中国科学技术大学
University of Science and Technology of China

Ch3.CPU设计-单周期、多周期

王 超

中国科学技术大学计算机学院
高能效智能计算实验室

2021年春

主要内容

- 处理器的基本结构
- 单周期CPU设计
  - ✓功能部件
  - ✓数据通路
- 多周期处理器设计
  - ✓功能部件
  - ✓数据通路
  - ✓控制信号

指令数据通路的Overview

- CPU性能因素
  - ✓指令数目:取决于ISA和编译器
  - ✓CPI 和时钟周期长度:取决于CPU硬件
- 我们将要考察三个RISC-V的实现
  - ✓一个是简化版本
  - ✓一个是多周期版本
  - ✓一个是更实际的流水线版本
- 用ISA的简单子集展现CPU设计的多数方面
  - ✓存储器访问：ld, sd
  - ✓算术/逻辑运算：add, sub, and, or
  - ✓控制转移：beq

指令数据通路Overview

- 实现不同指令的多数工作都是相同的，与指令类型无关
  - ✓取指：将PC送往MEM
  - ✓取数：根据指令字中的地址域读寄存器
- 执行操作各个指令不同，但同类指令非常类似
  - ✓算术逻辑指令
    - R类ALU指令 **add x1, x2, x3**
    - I类ALU指令 **addi x3,x3,4**
  - ✓不同类型指令也有相同之处，如都要使用ALU
    - 访存指令使用ALU计算地址 **lw x1,100(x2)**
    - 算逻指令使用ALU完成计算 **add x1, x2, x3**
    - 分支指令使用ALU进行条件比较 **beq x1, x2, name**
  - ✓其后，各个指令的工作就不同了
    - 访存指令对存储器进行读写 **lw x1,100(x2)**
    - 算逻指令将ALU结果写回寄存器 **add x1, x2, x3**
    - 分支指令将基于比较结果修改下一条指令的地址 **beq x1, x2, name**

RISC-V指令编码格式总结

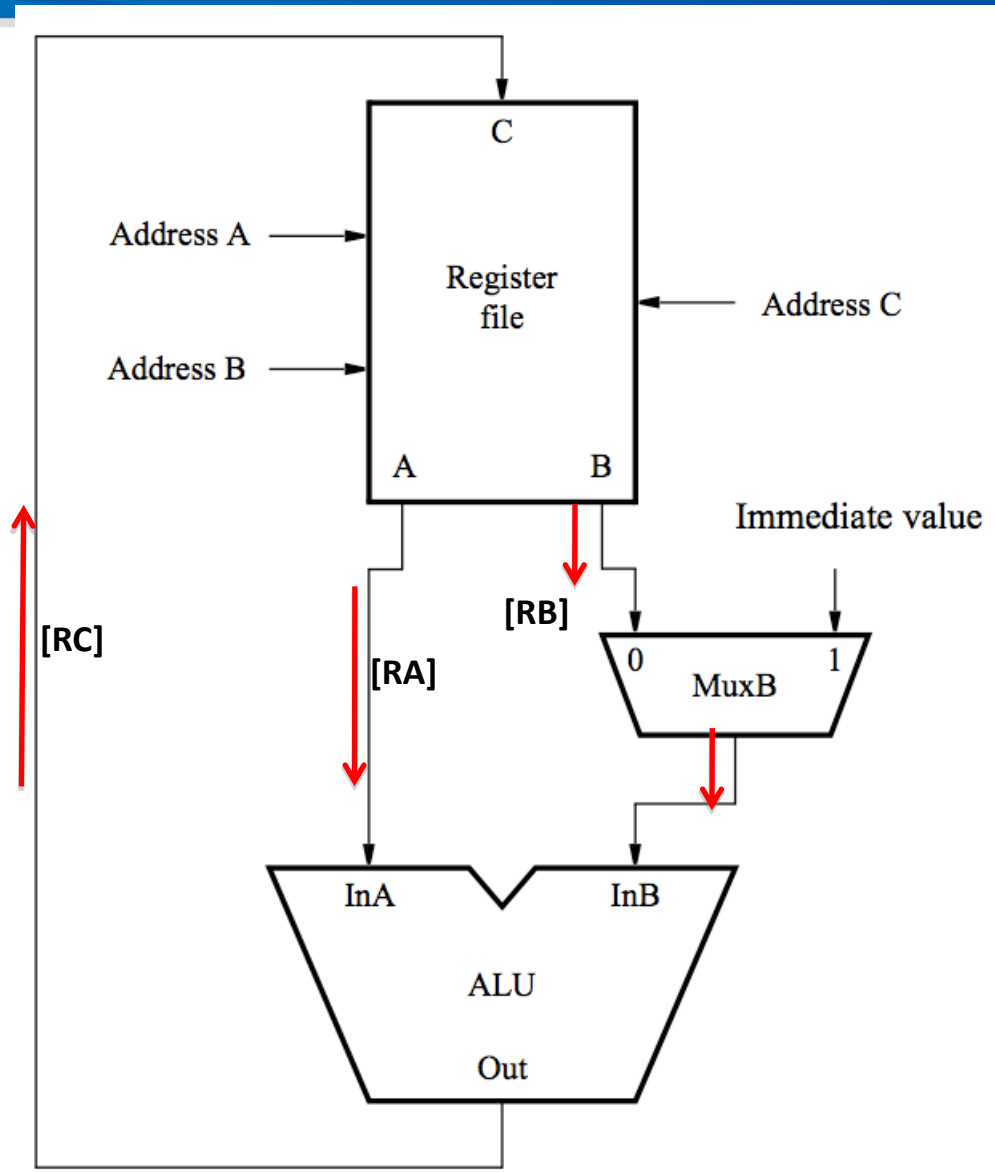
Name (Field Size)	Field 7 bits	5 bits	5 bits	3 bits	5 bits	7 bits	Comments
R-type	funct7	rs2	rs1	funct3	rd	opcode	Arithmetic instruction format
I-type	immediate[11:0]		rs1	funct3	rd	opcode	Loads & immediate arithmetic
S-type	immed[11:5]	rs2	rs1	funct3	immed[4:0]	opcode	Stores
SB-type	immed[12,10:5]	rs2	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

R-type Instructions	funct7	rs2	rs1	funct3	rd	opcode	Example
add (add)	0000000	00011	00010	000	00001	0110011	add x1, x2, x3
sub (sub)	0100000	00011	00010	000	00001	0110011	sub x1, x2, x3
I-type Instructions	**immediate**		**rs1**	**funct3**	**rd**	**opcode**	**Example**
addi (add immediate)	001111101000		00010	000	00001	0010011	addi x1, x2, 1000
ld (load doubleword)	001111101000		00010	011	00001	0000011	ld x1, 1000 (x2)
S-type Instructions	**immed -iate**	**rs2**	**rs1**	**funct3**	**immed -iate**	**opcode**	**Example**
sd (store doubleword)	0011111	00001	00010	011	01000	0100011	sd x1, 1000(x2)

寄存器计算指令的数据流

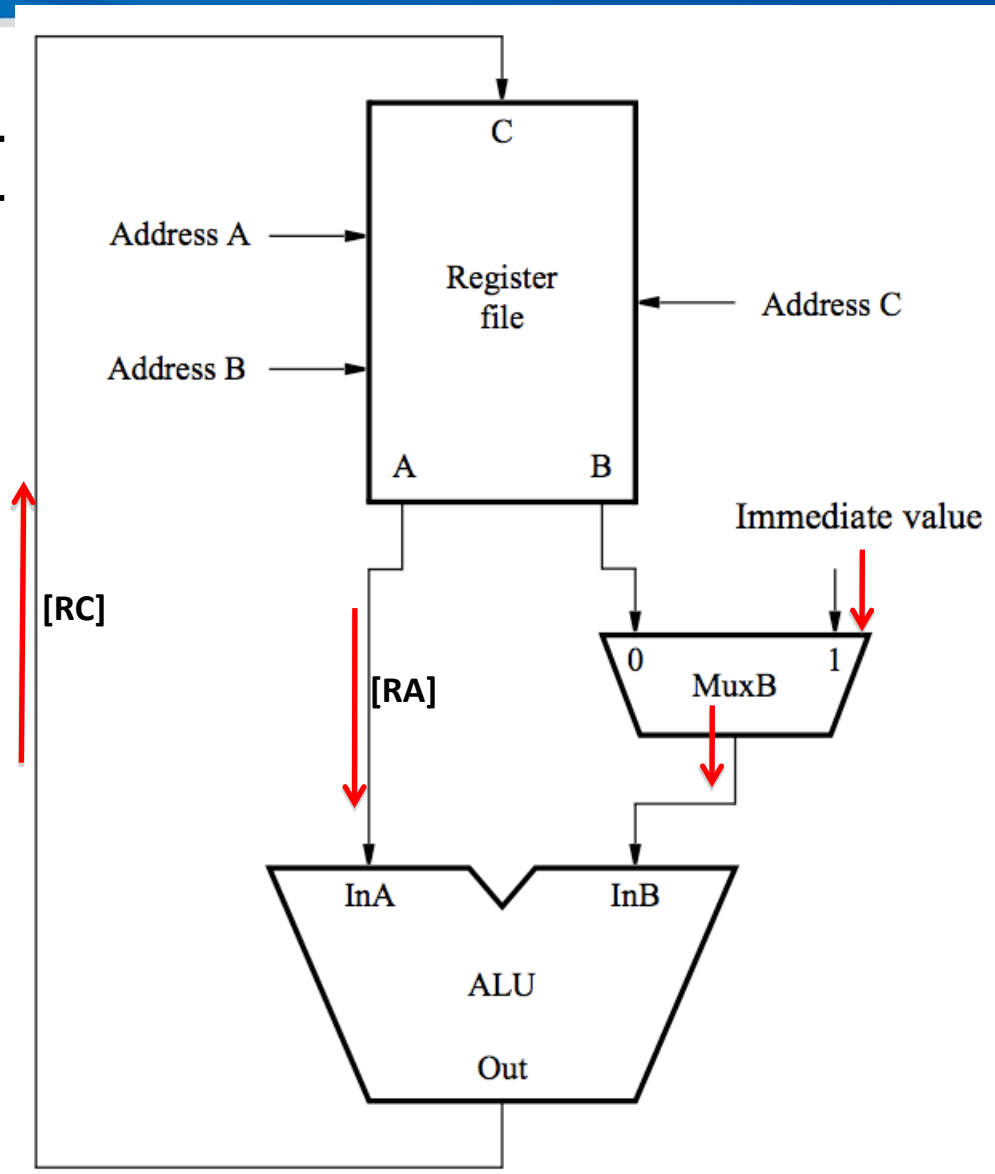
- 源操作数和目的操作数都在寄存器中

add x1, x2, x3; x2+x3->x1



立即数计算指令的数据流

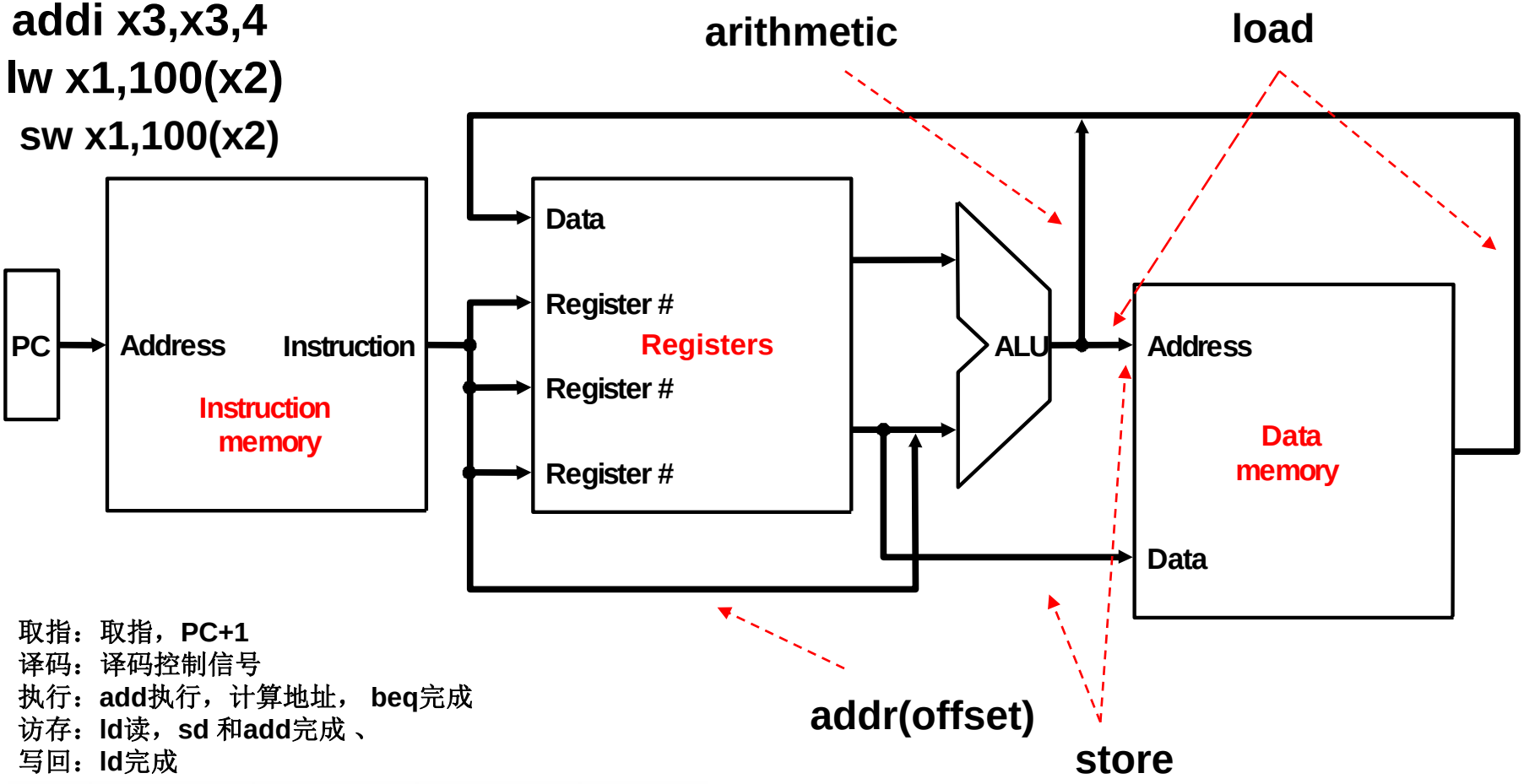
- 有一个源寄存器在立即数寄存器中



addi x3,x3,4；x3 = x3 + 4

CPU指令数据通路总图

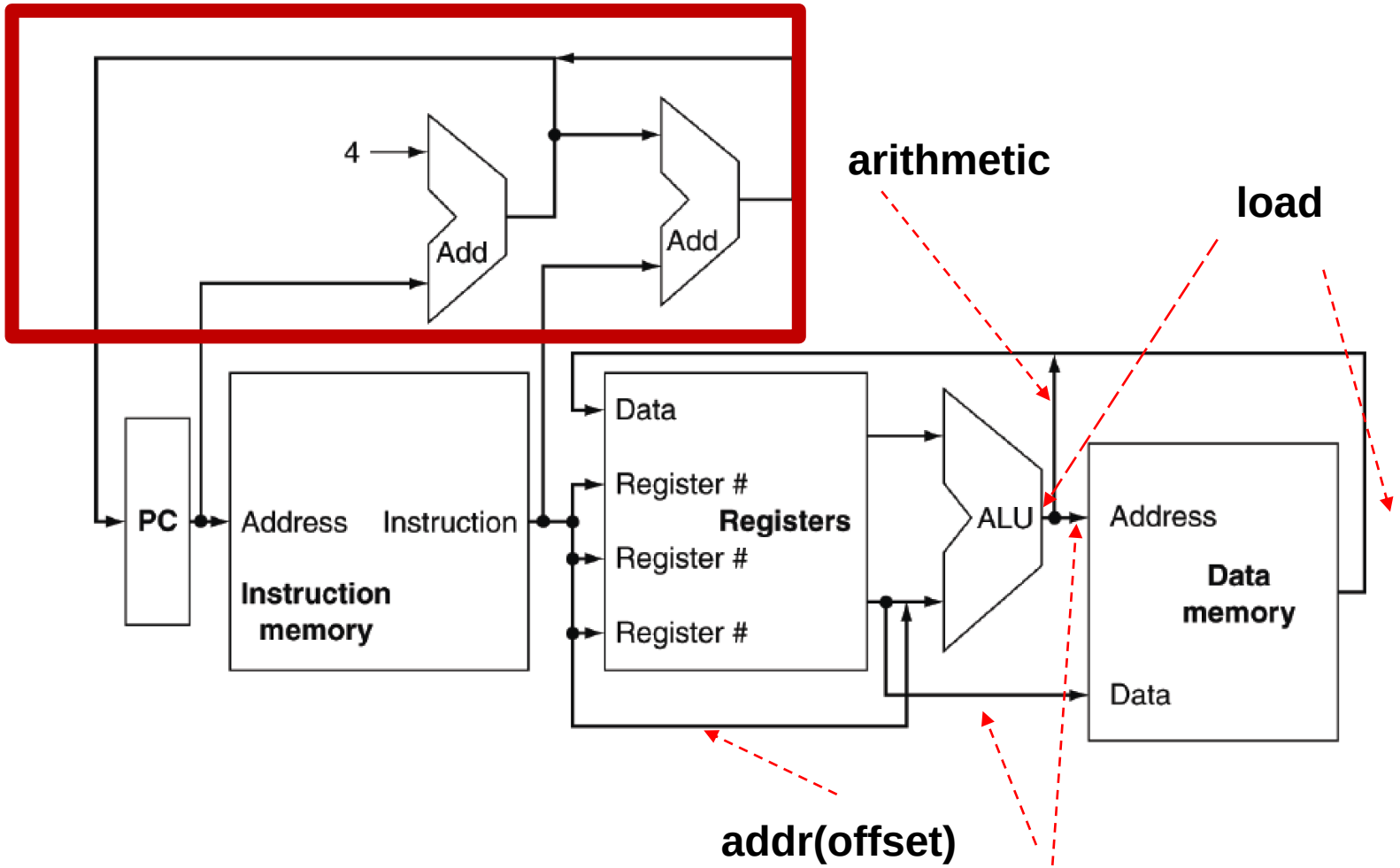
add x1, x2, x3
addi x3,x3,4
lw x1,100(x2)
sw x1,100(x2)



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：ld读，sd 和add完成 、
写回：ld完成

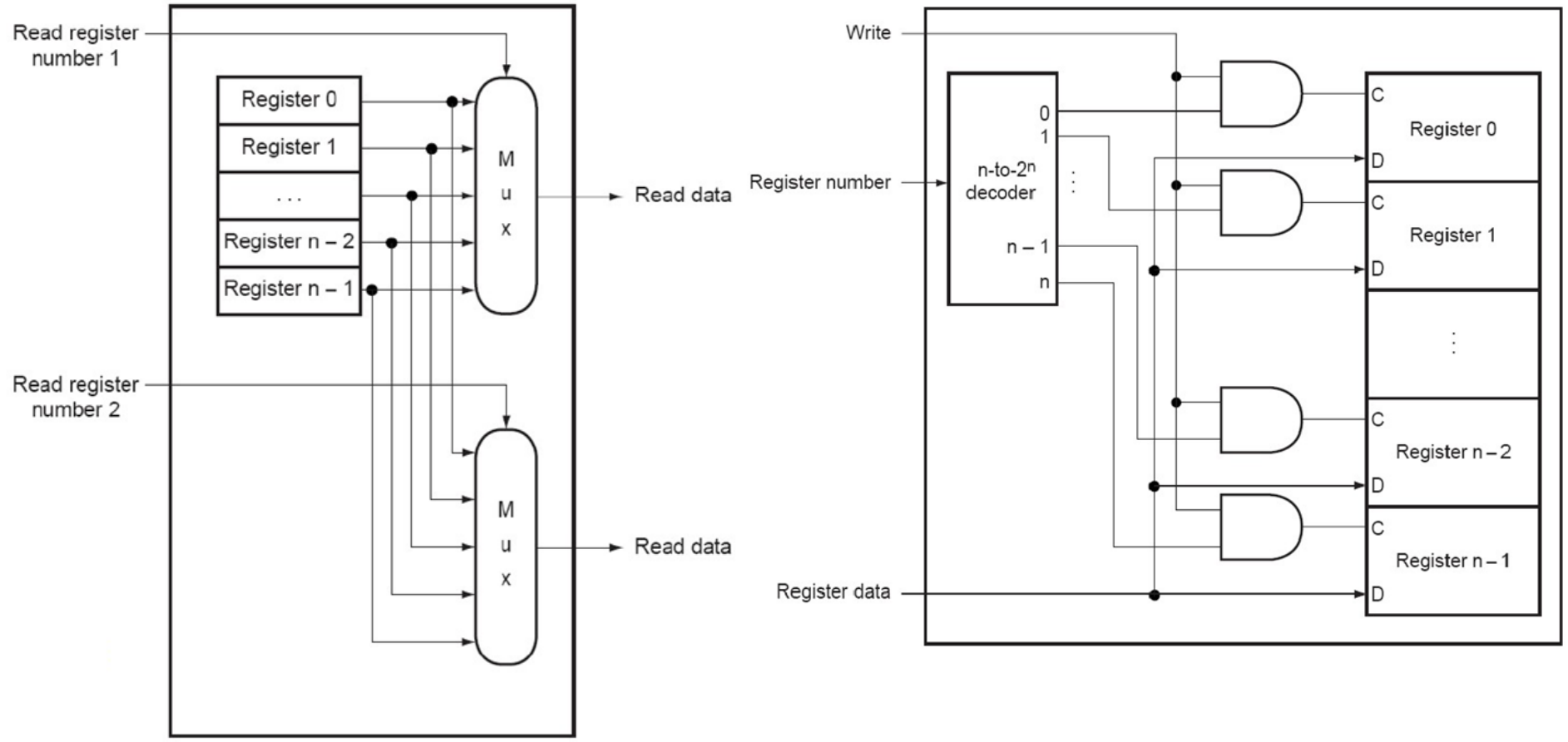
R-type	funct7	rs2	rs1	funct3	rd	opcode
I-type	immediate[11:0]		rs1	funct3	rd	opcode
S-type	immed[11:5]	rs2	rs1	funct3	immed[4:0]	opcode
B-type	immed[12,10:5]	rs2	rs1	funct3	immed[4:1,11]	opcode

CPU指令数据通路总图



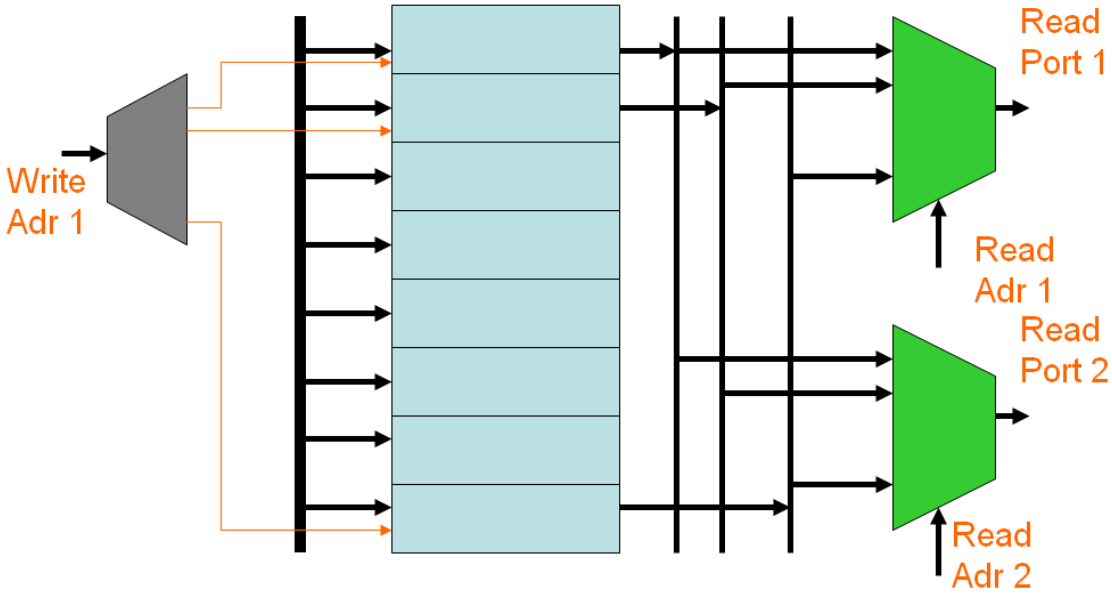
R-type	funct7	rs2	rs1	funct3	rd	opcode
I-type	immediate[11:0]		rs1	funct3	rd	opcode
S-type	immed[11:5]	rs2	rs1	funct3	immed[4:0]	opcode
B-type	immed[12,10:5]	rs2	rs1	funct3	immed[4:1,11]	opcode

