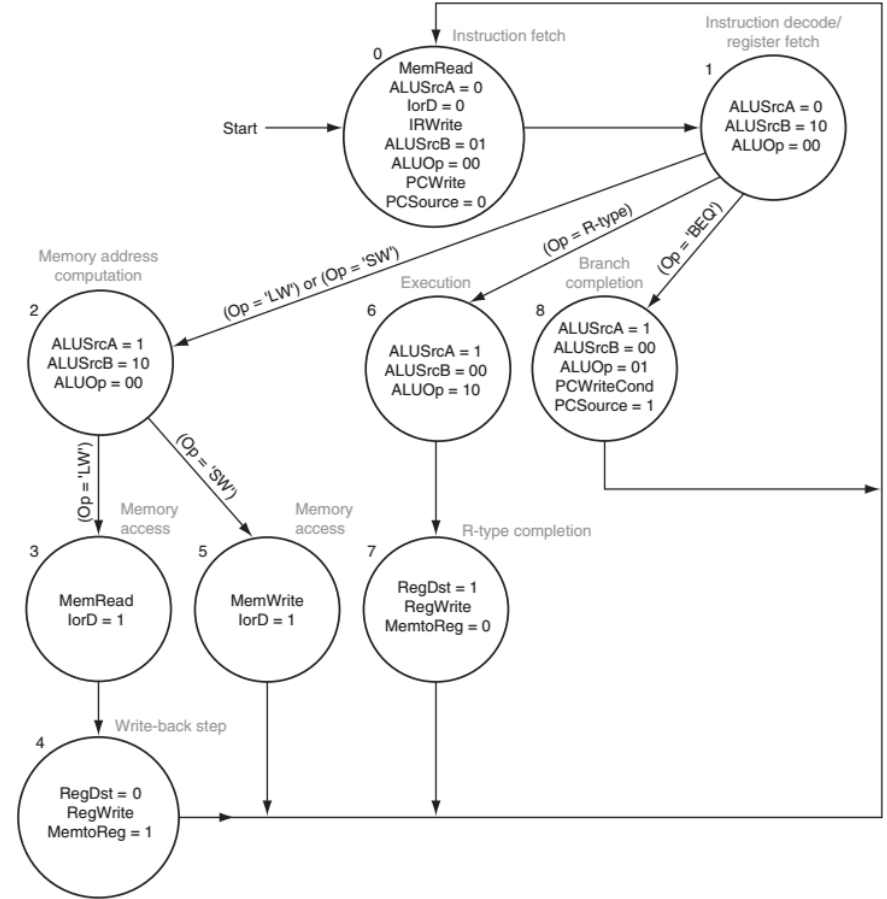


FIGURE e4.5.13 using a block of combinational logic and a register to hold the current state



