回顾第19次课

- **计算机性能**
  - **CPI+指令条数+时钟周期宽度**
- **CPU**
  - **执行部件（操作元件，存储元件）**
  - **控制部件（操作元件，存储元件）**

以下以RISC-V指令系统为例介绍非总线式CPU的设计。

设计处理器的步骤

ISA确定后，进行处理器设计的大致步骤

第一步：分析每条指令的功能，并用RTL(Register Transfer Language)来表示。

第二步：根据指令的功能给出所需的元件，并考虑如何将他们互连。

第三步：确定每个元件所需控制信号的取值。

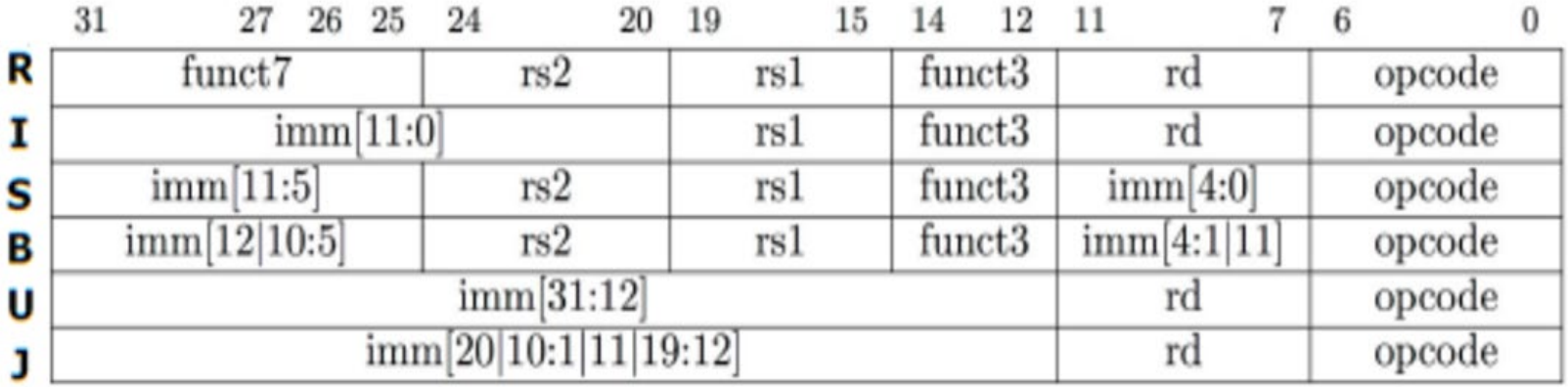
第四步：汇总所有指令所涉及到的控制信号，生成一张反映指令与控制信号之间关系的表。

第五步：根据表得到每个控制信号的逻辑表达式，据此设计控制器电路。

- **处理器设计涉及到数据通路的设计和控制器的设计**
- **数据通路中有两种元件**
  - **操作元件：由组合逻辑电路实现**
  - **存储（状态）元件：由时序逻辑电路实现**

RISC-V指令格式

- **所有指令都是32位宽，须按字地址对齐**
  字地址为4的倍数或2的倍数（RV32C压缩指令格式）！
- **有6种指令格式**



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

数据通路实现目标包括：

3条R-型指令：add rd, rs1, rs2、slt rd, rs1, rs2、sltu rd, rs1, rs2

2条I-型指令：ori rd, rs1, imm12、lw rd, imm12(rs1)

1条S-型指令：sw rs2, imm12(rs1)

1条B-型指令：beq rs1, rs2, imm12

1条U-型指令：lui rd, imm20

1条J-型指令：jal rd, imm20

具有很强的代表性：
算术/逻辑运算，取数/存数；
短立即数，长立即数；
条件转移，无条件转移；
带符号数判断大小，无符号数判断大小

目标指令功能描述

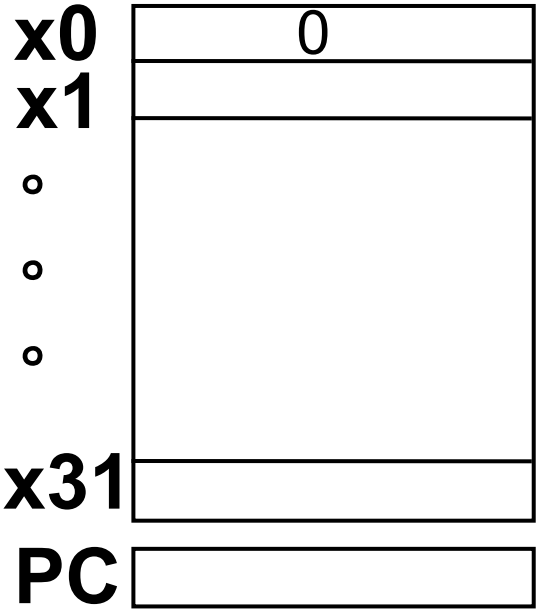
注意：每条指令的第一步都是取指令并PC加4，使PC指向下条指令（表中除第一条add指令外，其余指令都省略了对第一步的描述）

注意：I-型指令的立即数为符号扩展，即使是逻辑运算，立即数也是符号扩展！

指　令	功　能
add rd, rs1, rs2	M[PC]，PC←PC + 4 R[rd]←R[rs1] + R[rs2]
slt rd, rs1, rs2	if (R[rs1] < R[rs2]) R[rd]←1 else R[rd]←0
sltu rd, rs1, rs2	if (R[rs1] < R[rs2]) R[rd]←1 else R[rd]←0
ori rd, rs1, imm12	R[rd]←R[rs1] | SEXT(imm12)
lui rd, imm20	R[rd]←imm20||000H
lw rd, rs1, imm12	Addr←R[rs1] + SEXT(imm12) R[rd]←M[Addr]
sw rs1, rs2, imm12	Addr←R[rs1] + SEXT(imm12) M[Addr]←R[rs2]
beq rs1, rs2, imm12	Cond←R[rs1] − R[rs2] if (Cond eq 0) PC←PC+(SEXT(imm12)×2)
jal rd, imm20	R[rd]←PC + 4 PC←PC + (SEXT(imm20)×2)

RV32I指令中寄存器数据和存储器数据的指定

- 寄存器数据指定：
  - 31 x 32-bit GPRs （x0 = 0）
  - 寄存器编号占5 bit
  - PC: 程序计数器（无需编号）
  - 寄存器功能和2种汇编表示方式
- 存储器数据指定
  - 32- bit machine --> 可访问空间: 2^{32}bytes
  - Little Endian(小端方式)



- 只能通过Load/Store指令访问存储器数据
- 数据地址通过一个32位寄存器内容加12位偏移量得到
- 12位偏移量是带符号整数，采用符号扩展
- lw rd, imm12(rs1)， sw rs2, imm12(rs1)

- 数据不要求按边界对齐，执行到一条不按边界对齐的访存指令时，硬件抛出异常，由软件进行处理

SKIP

RV32I寄存器的功能定义和两种汇编表示

BACK to last

寄存器	ABI 名	功能描述	被调用过程保存?
x0	zero	硬编码 0	—
x1	ra	返回地址	否
x2	sp	栈指针	是
x3	gp	全局指针	—
x4	tp	线程指针	—
x5	t0	临时寄存器	否
x6~x7	t1~t2	临时寄存器	否
x8	s0/fp	保存寄存器/帧指针	是
x9	s1	保存寄存器	是
x10~x11	a0~a1	过程参数/返回值	否
x12~x17	a2~a7	过程参数	否
x18~x27	s2~s11	保存寄存器	是
x28~x31	t3~t6	临时寄存器	否

Registers are referenced either by number—x0, … x31, or by name —zero, ra, s1… t0.

功能设计需求的分析

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

除R-型外，其他5类都带有立即数

——立即数扩展器

核心运算类功能的实现 ——ALU

根据PC取指令和PC+4 ——取指令部件

指令的RTL最终实现 ——完整数据通路

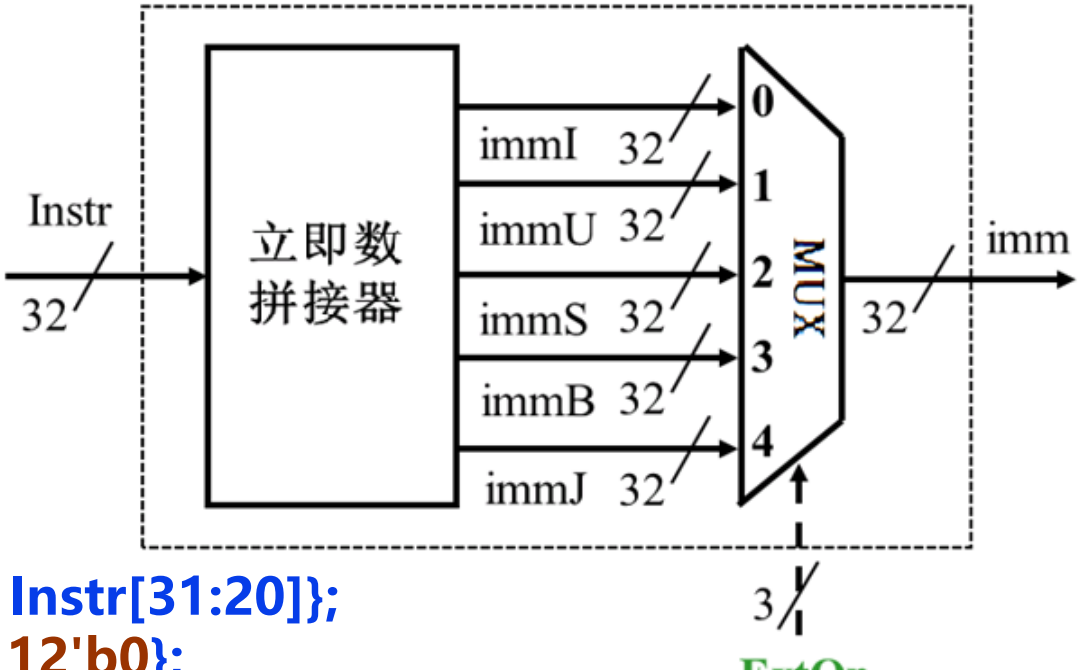
add rd, rs1, rs2	M[PC]，PC←PC + 4 R[rd]←R[rs1] + R[rs2]

扩展器部件的设计

除R-型外，其他5类都带有立即数

立即数拼接器：根据指令格式对指令中的立即数进行拼接和扩展，形成32位立即数

ExtOp：控制选择和输入指令相匹配的立即操作数输出




```
assign immI = {20{Instr[31]}, Instr[31:20]};
assign immU = {Instr[31:12], 12'b0};
assign immS = {20{Instr[31]}, Instr[31:25], Instr[11:7]};
assign immB = {20{Instr[31]}, Instr[7], Instr[30:25], Instr[11:8], 1'b0};
assign immJ = {12{Instr[31]}, Instr[19:12], Instr[20], Instr[30:21], 1'b0};
```

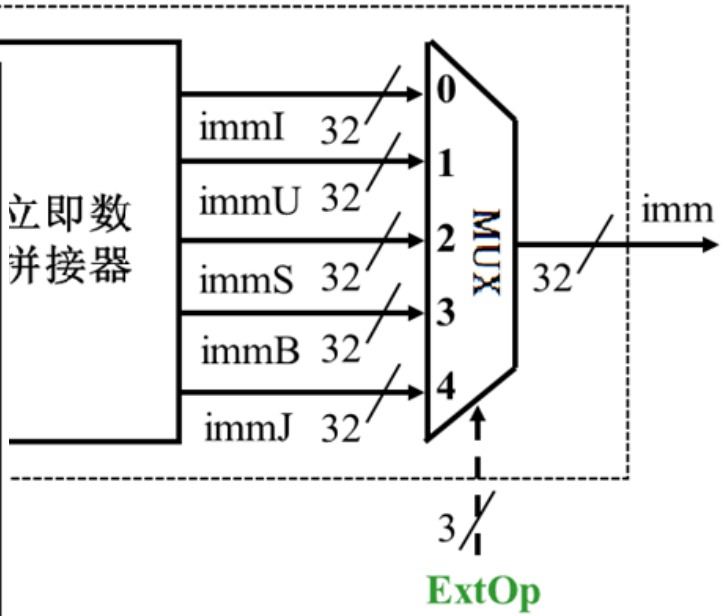
由于所有指令格式是预知的，每种指令中的立即数的位置是固定的，因此拼接器拿到一条32位指令之后，就可以同时完成五种拼接（把这个32位分别当成IUSBJ类指令）

但只有一个是正确的，其它都是错的，最终会按ExtOP输出那个正确的。

——不用等待其它信号，在译码结果（ ExtOP）还没出来之前，就可以把五种立即数拼好。

扩展器部件的设计

指　令	立即数编码类型	ExtOp<2:0>
add rd, rs1, rs2	无立即数	× × ×
slt rd, rs1, rs2		
sltu rd, rs1, rs2		
ori rd, rs1, imm12	I-型立即数（immI）	0 0 0
lui rd, imm20	U-型立即数（immU）	0 0 1
lw rd, rs1, imm12	I-型立即数（immI）	0 0 0
sw rs1, rs2, imm12	S-型立即数（immS）	0 1 0
beq rs1, rs2, imm12	B-型立即数（immB）	0 1 1
jal rd, imm20	J-型立即数（immJ）	1 0 0



问题：各指令的ExtOp取值如何？占几位？

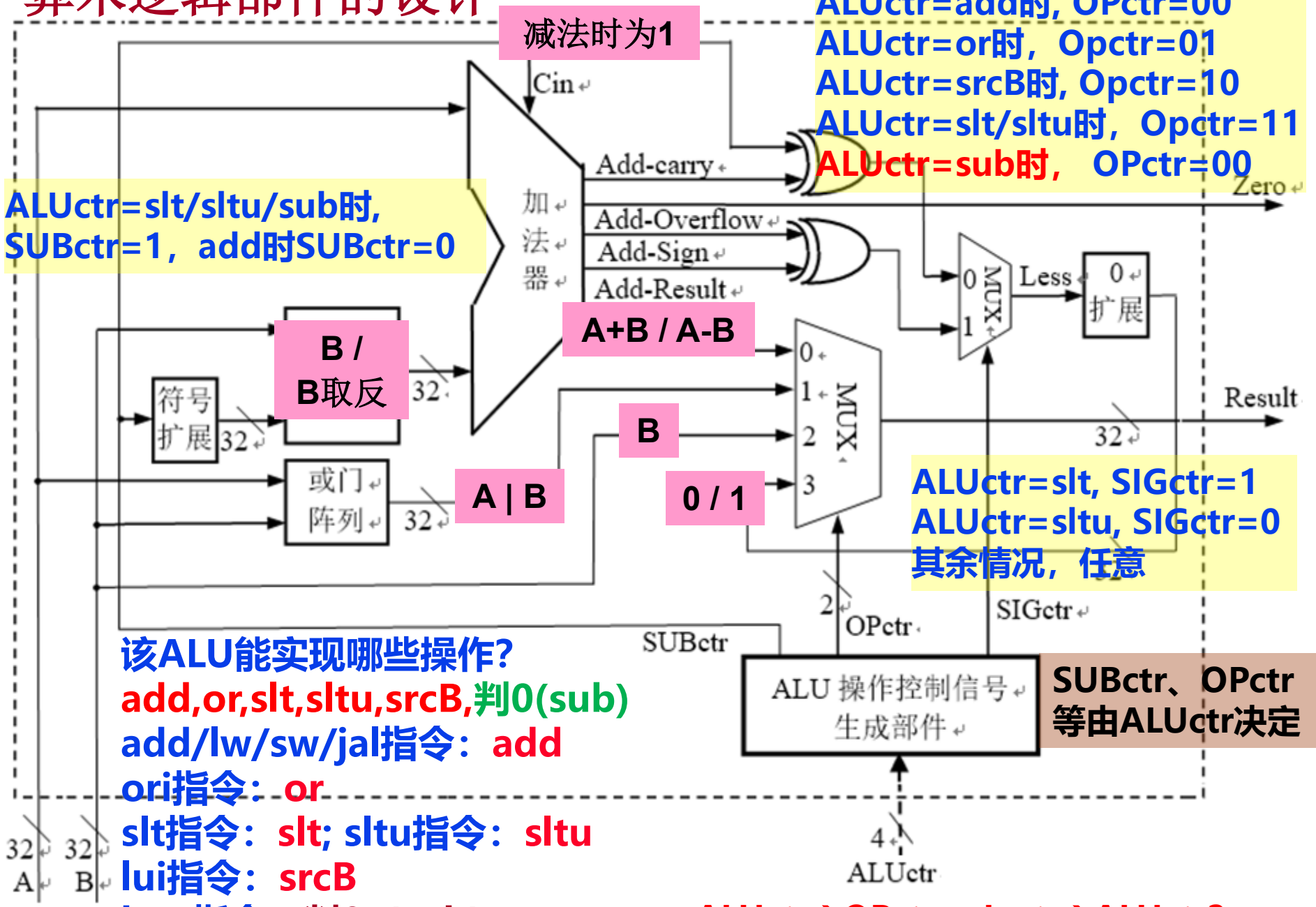
有5种情况，至少占3位！

如何根据指令编码Instr得到ExtOp？

通过指令译码器得到！

将在控制器设计部分介绍。

算术逻辑部件的设计



ALUctr=add时, OPctr=00
ALUctr=or时，Opctr=01
ALUctr=srcB时, Opctr=10
ALUctr=slt/sltu时，Opctr=11
ALUctr=sub时， OPctr=00

ALUctr=slt/sltu/sub时,
SUBctr=1，add时SUBctr=0

ALUctr=slt, SIGctr=1
ALUctr=sltu, SIGctr=0
其余情况，任意

SUBctr、OPctr
等由ALUctr决定

该ALU能实现哪些操作?
add,or,slt,sltu,srcB,判0(sub)
add/lw/sw/jal指令：add
ori指令：or
slt指令：slt; sltu指令：sltu
lui指令：srcB
beq指令：判0（sub）

ALUctr→OPctr, Instr→ALUctr?

算术逻辑部件的设计

(1) ALUctr如何决定OPctr等?