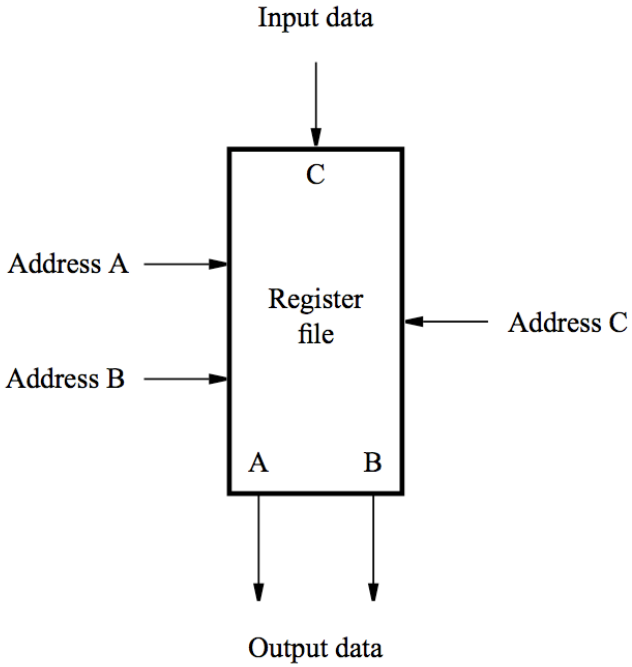
通路中的存储-RegisterFile



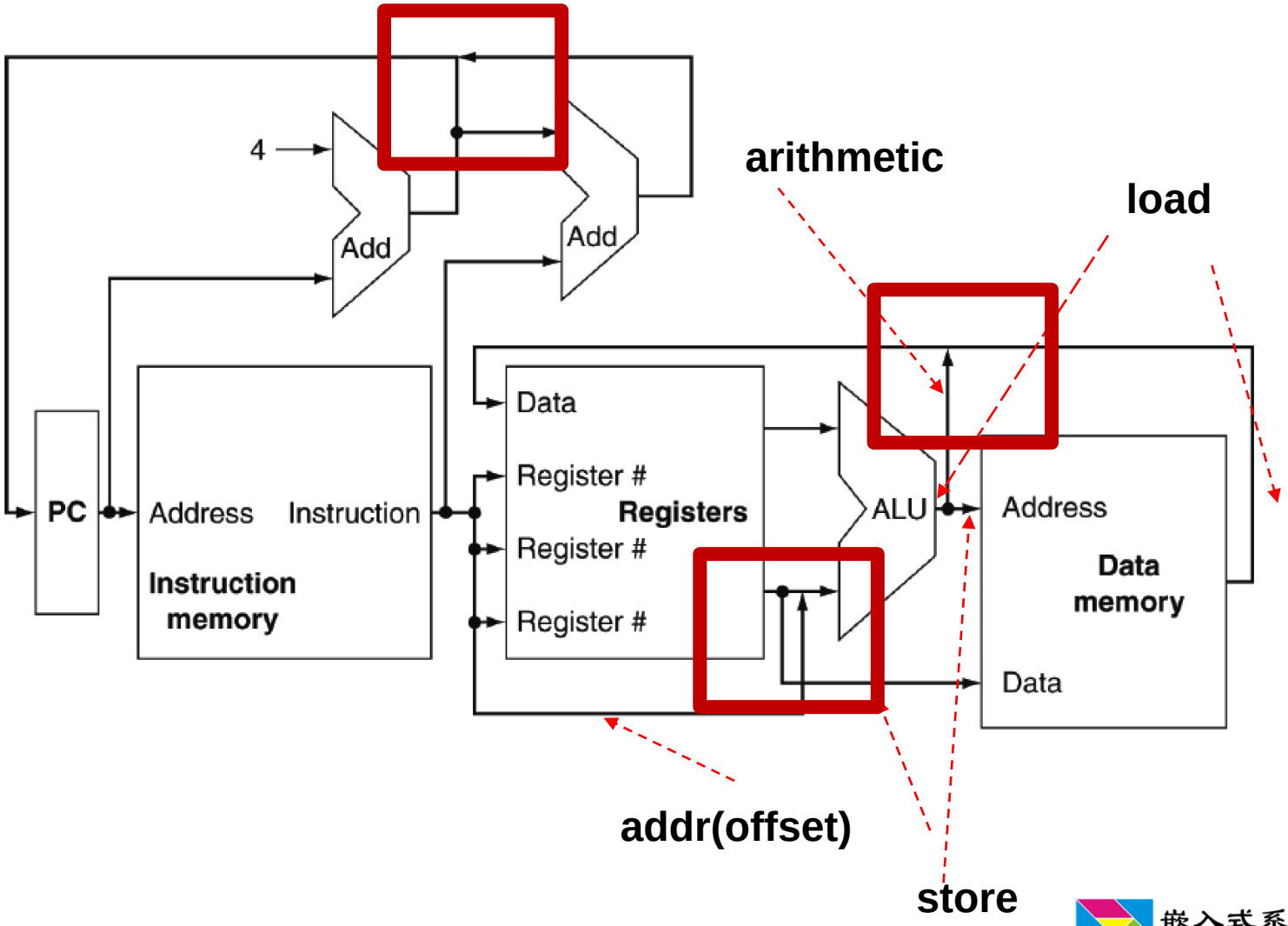
RegFile读写操作

- 寄存器文件在一个周期中有两个读和一个写操作。
  - ✓ 在一个周期内，某个REG可以同时完成读写操作，但读出的是上一个周期写入的值
- 寄存器文件不能同时进行读和写操作。两种设计方法：
  - ✓ 后写（late write）：在前半周期读数据，在后半周期写数据。
  - ✓ 先写（early write）：与后写相反。
  - ✓ 两种方式各有什么优缺点？（思考）

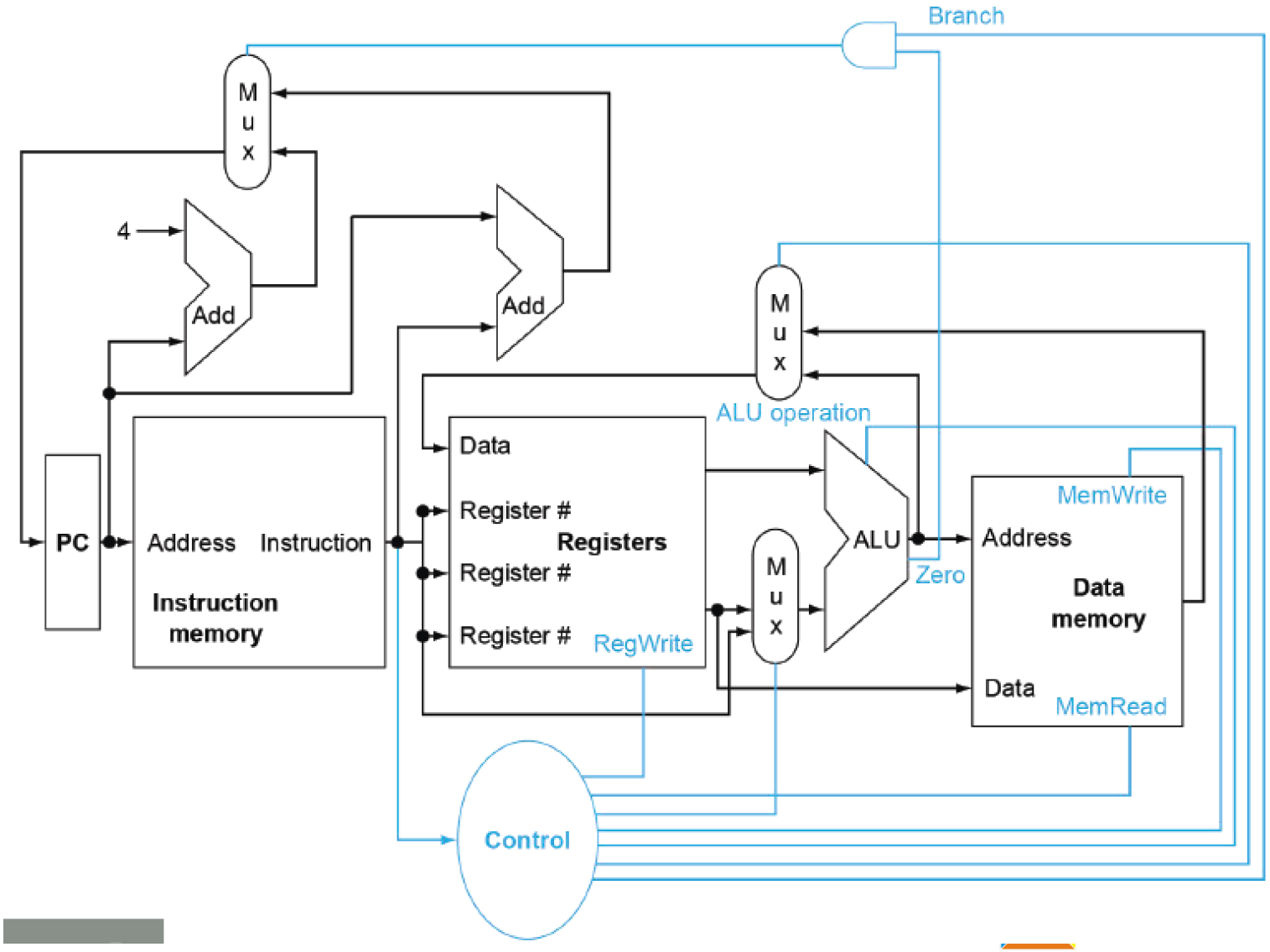




通道中的多路选择器


4
Add
Add
arithmetic
load
PC
Address
Instruction
Instruction memory
Data
Register #
Registers
Register #
Register #
ALU
Address
Data memory
Data
addr(offset)
store

通道中的多路选择器


Branch
M
u
x
4
Add
Add
M
u
x
ALU operation
PC
Address
Instruction
Instruction
memory
Data
Register #
Registers
Register #
Register #
RegWrite
M
u
x
ALU
Zero
MemWrite
Address
Data
memory
Data
MemRead
Control

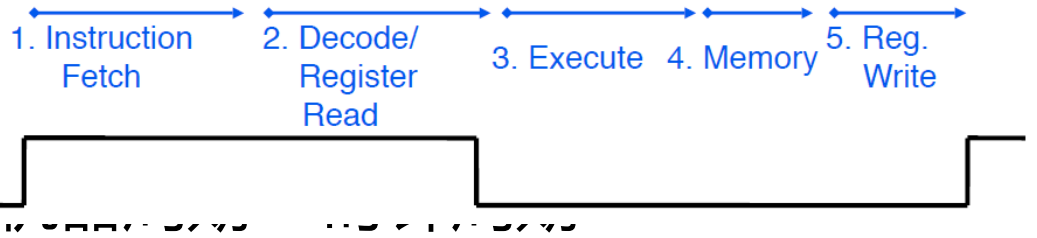
数据通路设计➔控制部件设计

控制信号的作用?

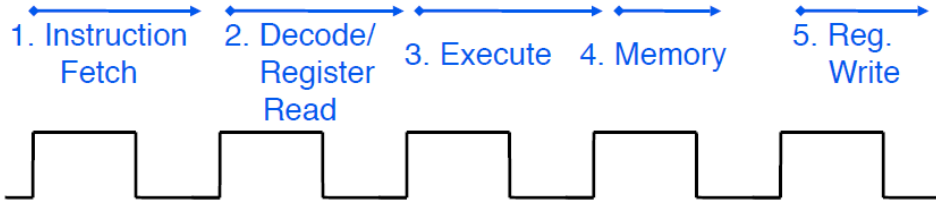
控制信号的来源?

实现 单周期->多周期

- 指令周期 = m 机器周期。机器周期 = 时钟周期
- 单周期实现：指令周期 = 1 机器周期 = 1 时钟周期
  - ✓ All stages of ainstructioare completed withione ***long*** clock cycle.
  - ✓ 所需控制信号同时生成

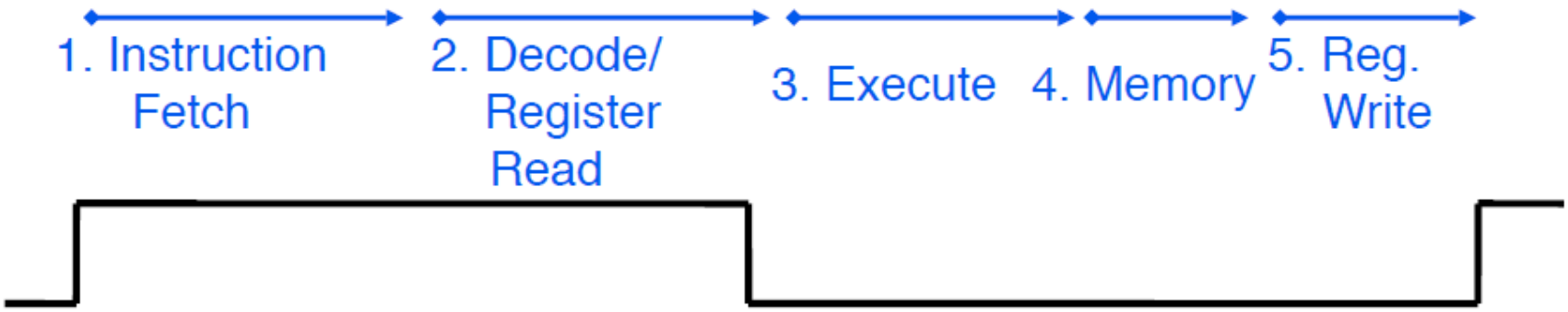


- 多周期实现：指令周期 =
  - ✓ Only one stage of instructioper clock cycle
  - ✓ 按时钟周期（= 机器周期）生成当前周期所需控制信号

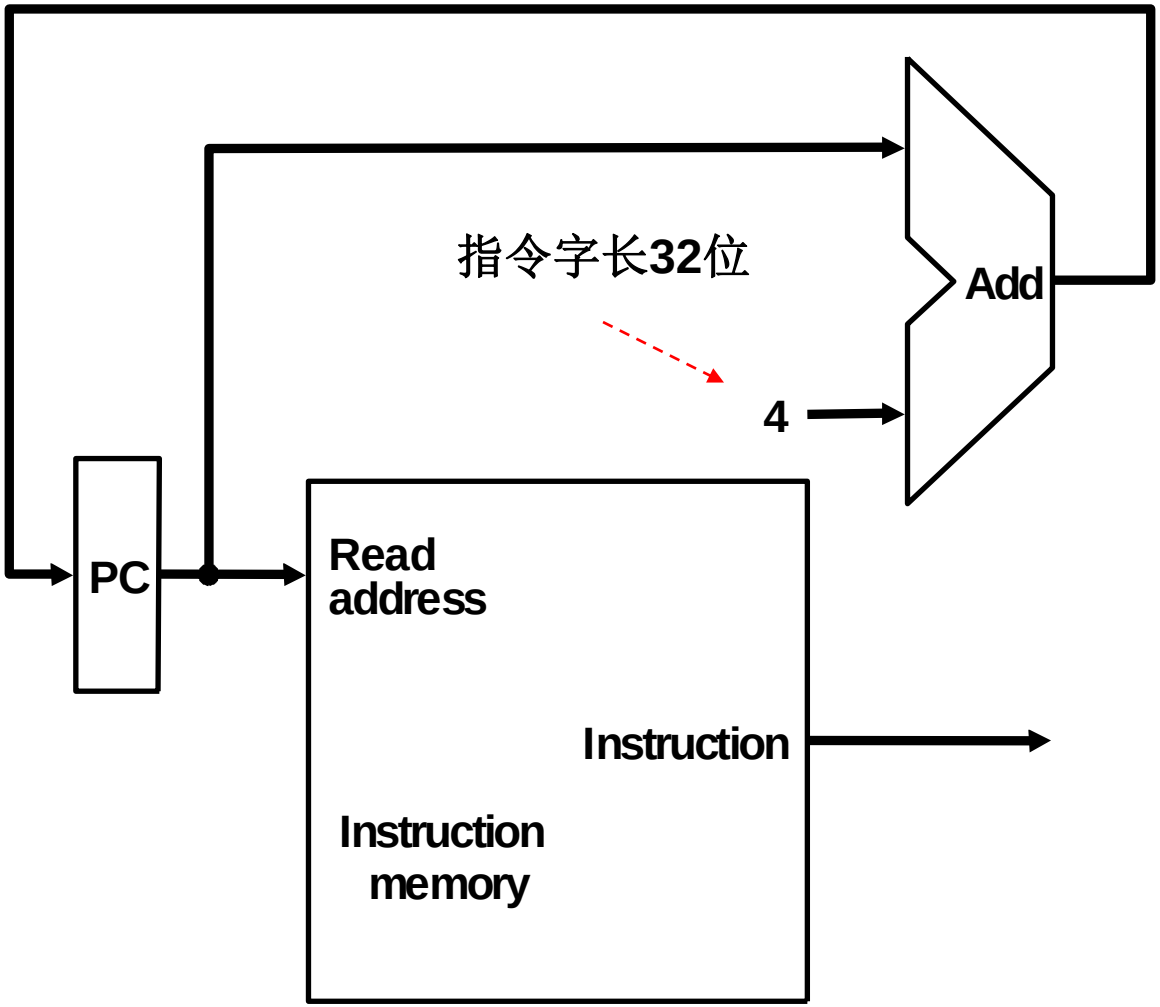


单周期

- 定长指令周期
  - ✓All stages of ainstructioare 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

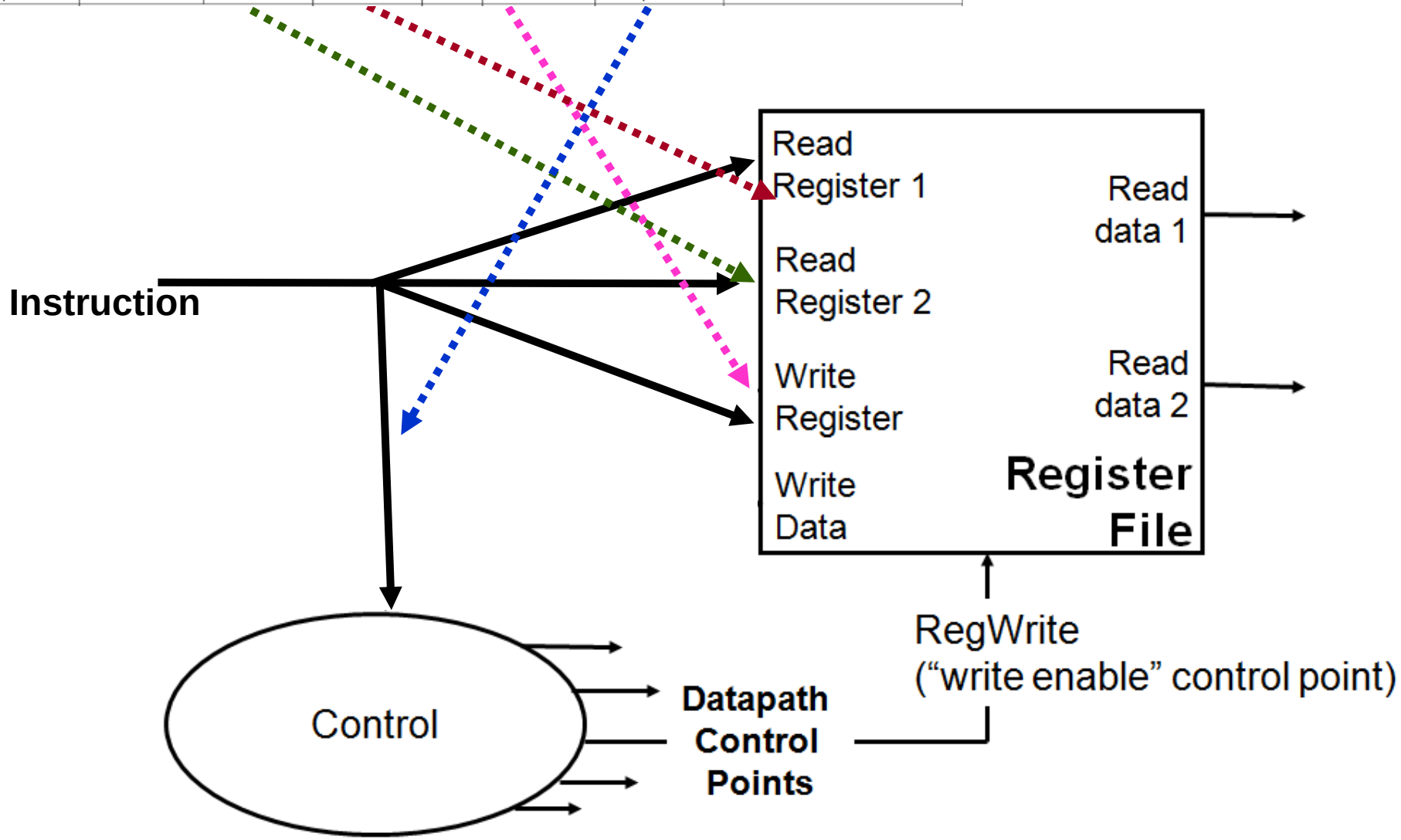


取指 Instruction Fetch

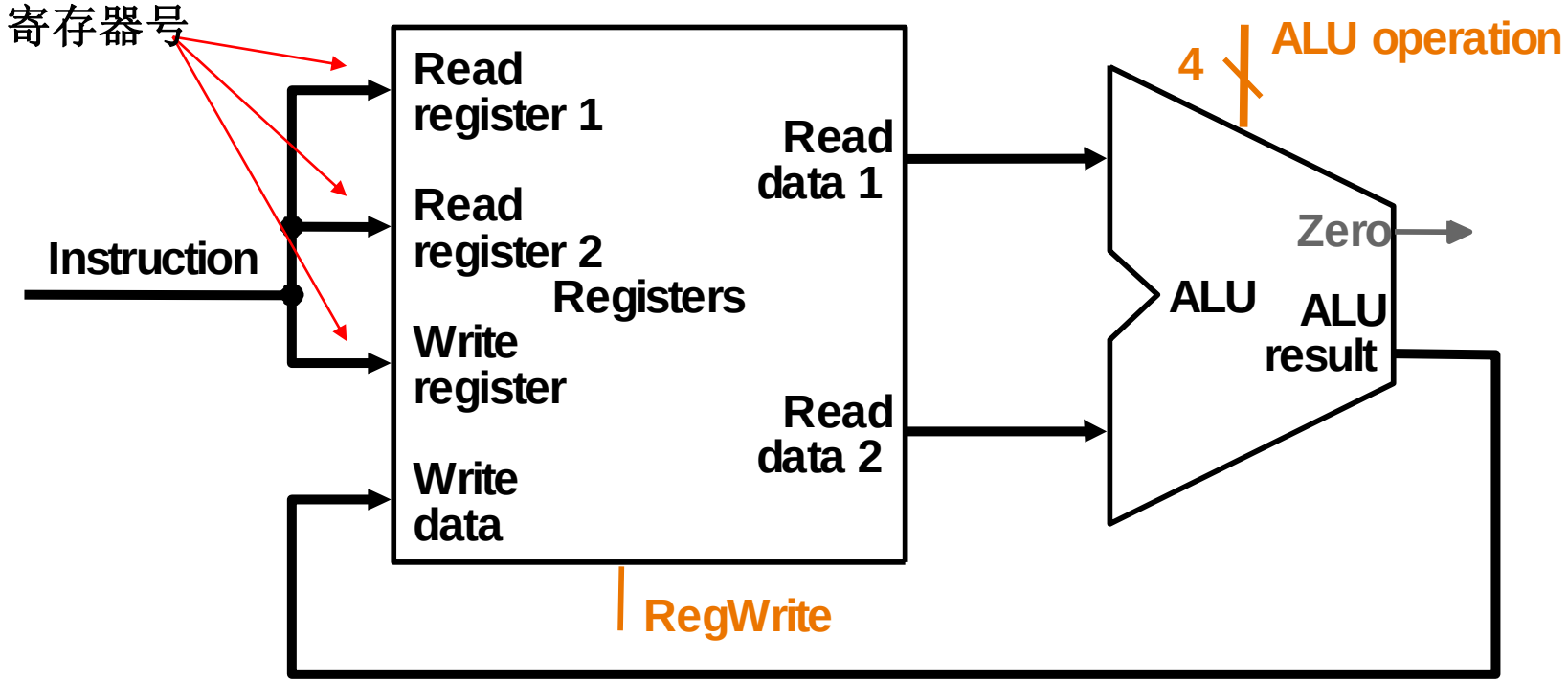


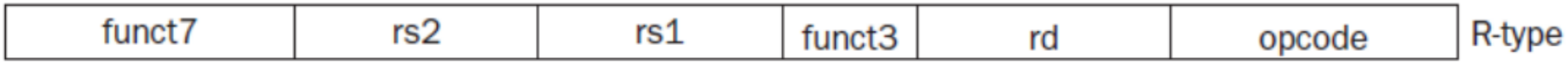
译码 Decode-RISC-V

Name (Field Size)	Field 7 bits	5 bits	5 bits	3 bits	5 bits	7 bits	Comments
R-type	funct7	rs2	rs1	funct3	rd	opcode	Arithmetic instruction format