另：参考唐10.1.3

FSM控制器实现：PLA方式

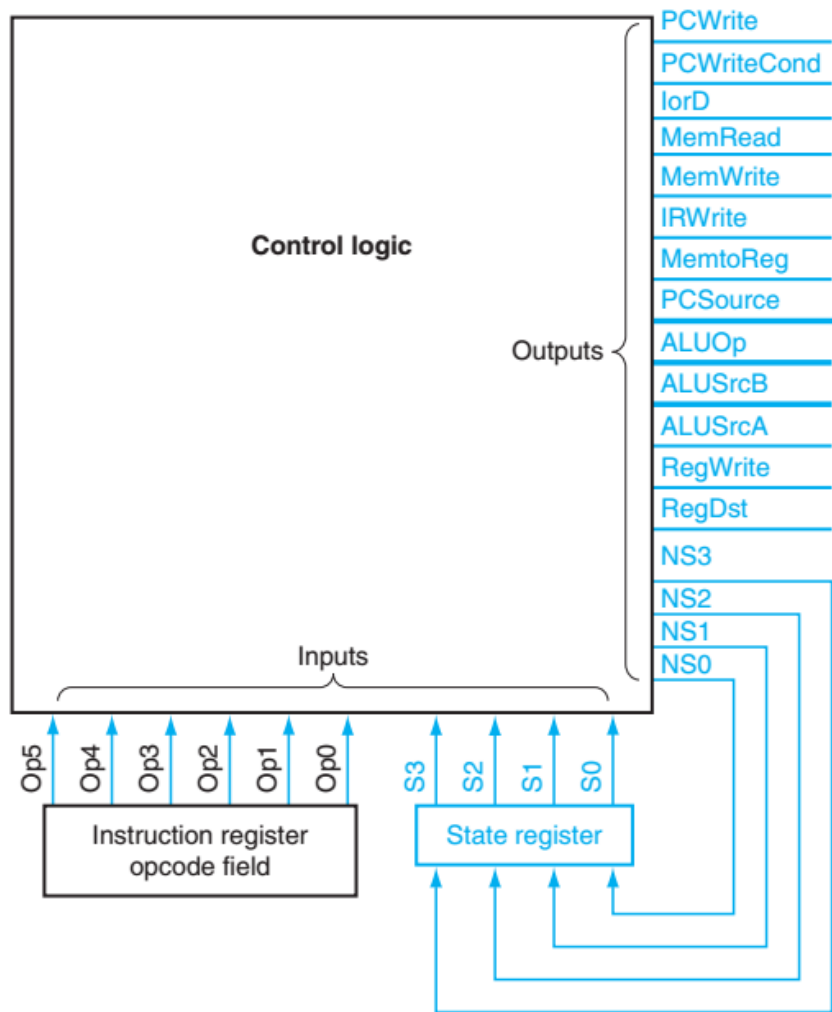


FIGURE C.3.2

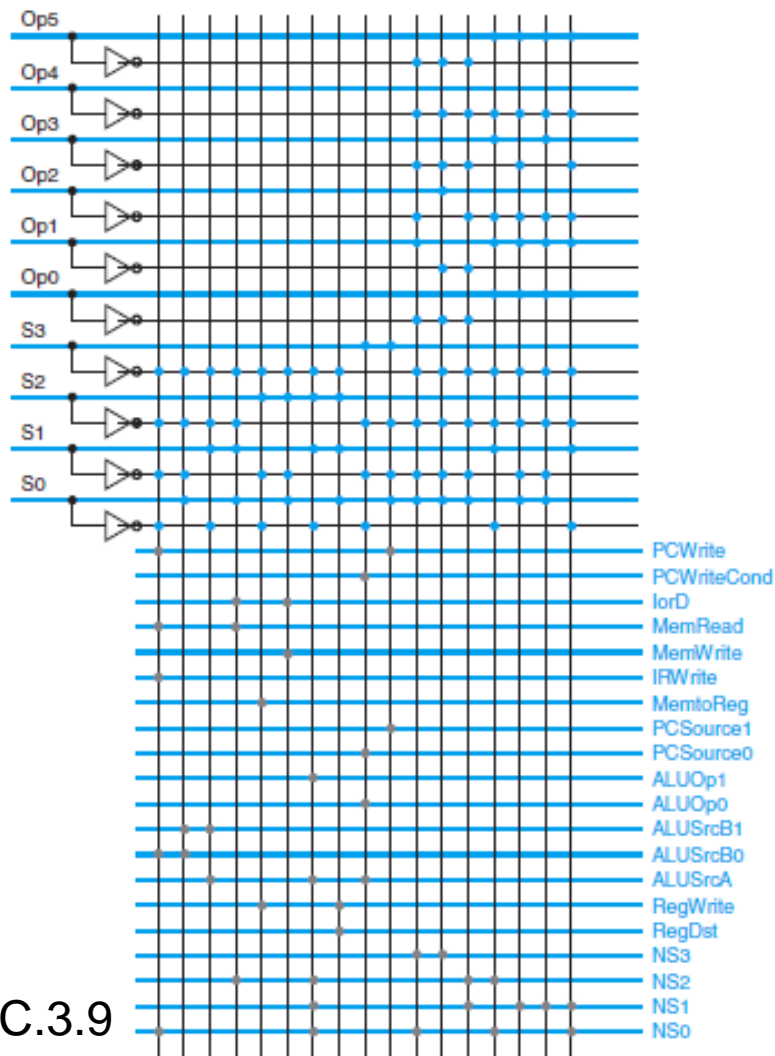


Fig C.3.9

顺序器实现

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

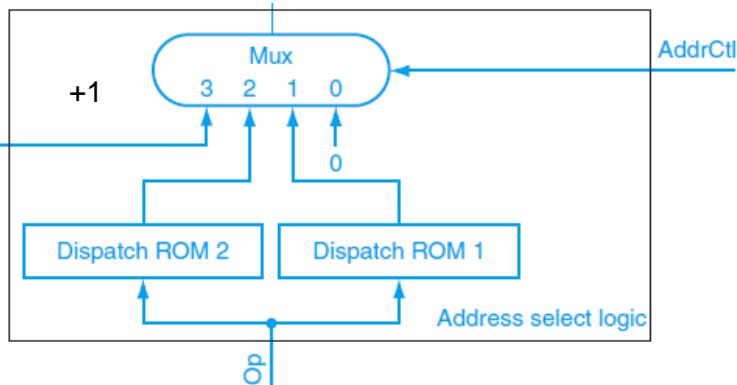