看操作和SUBctr、OPctr等的关系

ALUctr=slt/sltu/sub时, SUBctr=1，add时SUBctr=0

ALUctr=slt, SIGctr=1
ALUctr=sltu, SIGctr=0
其余情况，任意

ALUctr=add时, OPctr=00
ALUctr=or时，Opctr=01
ALUctr=srcB时, Opctr=10
ALUctr=slt/sltu时，Opctr=11
ALUctr=sub时， OPctr=00

ALUctr<3:0>	操作类型	SUBctr	SIGctr	OPctr<1:0>
0 0 0 0	add	0	×	0 0
0 0 0 1	（未用）			
0 0 1 0	slt	1	1	1 1
0 0 1 1	sltu	1	0	1 1
0 1 0 0	（未用）			
0 1 0 1	（未用）			
0 1 1 0	or	×	×	0 1
0 1 1 1	（未用）			
1 0 0 0	sub	1	×	0 0
其余	（未用）			
1 1 1 1	srcB	×	×	1 0

```
SUBctr = (~ALUctr<3> & ~ALUctr<2> & ALUctr<1>) | ALUctr<3>
SIGctr = ~ALUctr<0>
OPctr<1> = (~ALUctr<3> & ~ALUctr<2> & ALUctr<1>) |                    (slt, sltu)
           (ALUctr<3> & ALUctr<2> & ALUctr<1> & ALUctr<0>)
OPctr<0> = (~ALUctr<3> & ~ALUctr<2> & ALUctr<1>) |
           (~ALUctr<3> & ALUctr<2> & ALUctr<1> & ~ALUctr<0>)          (or)
```

(2) 指令Instr如何决定ALUctr?

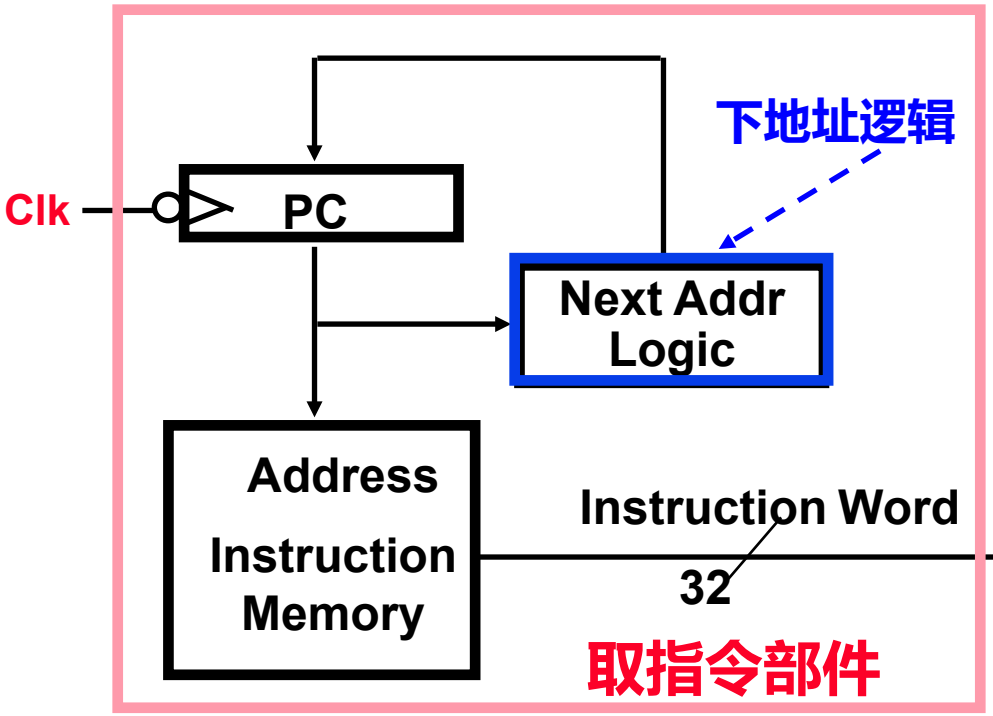
通过指令译码器得到!

在控制器设计部分介绍

取指令部件(Instruction Fetch Unit)的设计

- 每条指令都有的公共操作：
  - 取指令： M[PC]
  - 更新PC：PC ← PC + 4

转移（Branch and Jump）时，PC内容再次被更新为 “转移目标地址”



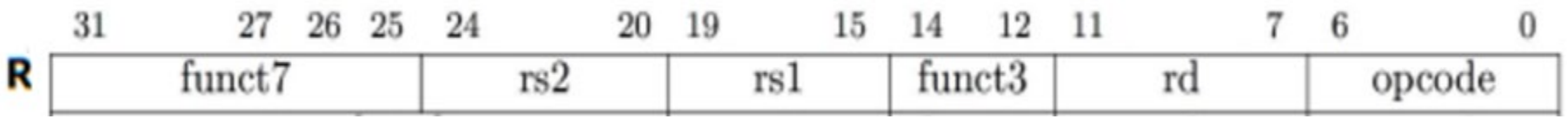
先取指令，再改PC的值（具体实现时，可以并行）

绝不能先改PC的值，再取指令

取指后，各指令功能不同，数据通路中信息流动过程也不同

下面分别对每条指令进行相应数据通路的设计

R-型指令的数据通路



R-型指令功能：

R[rd] ← R[rs1] op R[rs2]

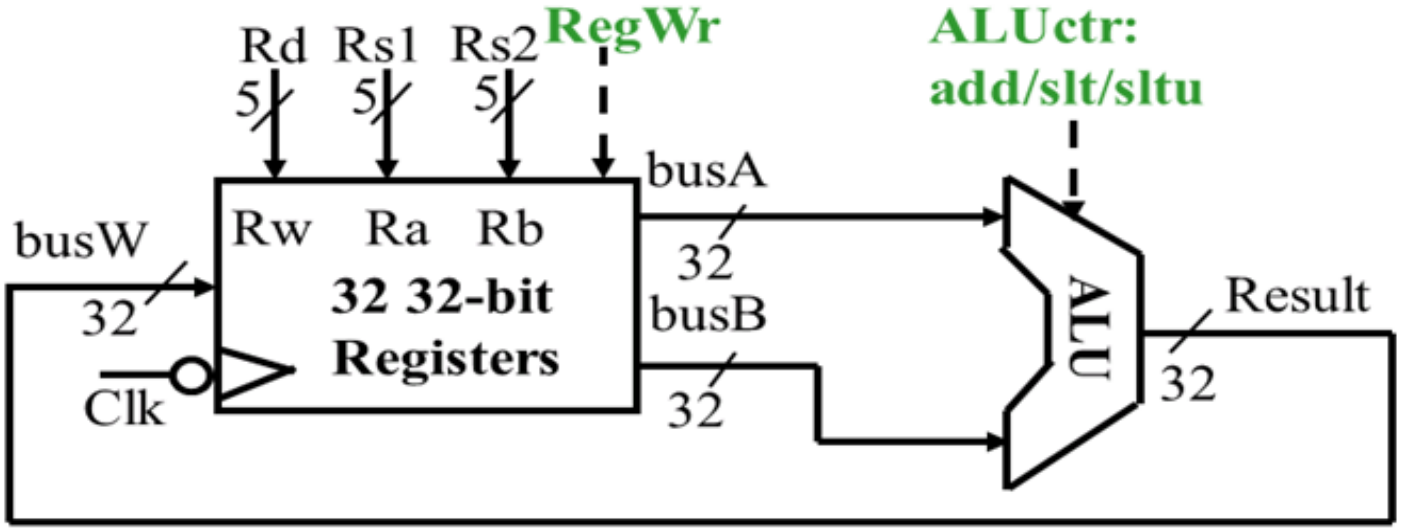
根据PC读取指令

以下相应字段送控制器

- **操作码opcode**
- **功能码funct7和funct3**

以下相应字段送寄存器堆

- **rs1送Rs1输入端**
- **rs2送Rs2输入端**
- **rd送Rd输入端**

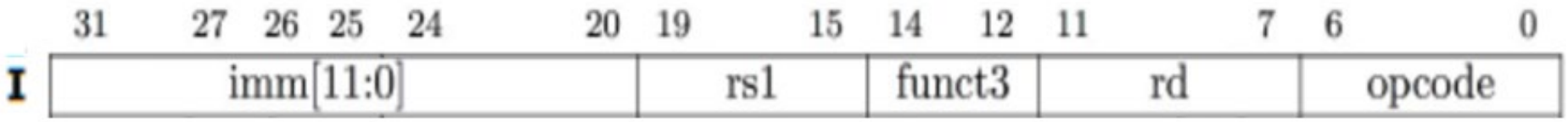


三条R-型指令：add、slt和sltu，分别对应ALU的三种操作，

即ALUctr为add、slt和sltu（比较结果为0（>=）或1（<））。

这三条指令都要写结果，故三条指令对应的控制信号RegWr=1

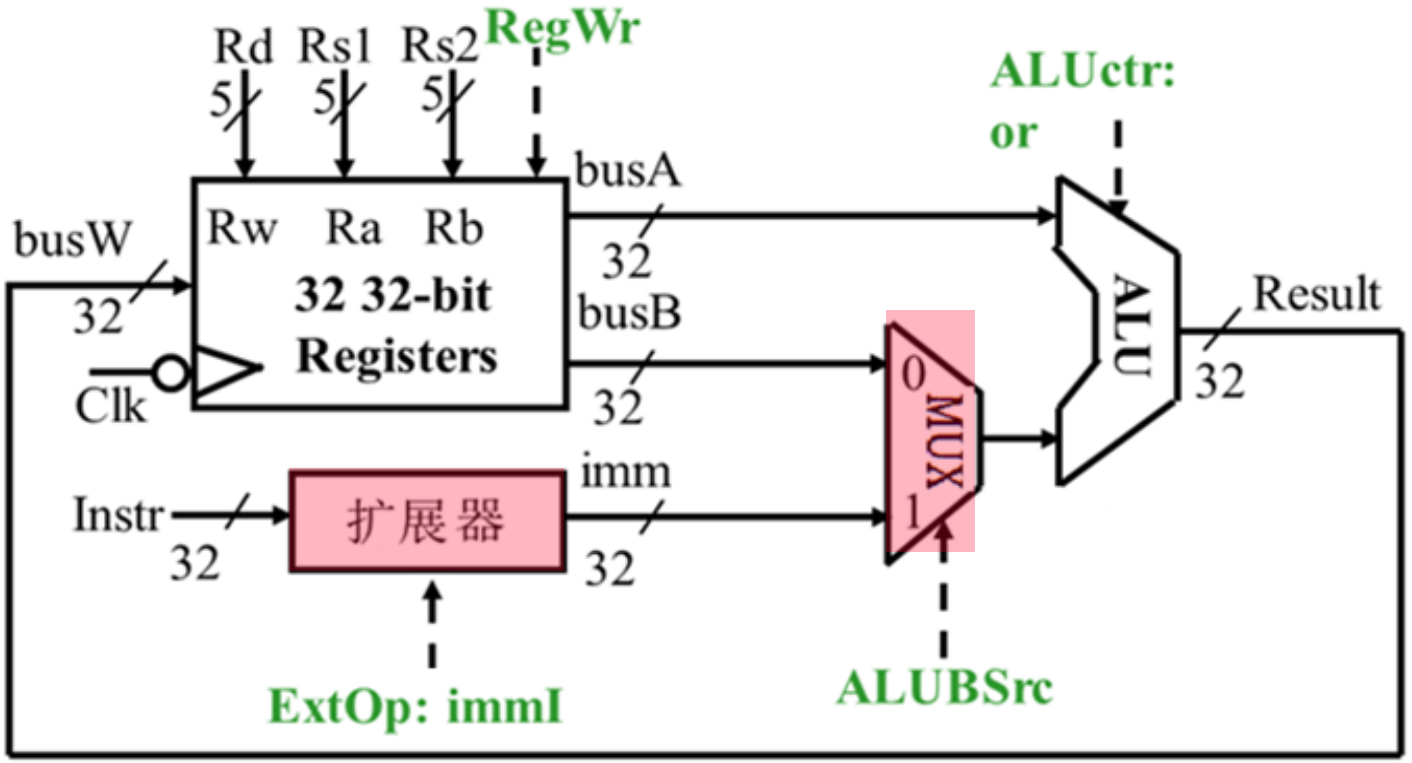
I-型运算指令ori的数据通路



I-型指令 ori 功能：R[rd] ← R[rs1] or SEXT(imm12)

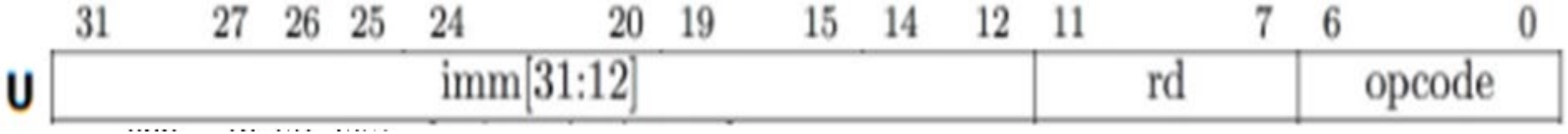
多了一个扩展器和多路选择器MUX

新增的控制信号在执行R型指令时，也需要给出正确的取值！下同

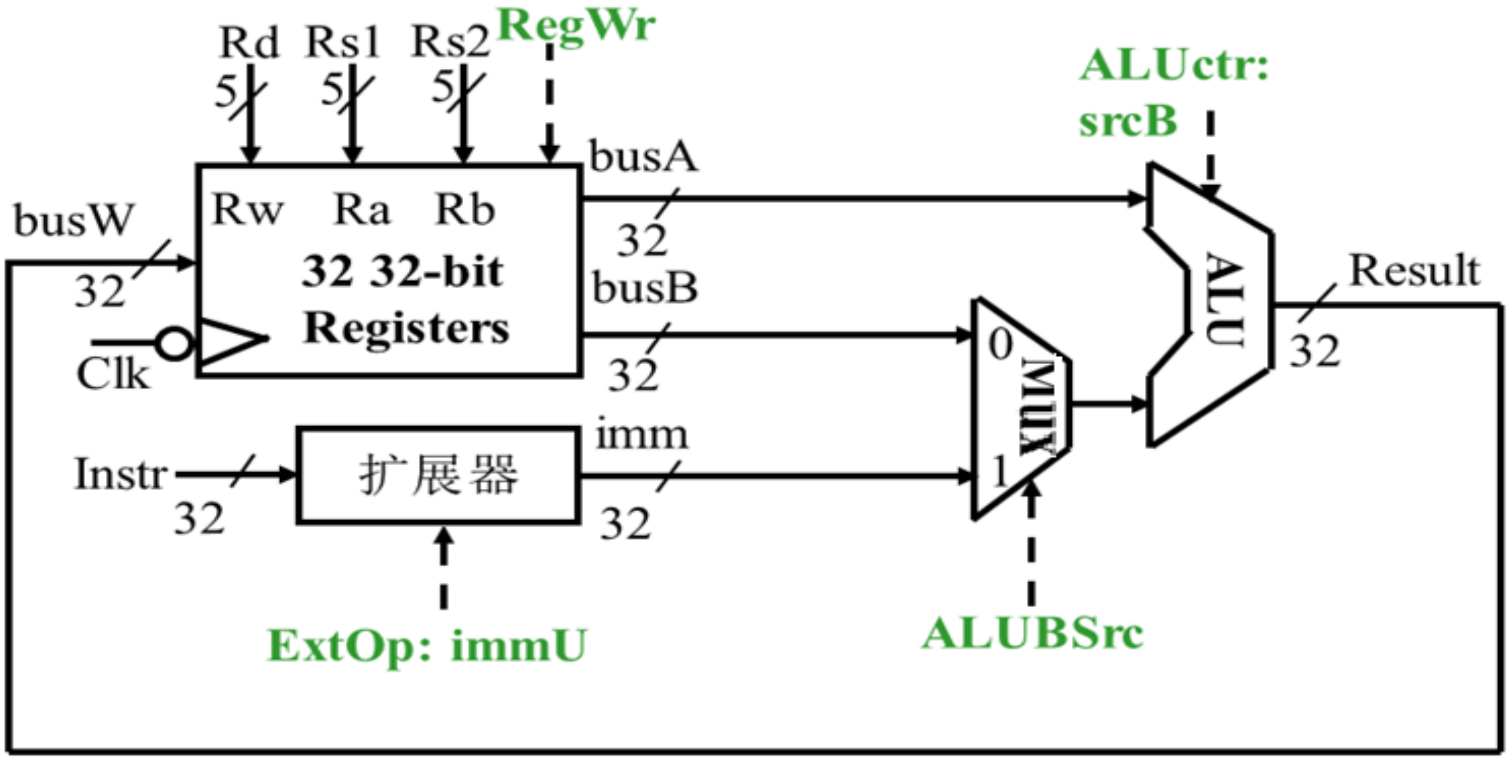


9条目标指令中，ori指令为I-型，其ALUctr为or。第二操作数为I-型立即数 immI ，ExtOp取值为000，ALUBSrc=1，RegWr=1

U-型指令的数据通路



U-型指令lui的功能：R[rd]← imm20||000H 即直接将扩展器结果输出

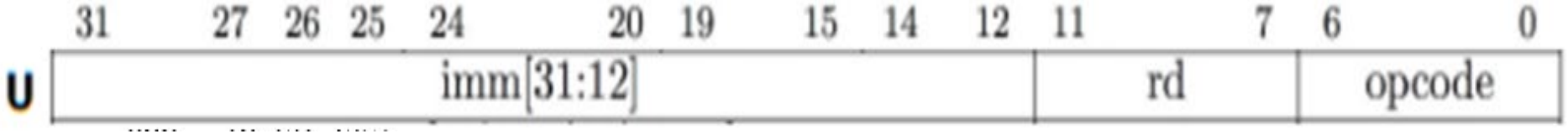


思考：

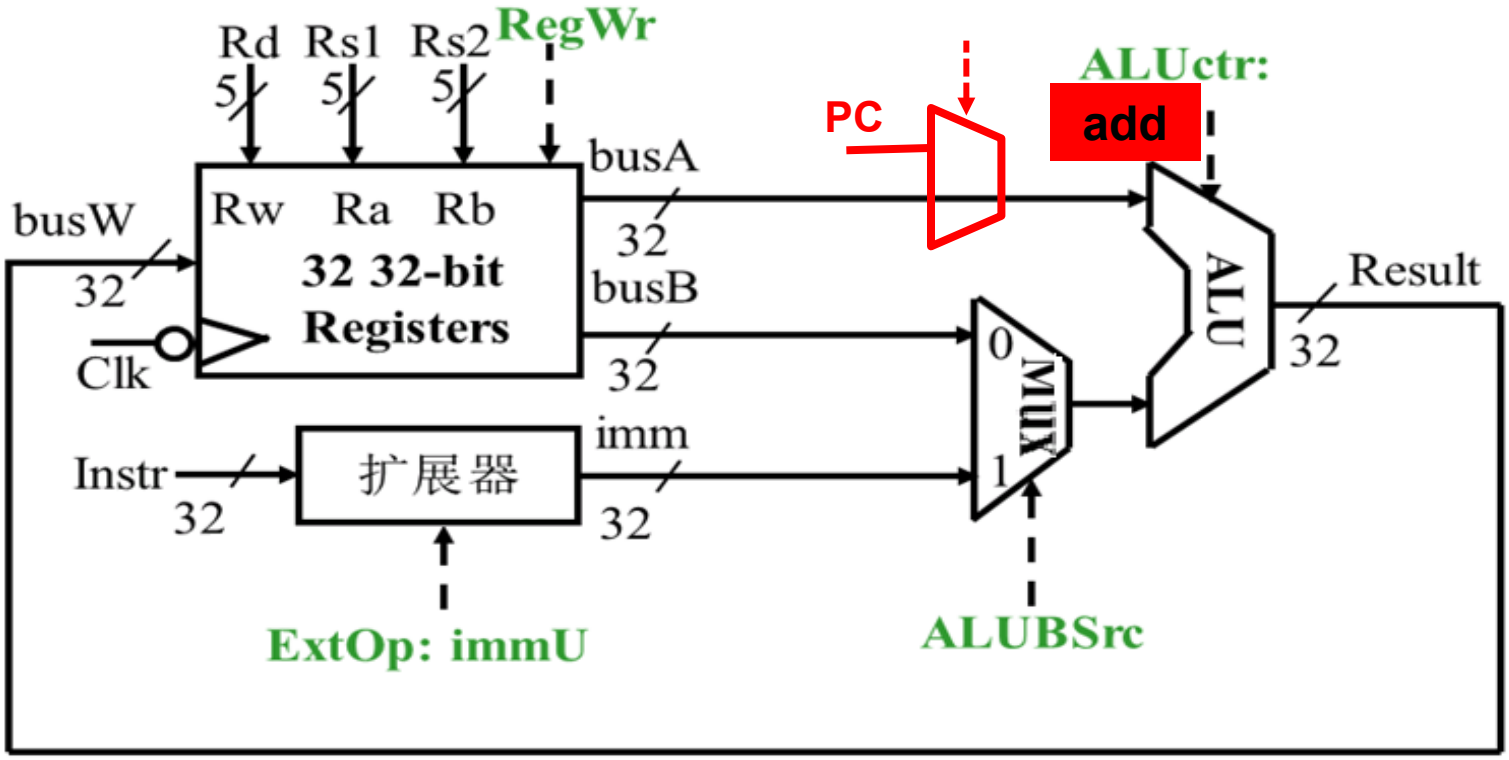
auipc的数据通路?