R-type指令的执行



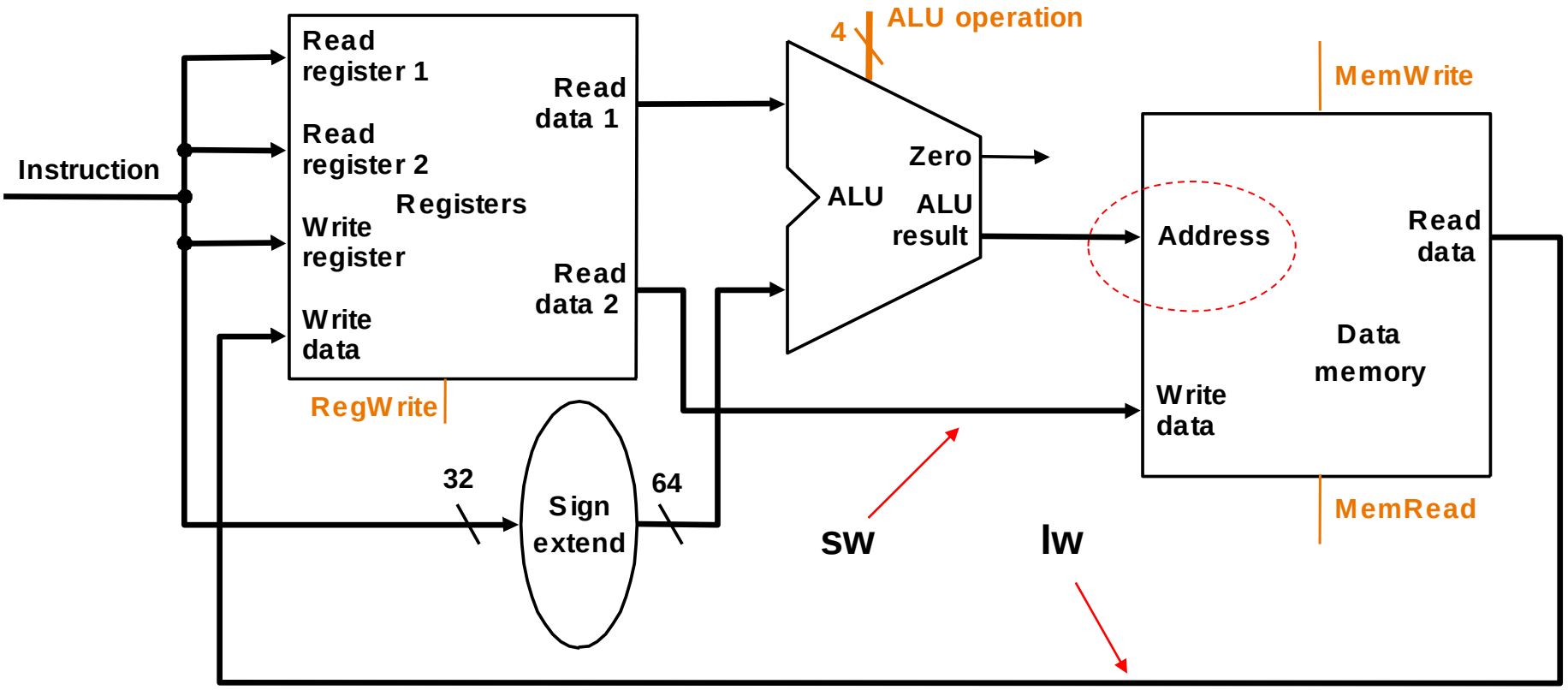


- 寄存器堆操作
  - 读：给出寄存器编号，则寄存器的值自动送到输出端口
  - 写：需要寄存器编号和控制信号**RegWrite**，时钟边沿触发
  - 在一个周期内，**REG**可以同时完成读写操作，但读出的是上一个周期写入的值(读后写)

访存指令的执行-ld

rs：基址
rt：ld目的寄存器

immediate[11:0]	rs1	funct3	rd	opcode	I-type

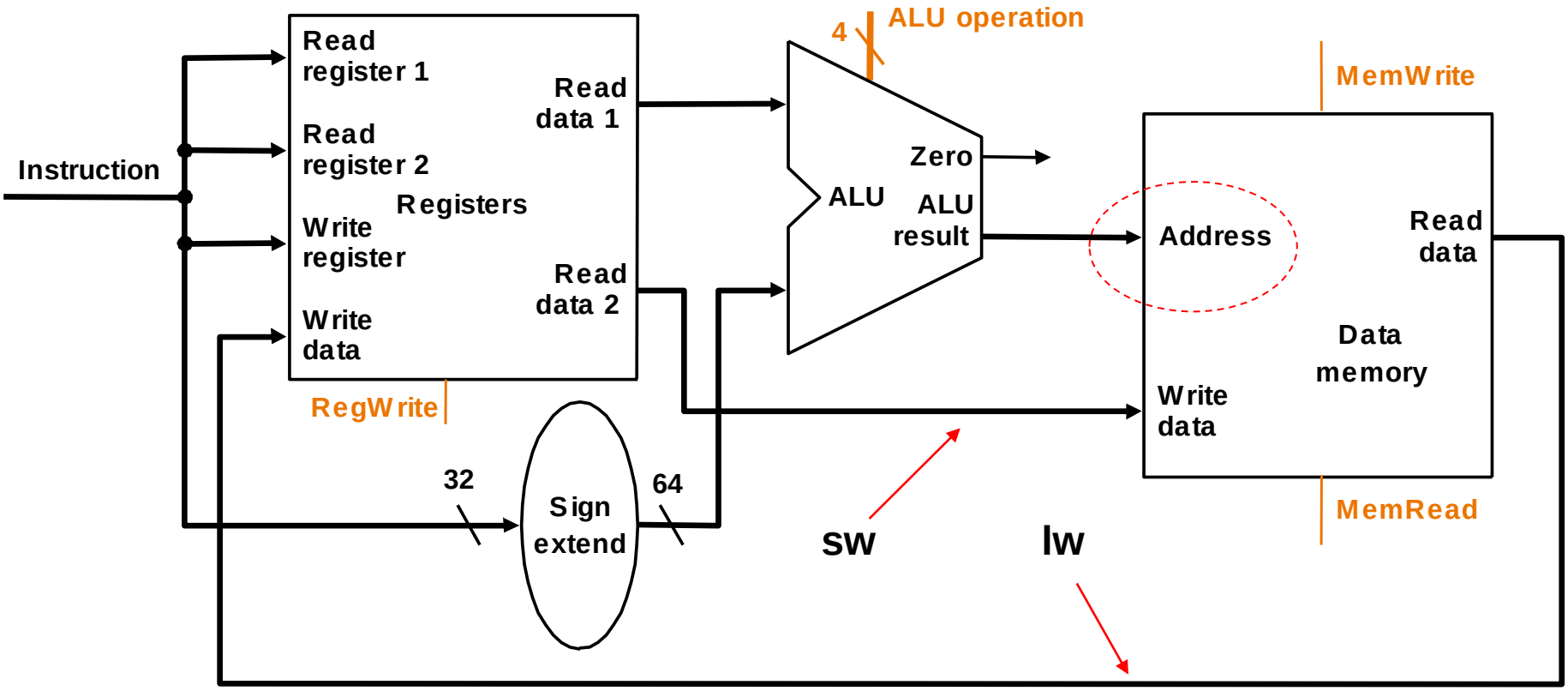


- lw x1, offset(x2); M(x2+offset) -> x1
- 需要对指令字中的32位偏移进行64位带符号扩展（或者16位偏移进行32位扩展）

访存指令的执行-sw

rs：基址
rt：sw源寄存器

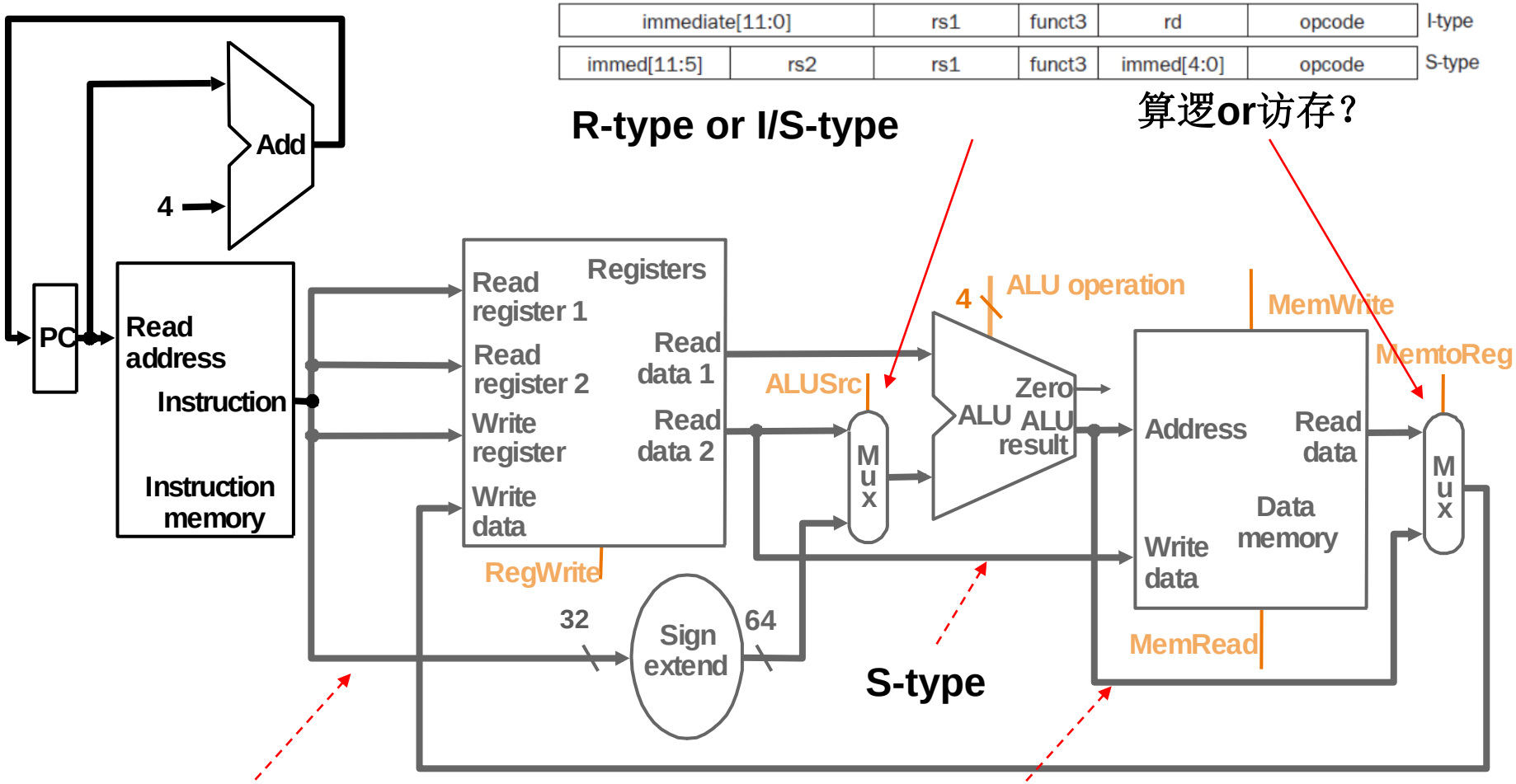
immed[11:5]	rs2	rs1	funct3	immed[4:0]	opcode	S-type



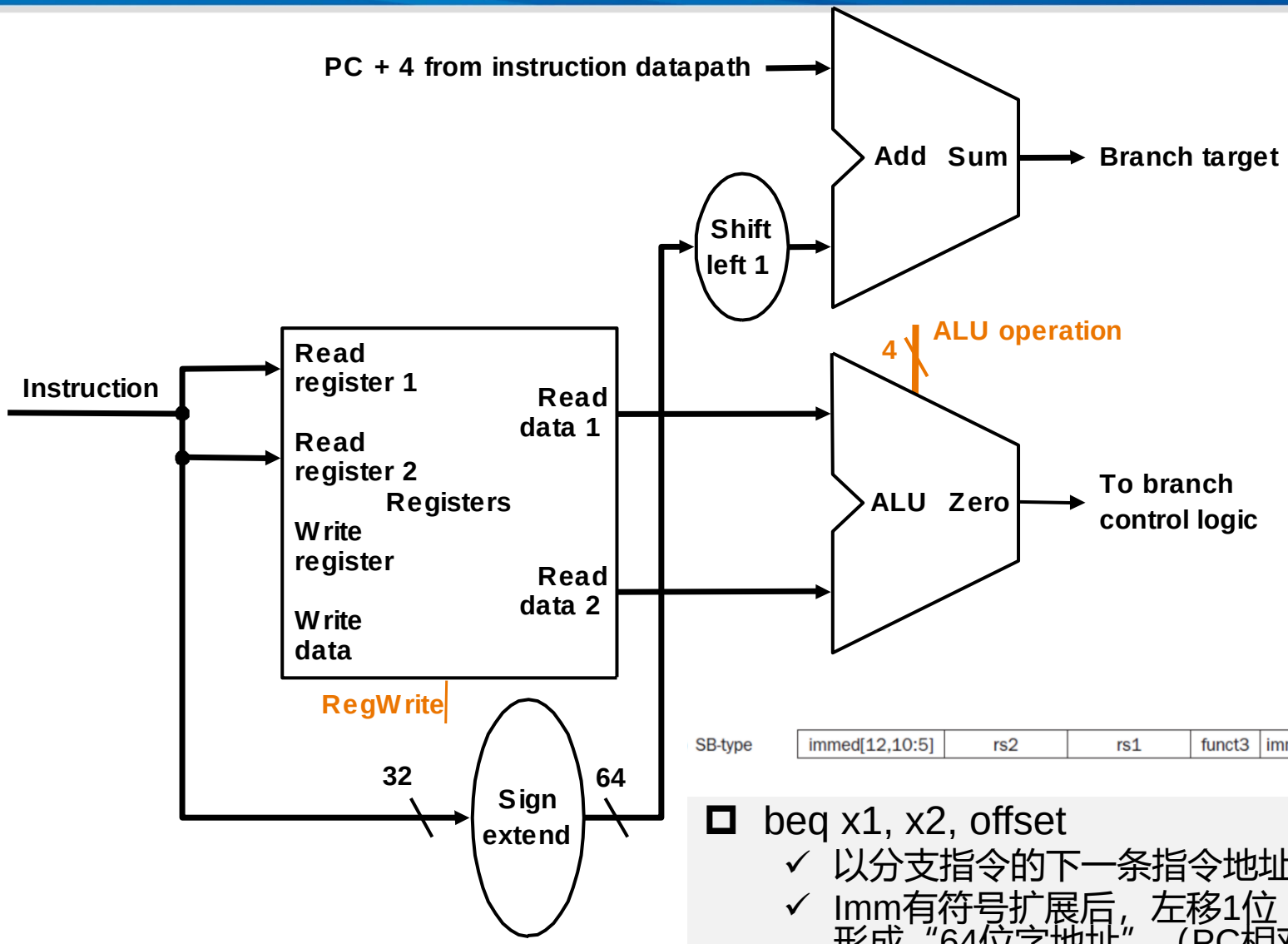
- sw x1, offset(x2)； x1 -> M(x2+offset)
- 需要对指令字中的32位偏移进行64位带符号扩展（或者16位偏移进行32位扩展）

访存指令和算逻指令的数据通路综合

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



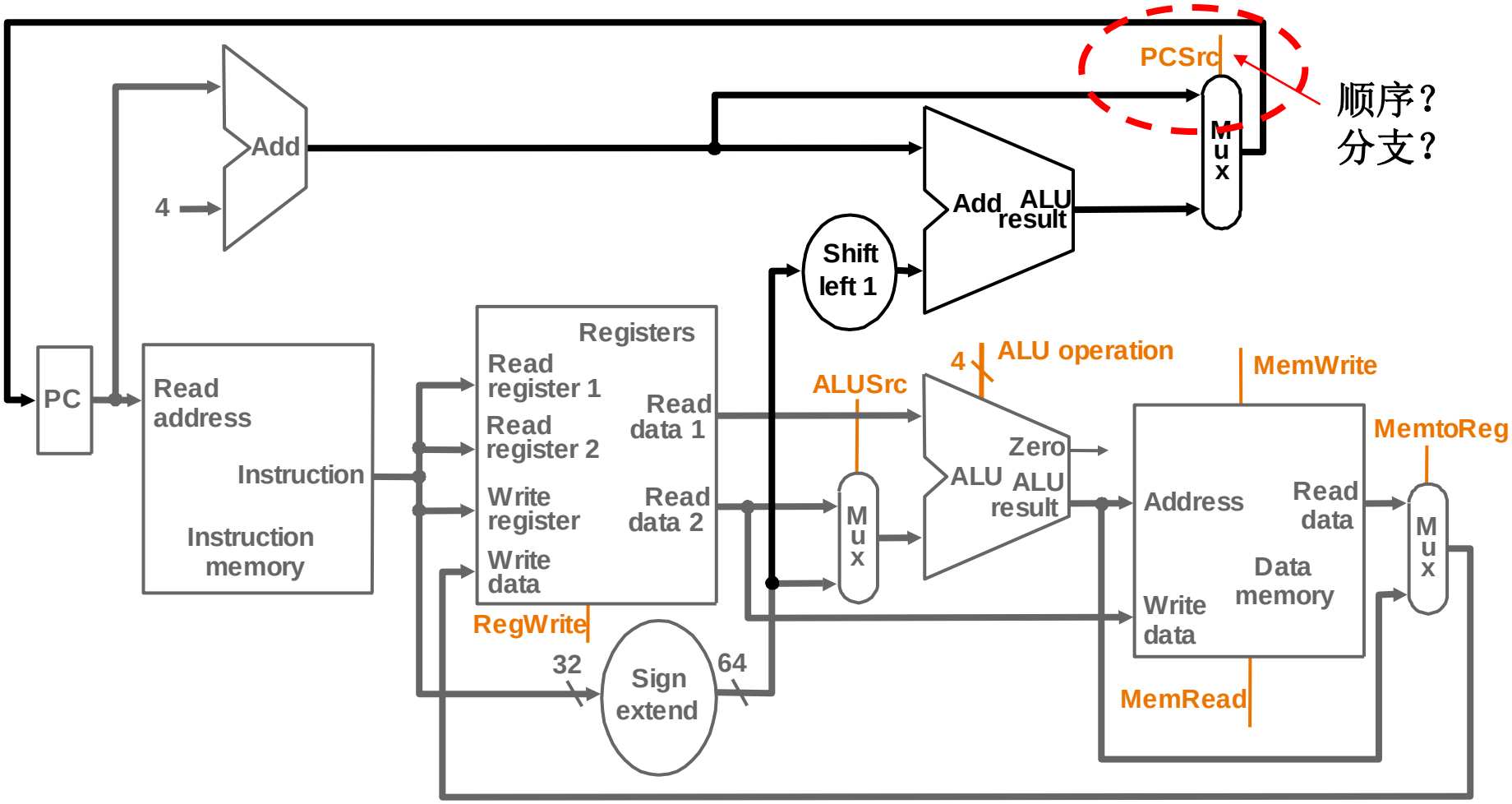
条件转移beq



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

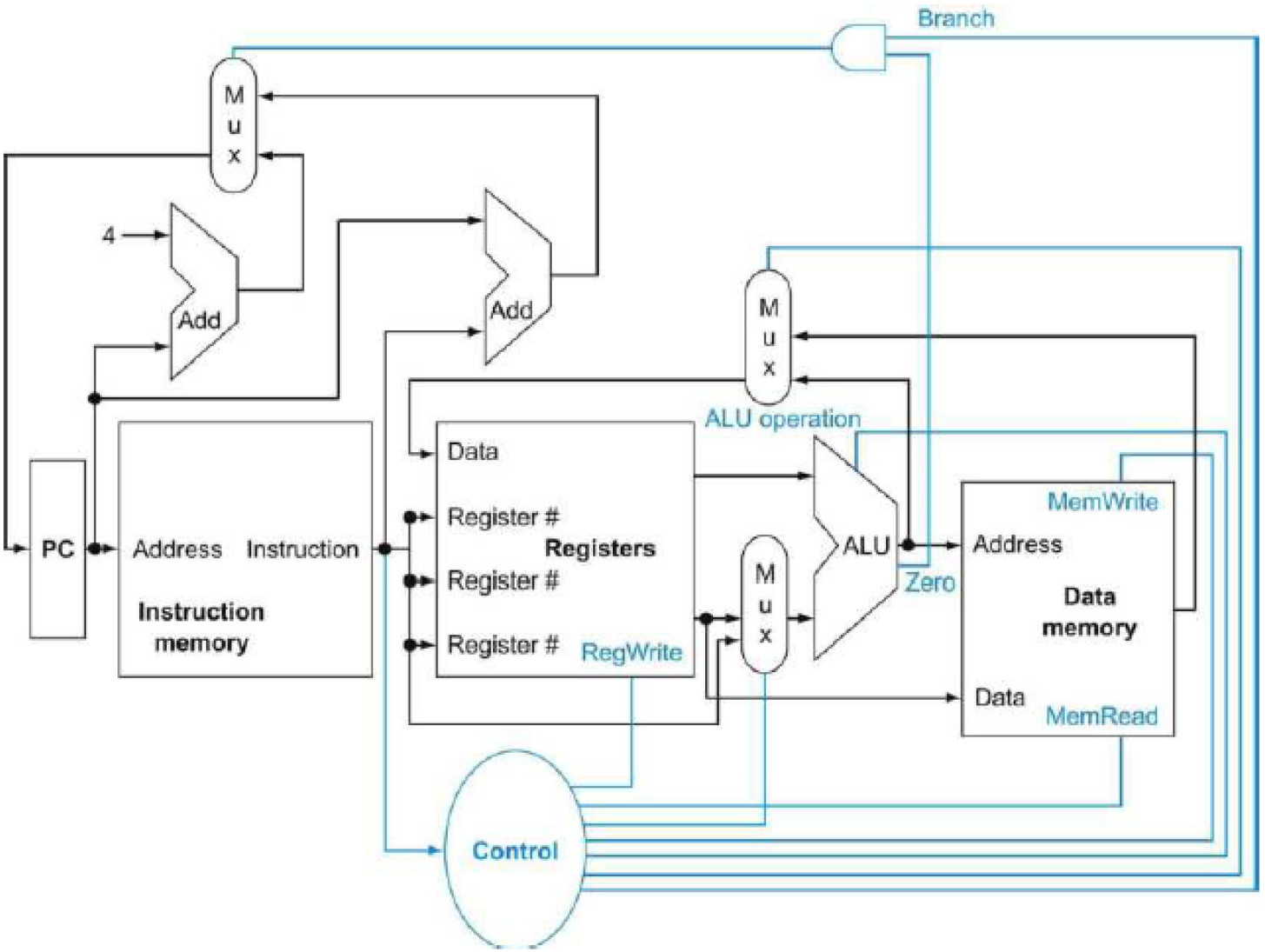
- beq x1, x2, offset
  - ✓ 以分支指令的下一条指令地址为基址
  - ✓ Imm有符号扩展后，左移1位（半字对齐），形成“64位字地址”（PC相对寻址）

R-/I-/S/SB-type数据通路总图



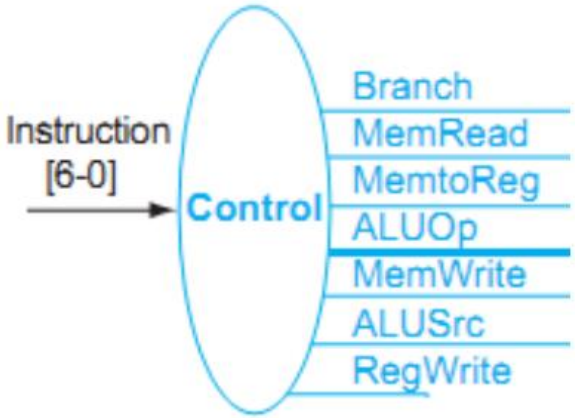
控制信号如何处理？

复习控制器-示意图


Branch
M
u
x
4
Add
Add
M
u
x
ALU operation
PC
Address
Instruction
Instruction
memory
Data
Register #
Registers
Register #
Register #
RegWrite
M
u
x
ALU
Zero
MemWrite
Address
Data
memory
Data
MemRead
Control