Fig C.4.2

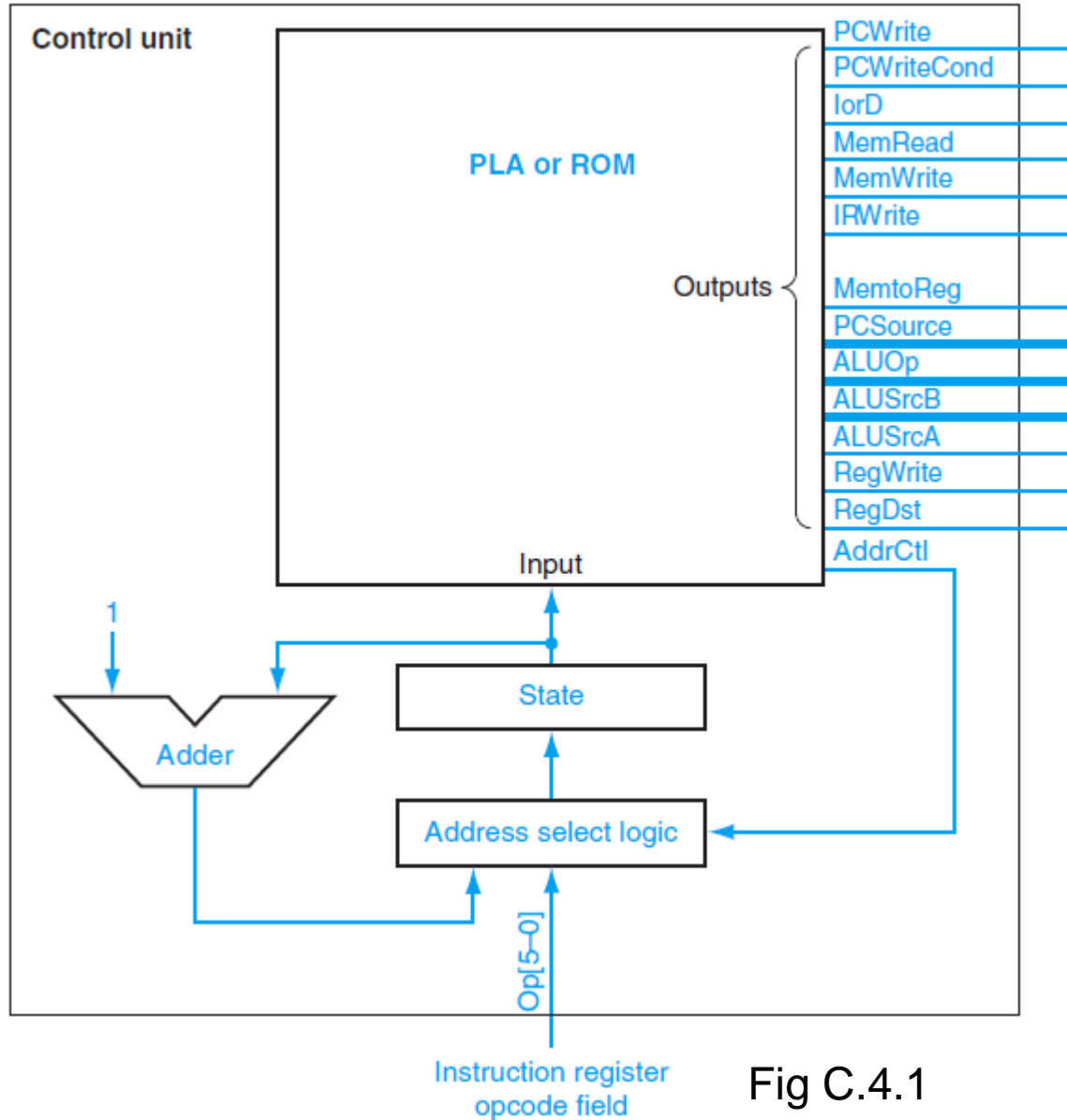


Fig C.4.1

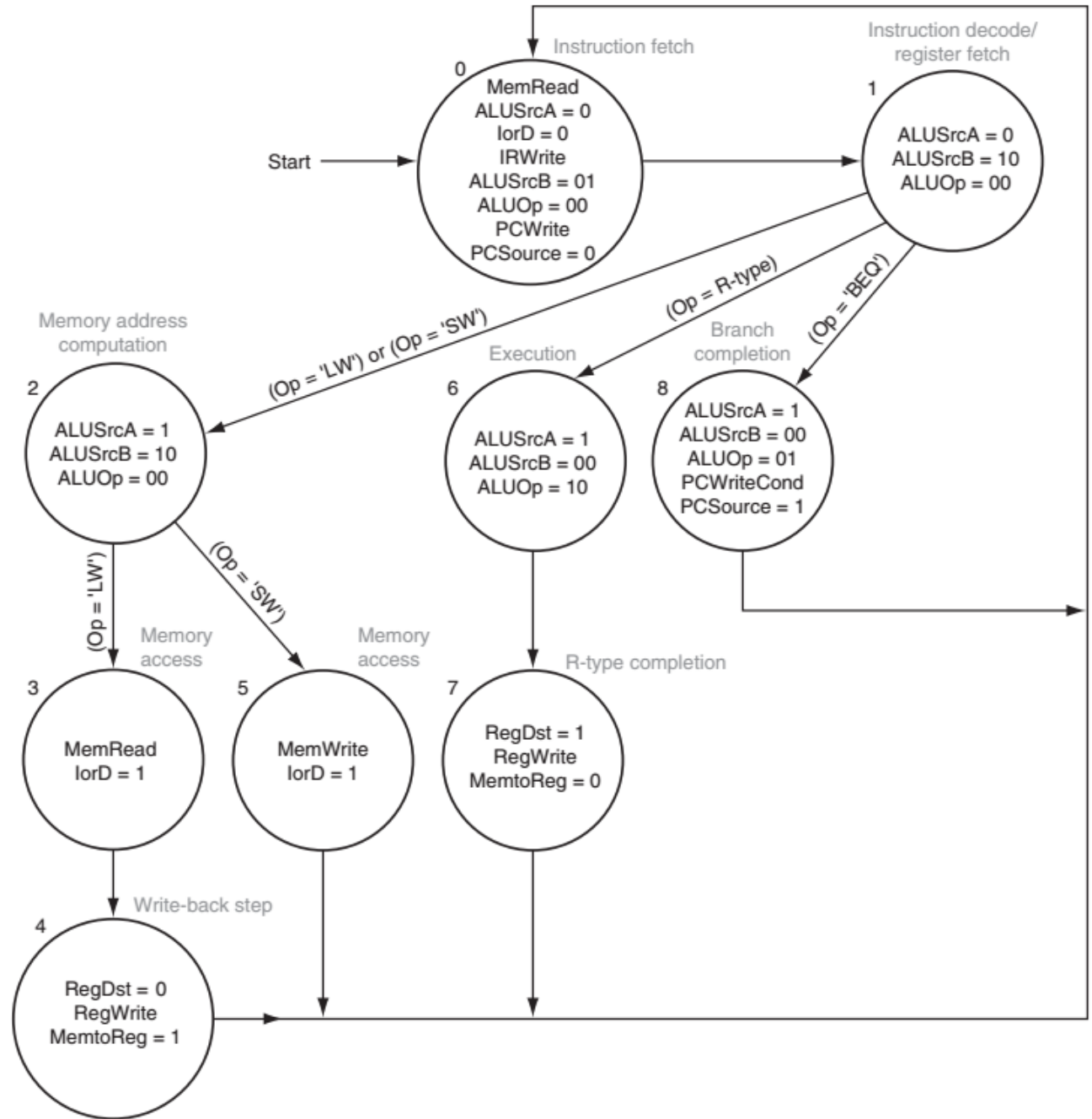
J-type?

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

FIGURE C.3.1

jal、jalr指令?
lui指令?

nop指令?
异常?



多周期微结构中，CPI = ?