9条目标指令中，lui指令为U-型，其ALUctr为srcB 。第二操作数为U-型立即数 immU ，ExtOp取值为001，ALUBSrc=1，RegWr=1

U-型指令的数据通路（auipc)



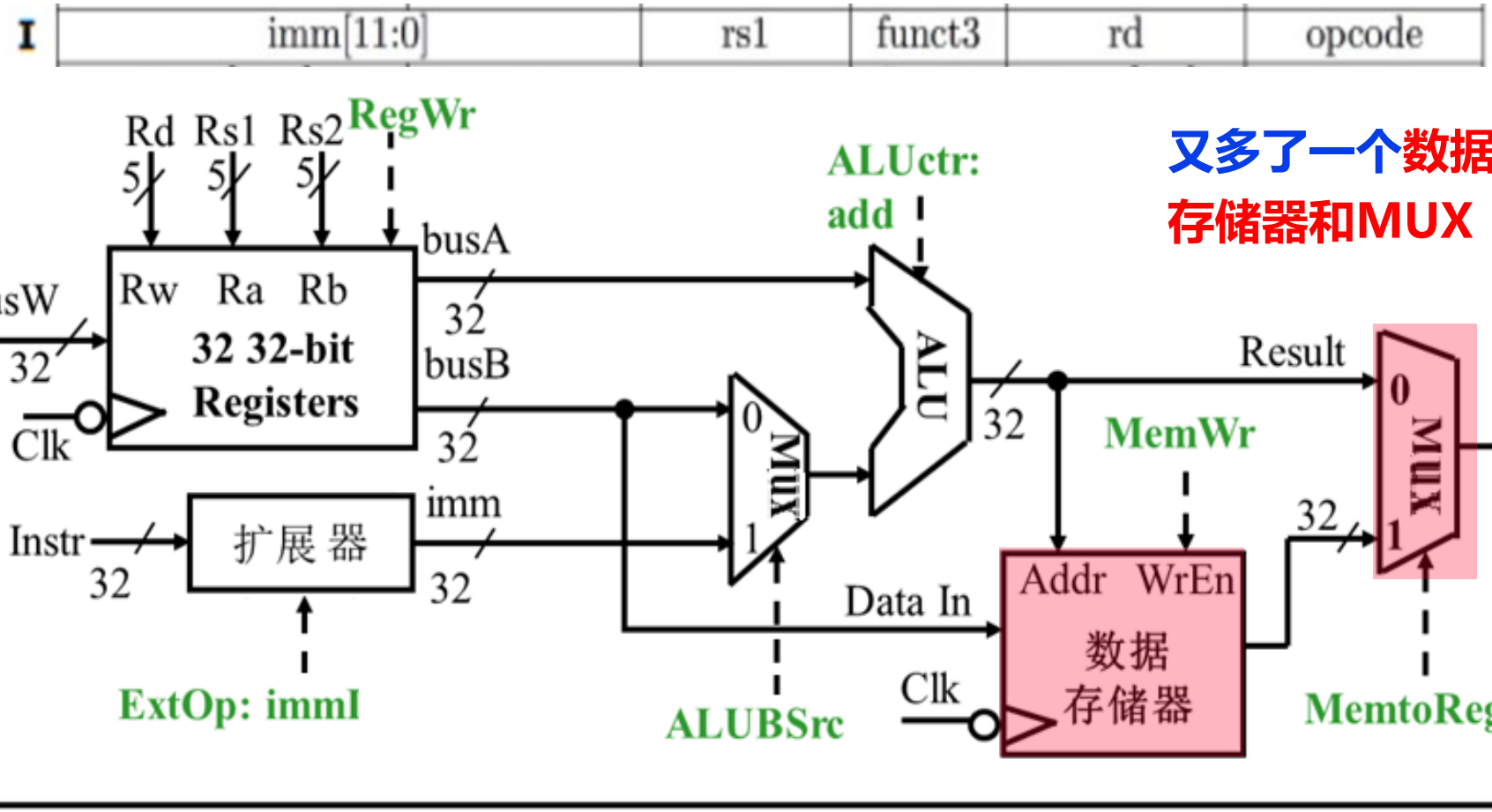
指令auipc的功能：R[rd]← PC+imm20||000H



auipc指令为U-型，其ALUctr为add。第二操作数为U-型立即数 immU，
ExtOp取值为001，ALUBSrc=1，RegWr=1，...

Load指令的数据通路

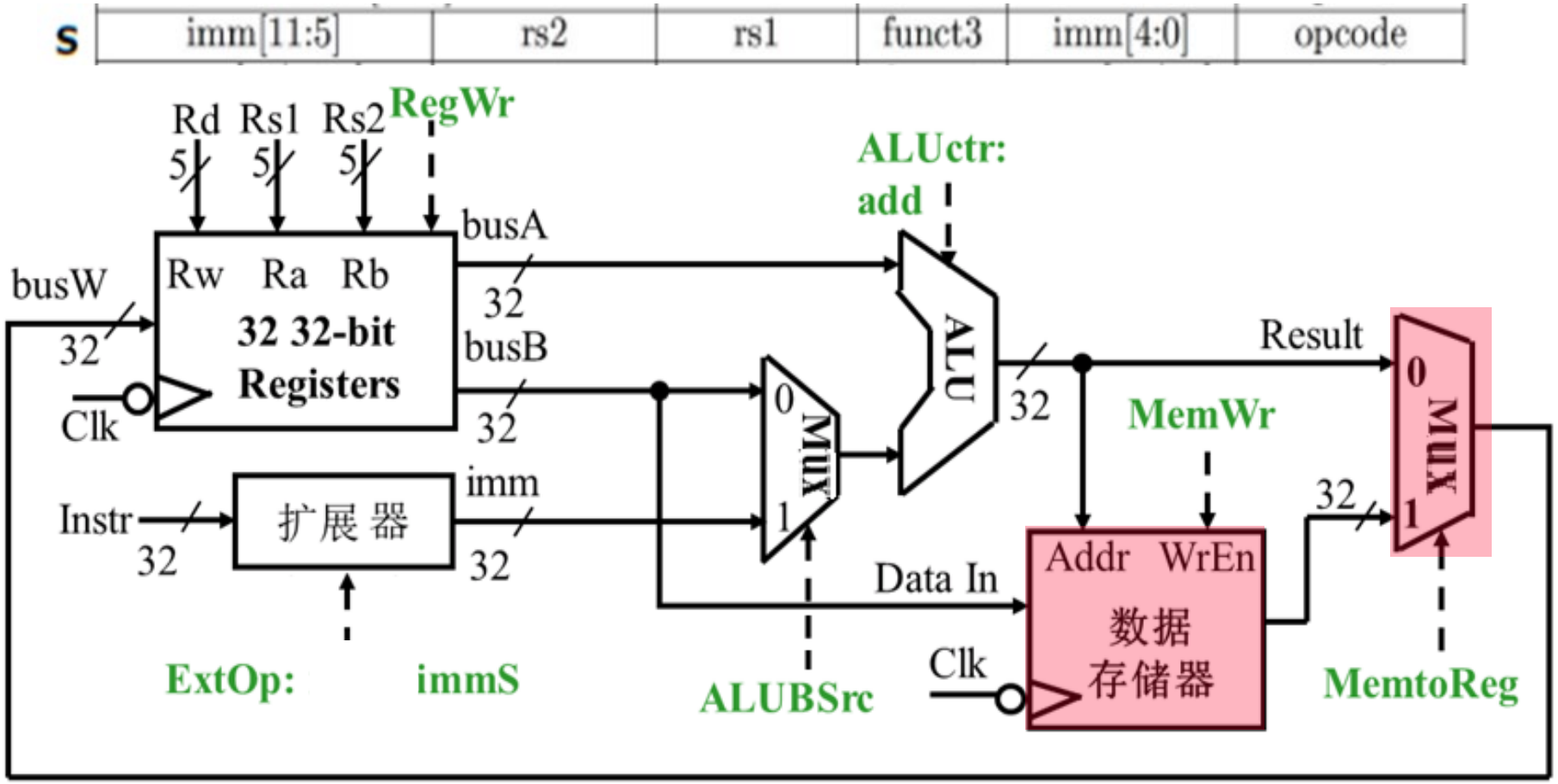
I-型的lw指令的功能：R[rd]← M[R[rs1]+SEXT(imm12)]



ALUctr是add，第二操作数为immI，故ExtOp为000；ALUBSrc=1；MemWr为0；MemtoReg为1；RegWr为1

Store指令的数据通路

S-型的sw指令的功能：M[R[rs1]+SEXT(imm12)]←R[rs2]

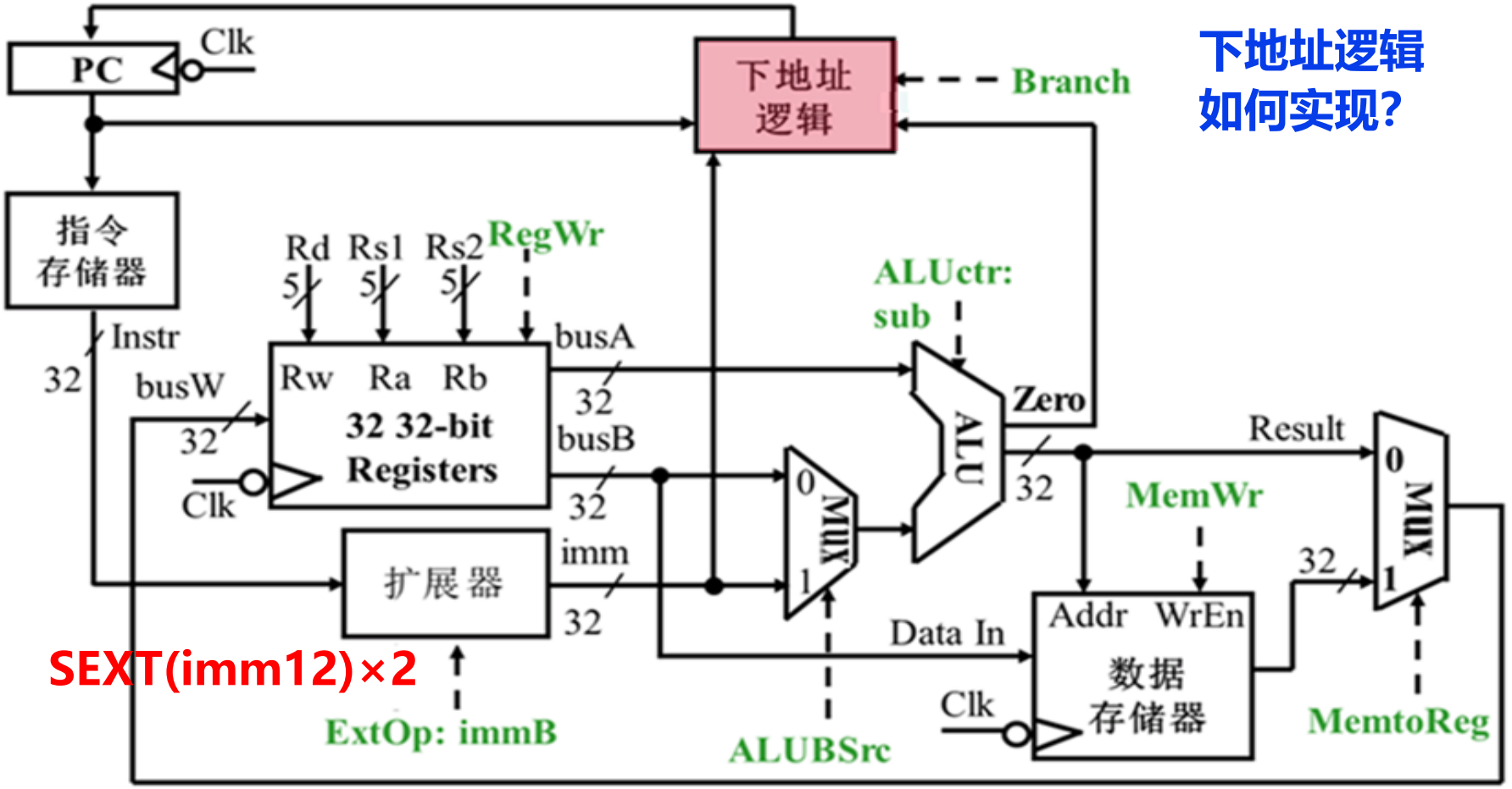


ALUctr是add，第二操作数为immS，故ExtOp为010；ALUBSrc=1；MemWr为1；MemtoReg为x（x表示任意）；RegWr为0

B-型指令的数据通路

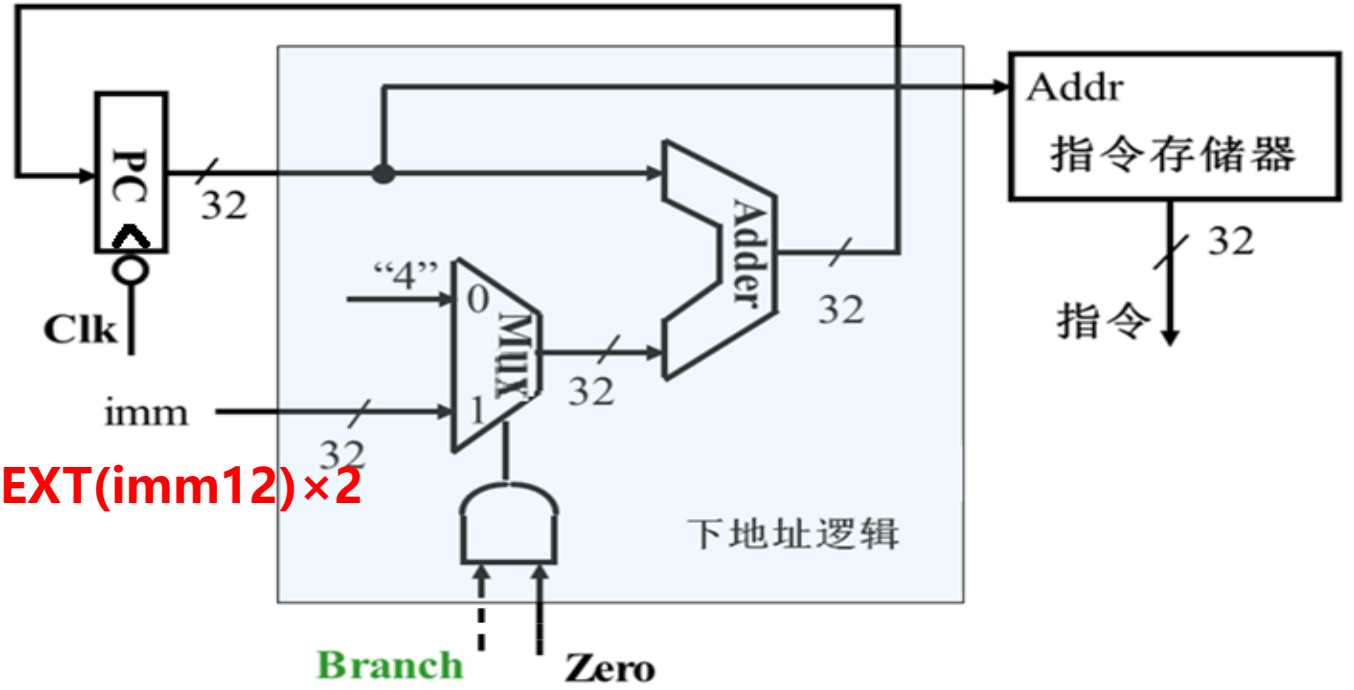
B-型的beq指令的功能是：

if (R[rs1]=R[rs2]) PC←PC+(SEXT(imm12)×2) else PC←PC+4



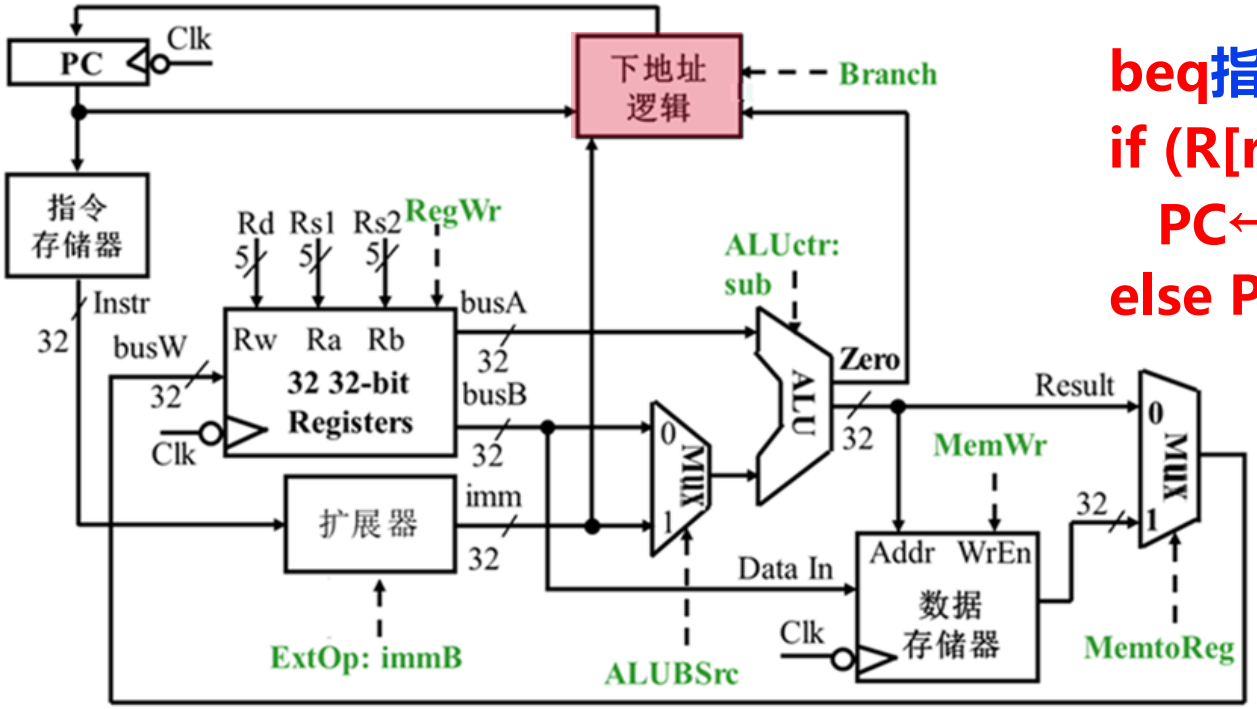
ALUctr是sub，立即数为immB，故ExtOp=011；ALUBSrc=0；
MemWr=0；MemtoReg=x；RegWr=0；Branch=1