控制器：控制信号生成

- op域（7位）译码产生的控制信号：7个
  - ✓ RegWrite：寄存器写操作控制
  - ✓ ALUSrc：ALU的第二个操作数来源
    - R-type指令（0）与 I/S/SB-type指令（1）
  - ✓ ALUOp：R-type指令（2位）
  - ✓ MemRead：存储器读控制，load指令
  - ✓ MemWrite：存储器写控制，store指令
  - ✓ MemtoReg：目的寄存器数据来源
    - R-type（I类ALU）指令与load指令二选一
  - ✓ Branch：是否分支指令（产生PCSrc）
    - PCSrc：nPC来源控制，顺序与分支成功二选一
    - “是否beq指令（Branch）” & “ALU的Zero状态有效”



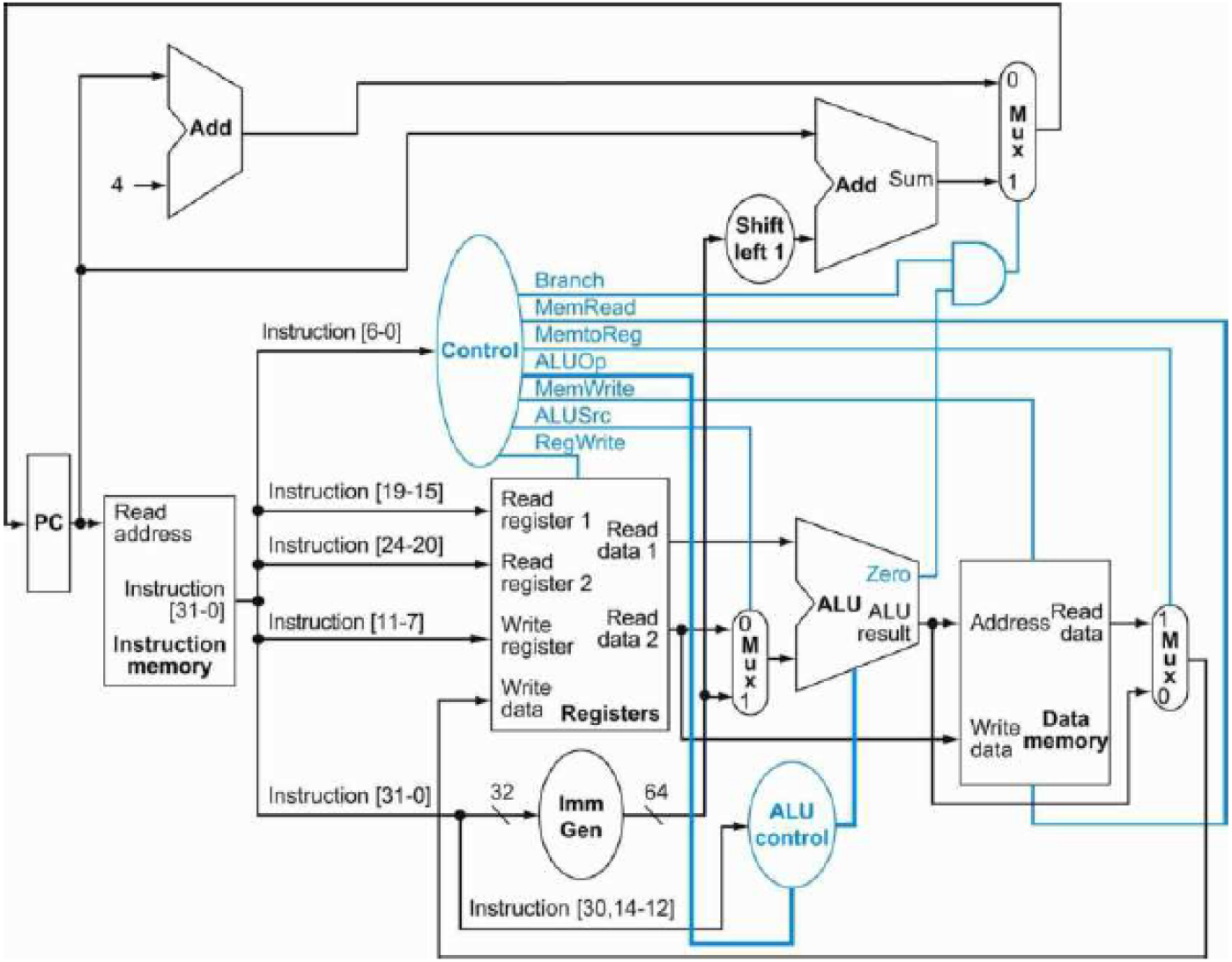
ALU控制信号

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

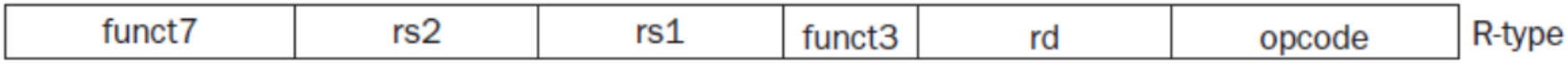
2位ALUop与Funct7、Funct3组合，产生ALU_ctrl_input（即4位ALU operation）

ALU控制选择

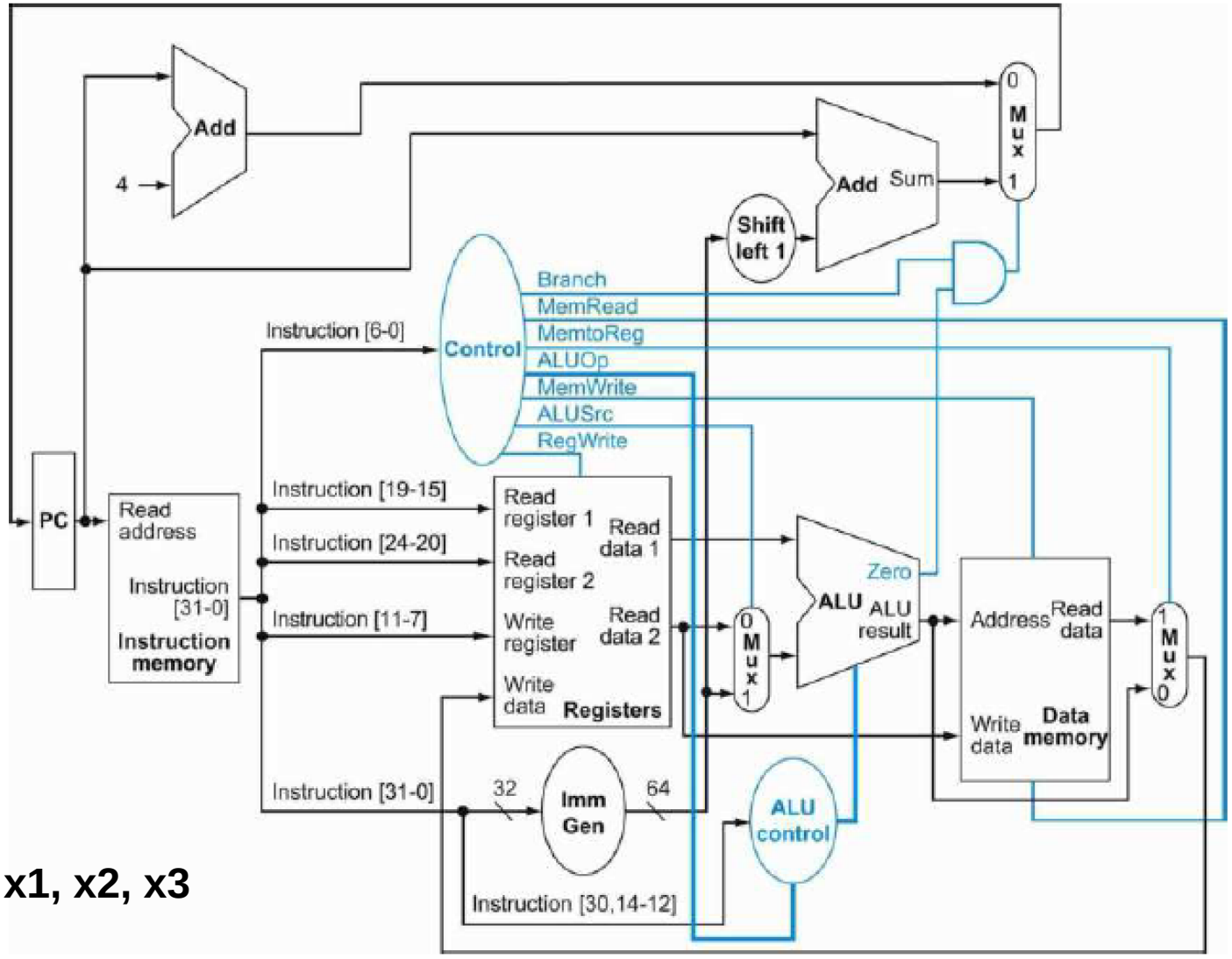

Add
4
Add
Sum
0
M
u
x
1
Shift
left 1
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Control
Instruction [6-0]
PC
Read
address
Instruction
[31-0]
Instruction
memory
Instruction [19-15]
Instruction [24-20]
Instruction [11-7]
Read
register 1
Read
register 2
Write
register
Write
data
Read
data 1
Read
data 2
Registers
Zero
ALU
ALU
result
Address
Read
data
Write
data
Data
memory
1
M
u
x
0
Instruction [31-0]
32
Imm
Gen
64
ALU
control
Instruction [30,14-12]

R-type指令的执行过程

- add x1, x2, x3;　　x2+x3->x1
- 在一个周期内完成如下动作
  - ✓第一步：取指和PC + 4
  - ✓第二步：读两个源操作数寄存器x2和x3
  - ✓第三步：ALU操作
  - ✓第四步：结果写回目的寄存器x1



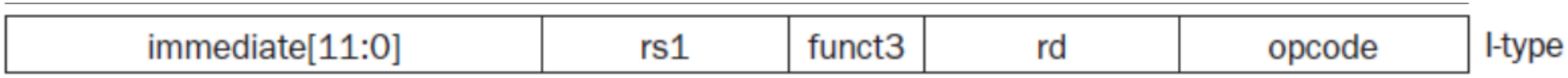
R-type指令的执行路径



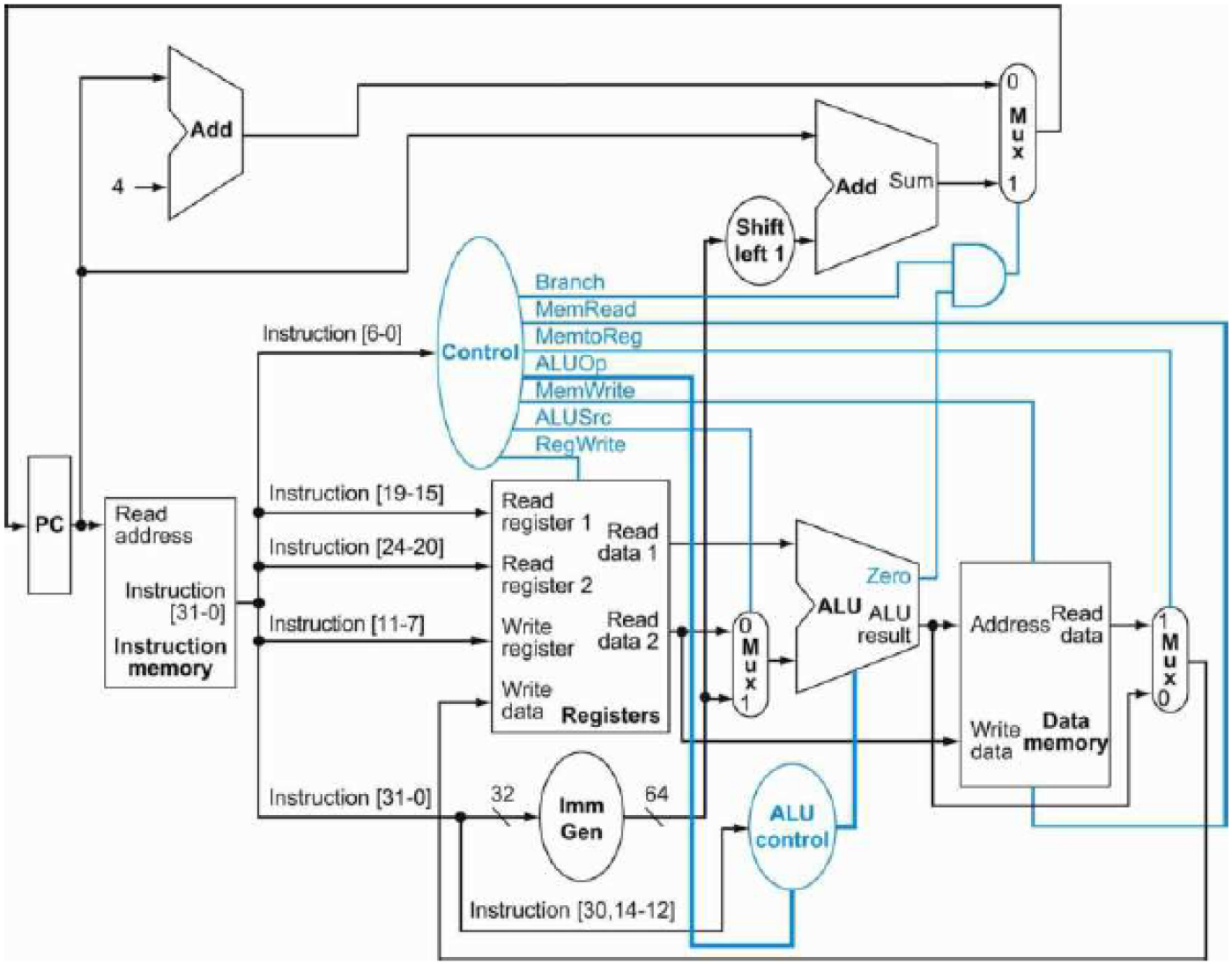
add x1, x2, x3

lw指令的执行过程

- lw x1, offset(x2);　　M(x2+offset) -> x1
- 第一步：取指和PC + 4
- 第二步：读寄存器x2
- 第三步：ALU操作完成x2与符号扩展后的offset加
- 第四步：ALU的结果作为访存地址，送往数据MEM
- 第五步：内存中的数据送往x1



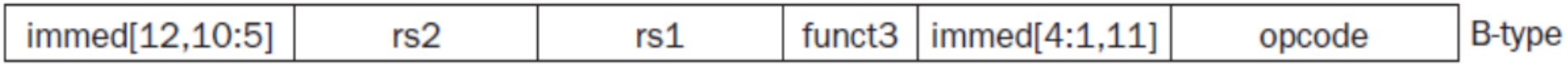
lw指令的执行路径



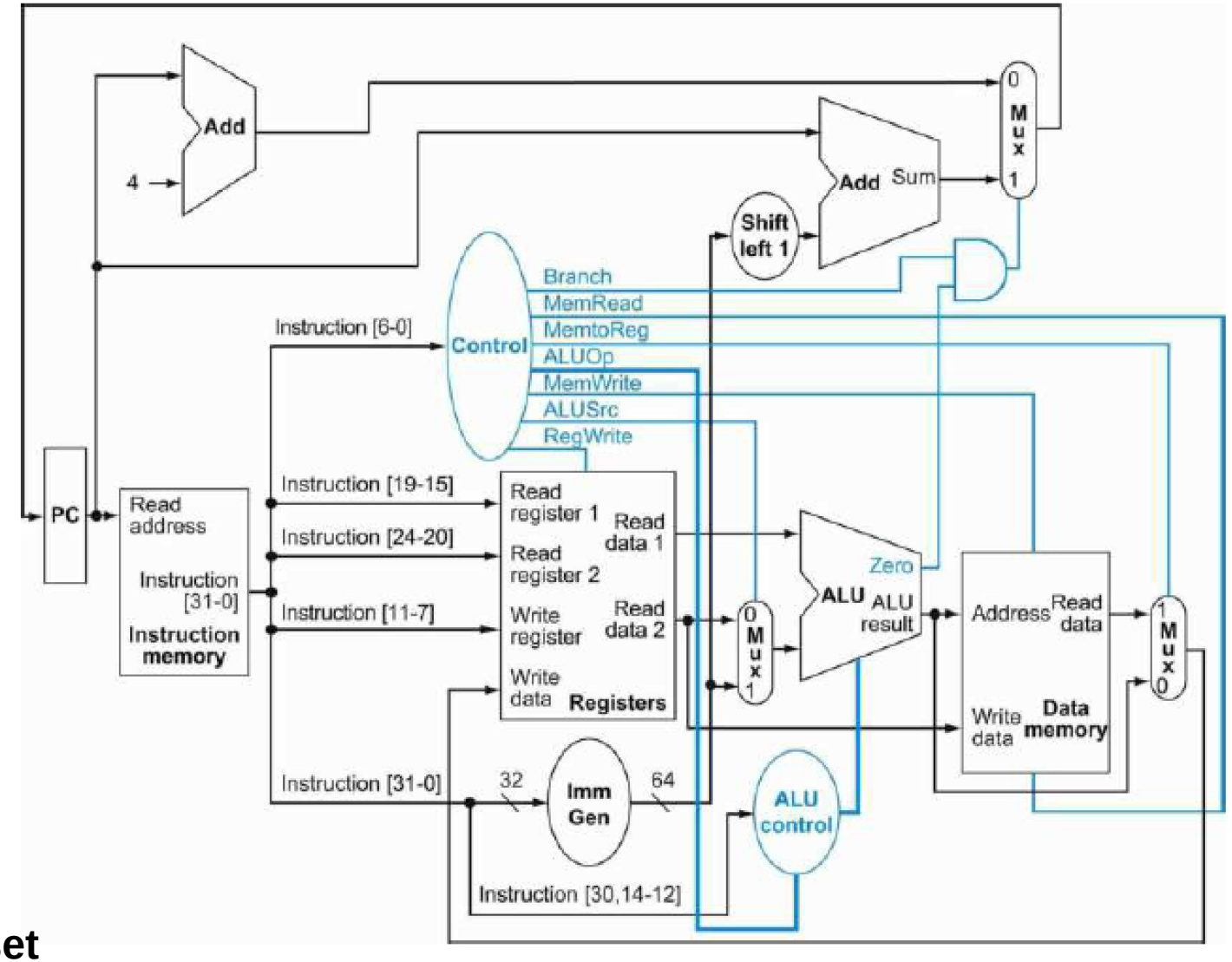
lw x1, offset(x2);

beq指令的执行过程

- beq x1, x2, offset
- 第一步：取指和PC + 4
- 第二步：读寄存器x1，x2
- 第三步：ALU将x1和x2相减；PC+4与被左移1位并进行符号扩展后的offset相加，作为分支目标地址
- 第四步：ALU的Zero确定应送往PC的值



beq的执行路径

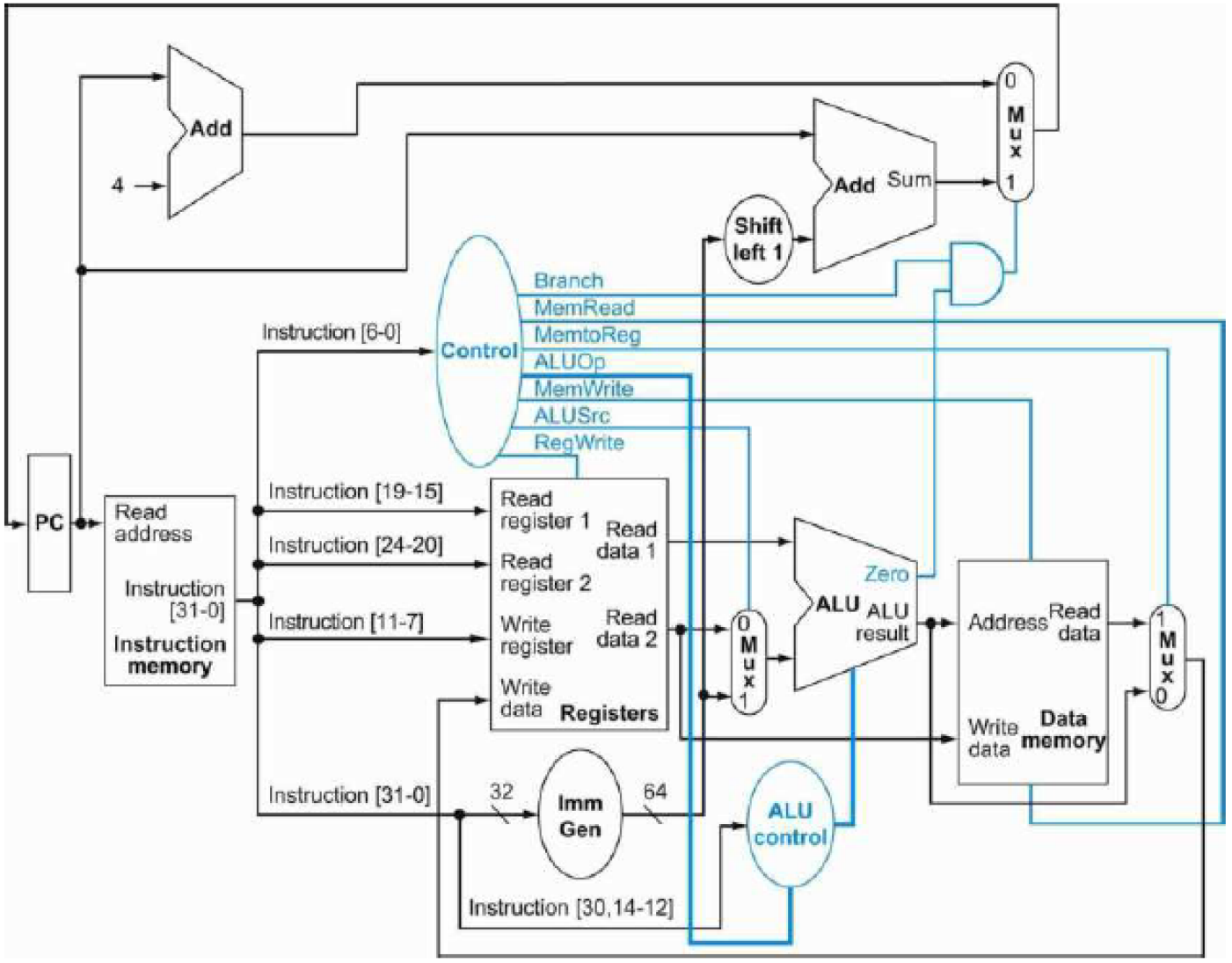


beq x1, x2, offset

JAL指令的实现

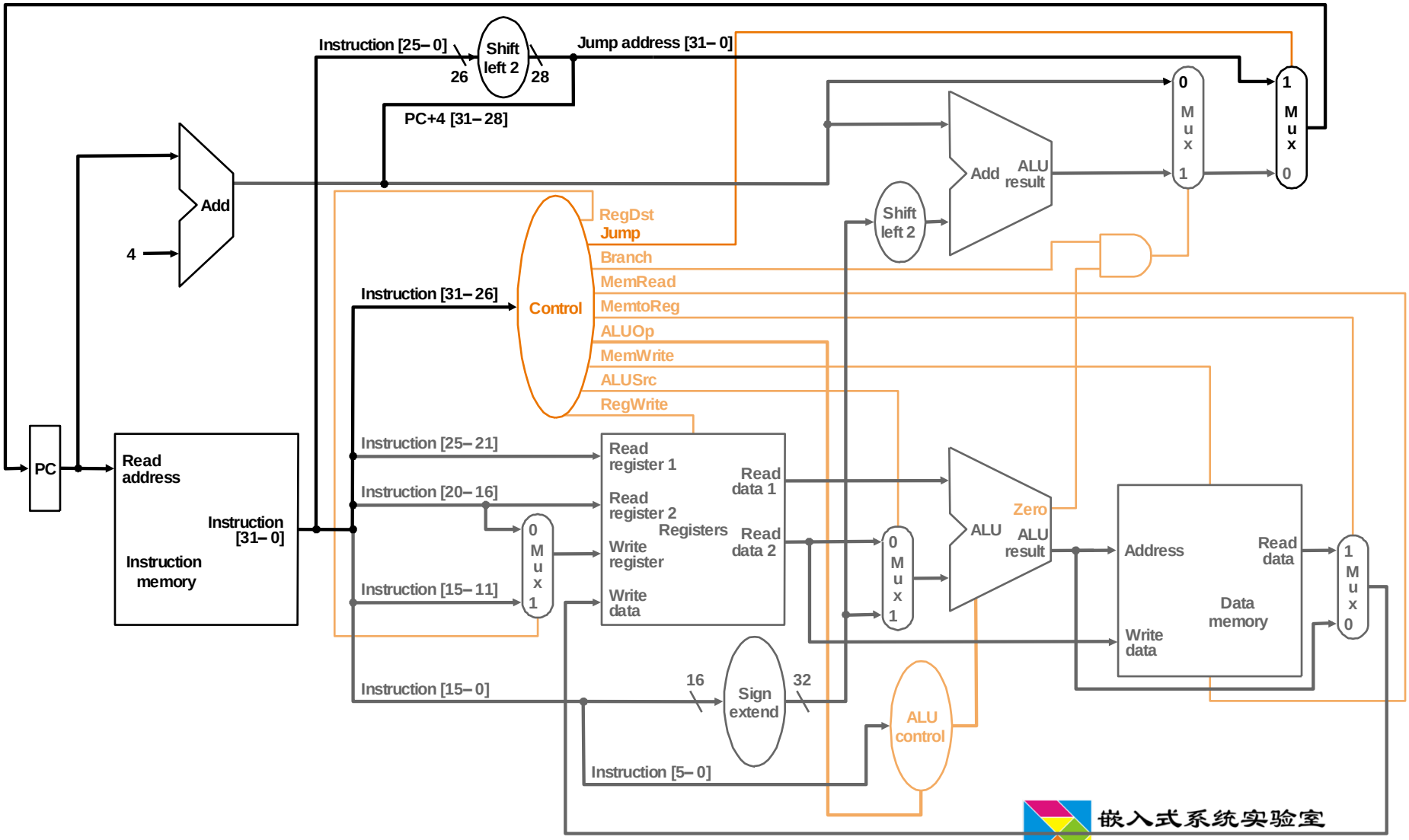
- UJ无条件转移，关键在于目标地址的生成
  - ✓PC±2^{19}
- 增加一个jump指令识别控制