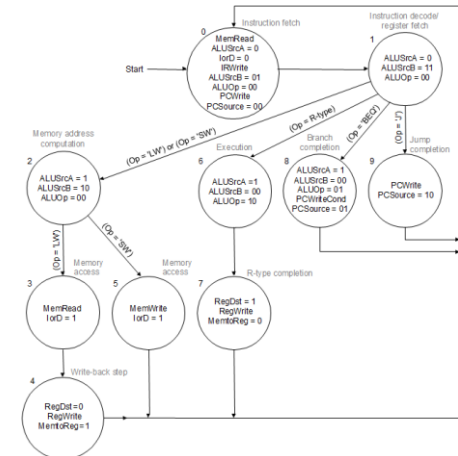
- CPI is always an **average** over a large number of instructions.
 - $CPI = \text{CPU clock cycles} / \text{Instruction count}$

• 设

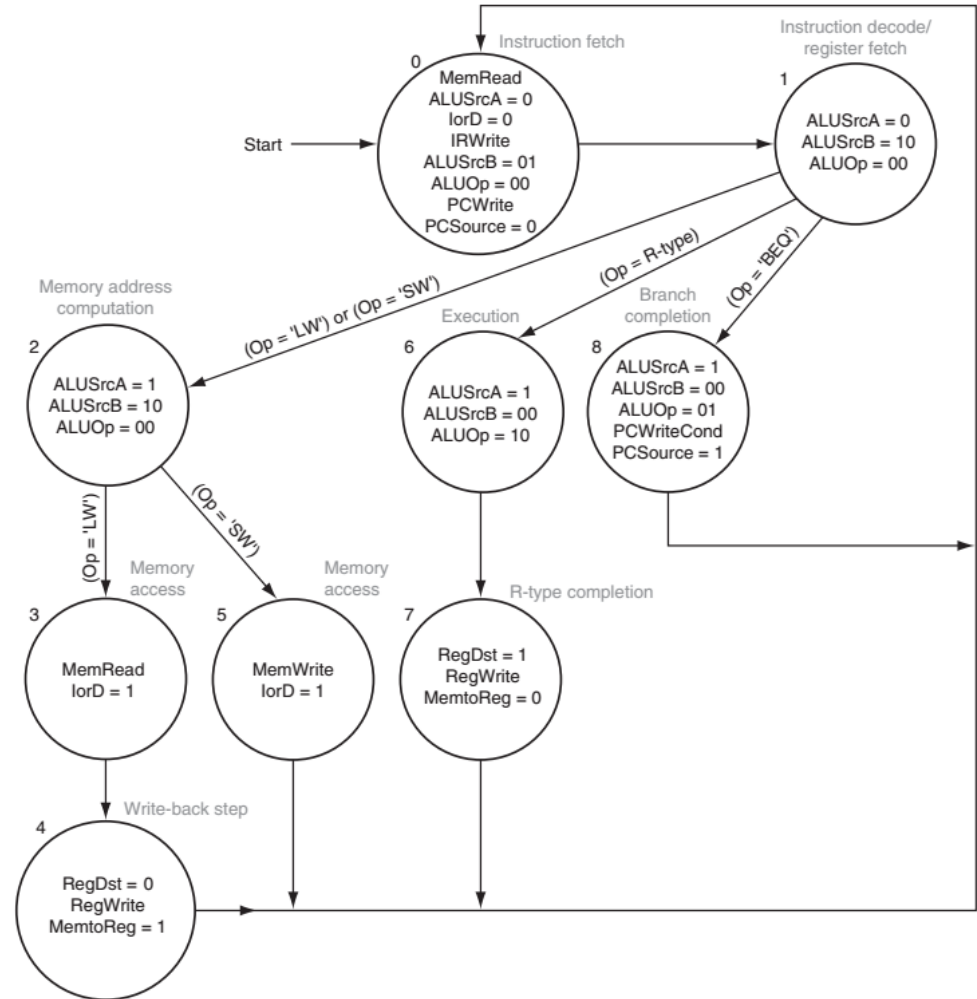
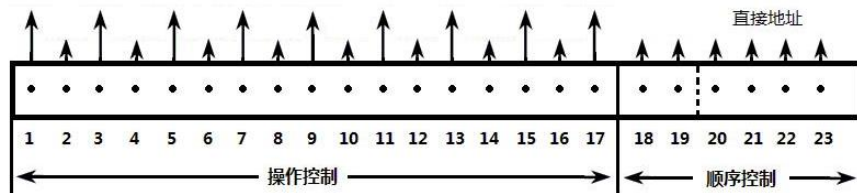
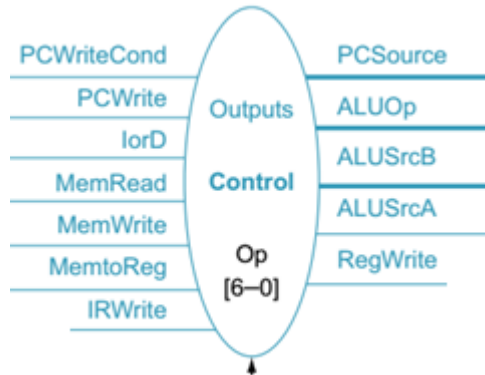
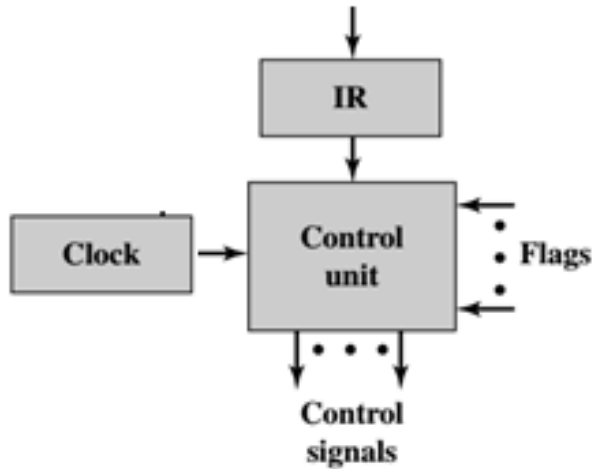
- SPECINT2000中，load为25%（取byte占1%，取word为24%），store为10%（存byte为1%，存word为9%），ALU占52%，branch为11%（6%beq，5%bne），jump为2%（1%jal+1%jr），则
- 指令周期定长时：CPI=5.0
- 指令周期不定长时：CPI=4.12

$$\bullet = 0.25 \times 5 + 0.10 \times 4 + 0.52 \times 4 + 0.11 \times 3 + 0.02 \times 3 = 4.12$$