B-型指令的数



SEXT(imm12)×2

下地址逻辑
如何实现?



beq指令的功能是:

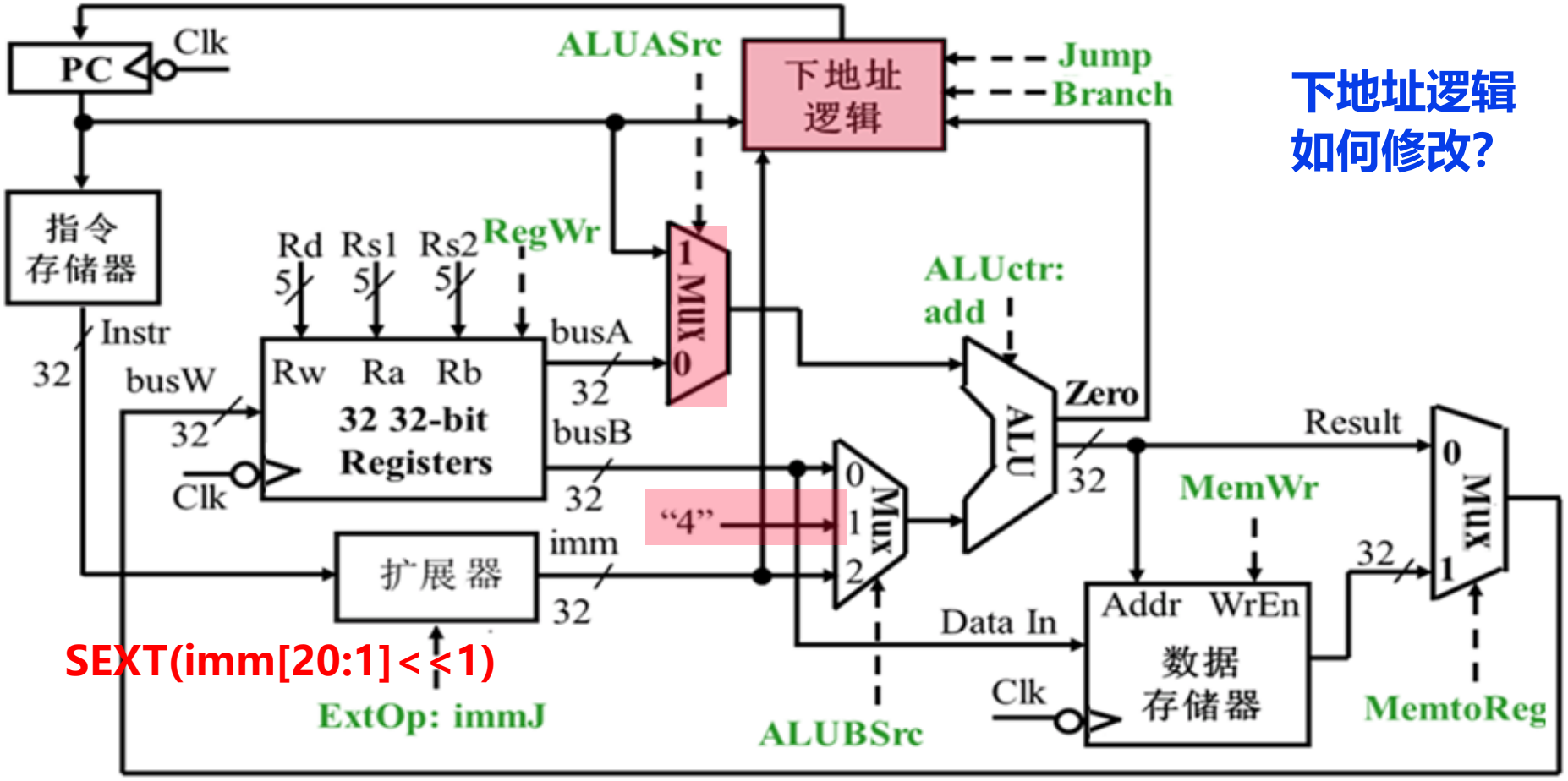
if (R[rs1]=R[rs2])
PC←PC+(SEXT(imm12)×2)
else PC←PC+4

J-型指令的数据通路

比前面的数据通路多了个MUX和“4”

jal是唯一的一条J-型指令，其功能是：

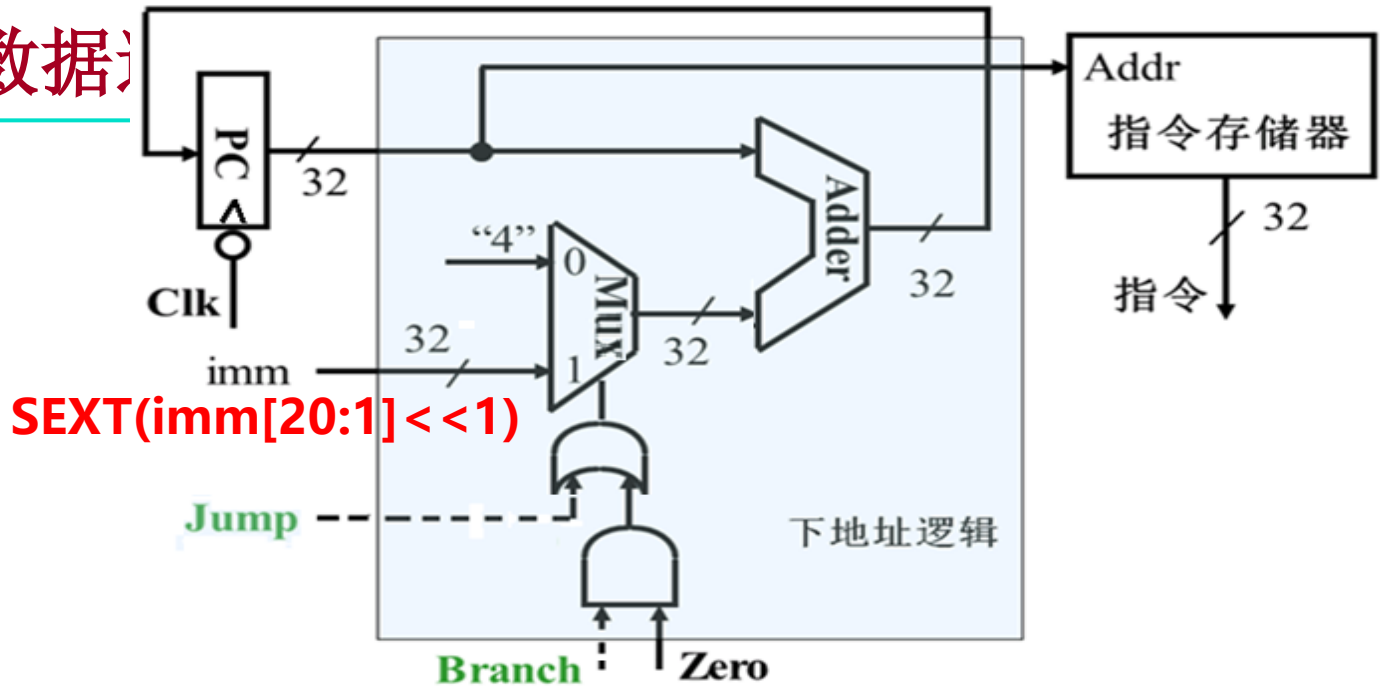
PC←PC+SEXT(imm[20:1]<<1)；R[rd]←PC+4



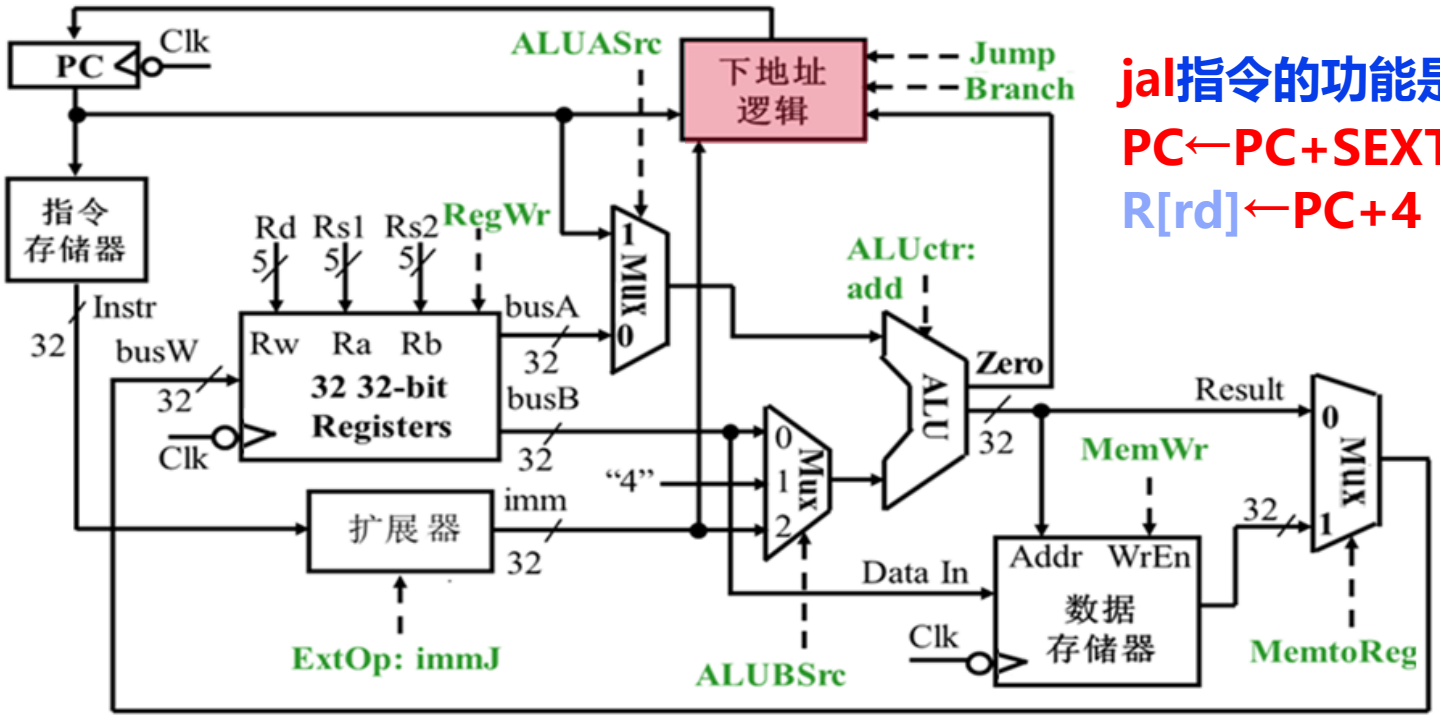
ALUctr是add，立即数为immJ，故ExtOp=100； ALUASrc=1； ALUBSrc=01；

MemWr=0；MemtoReg=0；RegWr=1；Branch=0；Jump=1

J-型指令的数据

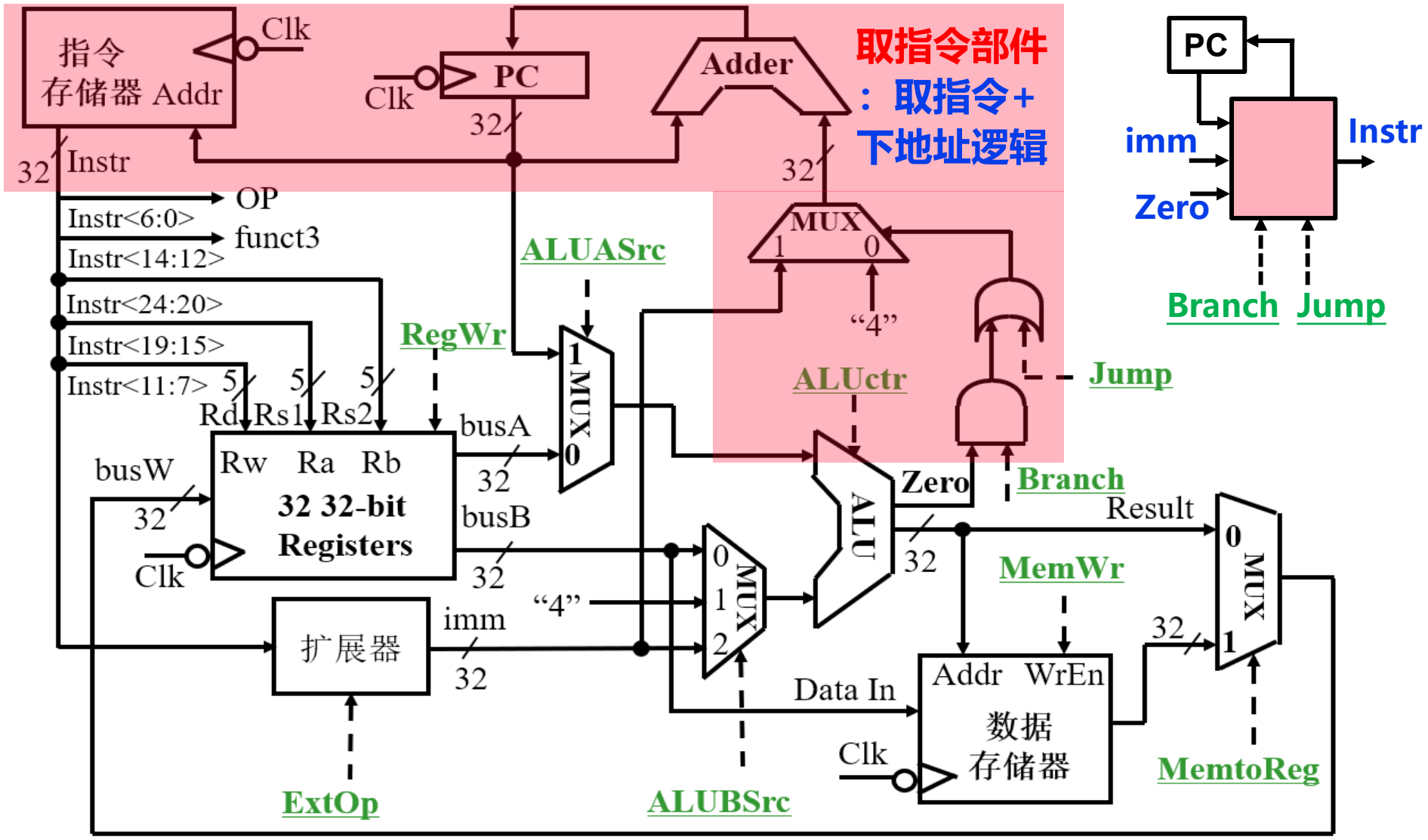


下地址逻辑
如何修改?



jal指令的功能是:
PC←PC+SEXT(imm[20:1]<<1);
R[rd]←PC+4

一个完整的单周期数据通路



指令执行结果总是在下个时钟到来时开始保存在 寄存器 或 存储器 或 PC 中!