JAL指令应该如何实现

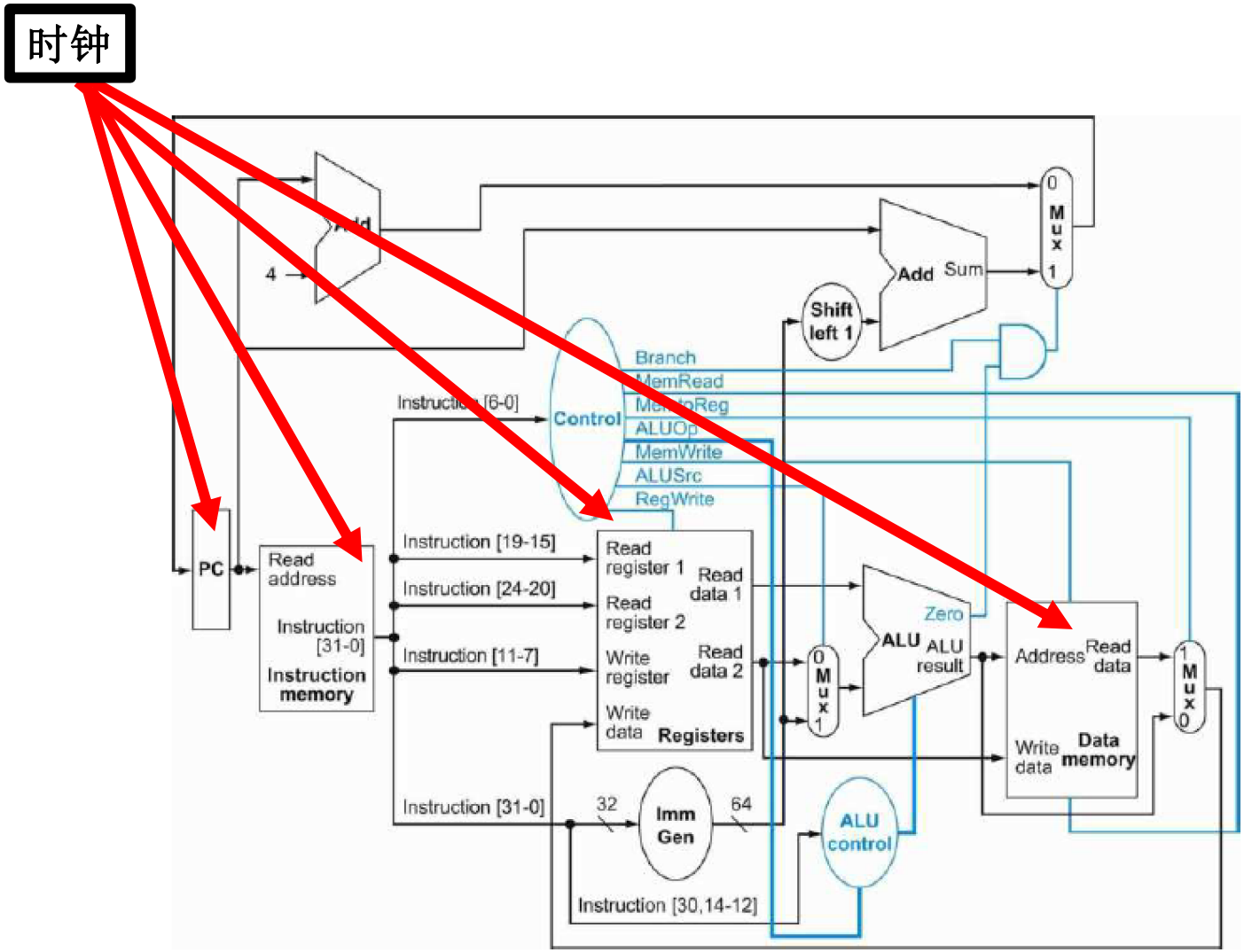


jal x0, 2000

参考：JAL指令的MIPS实现



Single-Cycle Processor: Clocking



时钟周期的确定方法

- 时钟周期由关键路径确定
- 关键路径的选择=哪种指令执行时间最长?
  - ✓PC时钟初始化=30ps　读内存=250ps
  - ✓寄存器读=150ps　寄存器写=20ps
  - ✓ALU计算=200ps　多个多路选择器共25ps（几个？）
- 画出关键路径，并计算主频

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

定长单周期or不定长单周期?

- 设程序中load有24%，store有12%，R-type有44%，beq有18%，jump有2%。假设load, store,R,beq,jump指令的时间分别是8,7,6,5,2ns（如下表）。如果时钟周期固定，单周期的时钟为8ns；如果时钟周期不定长，单周期的时钟可以是2ns～8ns。试比较时钟定长单周期实现和不定长单周期实现的性能。

指令	inst MEM	Reg Read	ALU	Data MEM	Reg Write	Total
R-Type	2	1	2		1	6ns
lw	2	1	2	2	1	8ns
sw	2	1	2	2		7ns
beq	2	1	2			5ns
JAL	2					2ns

单周期实现

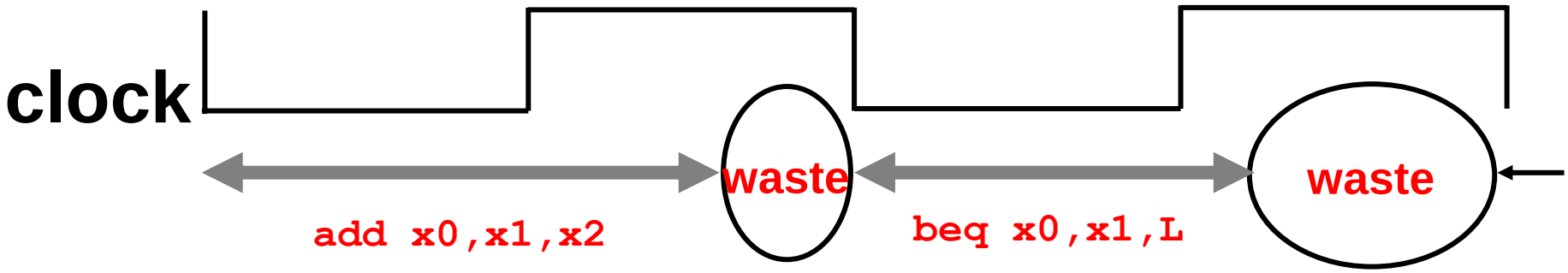
- 程序执行时间 = 指令数×CPI×时钟周期时间
- 指令周期 = 机器周期 = 时钟周期
  - ✓ CPI = 1

- 平均指令执行时间 = 8×24% + 7×12% + 6×44% + 5×18% + 2×2% = 6.3ns
- 因此，变长单周期实现较定长单周期实现快8/6.3=1.27倍

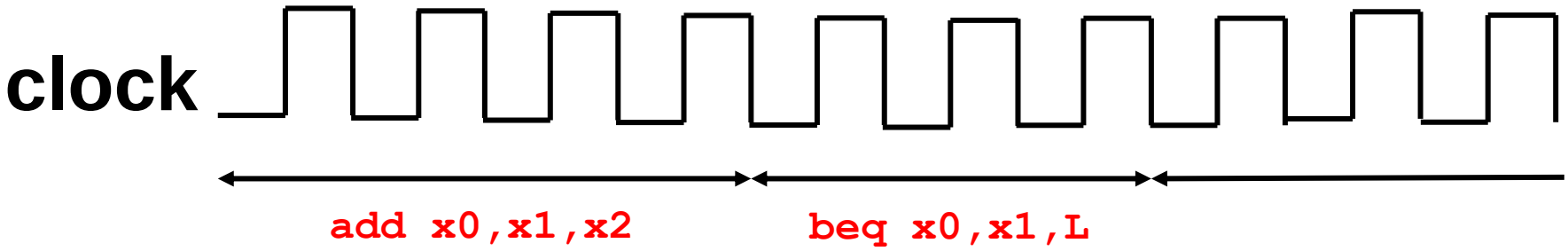
变长单周期实现?

定长单周期 vs. 不定长多周期

***Single-cycle Implementation*：定长指令周期**



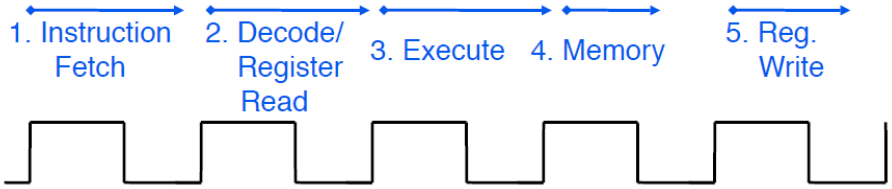
***Multicycle Implementation*：不定长指令周期**



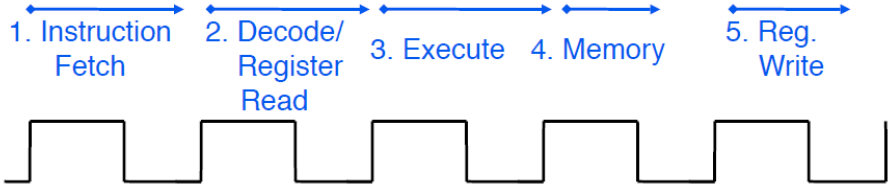
- **Multicycle Implementation:**
  less waste = higher performance

多周期实现

- 根据指令执行所使用的功能部件，将执行过程划分成多个阶段，每个阶段一个周期（机器周期）
  - ✓早结束的指令可以提升性能
- 时钟周期确定
  - ✓每个周期的工作尽量平衡
  - ✓假设一个周期内可以完成
    - 一次MEM访问，or
    - 一次寄存器访问（2 reads or one write），or
    - 一次ALU操作



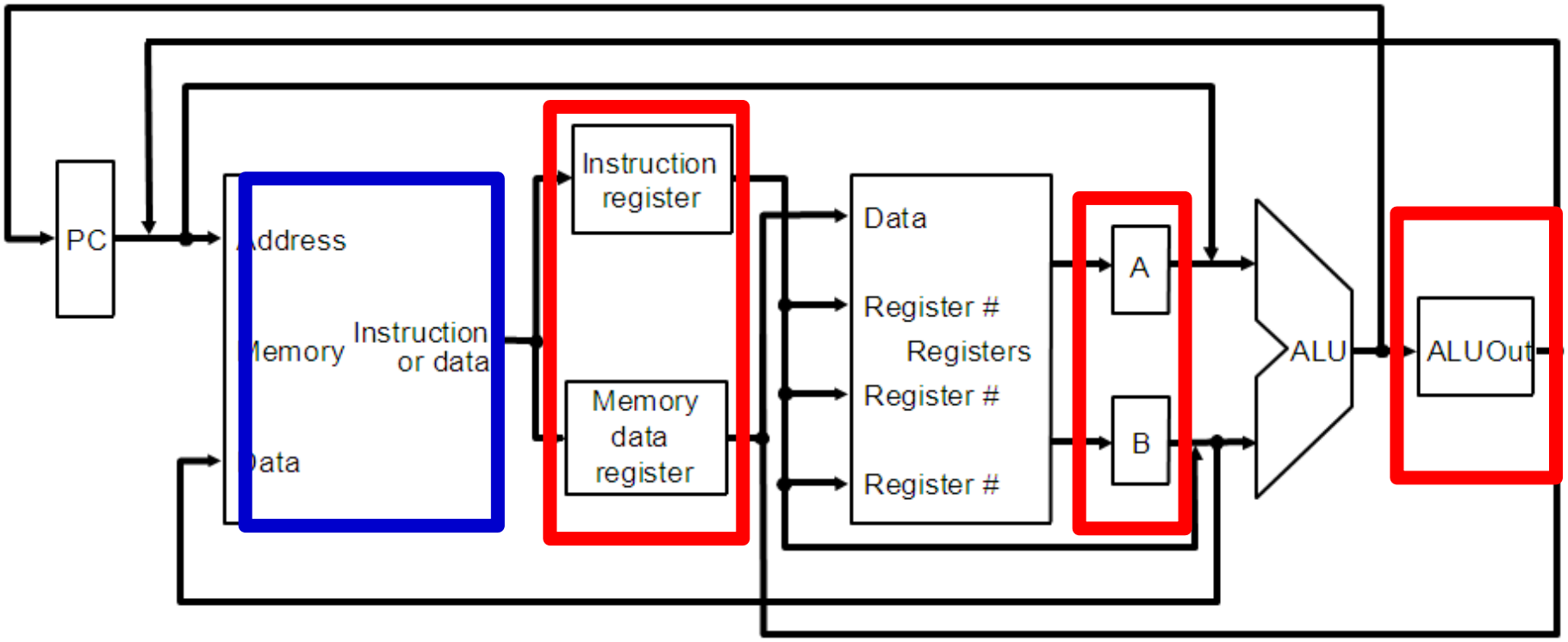
指令执行的多个周期



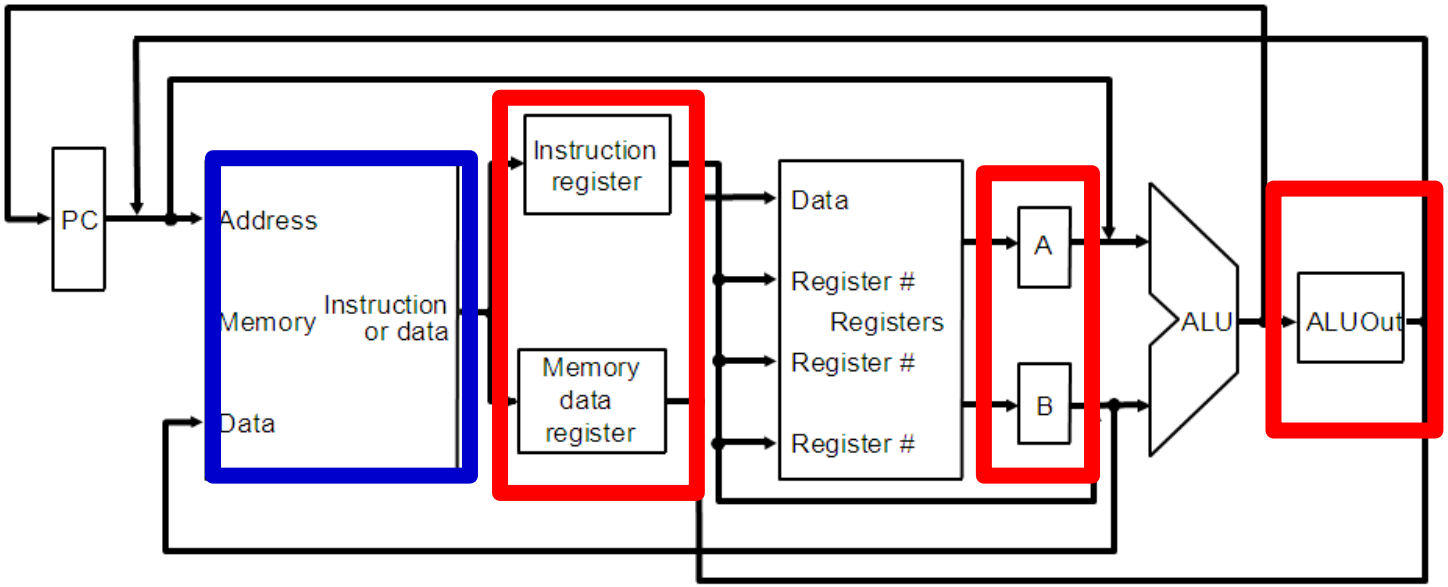
- 共5个阶段（周期）
  - ✓取指
  - ✓译码阶段、计算beq目标地址
  - ✓执行：R-type指令执行、访存地址计算，分支完成阶段
  - ✓访存：lw读，store和R-type指令完成阶段
  - ✓写回：lw完成阶段
- 注意
  - ✓定长机器周期：机器周期=时钟周期
  - ✓不定长指令周期：分别为3、4、5个机器周期
- 控制器根据机器周期标识发出控制信号