- 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

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。

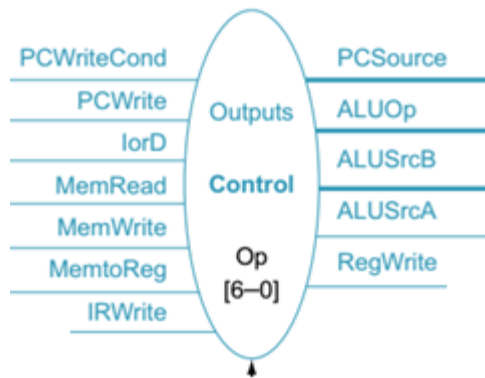
1964年IBM System/360首次采用了此技术。

优势：1) 避免组合逻辑控制器设计复杂性；2) 易于扩展和重构。



微程序控制器, §C.5

- Microcode
 - 控存



控制字段 顺序字段

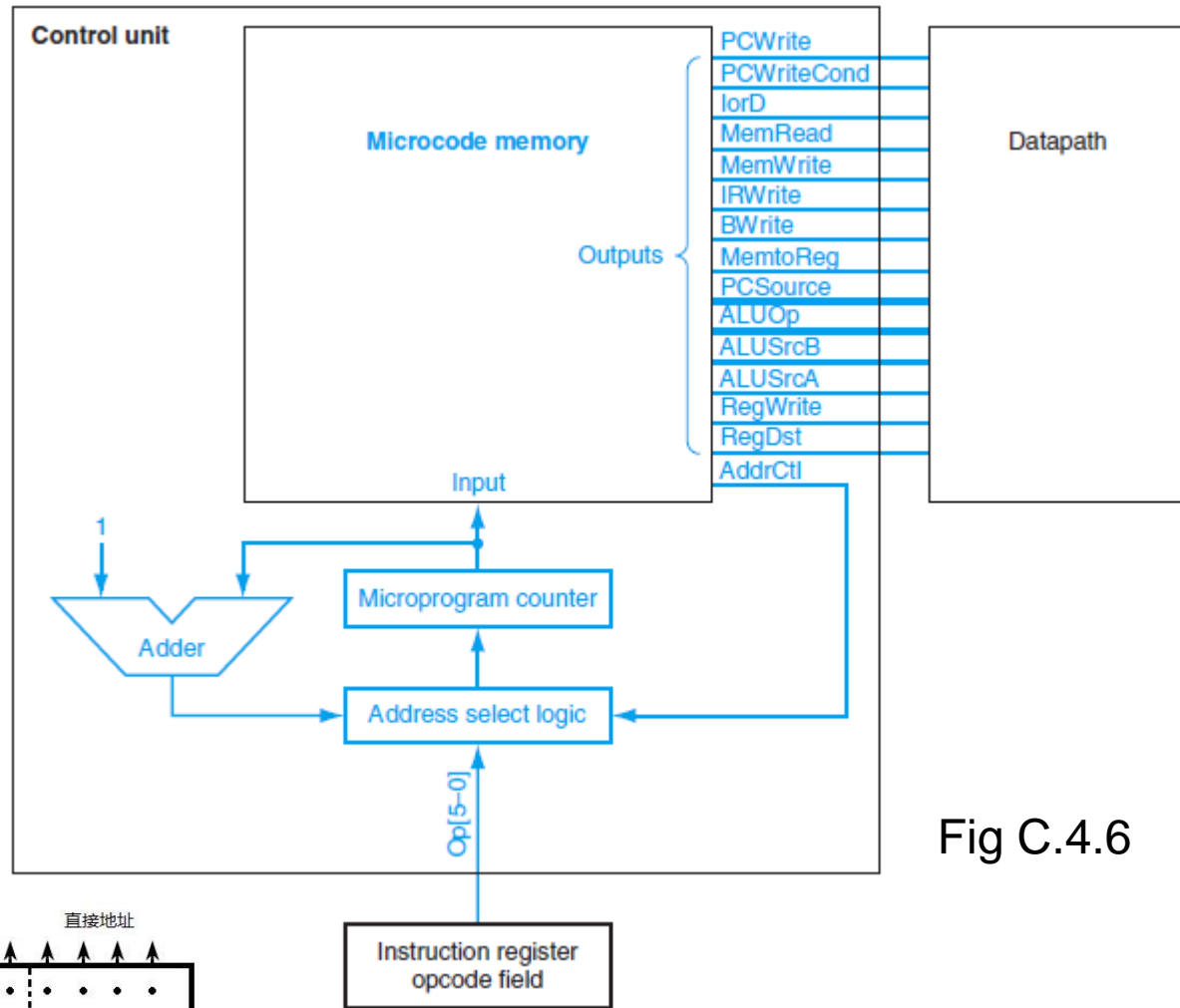
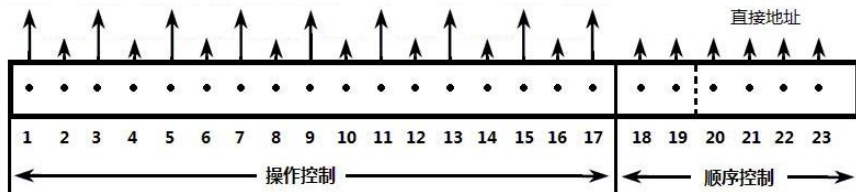
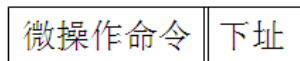
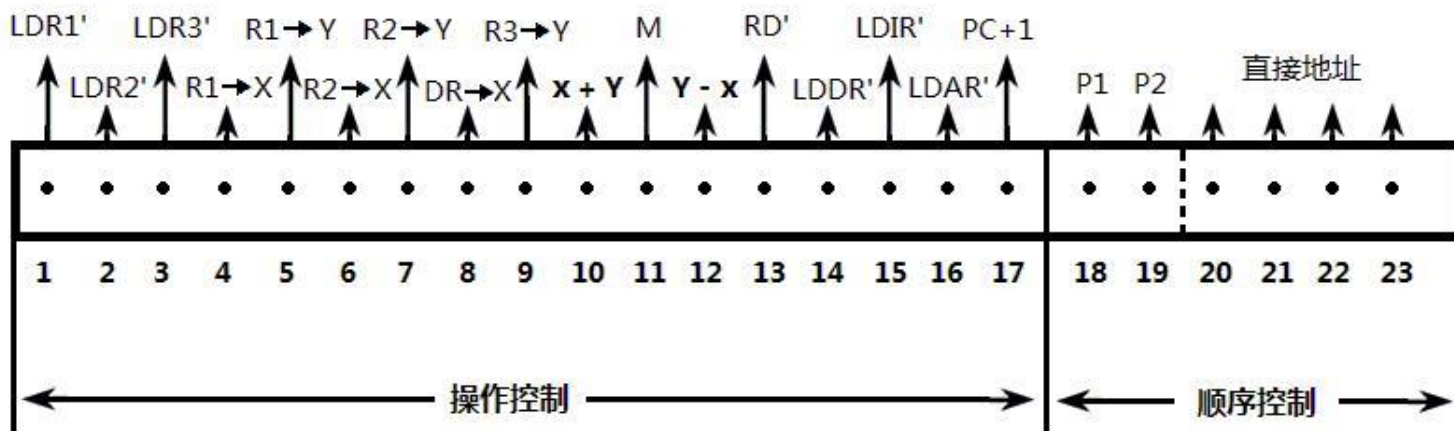
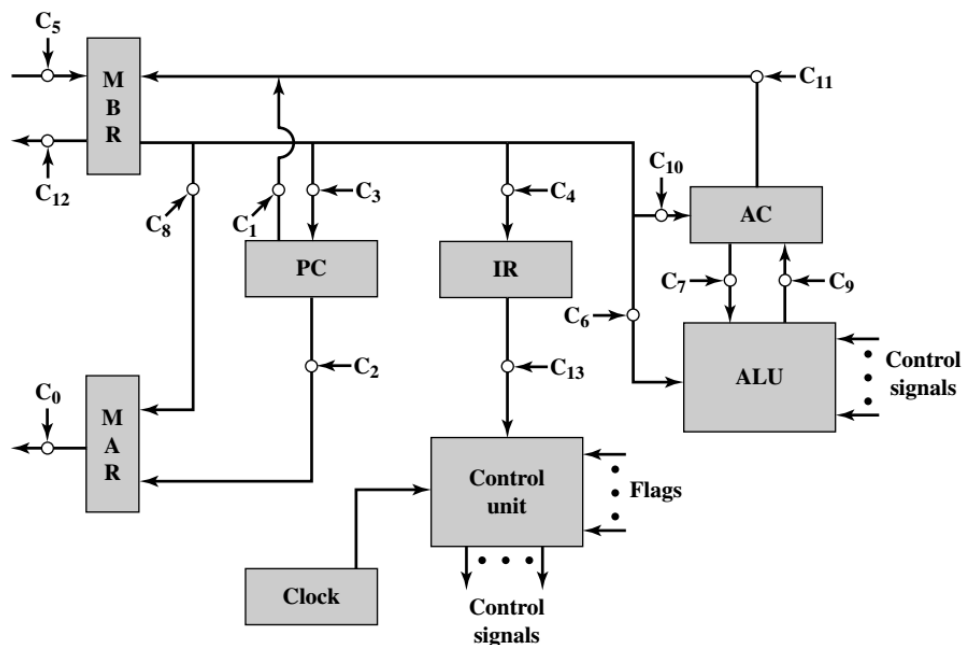