下一讲考虑：如何产生控制信号！ （控制器的设计内容）

第二讲小结

- CPU设计直接决定了时钟周期宽度和CPI，所以对计算机性能非常重要!
- CPU主要由数据通路和控制器组成
  - 数据通路：实现指令集中所有指令的操作功能
  - 控制器：控制数据通路中各部件进行正确操作
- 数据通路中包含两种元件
  - 操作元件（组合电路）：ALU、MUX、扩展器、Adder、Reg/Mem Read等
  - 状态 / 存储元件（时序电路）：PC、Reg/Mem Write
- 数据通路的定时
  - 数据通路中的操作元件没有存储功能，其操作结果必须写到存储元件中
  - 在时钟到达后clk-to-Q时存储元件开始更新状态
- RV32I指令集的一个子集作为CPU的实现目标
  - 公共操作：取指令和PC+4
  - 下址计算：PC，三路选择：顺序、Branch（结合标志Zero）、Jump
  - R-型：ALU两个操作数来自rs1和rs2，结果写到rd
  - 访存：符号扩展，数据在rt和主存单元中交换
  - 立即数：不同方式的扩展操作数imm送到ALU的一个输入端

第8章 中央处理器（一）

第一讲 中央处理器概述

第二讲 单周期数据通路的设计

第三讲 单周期控制器的设计

第四讲 多周期处理器的设计

第五讲 流水线处理器设计

第六讲 流水线冒险及其处理

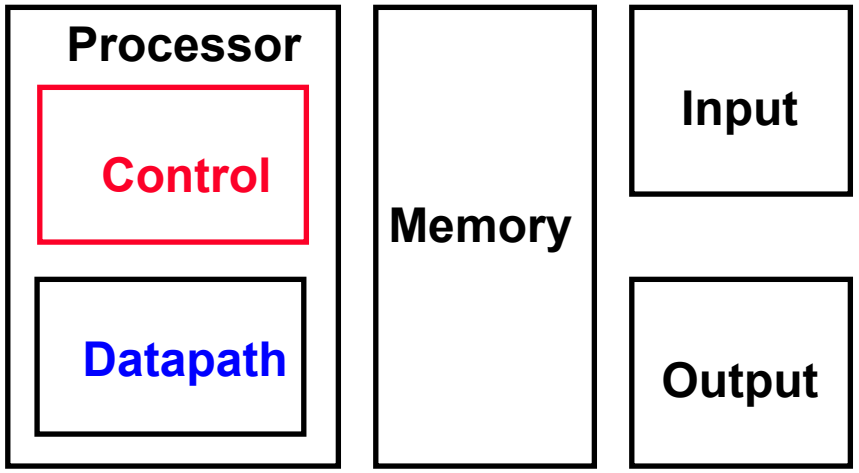
第七讲 高级流水线技术

第三讲 单周期控制器的设计

主 要 内 容

- 考察每条指令在数据通路中的执行过程和涉及到的控制信号的取值
  - 公共操作：取指令和计算下址PC
  - R-型指令（add / slt / sltu）
  - I-型运算类指令 （ori）
  - U-型指令（lui）
  - 访存指令（lw / sw）
  - B-型指令 （beq）
  - J-型指令 （jal）
- 汇总各指令的控制信号取值
- 设计主控制单元

下一个目标：设计单周期数据通路的控制器



设计方法：

1） 根据每条指令的功能，分析控制信号的取值，并在表中列出。

2） 根据列出的指令和控制信号的关系，写出每个控制信号的逻辑表达式。

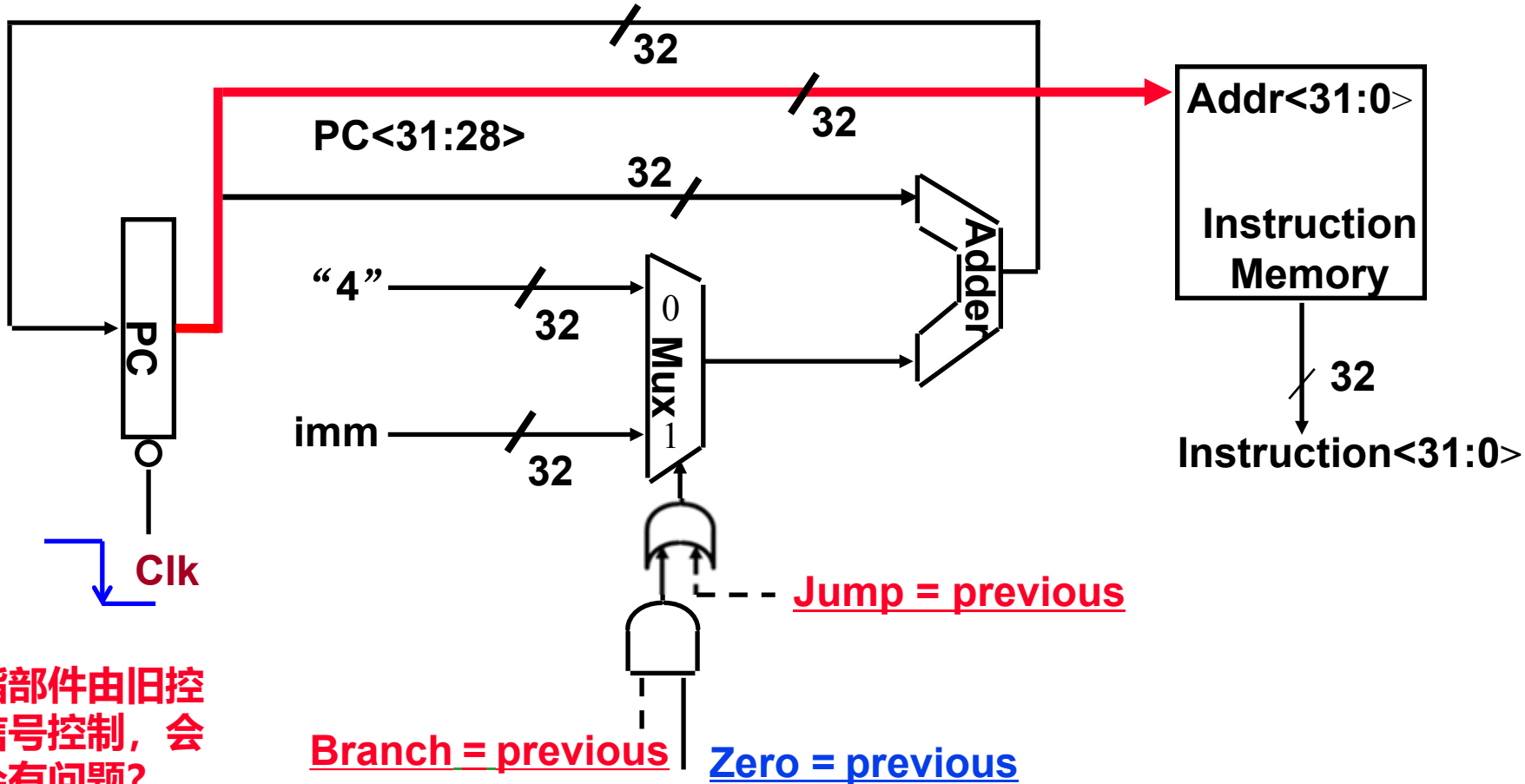
指令开始时取指部件中的动作

取指令: Instruction ← M[PC]

- 所有指令都相同

新指令还没有取出译码，所以控制信号的值还是原来指令的旧值。

新指令还没有执行，所以标志也为旧值。



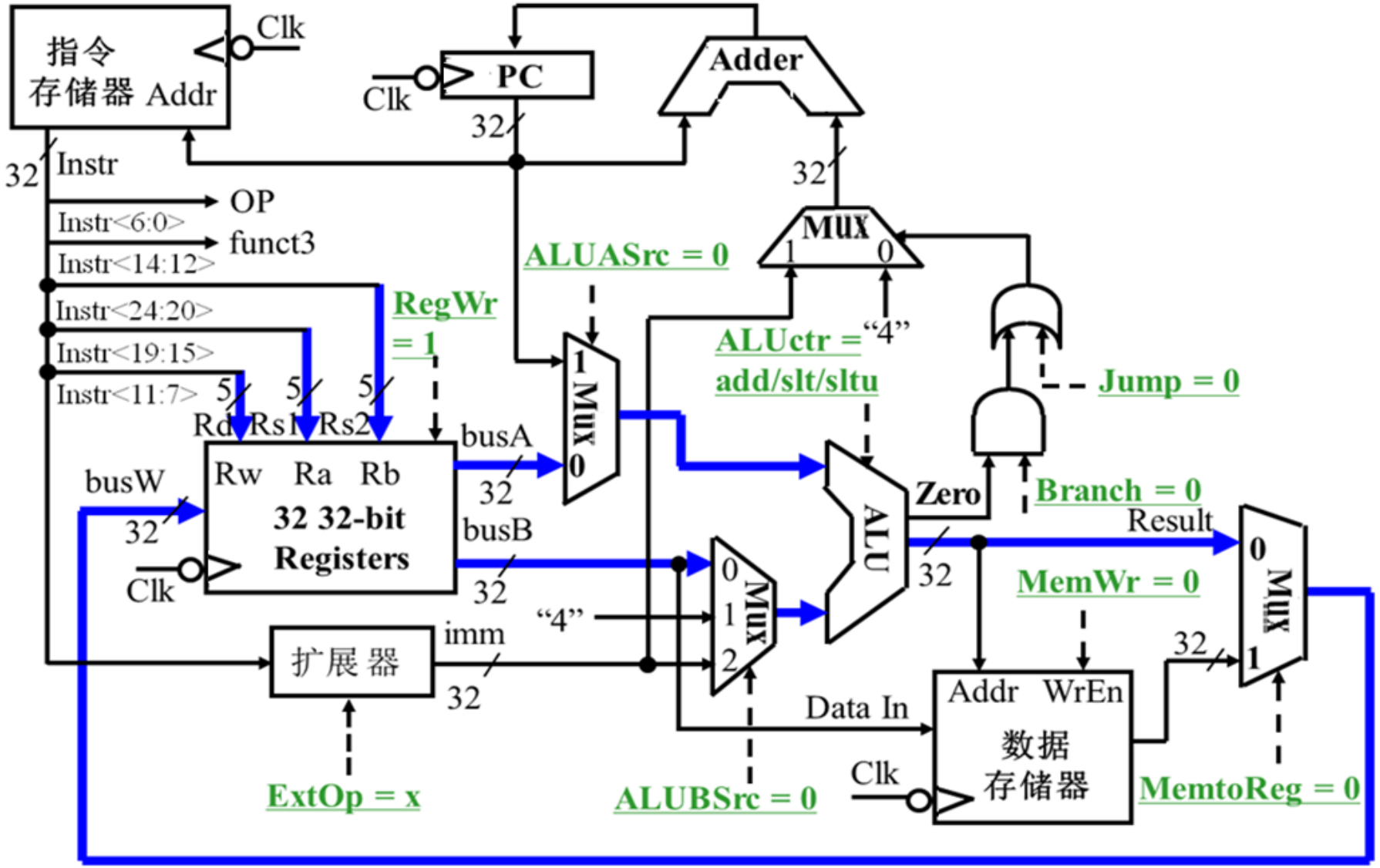
取指部件由旧控制信号控制，会不会有问题?

没有问题！Why?

因为在下个Clk到来之前PC输入端的值不会写入，只要保证下个Clk来之前能产生正确的PC即可！

指令译码后R-型指令操作过程

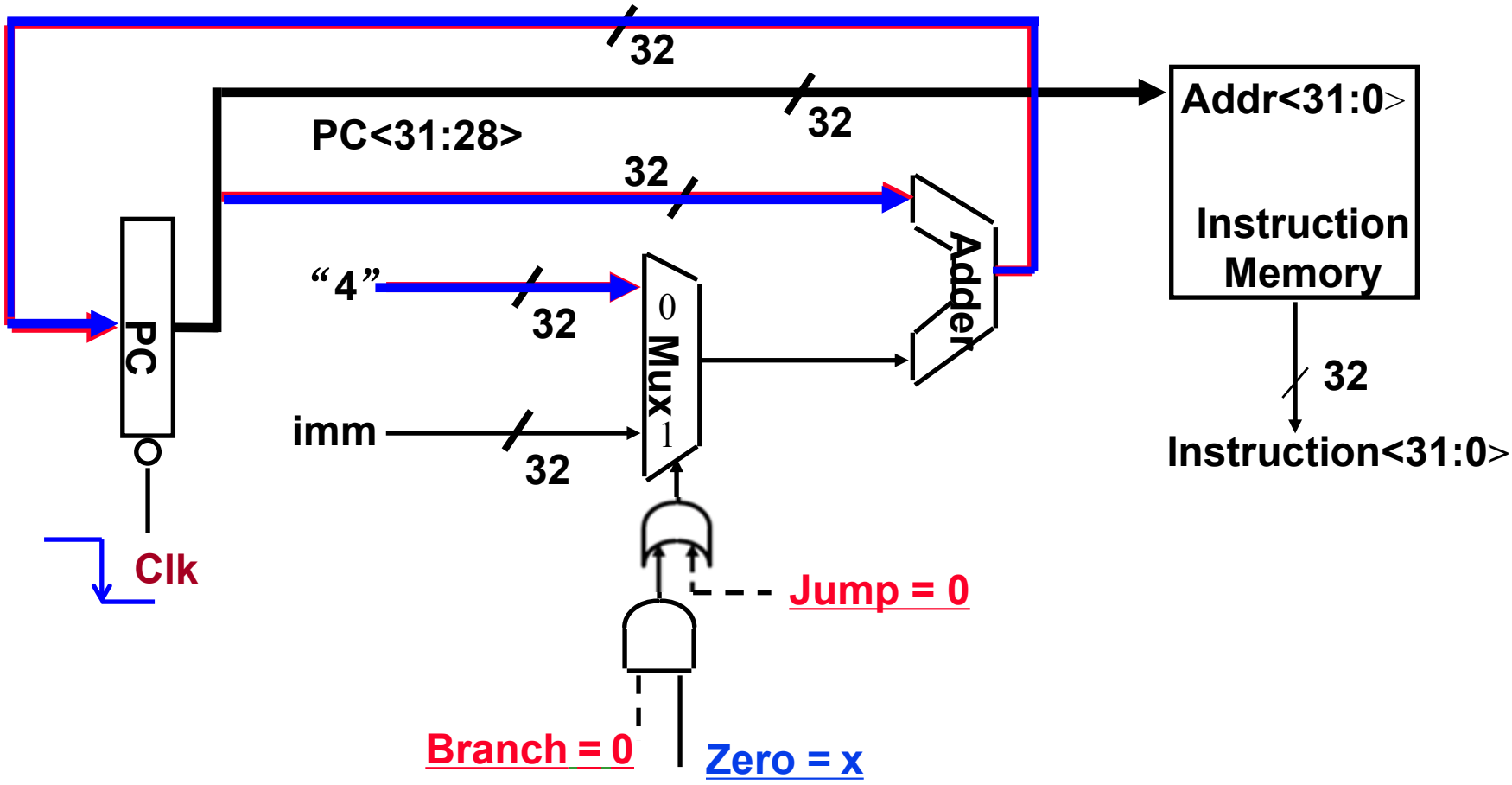
R-型指令功能：R[rd] ← R[rs1] op R[rs2]



R-型指令最后阶段取指部件中的动作

° PC ← PC + 4

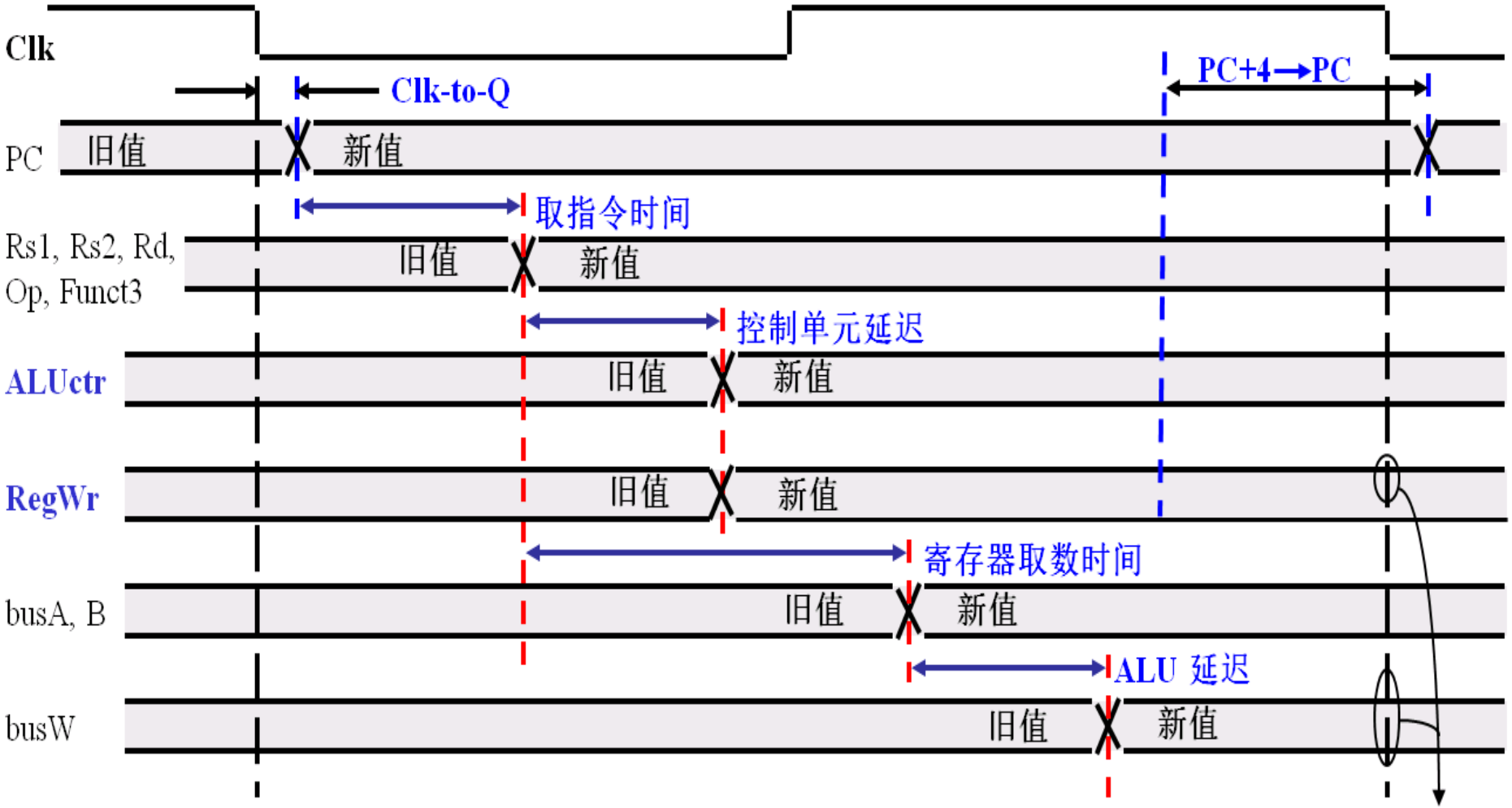
- 除 Branch and Jump以外的指令都相同



新的控制信号保证了正确的PC值的产生，在足够长的时间后，下个时钟Clk到来！

R-型指令的操作定时

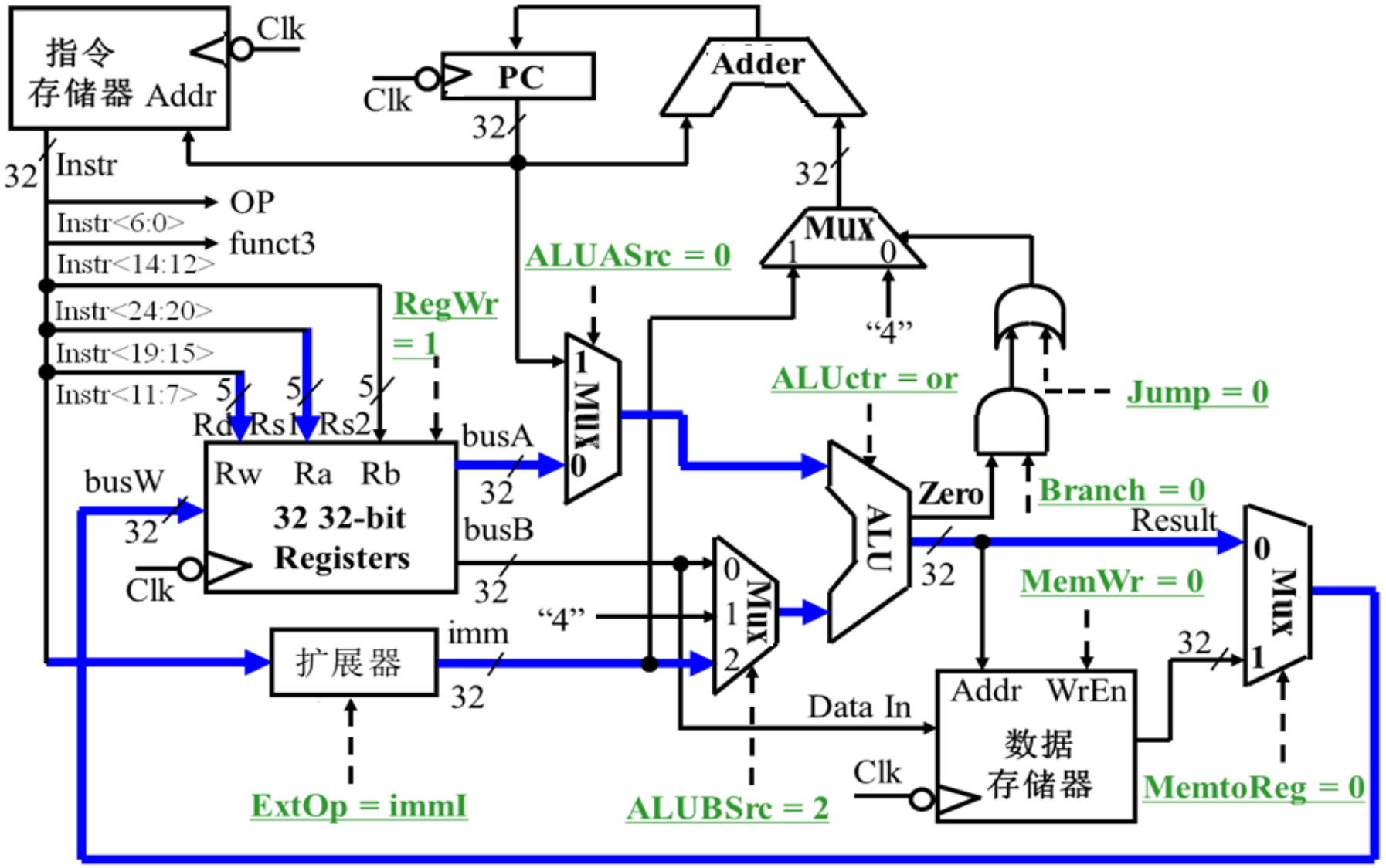
R-型指令功能：R[rd] ← R[rs1] op R[rs2]