多周期简单数据通路

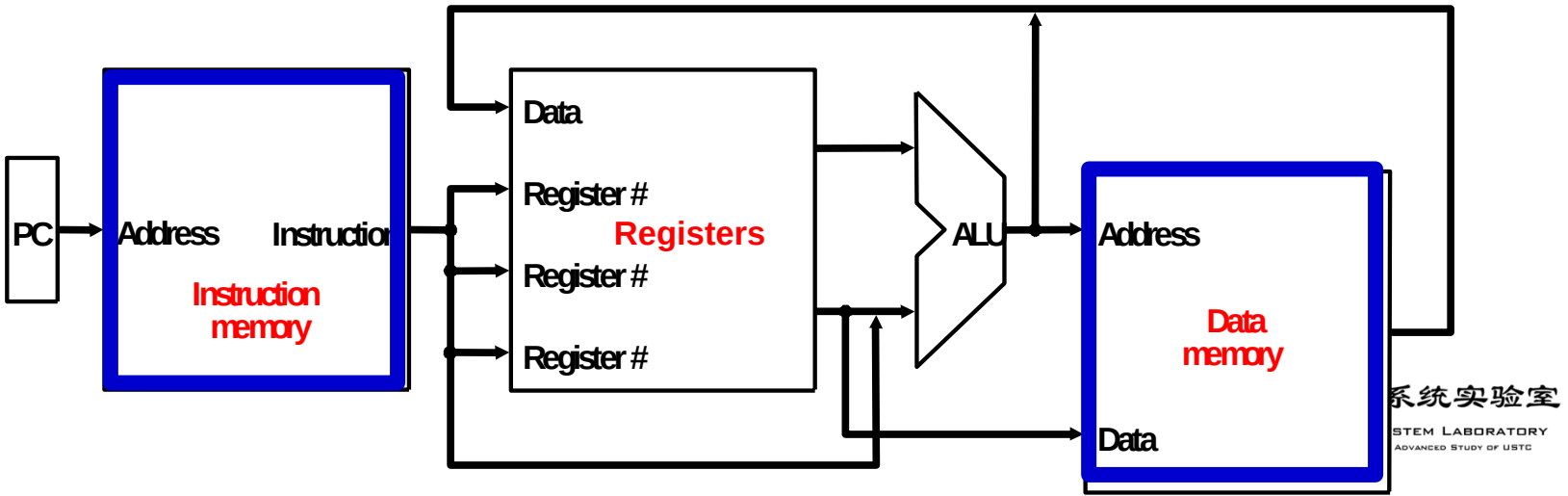
用哪些寄存器来实现多周期



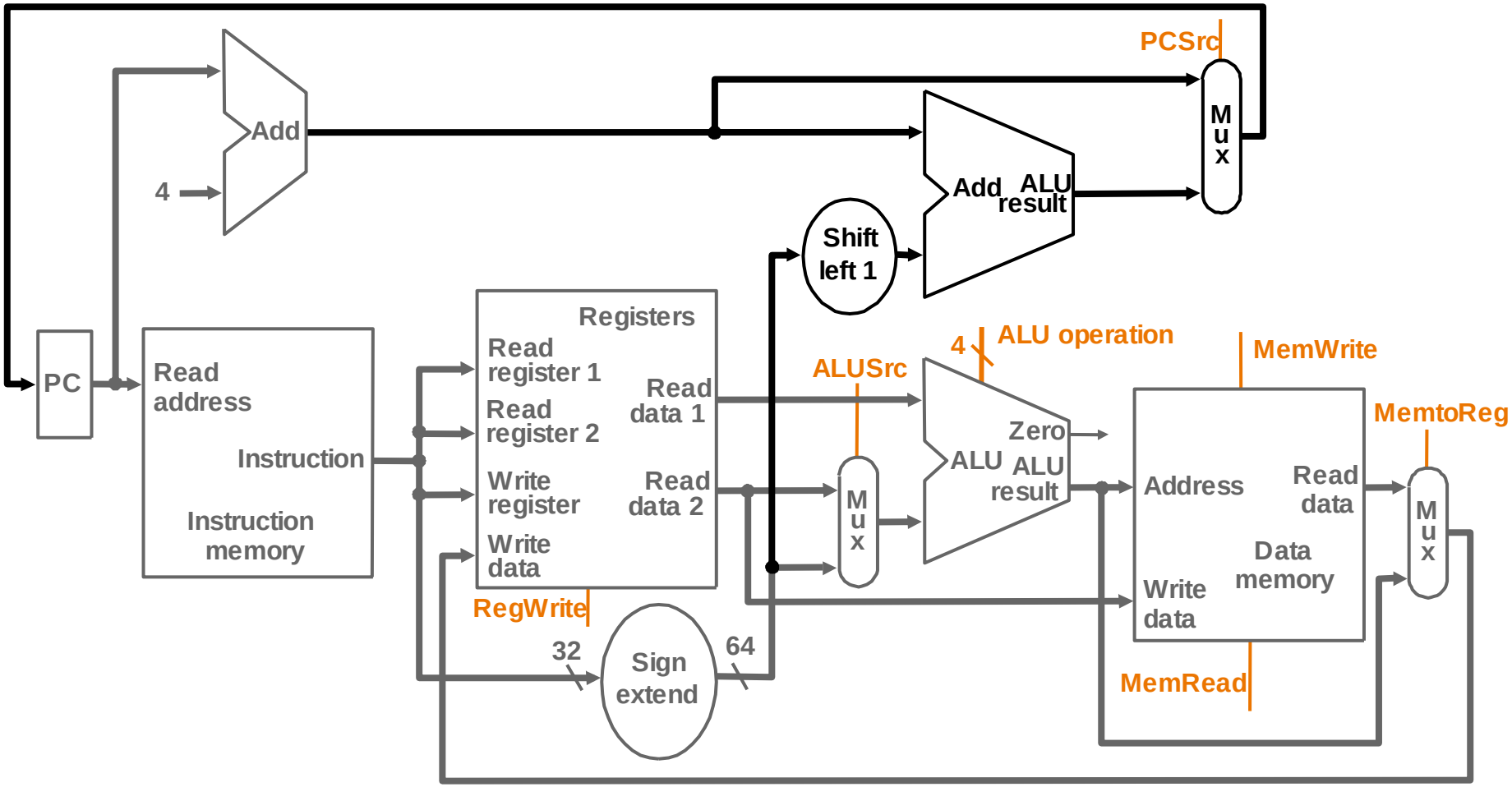
单周期和多周期简单数据通路区别



ALU、寄存器、存储器、控制信号

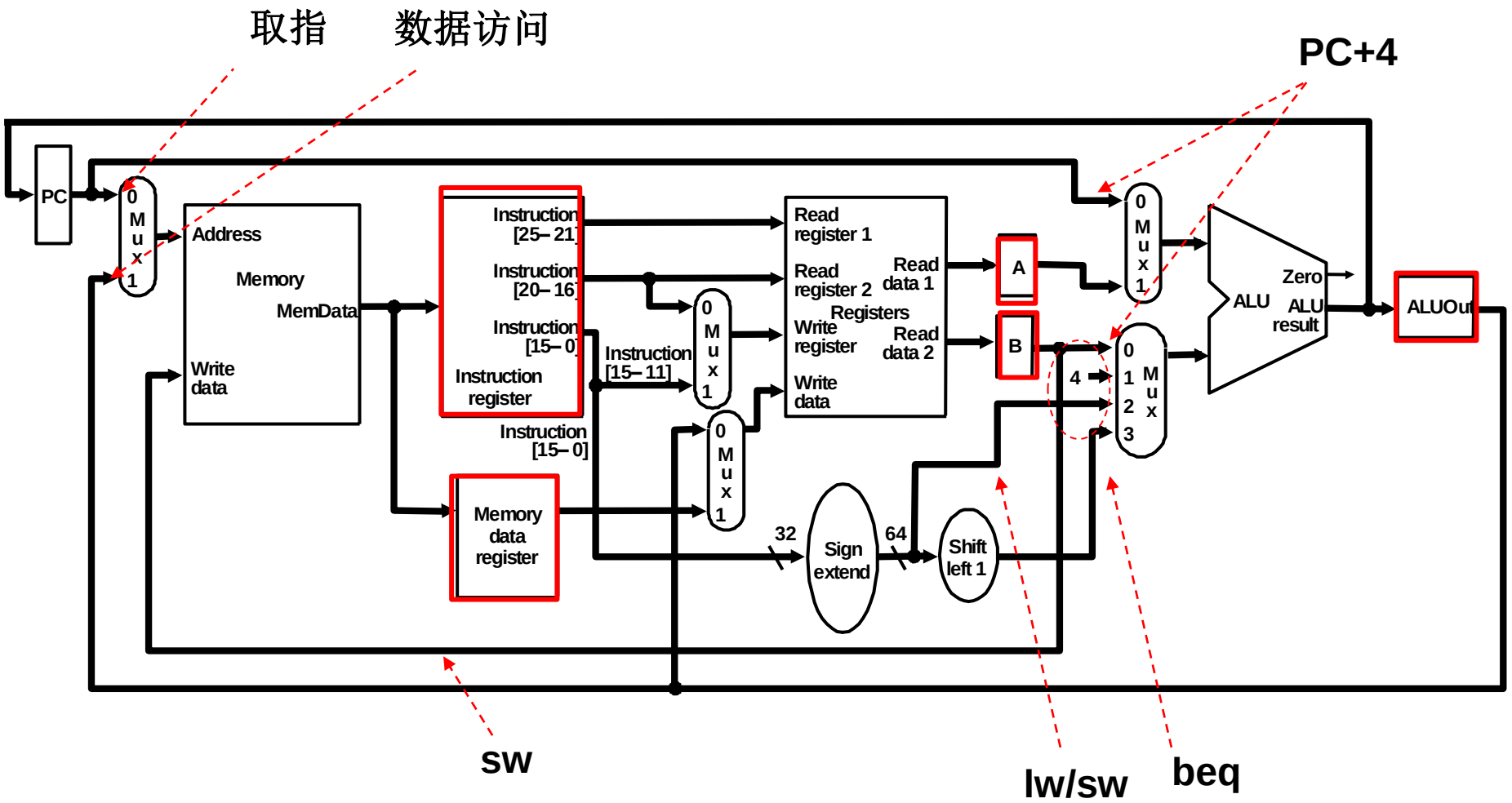


单周期数据通路


PCSrc
Add
4
M
u
x
Add
ALU
result
Shift
left 1
PC
Read
address
Instruction
Instruction
memory
Registers
Read
register 1
Read
register 2
Write
register
Write
data
Read
data 1
Read
data 2
RegWrite
32
Sign
extend
64
ALUSrc
4
ALU operation
MemWrite
MemtoReg
Zero
ALU
ALU
result
Address
Read
data
Data
memory
Write
data
MemRead

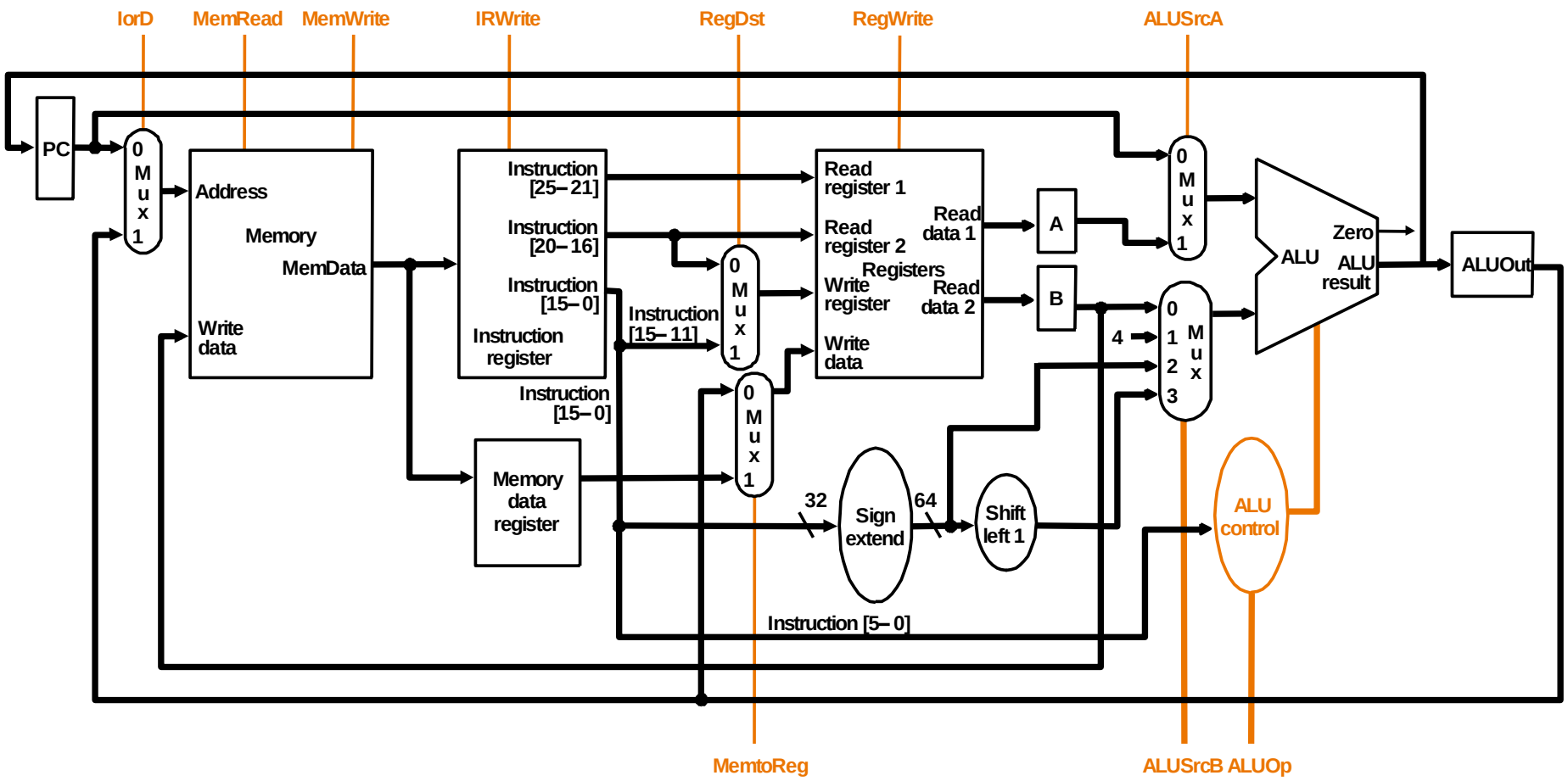
多周期数据通路 (MIPS/RISC-V)

Name (Field Size)	Field 7 bits	5 bits	5 bits	3 bits	5 bits	7 bits	Comments
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



需要在哪里加控制？

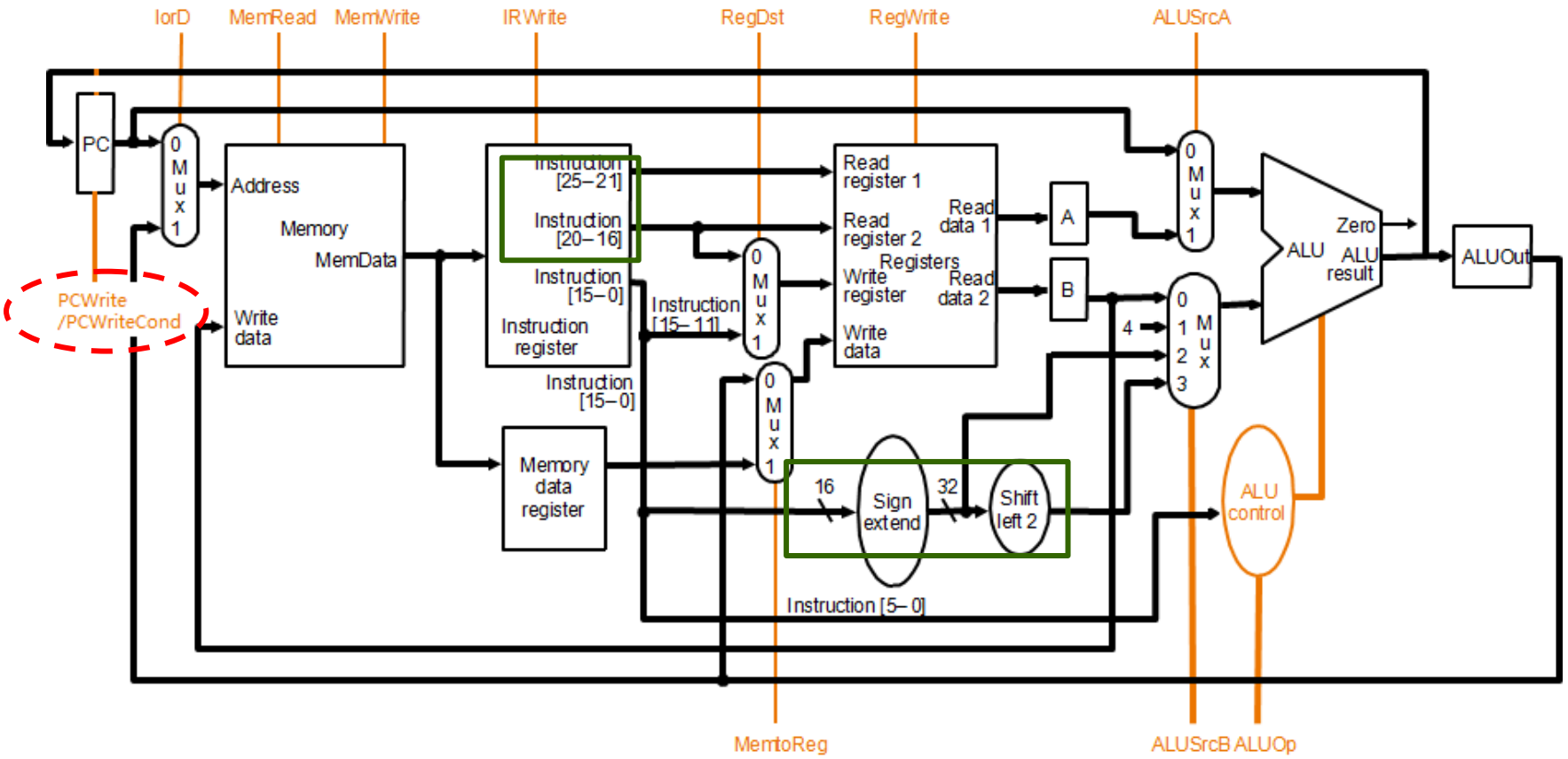
多周期控制信号



思考：为何单周期实现中beq＝NPC＋offset，而多周期是PC＋offset？

还需要对PC的写控制信号

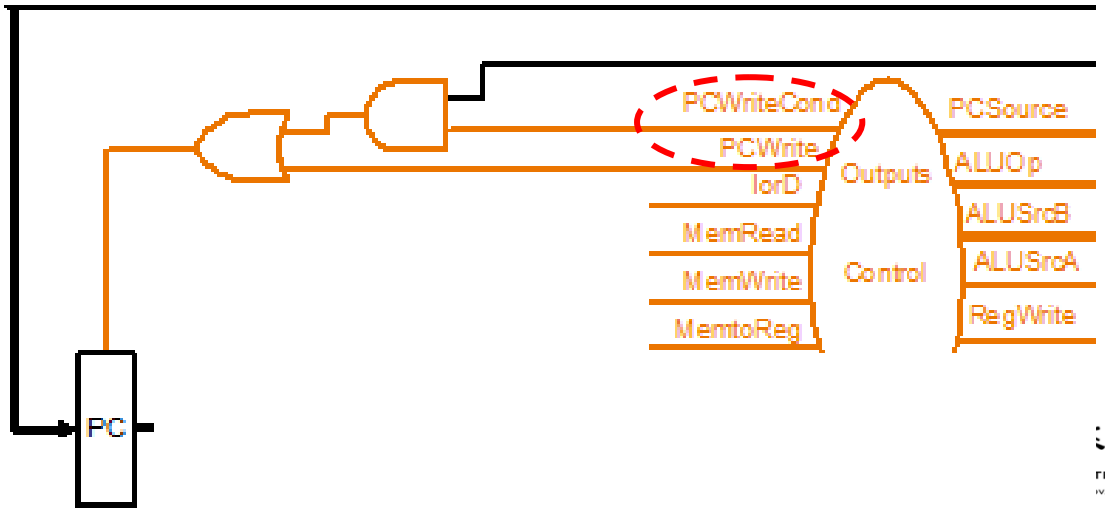
- “在一个指令周期内，PC不能变”



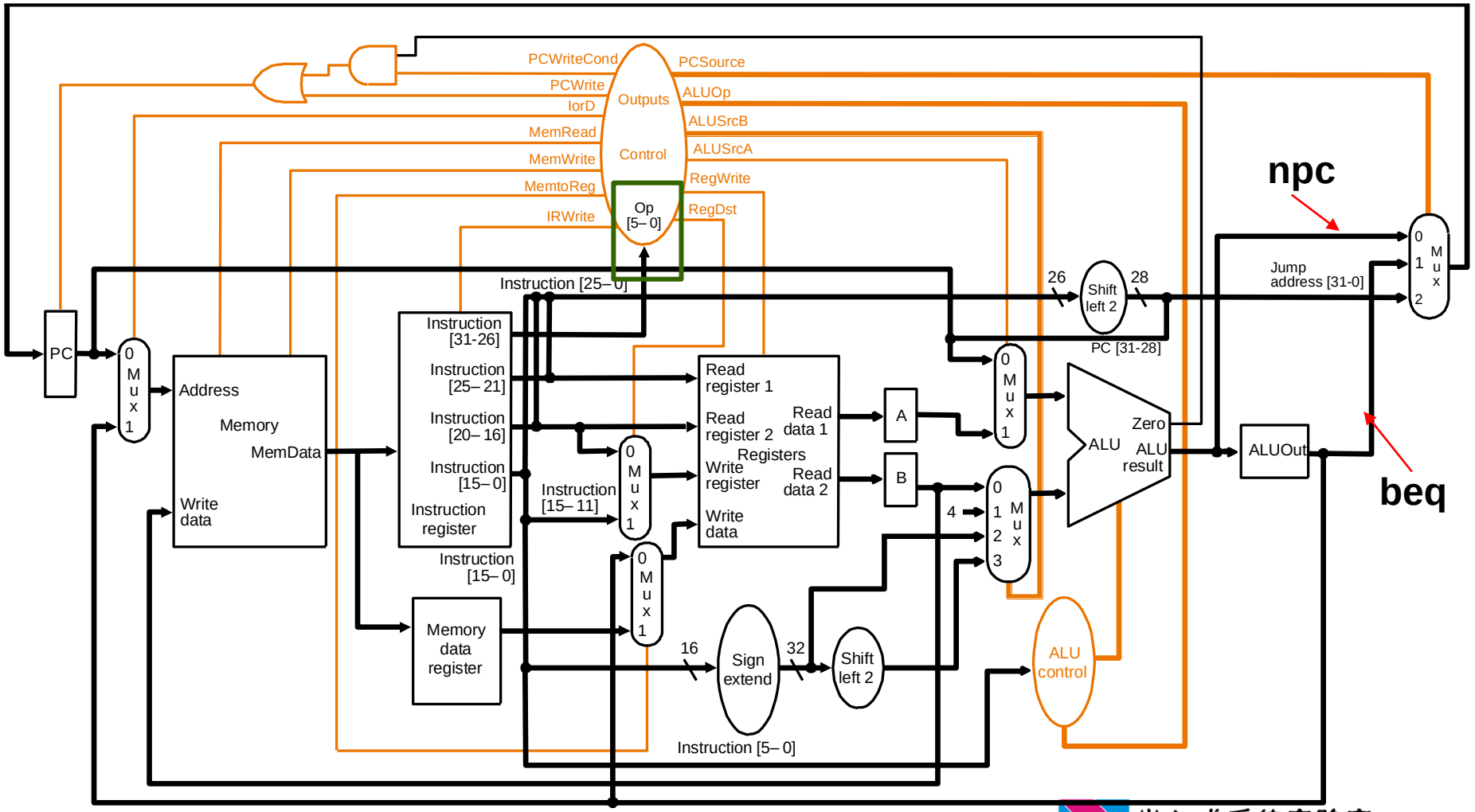
- 思考：为何单周期不需要PC写控制？

PC的写控制

- 3种情况
  - ✓ ALU：PC + 4的输出直接存入PC，无条件写（取指阶段）
  - ✓ ALU：beq指令的目标地址（译码阶段）
  - ✓ ALU：JAL指令 UJ-type（可在译码阶段）
- 需要两个写控制
  - ✓ 无条件写PCWrite：PC+4, JAL
  - ✓ 有条件写PCWriteCond：beq

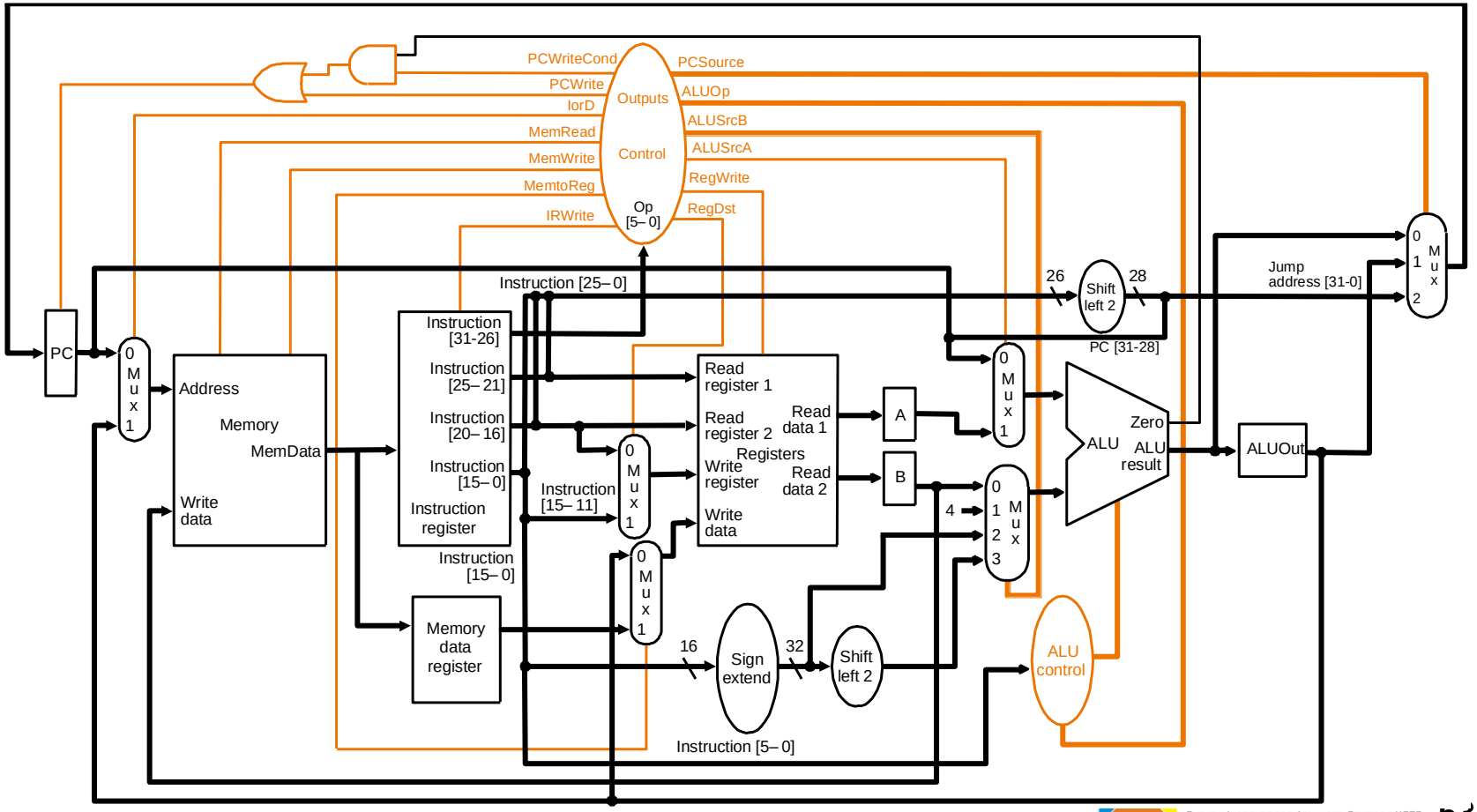


主控制部件


PCWriteCond
PCWrite
IorD
MemRead
MemWrite
MemtoReg
IRWrite
Outputs
Control
Op
[5– 0]
PCSource
ALUOp
ALUSrcB
ALUSrcA
RegWrite
RegDst
npc
Jump
address [31-0]
26
Shift
left 2
28
PC [31-28]
Instruction [25– 0]
PC
0
M
u
x
1
Address
Memory
MemData
Write
data
Instruction
[31-26]
Instruction
[25– 21]
Instruction
[20– 16]
Instruction
[15– 0]
Instruction
register
Instruction
[15– 0]
Instruction
[15– 11]
Read
register 1
Read
register 2
Registers
Write
register
Write
data
Read
data 1
Read
data 2
A
B
4
ALU
Zero
ALU
result
ALUOut
beq
Memory
data
register
16
Sign
extend
32
Shift
left 2
ALU
control
Instruction [5– 0]

取指阶段

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



译码阶段

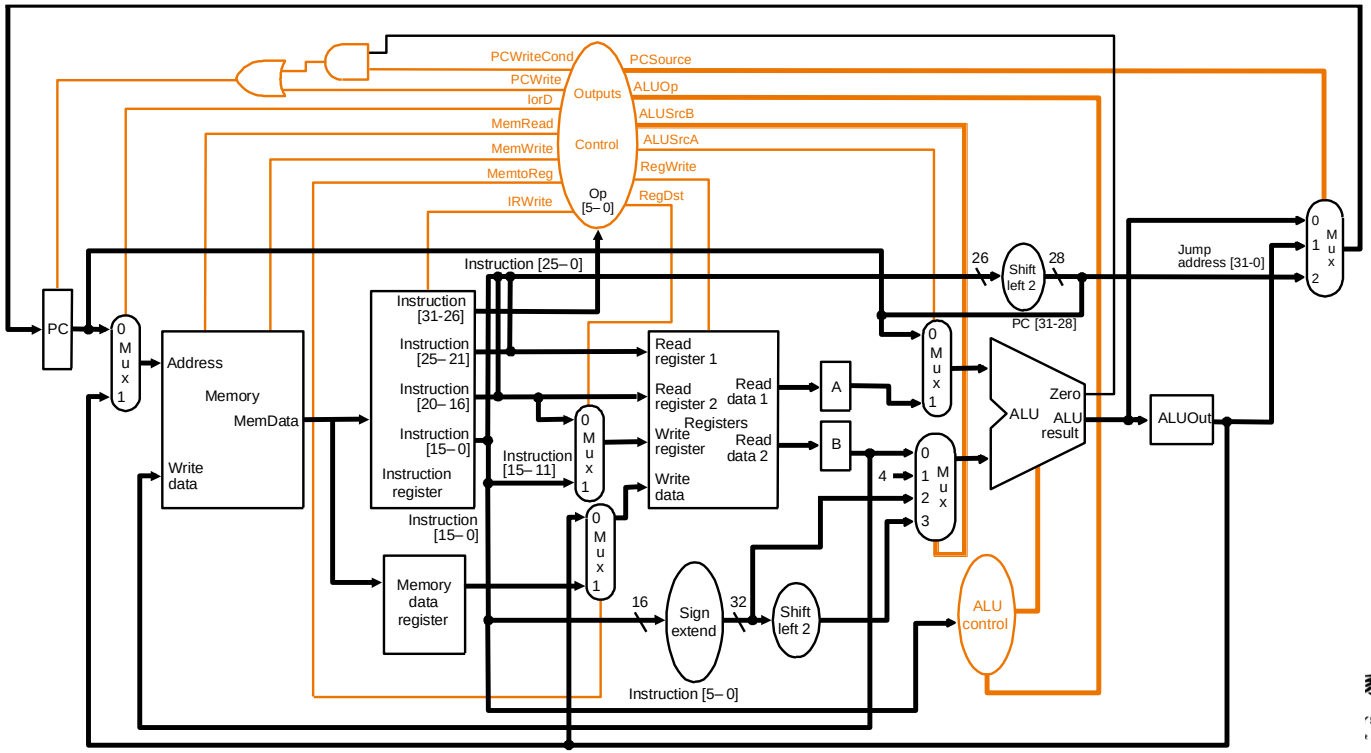
- 指令译码和读寄存器
  - ✓ 将rs和rt送往A和B：A=Reg[IR[25-21]], B=Reg[IR[20-16]]