Fig C.4.6

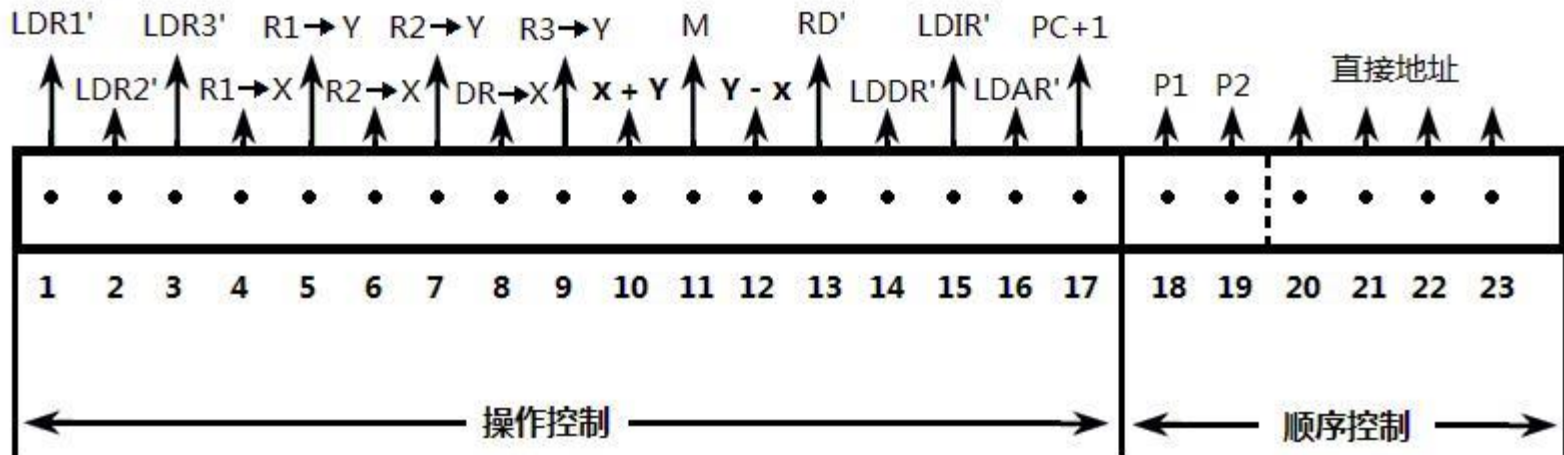
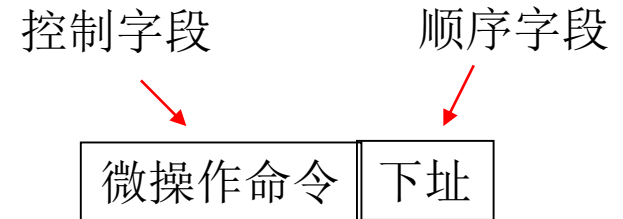
Overview: 微指令、微命令, 唐本

- 取指周期: C2、C5、C4、...
 - T_0 : PC \rightarrow MAR, 1 \rightarrow R
 - T_1 : M(MAR) \rightarrow MDR, PC+1 \rightarrow PC
 - T_2 : MDR \rightarrow IR, OP(IR) \rightarrow ID
- 间址周期:
 - T_0 : Ad(IR) \rightarrow MAR, 1 \rightarrow R
 - T_1 : M(MAR) \rightarrow MDR
 - T_2 : MDR \rightarrow Ad(IR)



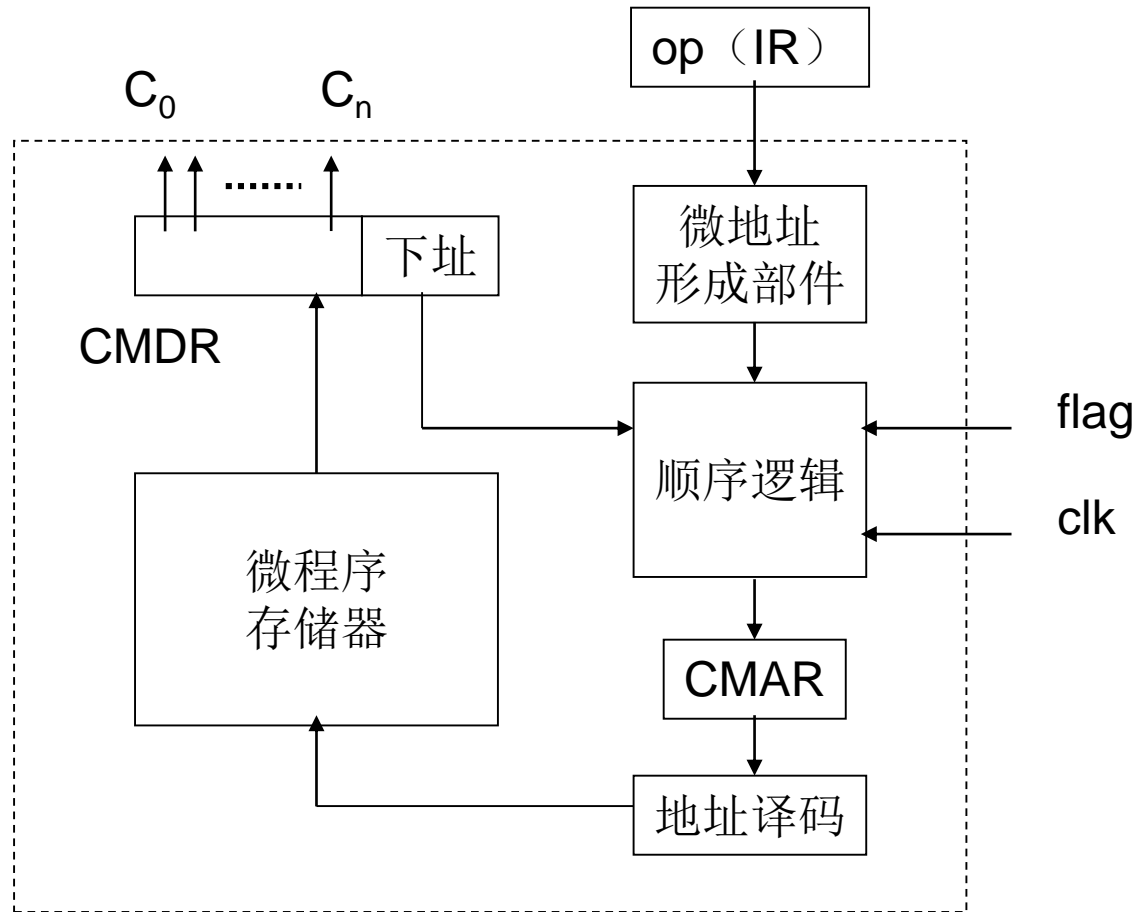
微程序

- 用一个微程序实现一条指令：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：水平，垂直
 - 微程序存储在“控存”中
- 微指令的格式：水平微指令