时钟周期就是这样算出来的，但此处还不是最终结果

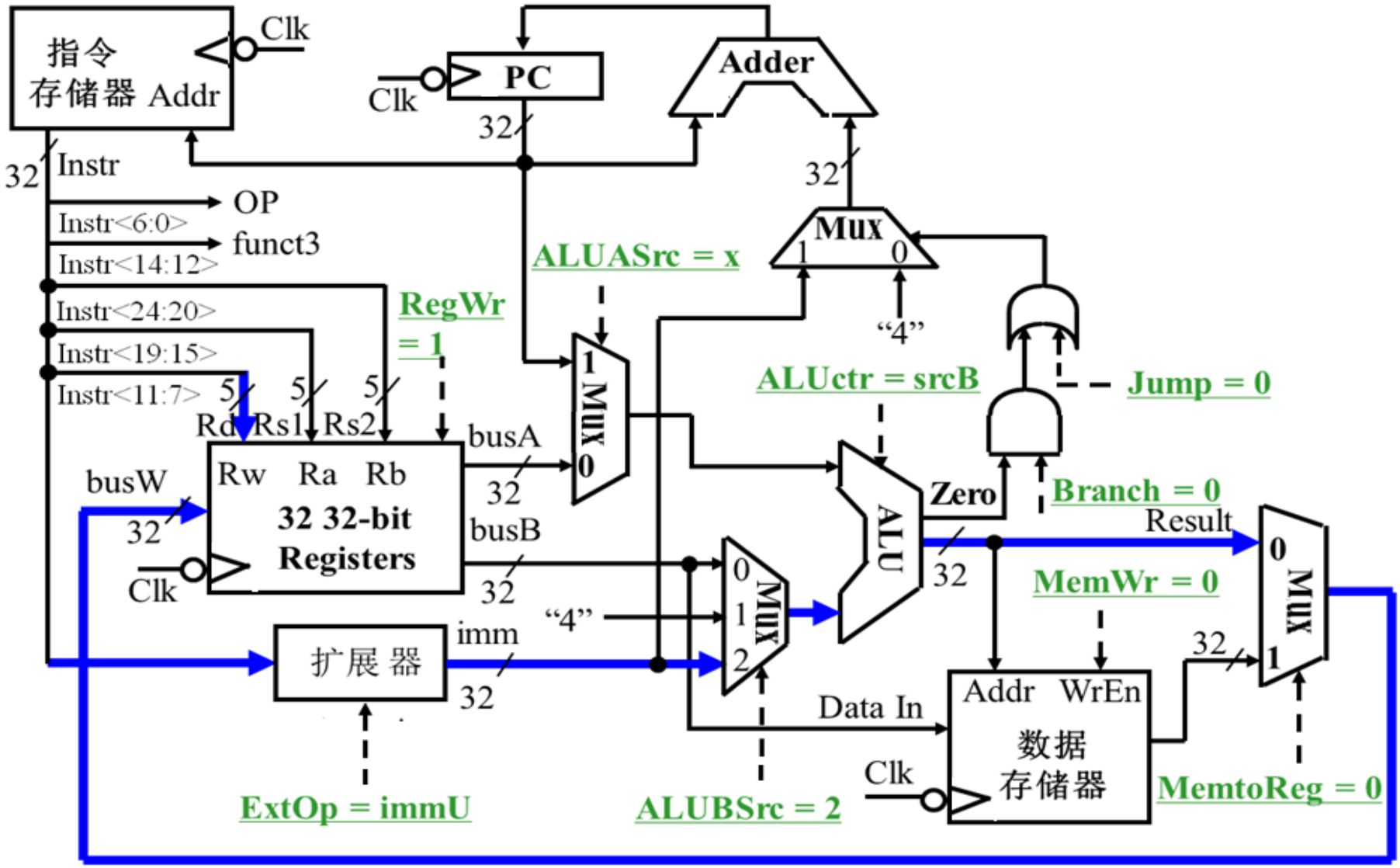
指令译码后I-型运算指令操作过程

I-型指令 ori 功能：R[rd] ← R[rs1] or SEXT(imm12)



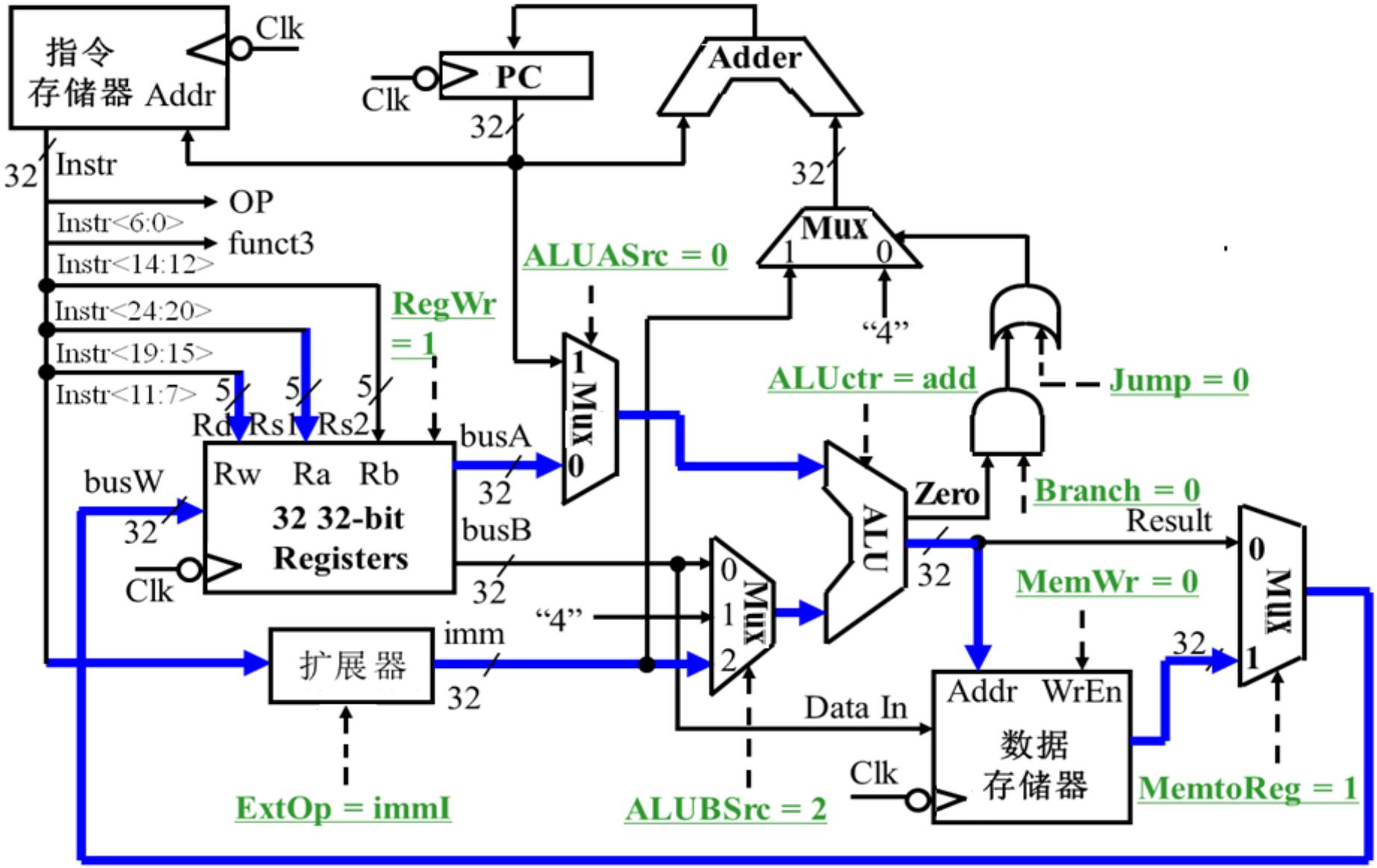
指令译码后U-型指令操作过程

U-型指令lui的功能：R[rd]← imm20||000H 即直接将扩展器结果输出



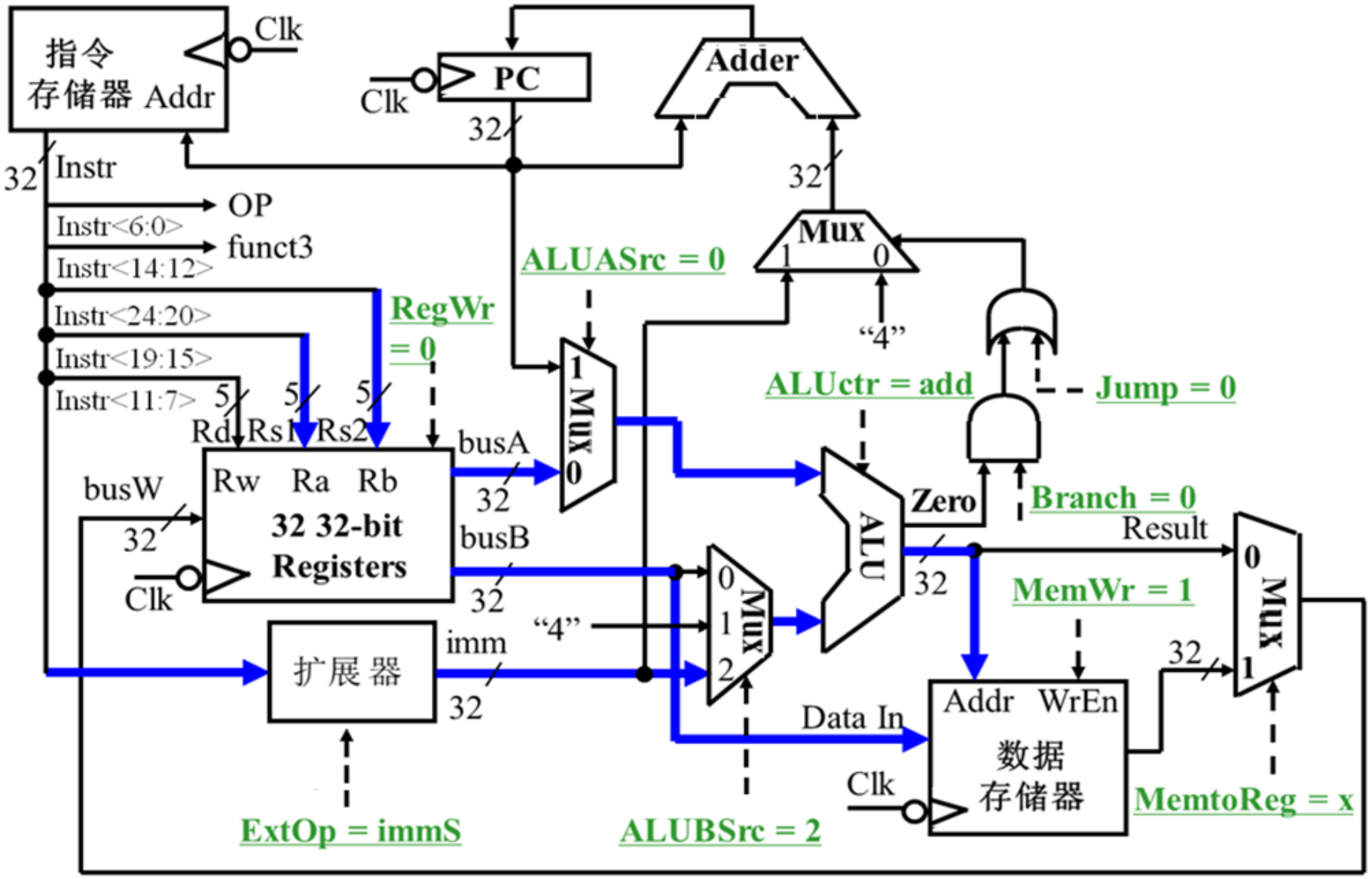
指令译码后Load指令操作过程

I-型的lw指令的功能：R[rd]← M[R[rs1]+SEXT(imm12)]



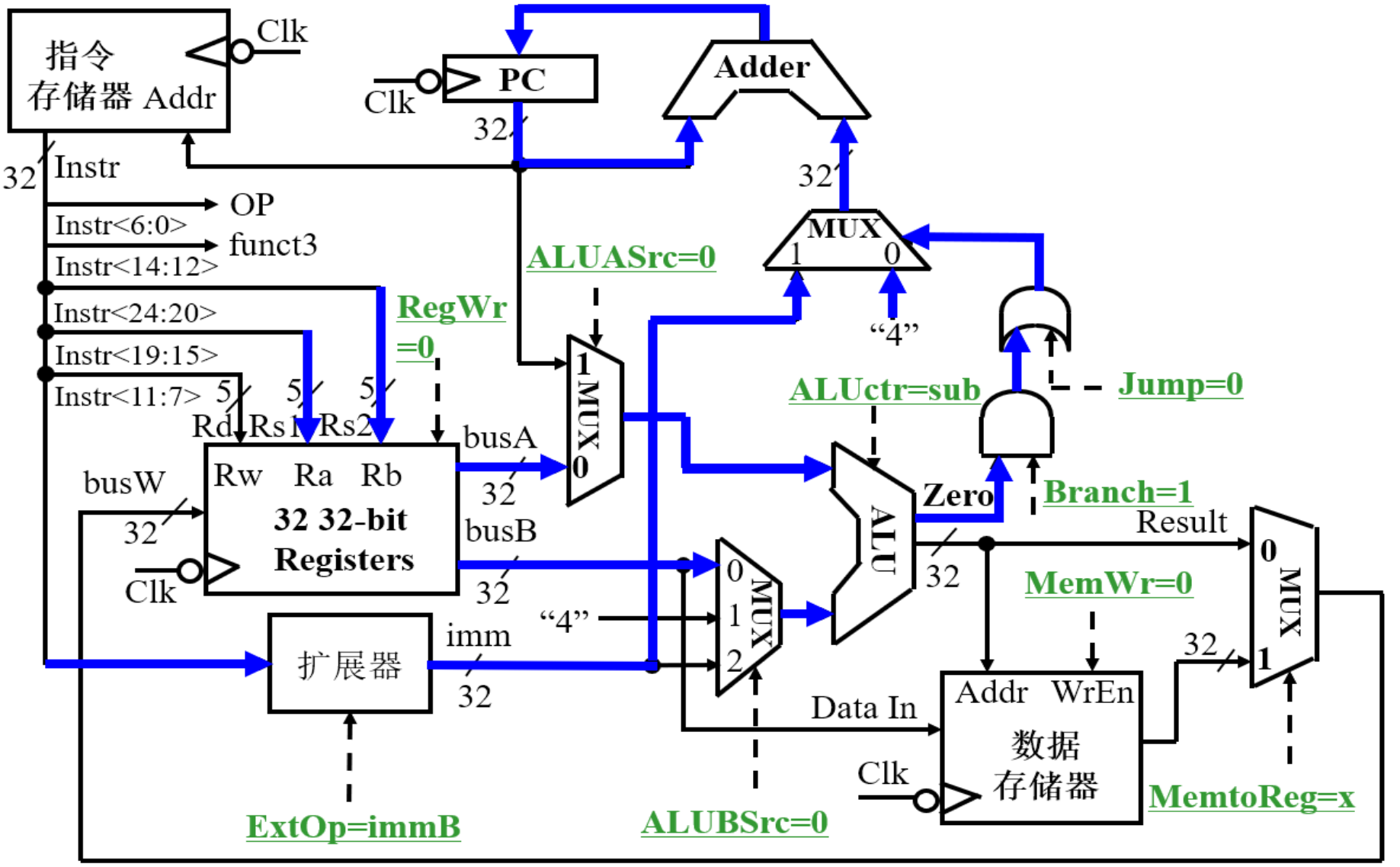
指令译码后Store指令操作过程

S-型的sw指令的功能：M[R[rs1]+SEXT(imm12)]←R[rs2]