A=rs1 B=rs2

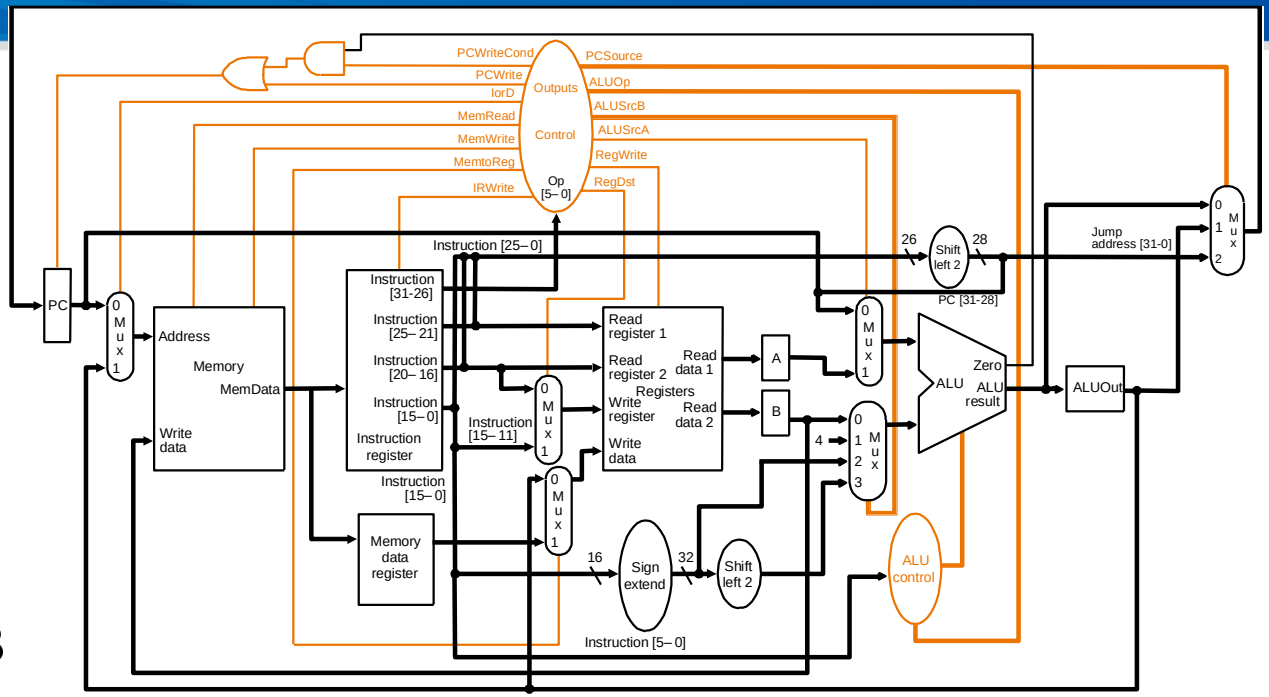
- 计算beq目标地址
  - ✓ ALUOut=PC+(sign-extend(IR[15-0] <<2)

PC+offset

    - 由于此时尚不知是何指令，所以读寄存器和计算分支地址可能无效
- 控制信号：ALUSrcA, ALUSrcB, ALUOp



R-type执行、访存地址计算、分支完成阶段



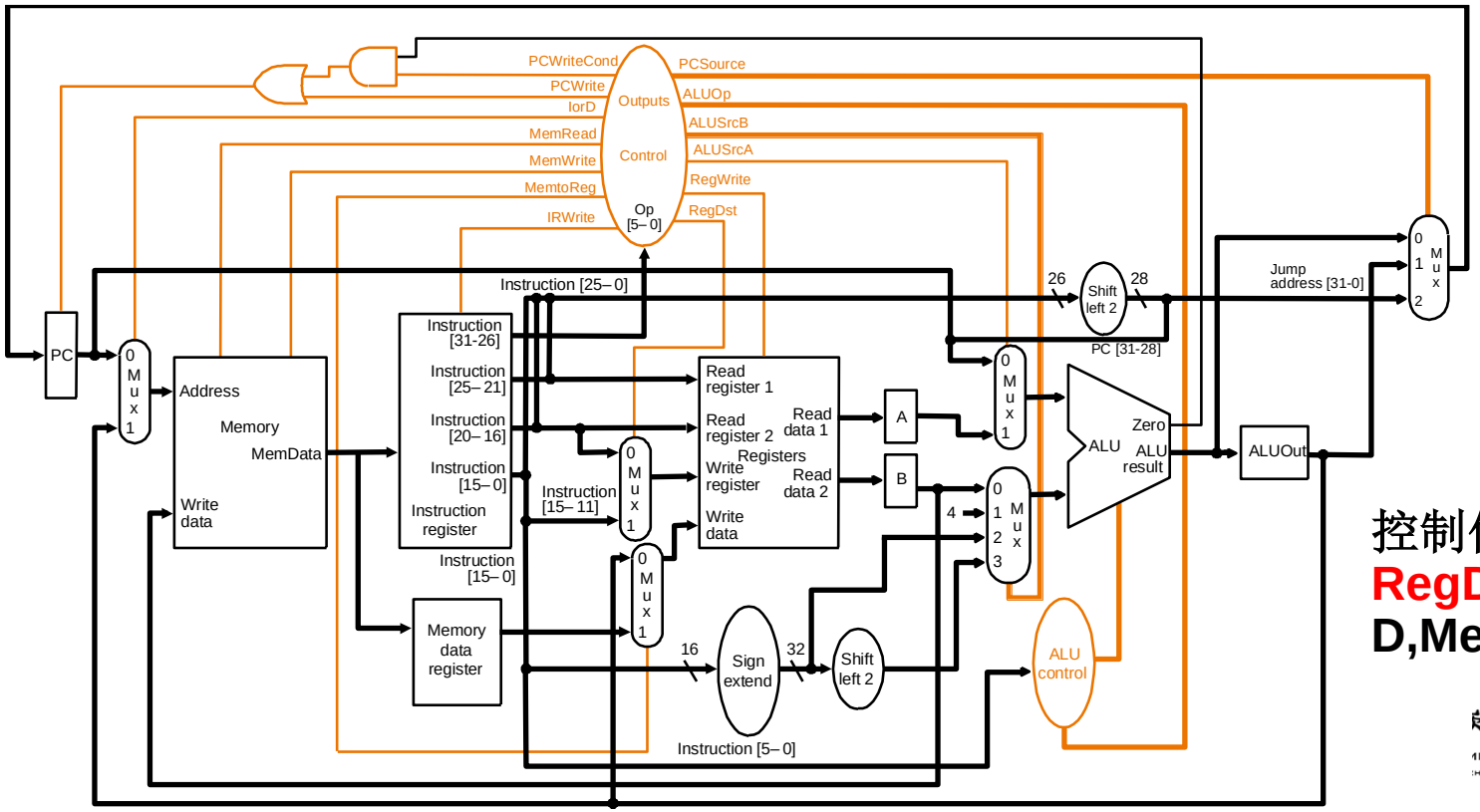
- 依赖于指令类型
  - ✓ R-type指令
    - ALUOut= A op B
  - ✓ 访存指令：计算访存地址
    - ALUOut=A+(sign-extend(IR[15-0]) **imm**
  - ✓ beq指令
    - if (A == B) PC=ALUOut；

控制信号：**ALUSrcA, ALUSrcB, ALUOp, PCWriteCond,PCSource, PCWrite**

  - ✓ jump指令 **PC+offset**
    - PC = PC[31-28] || (IR[25-0] << 2)　MIPS
    - PC = PC+IR[0-20]　RISC-V

R-type和sw完成、lw读阶段

- 第四阶段
  - ✓ R-type完成：结果写回 **rd** Reg[IR[15-11]] = ALUOut
  - ✓ sw完成：写入MEM MEM[ALUOut] = B
  - ✓ lw读： MDR = MEM[ALUOut]



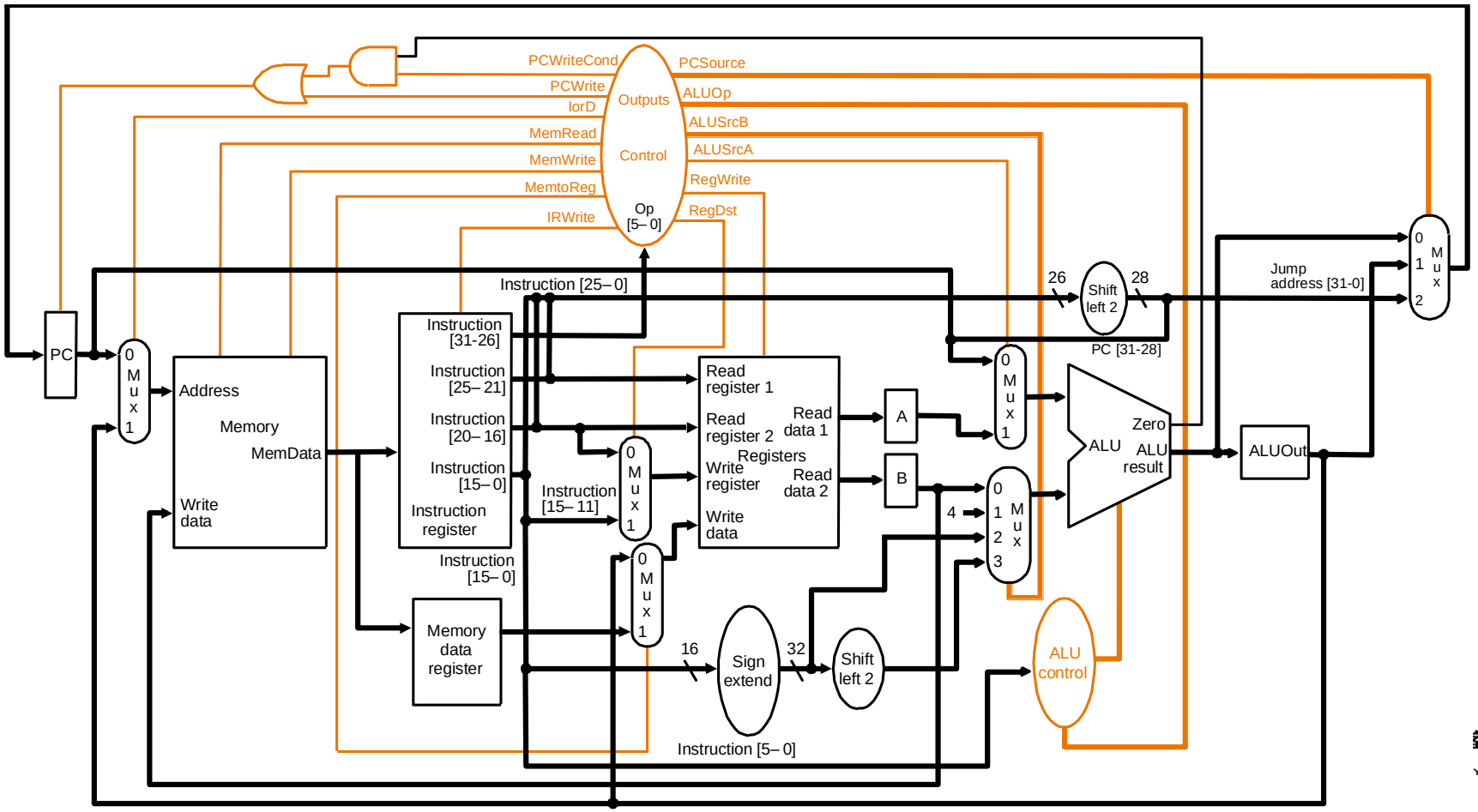
控制信号：**RegWrite, RegDst,MemtoReg,IorD,MemRead,MemWrite**

lw写阶段

- 第五阶段
  - ✓lw写回：Reg[IR[15-11]] = MDR

rd

控制信号：**RegWrite, RegDst,MemtoReg**



控制信号与操作的关系

1 ALUSrcA, ALUSrcB, ALUOp (ALU操作= ALU指令or算地址)

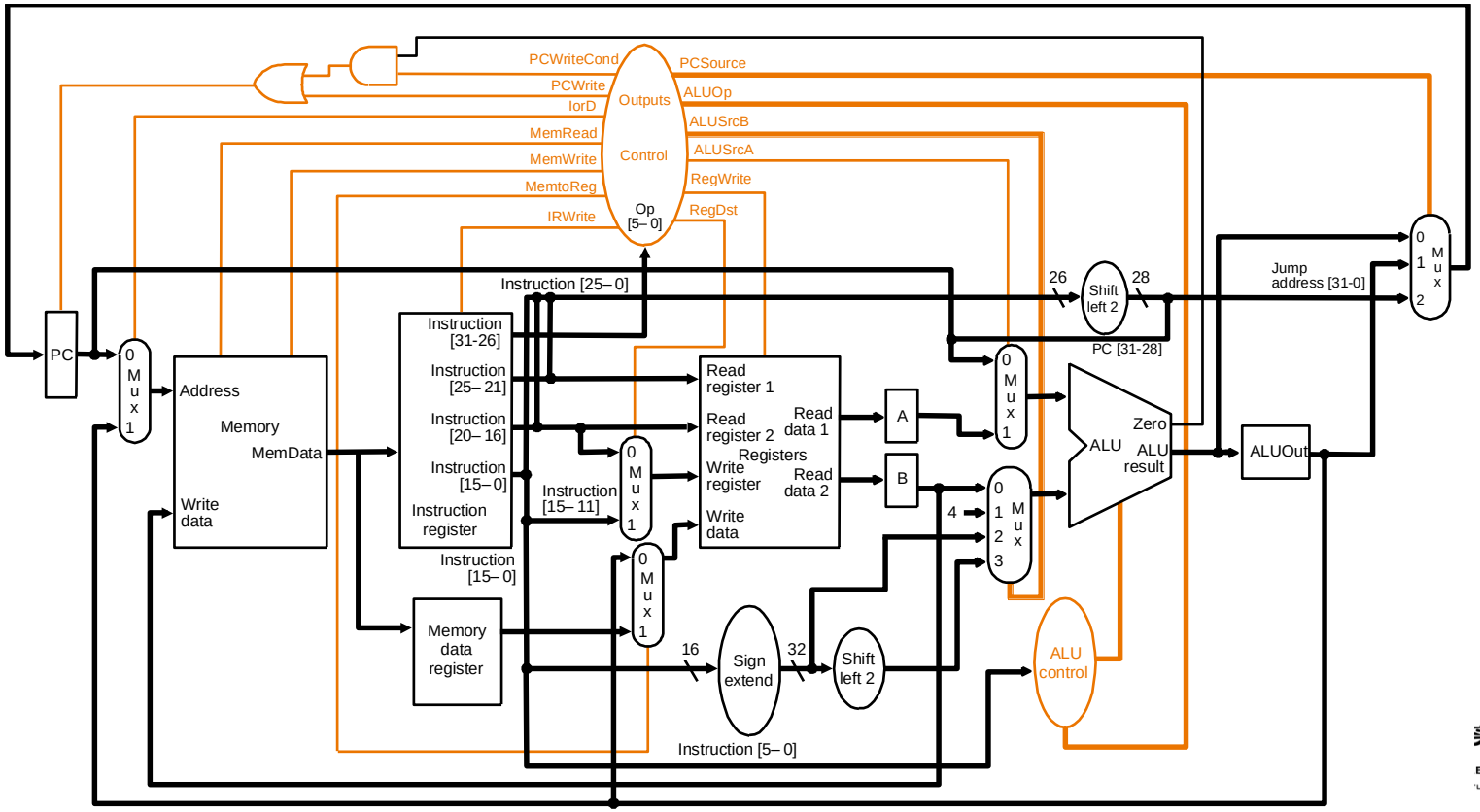
2 RegWrite, RegDst,MemtoReg (写Reg=ALU指令or LW指令)

3. MemRead, IRWrite, IorD (读存储器、写IR=取指)

4 PCWriteCond,PCSource, PCWrite(写PC=顺序、分支)

5. IorD,MemRead（读存储器=数据）

6. IorD,MemWrite（SW）



Multicycle RTL（参考）

Step	R-Type	lw/sw	beq/bne	j
IF	IR = Mem[PC] PC = PC + 4			
ID	A = Reg[IR[25-21]] B = Reg[IR[20-16]] ALUOut = PC + (SE(IR[15-0]) << 2)			
EX	ALUOut = A op B	ALUOut = A +SE(IR[15-0]) 访存指令PC+offset	If (A==B) then PC = ALUOut	PC = PC[31-28] || (IR[25-0]<<2
MEM	Reg[IR[15-11]] = ALUOut	MDR=Mem[ALUOut] Mem[ALUOut] = B		
WB		Reg[IR[20-16]] = MDR		

A=rs1 B=rs2
B指令PC+offset

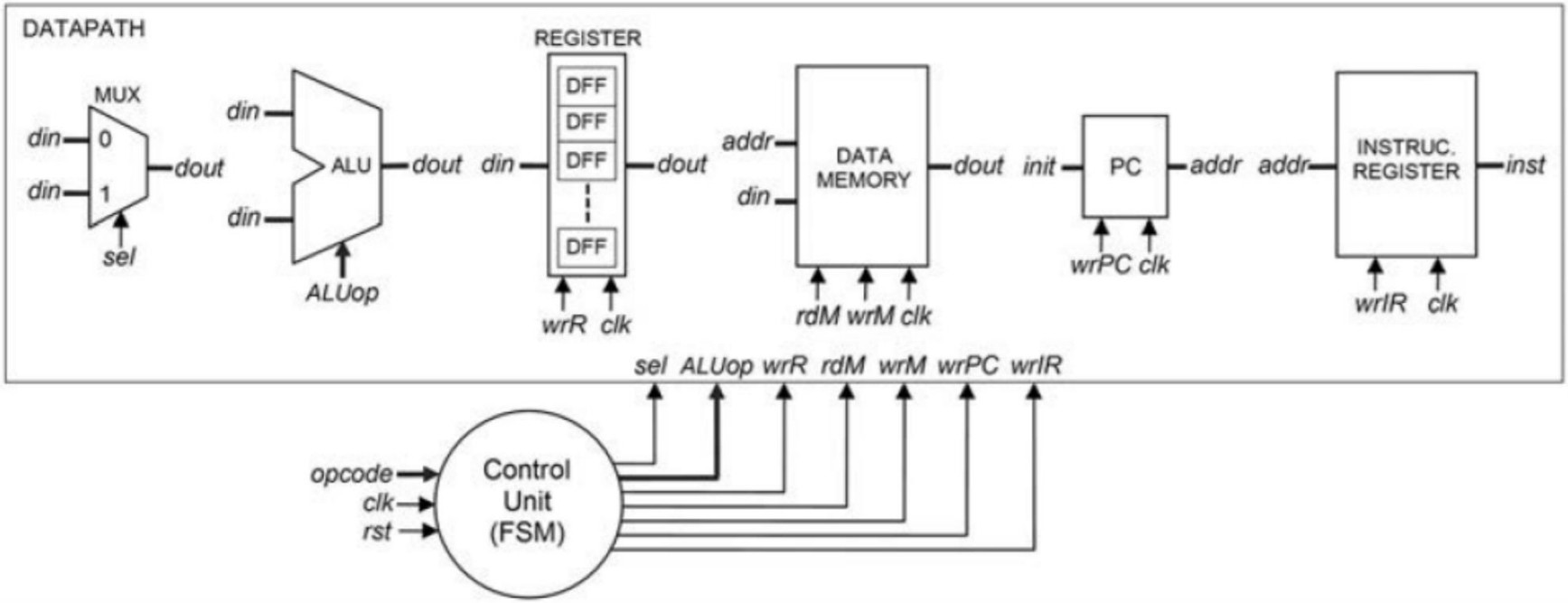
rd

跳转指令PC+offset

rd

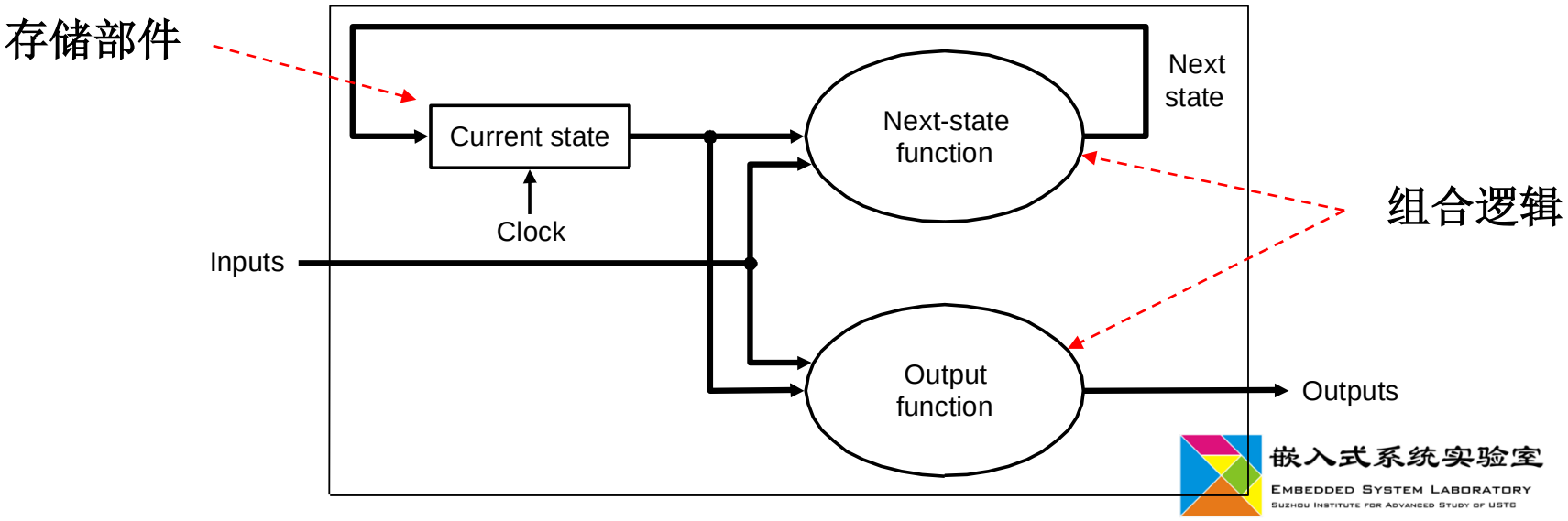
控制信号产生

- FSM: Finite State Machine



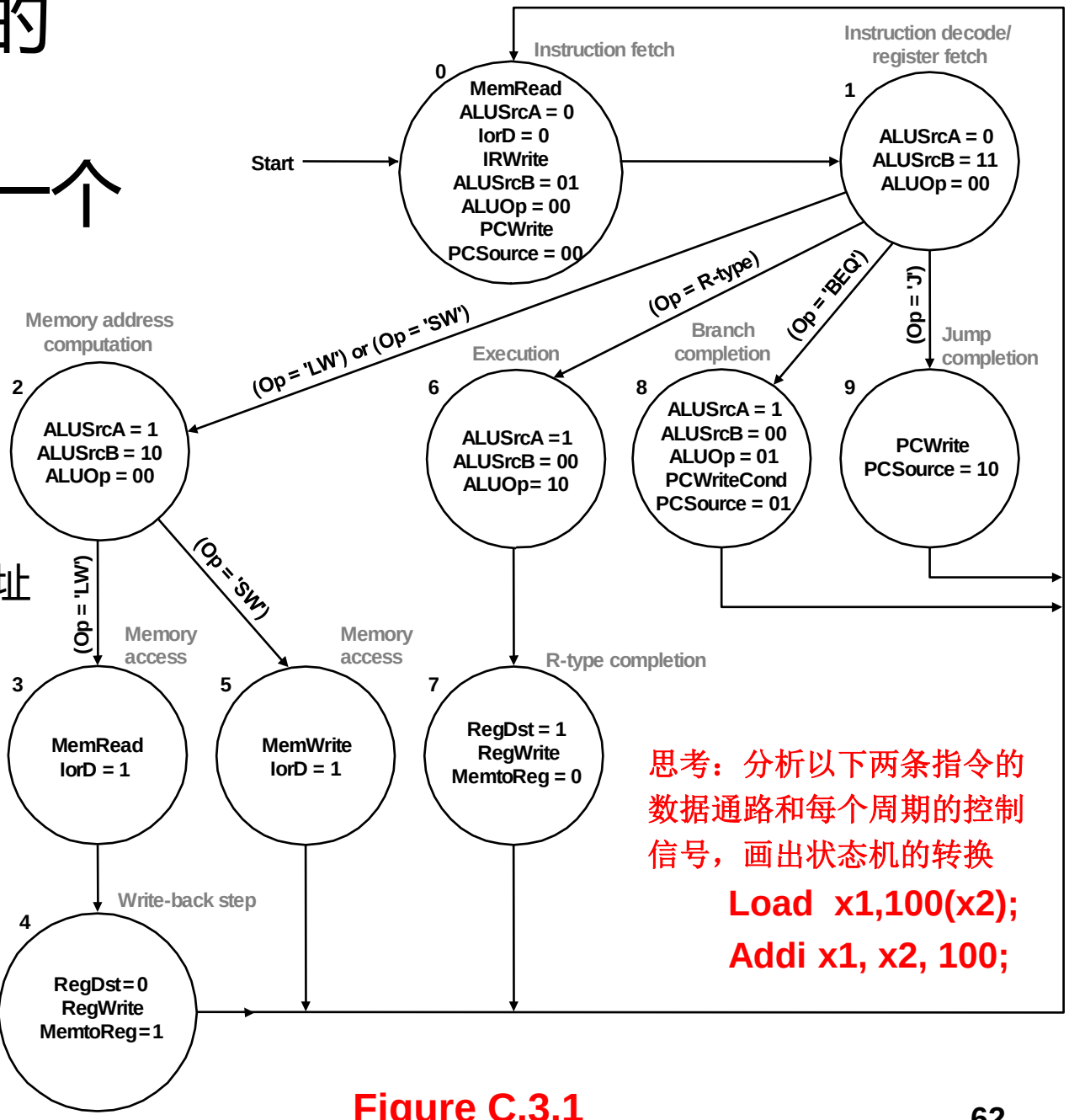
FSM控制部件实现

- FSM: Finite State Machine
- Moore型（Edward Moore），Mealy型（George Mealy）
  - ✓ Moore型速度快（输出与输入无关，可以在周期一开始就提供控制信号）,输出与时钟完全同步。
  - ✓ Mealy型电路较小。输出与时钟不完全同步（一个tick内随输入而变）。
  - ✓ 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