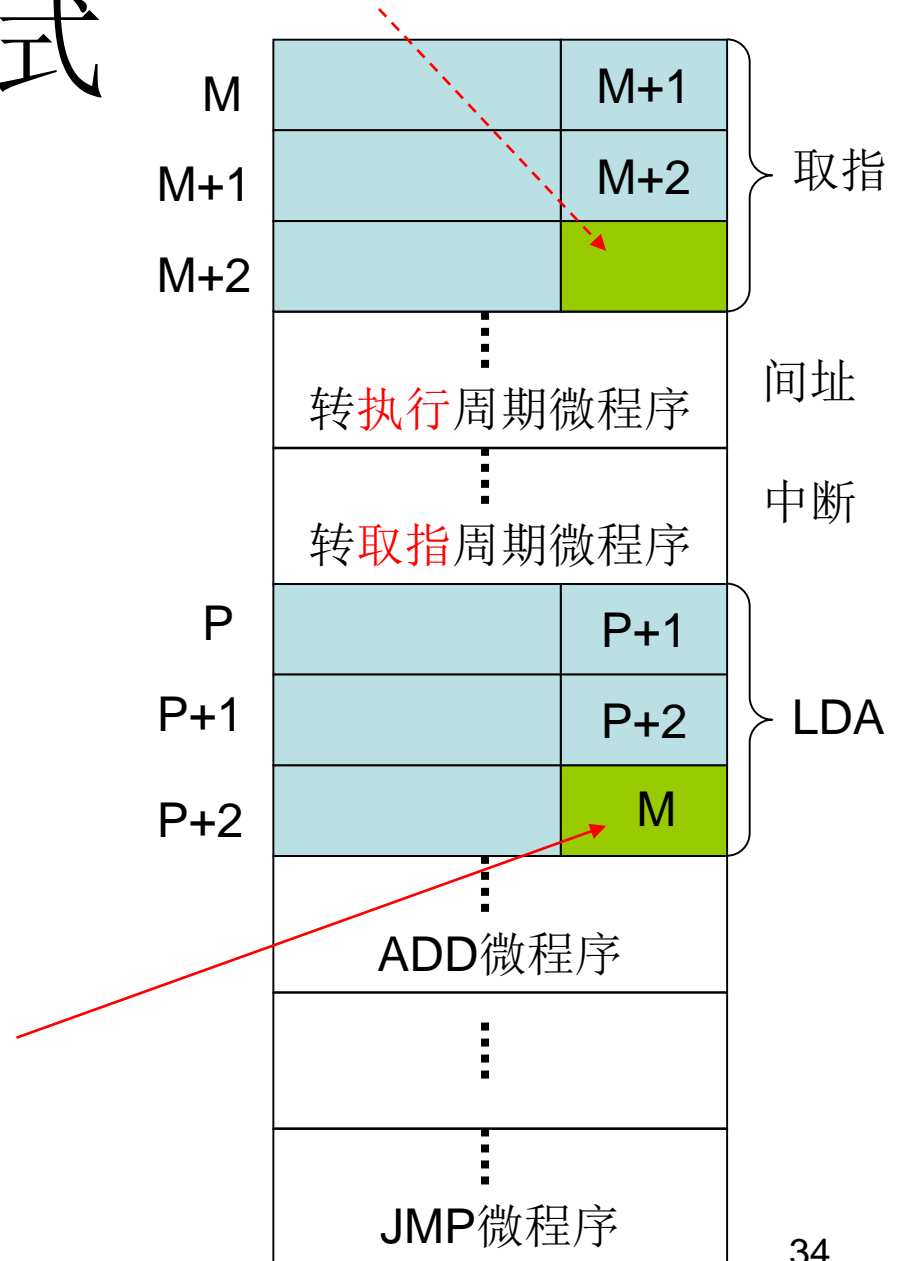
微程序控制部件的结构

- 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
- $C_0 \sim C_n$: 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 $3 + X$

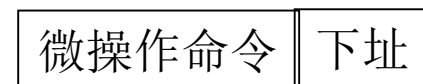


例：计算下址字段位数？

- 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**2**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 $32 \times 4 + 2 = 130$ 条微指令
 - $128 < 130 < 256$ ，因此需要**8**位寻址
 - 故下址字段至少**8**位。

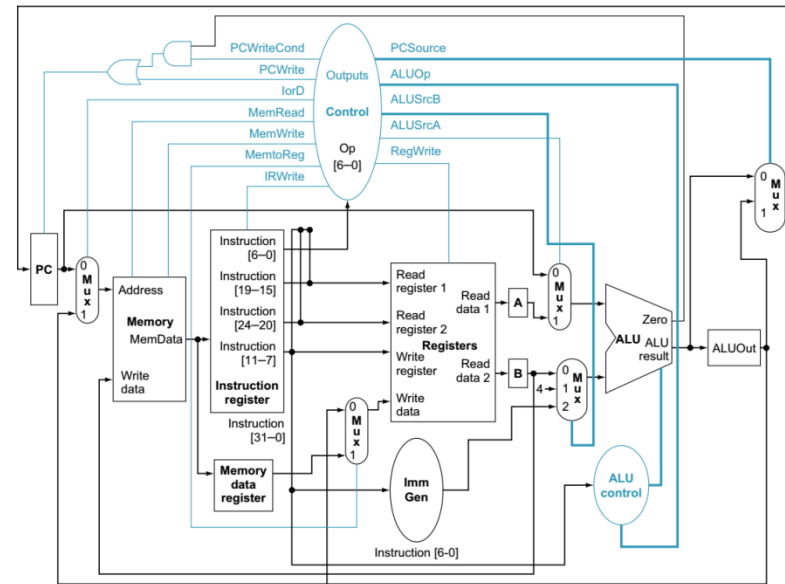
控制字段

顺序字段



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