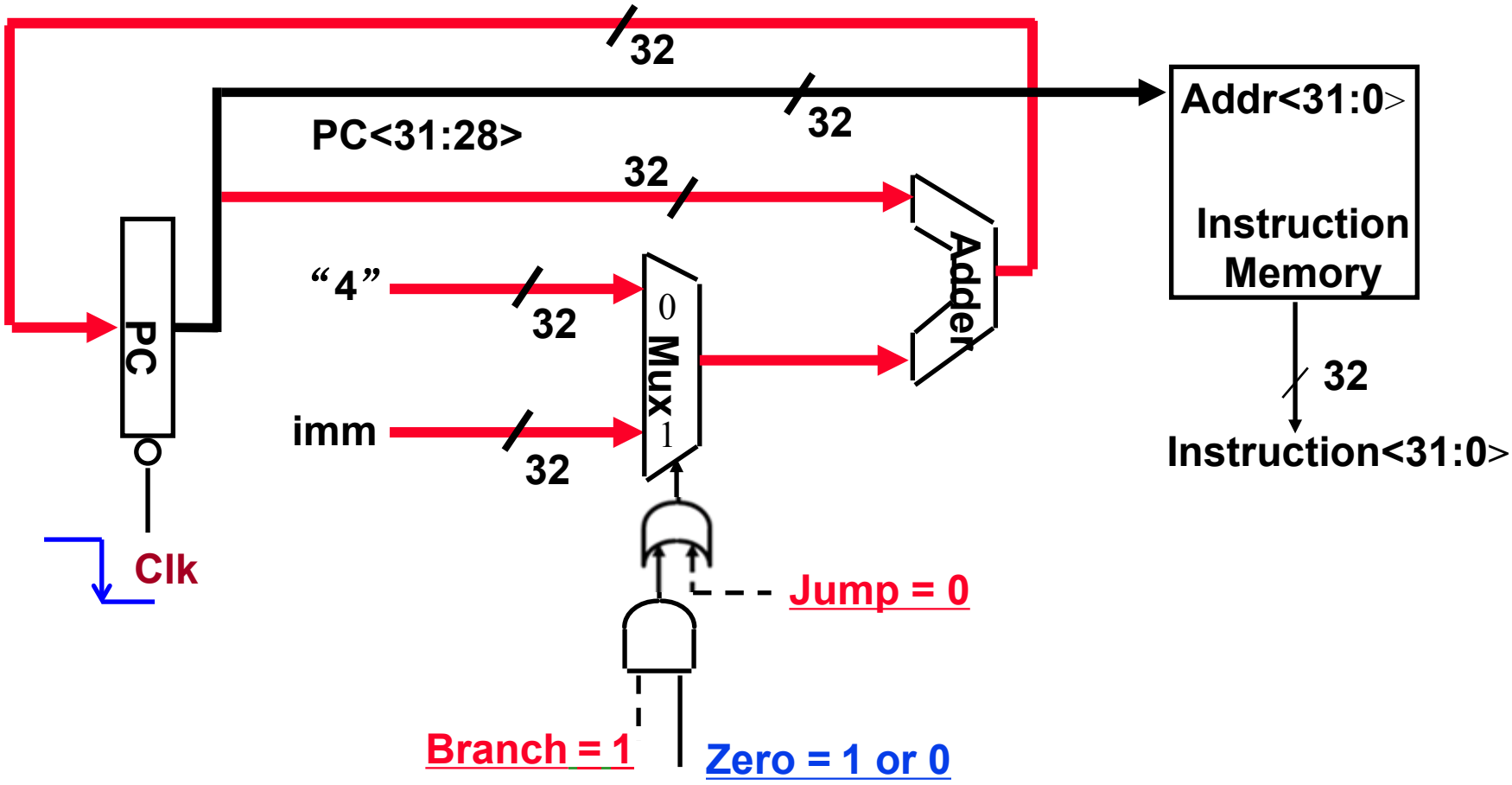
指令译码后B-型指令操作过程

beq功能：if (R[rs1]=R[rs2]) PC←PC+(SEXT(imm12)×2) else PC←PC+4



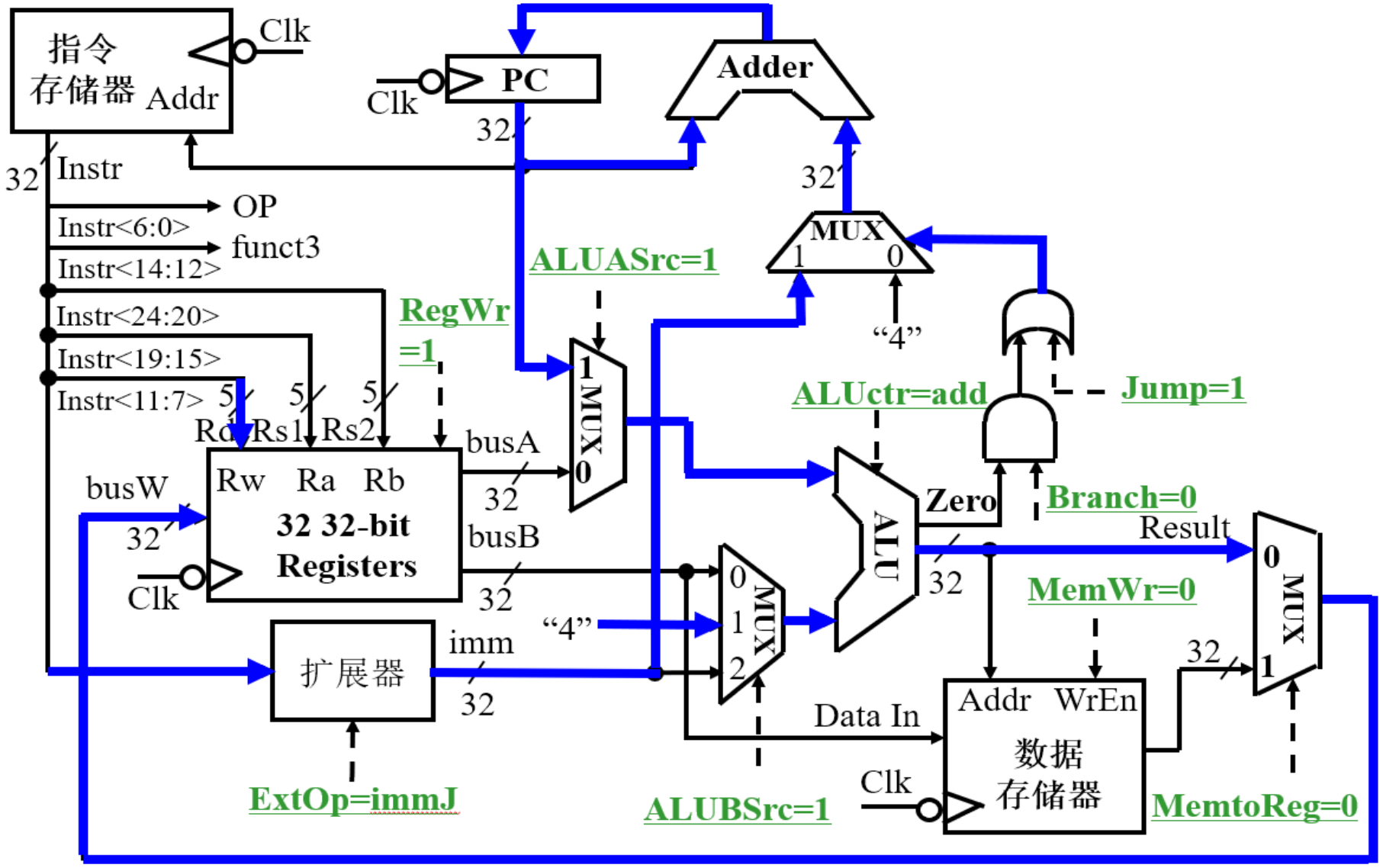
beq指令最后阶段取指部件中的动作

° if (Zero == 1) then PC = PC + imm else PC = PC + 4



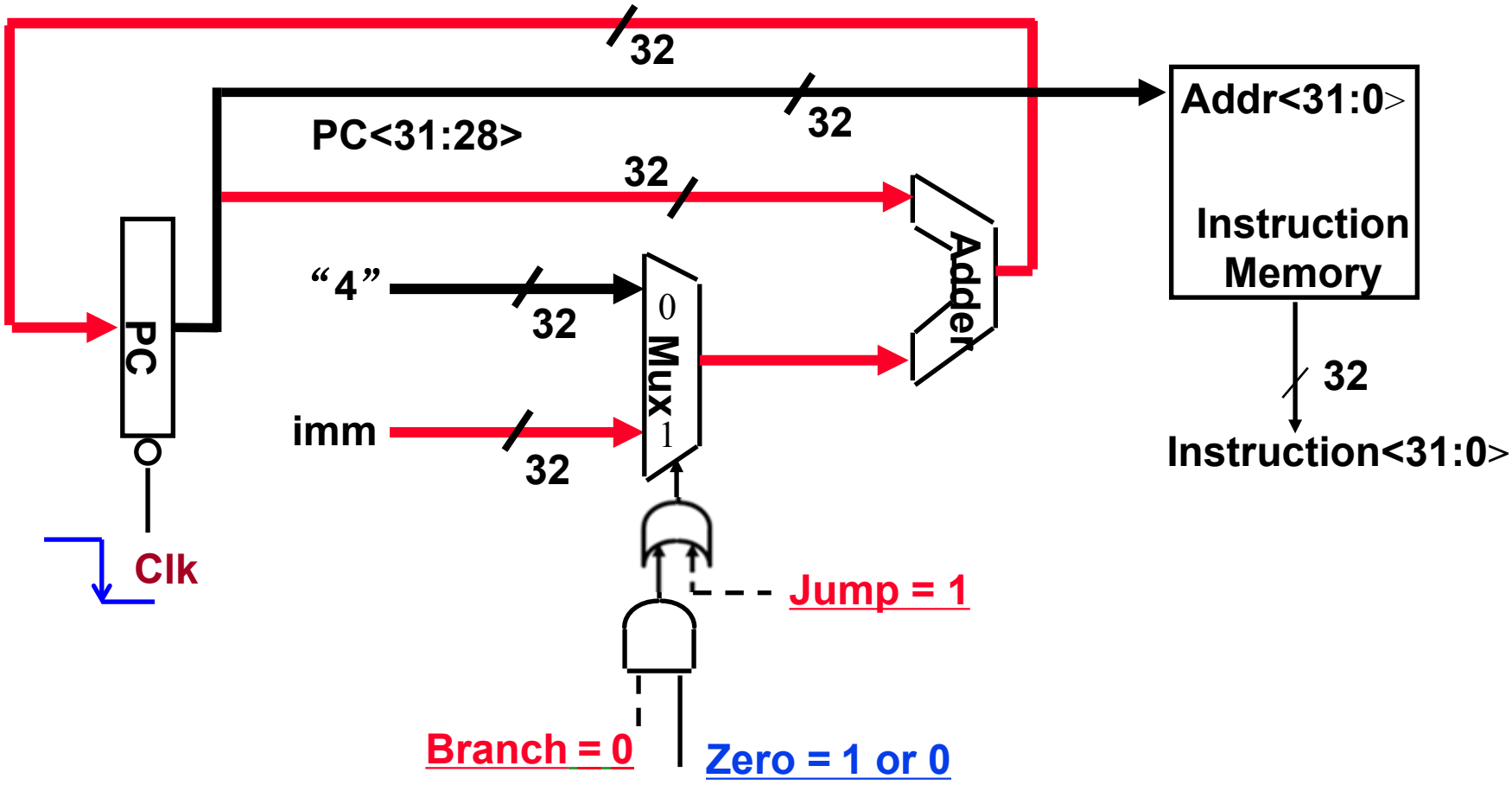
指令译码后J-型指令操作过程

J-型指令jal的功能：PC←PC+SEXT(imm[20:1]<<1)；R[rd]←PC+4



Jal指令最后阶段取指部件中的动作

° PC = PC + imm



综合分析结果，得到如下指令与控制信号的关系表

控制信号 \ funct3	000	010	011	110	无关	010	010	000	无关
op	0110011	0110011	0110011	0010011	0110111	0000011	0100011	1100011	1101111
	add	slt	sltu	ori	lui	lw	sw	beq	jal
Branch	0	0	0	0	0	0	0	1	0
Jump	0	0	0	0	0	0	0	0	1
ALUASrc	0	0	0	0	×	0	0	0	1
ALUBSrc<1:0>	00	00	00	10	10	10	10	00	01
ALUctr<3:0>	0000 (add)	0010 (slt)	0011 (sltu)	0110 (or)	1111 (srcB)	0000 (add)	0000 (add)	1000 (sub)	0000 (add)
MemtoReg	0	0	0	0	0	1	×	×	0
RegWr	1	1	1	1	1	1	0	0	1
MemWr	0	0	0	0	0	0	1	0	0
ExtOp<2:0>	×	×	×	000 immI	001 immU	000 immI	010 immS	011 immB	100 immJ

单值控制信号的逻辑表达式较易

有3个多值控制信号：ALUBSrc、ALUctr、ExtOp，每一位都有逻辑表达式

单值控制信号逻辑表达式的生成例子

funct3	000	010	011	110	无关	010	010	000	无关
op	0110011	0110011	0110011	0010011	0110111	0000011	0100011	1100011	1101111
控制信号	add	slt	sltu	ori	lui	lw	sw	beq	jal
Branch	0	0	0	0	0	0	0	1	0
Jump	0	0	0	0	0	0	0	0	1

Branch=op<6>&op<5>&~op<4>&~op<3>&~op<2>&op<1>&op<0>　(B-type)

注意：本例中beq的func3为000，但其它B型指令也一样
会让branch信号为1，所以此处就不用判断func3了

Jump=op<6>&op<5>&~op<4>&op<3>&op<2>&op<1>&op<0>
(J-type)

单值控制信号逻辑表达式的生成例子

funct3	000	010	011	110	无关	010	010	000	无关
op	0110011	0110011	0110011	0010011	0110111	0000011	0100011	1100011	1101111
控制信号	add	slt	sltu	ori	lui	lw	sw	beq	jal
RegWr	1	1	1	1	1	1	0	0	1

RegWr=(~op<6>& op<5>&op<4>&~op<3>&~op<2>&op<1>&op<0>) (R-type)
| (~op<6>&~op<5>&op<4>&~op<3>&~op<2>&op<1>&op<0>) (I-type-ALU)
| (~op<6>&op<5>&op<4>&~op<3>&op<2>&op<1>&op<0>) (lui)
| (~op<6>&~op<5>&~op<4>&~op<3>&~op<2>&op<1>& op<0>) (Load)
| (op<6>&op<5>&~op<4>&op<3>&op<2>&op<1>&op<0>) (J-type)

注意：类似于上页ppt所述，所以此处也都不用判断func3

多值控制信号逻辑表达式的生成

funct3	000	010	011	110	无关	010	010	000	无关
op	0110011	0110011	0110011	0010011	0110111	0000011	0100011	1100011	1101111
控制信号	add	slt	sltu	ori	lui	lw	sw	beq	jal
ALUBSrc<1:0>	00	00	00	10	10	10	10	00	01
ALUctr<3:0>	0000 (add)	0010 (slt)	0011 (sltu)	0110 (or)	1111 (srcB)	0000 (add)	0000 (add)	1000 (sub)	0000 (add)
ExtOp<2:0>	×	×	×	000 immI	001 immU	000 immI	010 immS	011 immB	100 immJ

以ALUctr为例介绍多值控制信号逻辑表达式生成。

从上表中观察到：R-型和I-型运算类指令的funct3与ALUctr后3位编码相同，而第1位为0，即ALUctr=0||funct3，lw、sw和jal都是对应add操作，即ALUctr=0000，综上得到ALUctr编码分配表如下：

	R-type	I-type-ALU	lui	Load/Store	B-type	J-type
ALUctr<3:0>	0 || funct3 (||为串接操作)		1111	0000	1000	0000

多值控制信号ALUctr逻辑表达式的生成

	R-type	I-type-ALU	Lui 0110111	Load/Store	B-type 1100011	J-type
ALUctr<3:0>	0 || funct3 (||为串接操作)		1111	0000	1000	0000

设funct3各位分别表示为fn<2> fn<1> fn<0>，ALUctr各位逻辑表达式如下：

ALUctr<3> =

(~op<6> & op<5> & op<4> & ~op<3> & op<2> & op<1> & op<0>)

(lui)

| (op<6> & op<5> & ~op<4> & ~op<3> & ~op<2> & op<1> & op<0>)

(B-type)

多值控制信号ALUctr逻辑表达式的生成

	R-type	I-type-ALU	Lui 0110111	Load/Store 0000011 0100011	B-type 1100011	J-type 1101111
ALUctr <3:0>	0 || funct3		1111	0000	1000	0000

设funct3各位分别表示为fn<2> fn<1> fn<0>，ALUctr各位逻辑表达式如下：

ALUctr<2> =

((~op<6> & op<5> & op<4> & ~op<3> & ~op<2> & op<1> & op<0>)
| (~op<6> & ~op<5> & op<4> & ~op<3> & ~op<2> & op<1> & op<0>)) & fn<2>
(R-type + I-type-ALU)
| (~op<6> & op<5> & op<4> & ~op<3> & op<2> & op<1> & op<0>)
(lui)