多周期控制器的MooreFSM，每个状态需要一个时钟周期。

- 取指：PC+4
- 译码: BEQ/Jump算地址
- 执行
  - ✓ R执行
  - ✓ LW/SW算地址
  - ✓ 分支完成
- 访存
  - ✓ R/SW完成
  - ✓ Load访存
- 写回
  - ✓ Load完成



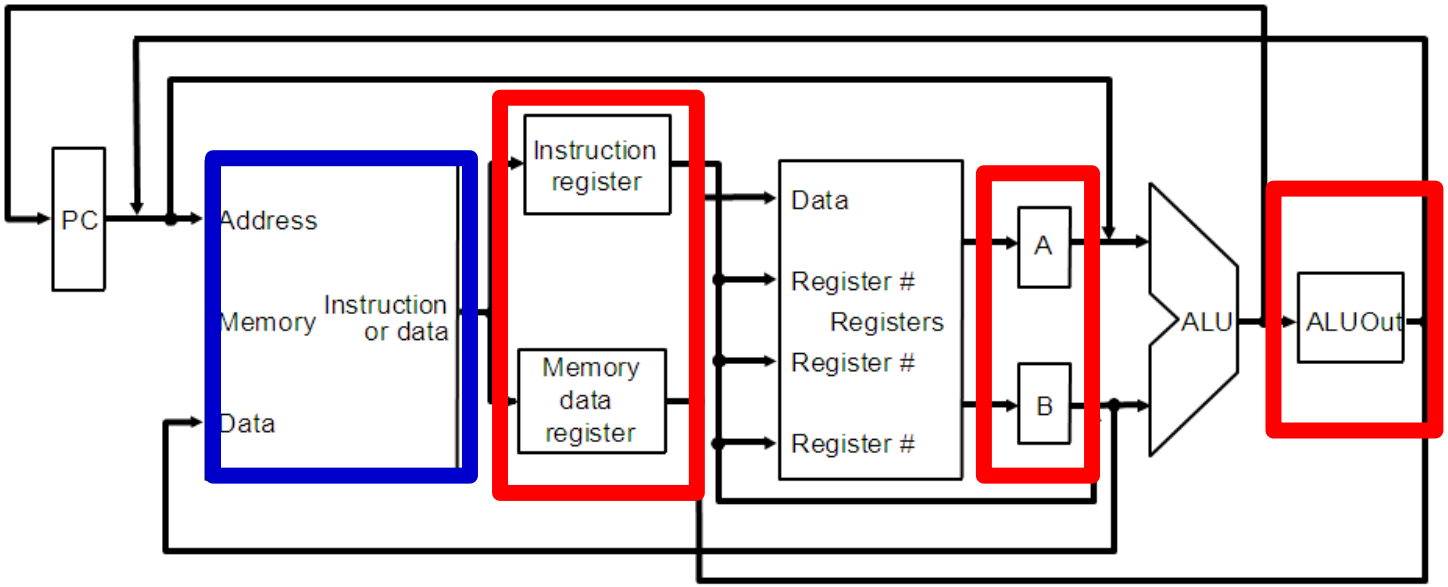
思考：分析以下两条指令的数据通路和每个周期的控制信号，画出状态机的转换

Load x1,100(x2);

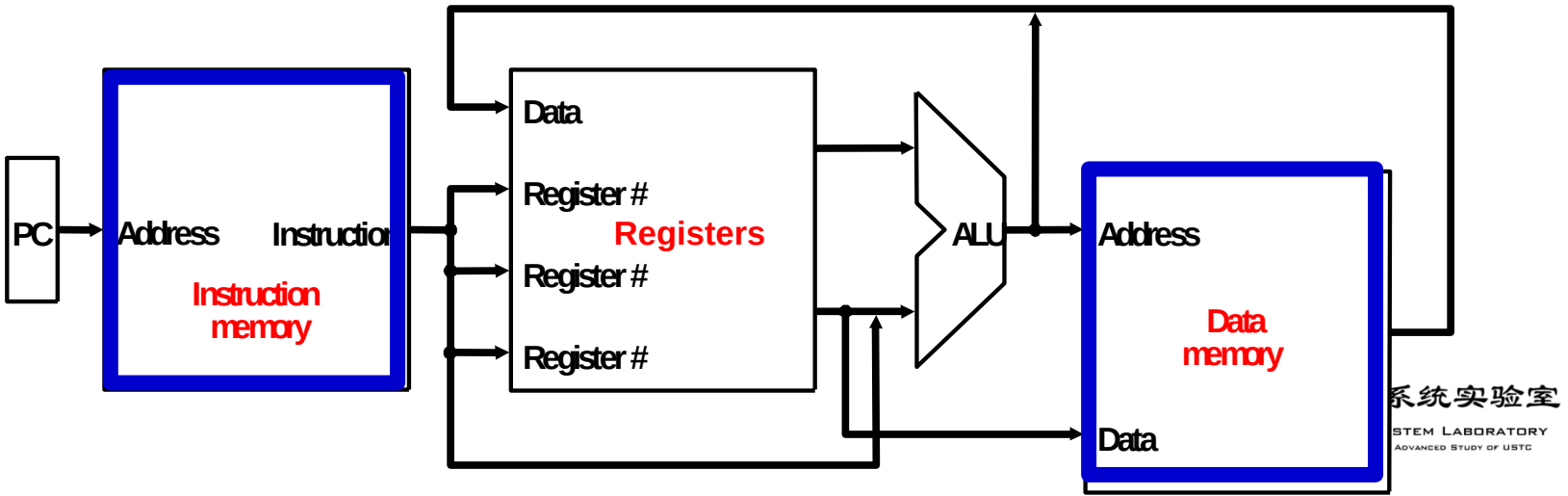
Addi x1, x2, 100;

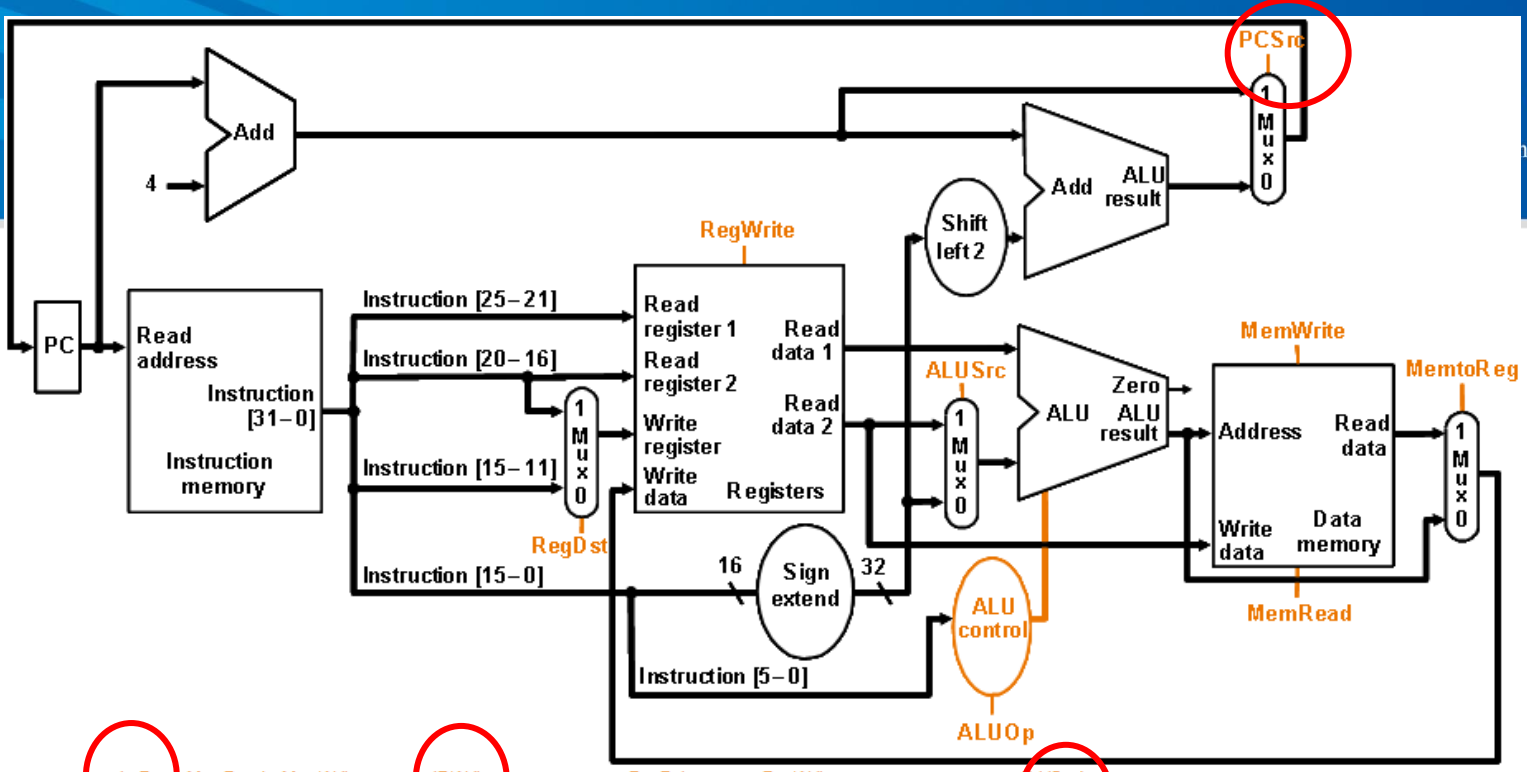
Figure C.3.1

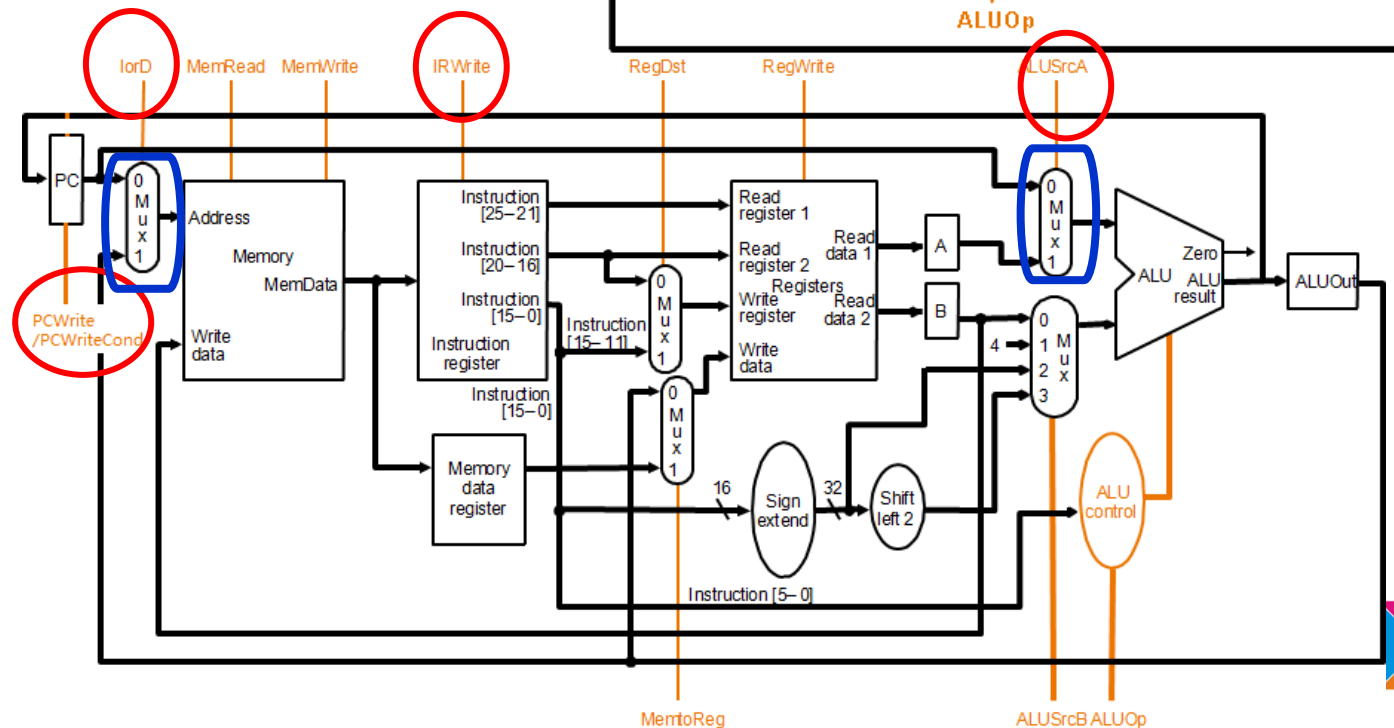
单周期和多周期简单数据通路区别
单周期:COD4 多周期：COD3 Ch5



ALU、寄存器、存储器、控制信号

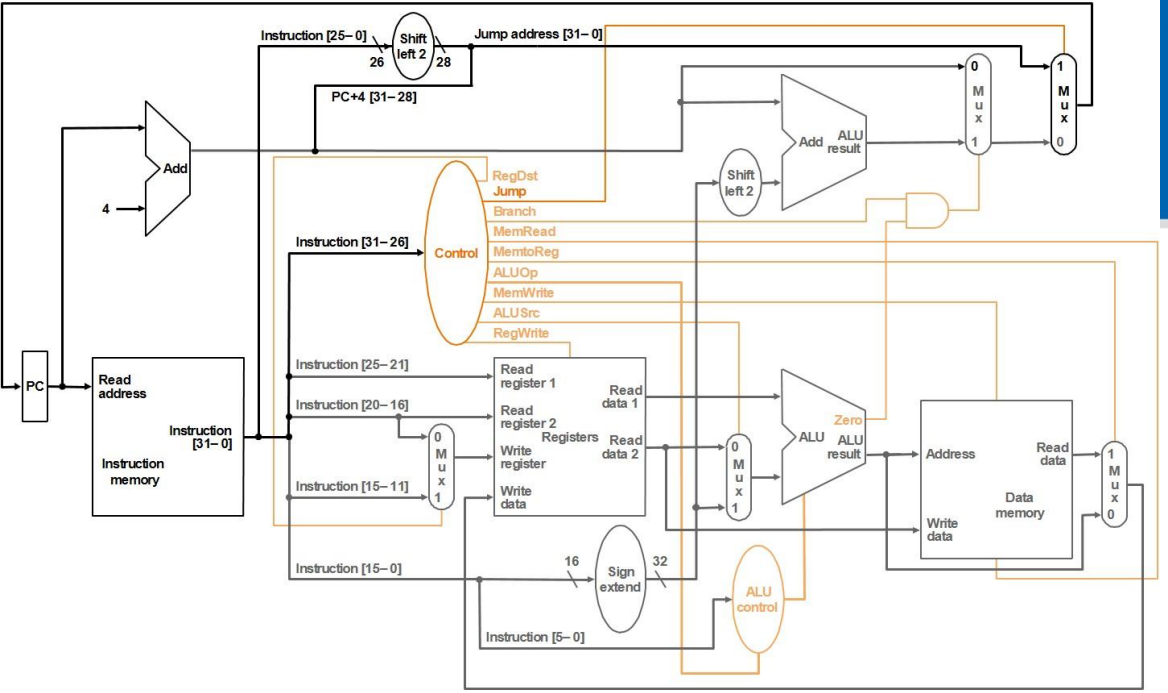






思考：找找两张图中相同及不同之处。

寄存器、存储器、ALU、多选



思考：找找两张图的区别

一、数据通路的区别

1.存储器

2.寄存器

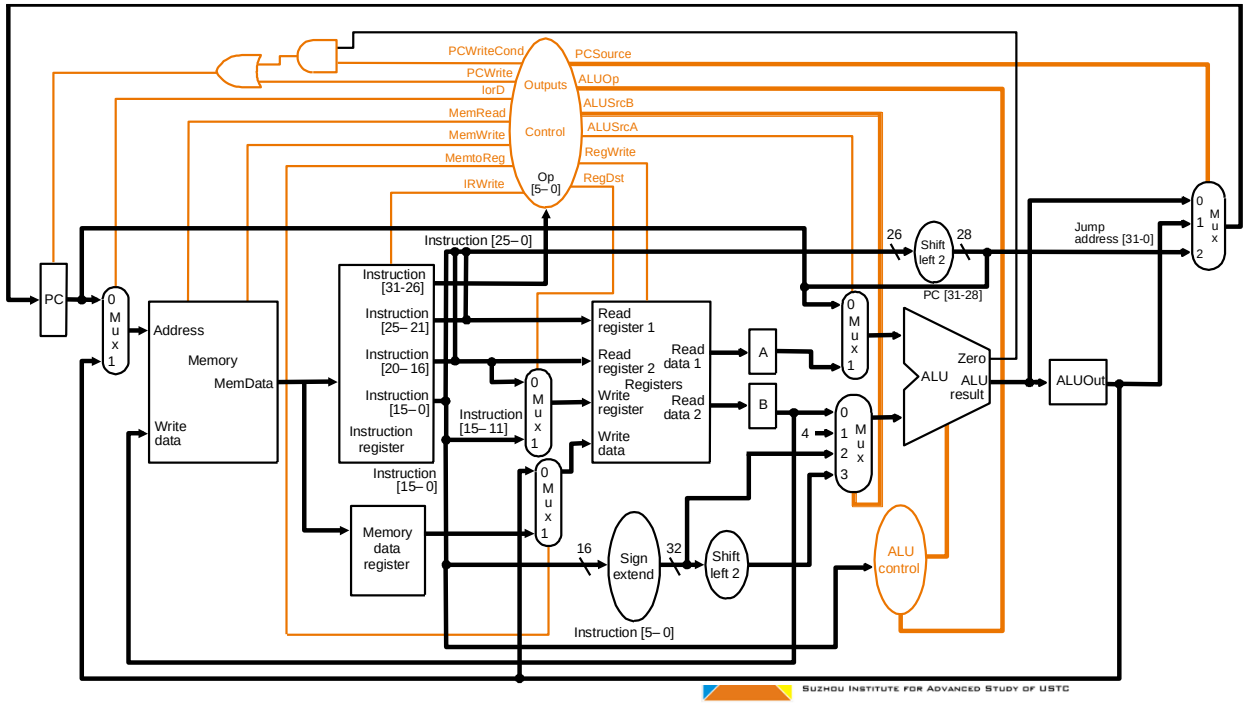
3.ALU

4.MUX

二、控制信号的区别

1. 寄存器文件写使能
2. 存储器读写使能
3. ALU控制
4. PC/IR 寄存器写
5. 多路选择器 2

时序单周期V.S.状态机



小结

- 如何设计一款CPU（单周期or多周期）
  - ✓ 确定指令➔实现操作➔数据通路➔运行部件➔RTL实现➔控制信号➔综合仿真➔开发板调试➔性能评测➔迭代
- 单周期与多周期
  - ✓ 单周期：在一个周期内完成指令的所有操作
    - 周期宽度如何确定,能否在一个clk内完成?
  - ✓ 多周期：一个周期完成指令的一步
    - 与单周期的区别：功能部件复用，中间结果保存
      - 可以实现不定长指令周期，提高性能
    - 按当前周期产生响应的控制信号（状态机）
      - 何时刷新PC，何时访存，何时写寄存器
- 作业RISC-V：
  1. 4.1，4.7，C.2
  2. 分析MIPS三种类型指令(R/I/J)的多周期设计方案中每个周期所用到的功能部件。

思考

- 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?
  - ✓ 控制逻辑有哪些实现方式?
  - ✓ 单周期的指令和数据内存分开，ALU和加法器分开，原因是什么?
- 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，中间结果保存
  - ✓ 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?
- 每一类指令的指令周期各含多少个时钟周期?
- 比较FSM控制器实现方式的特点。

例子

问：有一段程序共有100条指令，其中Load指令为20%，Store指令为20%，R类的ALU指令为40%，Beq指令为10%，JUMP指令为10%。这段程序分别在单周期和多周期的CPU上实现（多周期参考MIPS的CPI），其中单周期CPU时钟为200MHz，多周期CPU时钟为800MHz(考虑到存储器、ALU实现复杂)，问在何种CPU上运行时快，快多少?

答：略。

Multicycle RTL（参考）

Step	R-Type	lw/sw	beq/bne	j
IF	IR = Mem[PC] PC = PC + 4			
ID	A = Reg[IR[25-21]] B = Reg[IR[20-16]] ALUOut = PC + (SE(IR[15-0]) << 2)			
EX	ALUOut = A op B	ALUOut = A +SE(IR[15-0]) 访存指令PC+offset	If (A==B) then PC = ALUOut	PC = PC[31-28] || (IR[25-0]<<2
MEM	Reg[IR[15-11]] = ALUOut	MDR=Mem[ALUOut] Mem[ALUOut] = B		
WB		Reg[IR[20-16]] = MDR		

A=rs1 B=rs2
B指令PC+offset

rd

跳转指令PC+offset

rd

联系方式

Acknowledgements:
This slides contains materials from following lectures:
- **Computer Architecture (NUDT)**
- **CS 152 CS 252 (UC Berkeley)**

Research Area：
- 基于分布式系统, GPU，FPGA的神经网络、图计算加速
- 人工智能和深度学习（寒武纪）芯片及智能计算机

Contact：
- 中国科学技术大学计算机学院
- 嵌入式系统实验室（西活北一楼）
- 高效能智能计算实验室 （中科大苏州研究院）

cswang@ustc.edu.cn
http://staff.ustc.edu.cn/~cswang