多值控制信号ALUctr逻辑表达式的生成

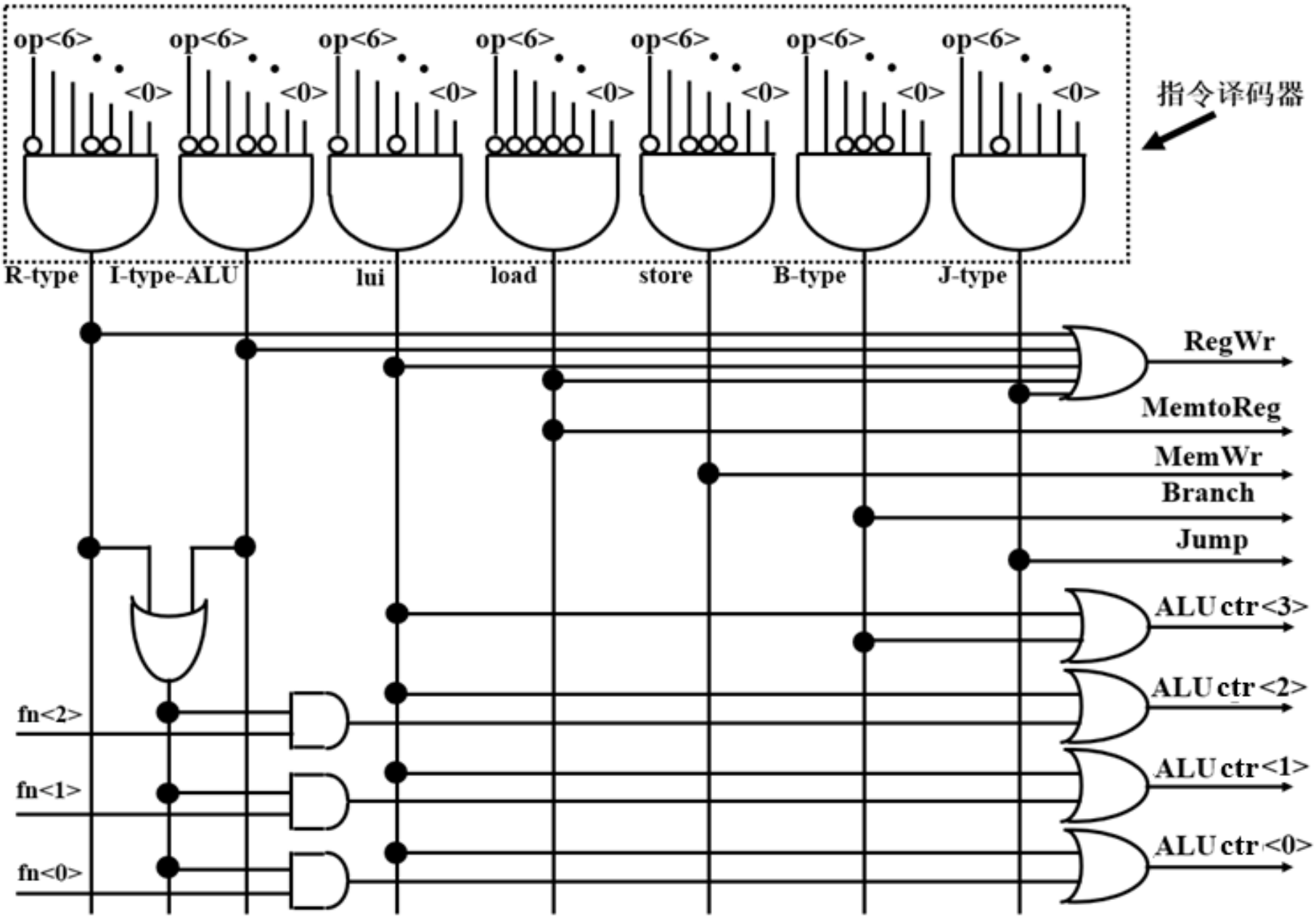
	R-type	I-type-ALU	lui	Load/Store	B-type	J-type
ALUctr<3:0>	0 || funct3（||为串接操作)		1111	0000	1000	0000

设funct3各位分别表示为fn<2> fn<1> fn<0>，ALUctr各位逻辑表达式如下：

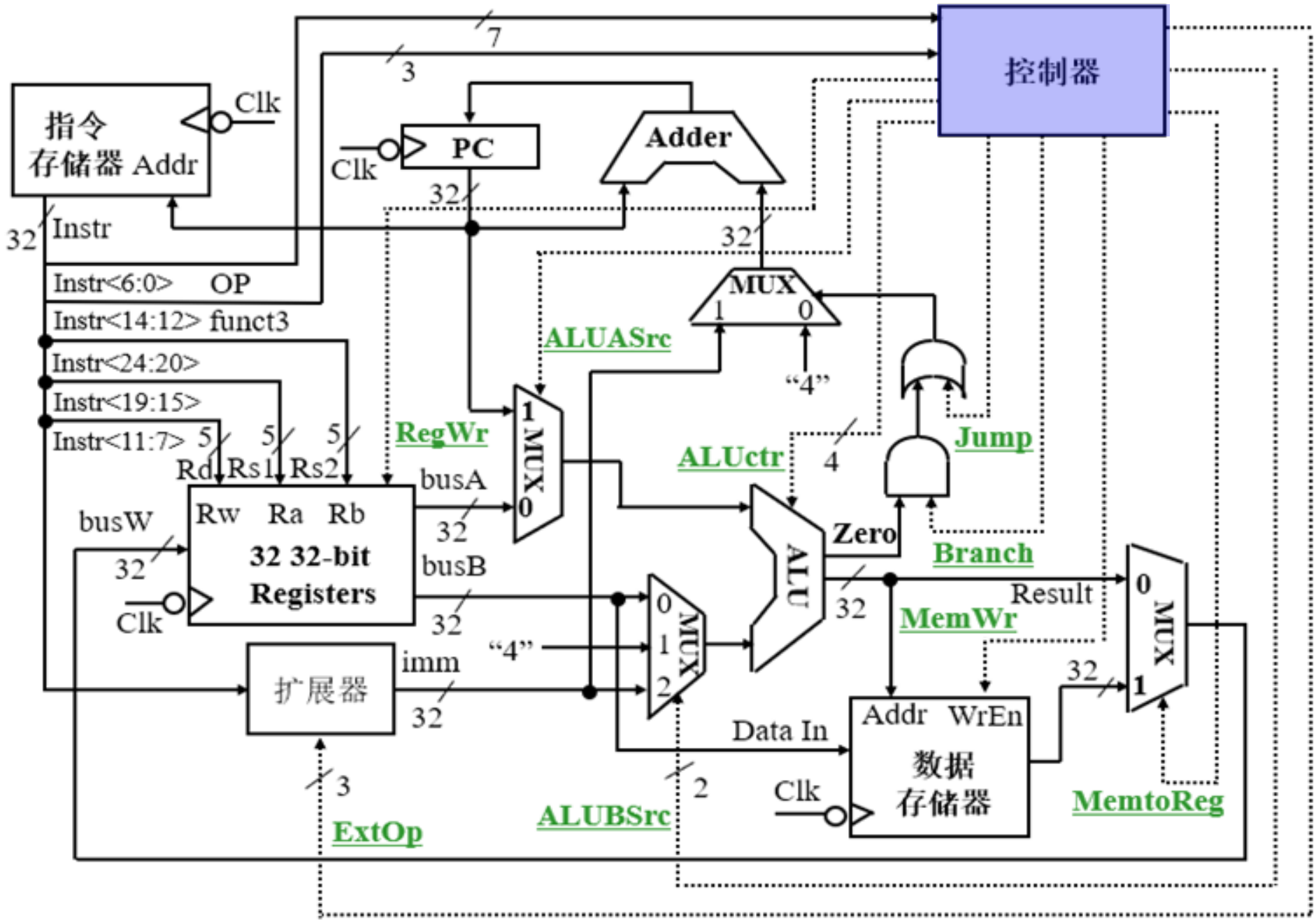
ALUctr<1> = ((~op<6> & op<5> & op<4> & ~op<3> & ~op<2> & op<1> & op<0>)
| (~op<6> & ~op<5> & op<4> & ~op<3> & ~op<2> & op<1> & op<0>)) & fn<1>
| (~op<6> & op<5> & op<4> & ~op<3> & op<2> & op<1> & op<0>)
(R-type + I-type-ALU) &fn<1> +(lui)

ALUctr<0> = ((~op<6> & op<5> & op<4> & ~op<3> & ~op<2> & op<1> & op<0>)
| (~op<6> & ~op<5> & op<4> & ~op<3> & ~op<2> & op<1> & op<0>)) & fn<0>
| (~op<6> & op<5> & op<4> & ~op<3> & op<2> & op<1> & op<0>)
(R-type + I-type-ALU) &fn<0> +(lui)

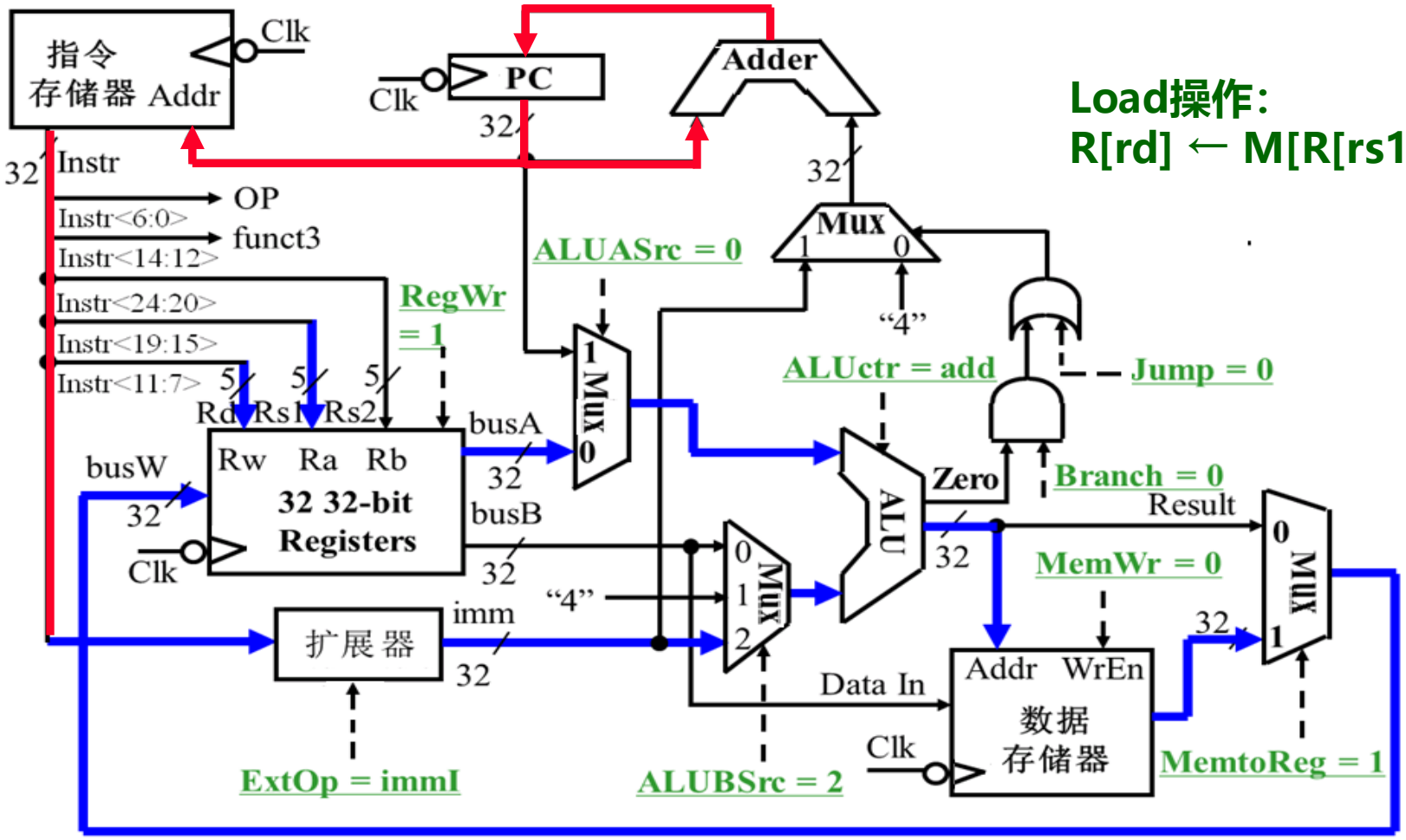
根据逻辑表达式实现控制器的PLA电路



执行前述9条指令完整的单周期处理器



时钟周期的确定：找数据通路中的关键路径

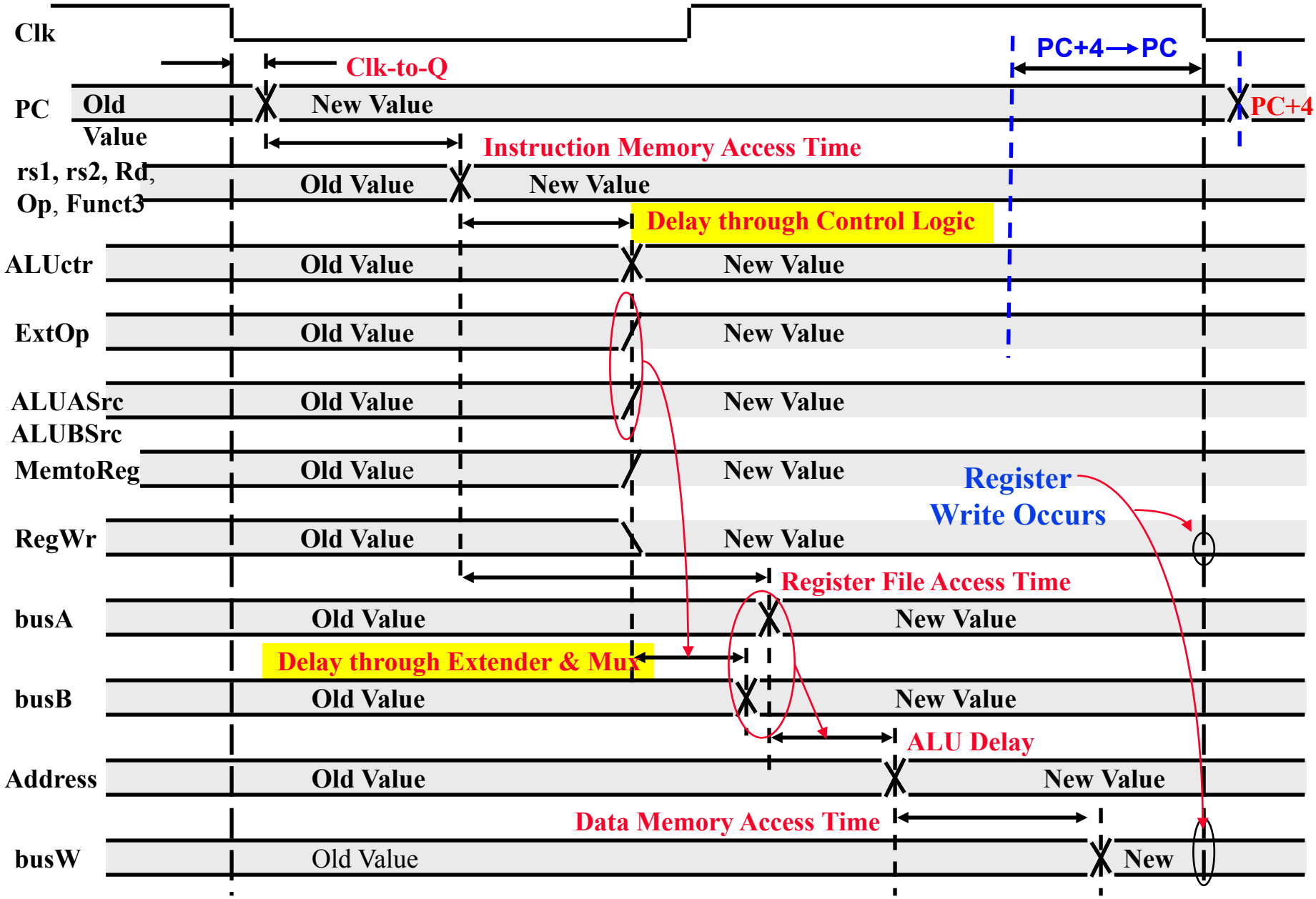


Load操作：
R[rd] ← M[R[rs1]+imm]

- 寄存器组和理想存储器的定时方式
  - 写操作时，作为时序逻辑电路。即：
    - 时钟到达前，输入需setup；时钟到达后经Clk-to-Q，写入数据到达输出端
  - 读操作时，作为组合逻辑电路。即：
    - 地址有效后经过 “access time”，输出开始有效

Critical Path (Load Operation)（时钟周期）=
PC’s prop time (Clk-to-Q) +
Instruction Memory’s Access Time +
Register File’s Access Time +
ALU to Perform a 32-bit Add +
Data Memory Access Time +
Setup Time for Register File Write +
Clock Skew

lw指令的执行时间最长，它所花时间作为时钟周期



单周期计算机的性能

- 单周期处理器的CPI为多少？ CPI=1!
  - 其他条件一定的情况下，CPI越小，则性能越好！
  - CPI=1，不是很好吗？
- 单周期处理器的性能会不会很好？为什么？
  - 计算机的性能除CPI外，还取决于时钟周期的宽度
  - 单周期处理器的时钟宽度为最复杂指令的执行时间
  - 很多指令可以在更短的时间内完成
    - 非load/store指令无需访问DM
    - J-指令无需读通用寄存器

单周期计算机的性能

假设在单周期处理器中，各主要功能单元的操作时间为：

- 存储单元：200ps\ ALU和加法器：100ps\ 寄存器堆（读/写）：50ps

假设MUX、控制单元、PC、扩展器和传输线路都没有延迟，则单周期实现方式（每条指令在一个固定长度的时钟周期内完成）中，CPU执行时间如何计算?

各类指令要求的时间长度为：

Instruction class	Instruction memory	Register read	ALU operation	Data memory	Register write	Total
R-type	200	50	100	0	50	400 ps
Load word	200	50	100	200	50	600 ps
Store word	200	50	100	200	0	550 ps
Branch	200	50	100	0	0	350 ps
Jump	200	0	100	0	50	350 ps

解：CPU执行时间=指令条数 x CPI x 时钟周期

单周期CPI都为1，时钟周期由最长指令来决定，应该是load指令，为600ps

所以：CPU执行时间 = 600x程序指令条数

第三讲 小结

- 考察每条指令在单周期数据通路中的执行过程
  - 每条指令在一个时钟周期内完成
  - 每个时钟到来时，都开始进入取指令操作
    - 经过clk-to-Q，PC得到新值，经过access time后得到当前指令
    - 按三种方式分别计算下条指令地址，在branch / zero / jump的控制下，选择其中之一送到PC输入端，但不会马上写到PC中，一直到下个时钟到达时，才会更新PC。三种下址方式为：
      - branch=jump=0：PC+4
      - branch=zero=1： PC+SEXT(imm12)*2 (ExtOP为ImmB)
      - jump=1： PC+SEXT(imm20)*2 (ExtOP为ImmJ)

第三讲 小结

本章作业：习题3、4、5、6、11、13、14、17、18

- 考察每条指令在单周期数据通路中的执行过程
  - 指令取出后被译码，产生指令对应的控制信号
- 3条R-型指令：add rd, rs1, rs2、slt rd, rs1, rs2、sltu rd, rs1, rs2
  - rd为目的寄存器，无访存操作，……
- I-型指令：ori rd, rs1, imm12、
  - rd为目的寄存器，符号扩展，无访存操作，……
- I-型指令： lw rd, imm12(rs1)
  - rd为目的寄存器，符号扩展，计算地址、读内存，……
- 1条S-型指令：sw rs2, imm12(rs1)
  - rs2为源寄存器，符号扩展，计算地址、写内存，……
- 1条B-型指令：beq rs1, rs2, imm12
  - rs1和rs2为源寄存器，符号扩展，无访存操作，PC有条件更新……
- 1条U-型指令：lui rd, imm20
  - rd为目的寄存器，立即数末尾补0，无访存操作，……
- 1条J-型指令：jal rd, imm20
  - rd为目的寄存器（PC+4），符号扩展，无访存操作， PC无条件更新……
- 汇总每条指令控制信号的取值，生成真值表，写出逻辑表达式，设计控制器逻辑