第八讲 微处理器设计

第1部分 单周期处理器

李学清
xueqingli@tsinghua.edu.cn

腾讯会议: 983-4578-6584

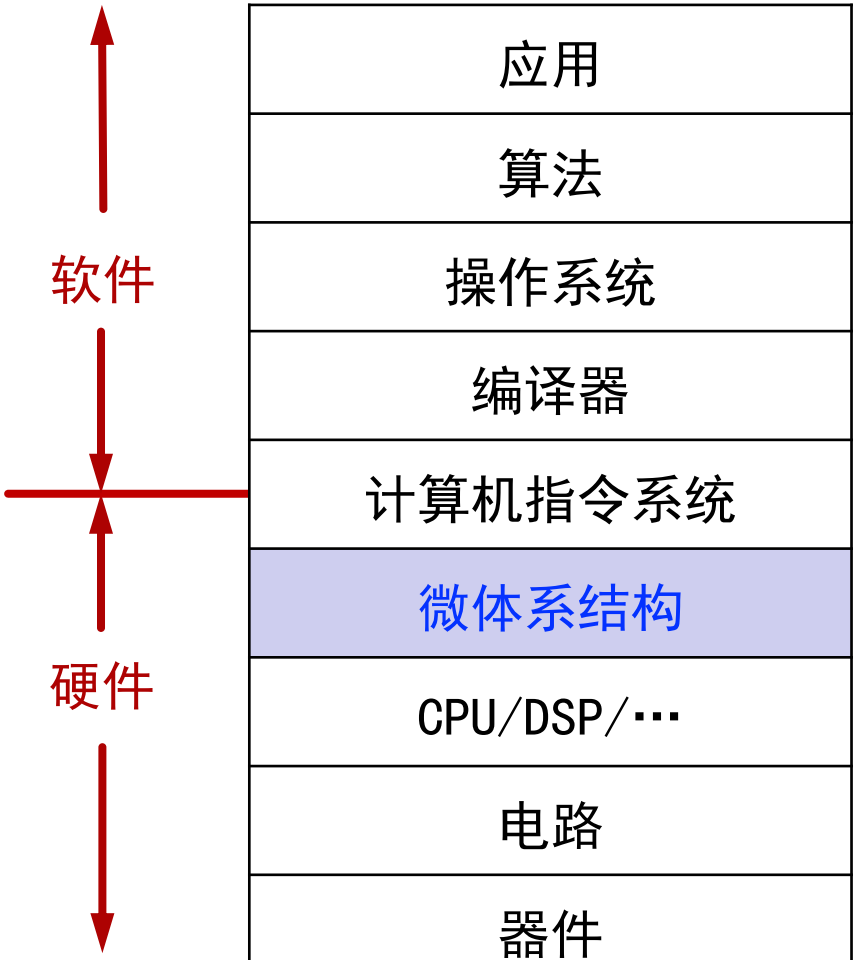
4/19/2022

致谢（to李响）：（1）ALUctr与ALUConf已统一为ALUConf；
（2）P75中ALUSrc的描述已更正

作业

- 要求
  - ddl: 8pm, 4/26/2022
  - 标记学号，姓名和班级
  - 不要抄袭或迟交
  - 线下上课的同学交纸质版，线上上课同学交电子版
- 作业
  - 教材单周期处理器章节习题3、4、5、6、7

计算机指令系统



计算机系统

周	日期	主要内容
1	2/22	概论
2	3/1	布尔代数
3	3/8	组合逻辑
4	3/15	时序逻辑
5	3/22	时序逻辑
6	3/29	计算机指令系统
7	4/10	期中考试
8	4/12	计算机指令系统
9	**4/19**	**微处理器设计**
10	4/26	微处理器设计
11	5/7	劳动节放假5/3->5/7
12	5/10	微处理器/流水线
13	5/17	流水线技术
14	5/24	流水线技术
15	5/31	存储器技术
16	6/7	存储器-IO/总线/总结

“我今天在紫荆食堂吃到了一份数逻”

注：版权原作者所有

本节课学习目标

- 知识点：概念、原理和指标
  - 高级语言 → 汇编语言
  - 硬件上的指令执行
- 能力
  - 掌握设计和分析单周期处理器的方法
  - 掌握设计折衷和评估方法
- 思想
  - 同步和结构化的计算
  - 折衷

目录

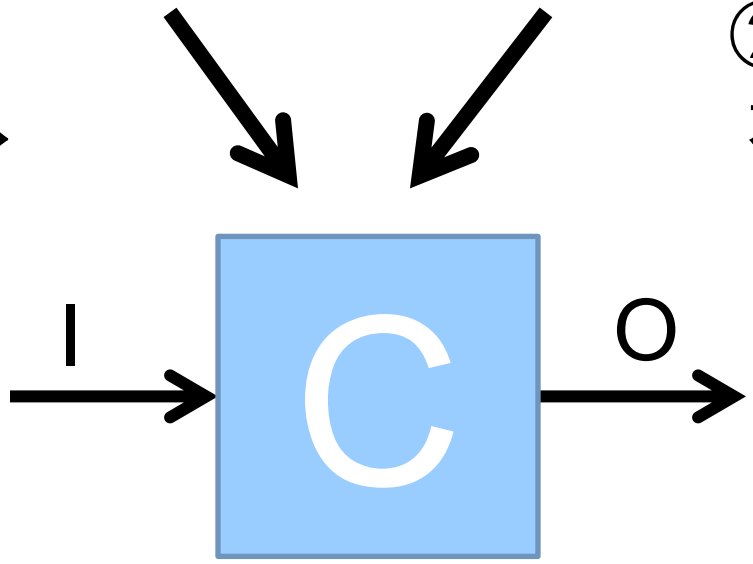
回顾: 两种实现方法

① 定制硬件: 一种算法, 一种电路 (ASIC)
② 通用硬件: 多种算法, 一种电路

$$f: x \rightarrow y$$

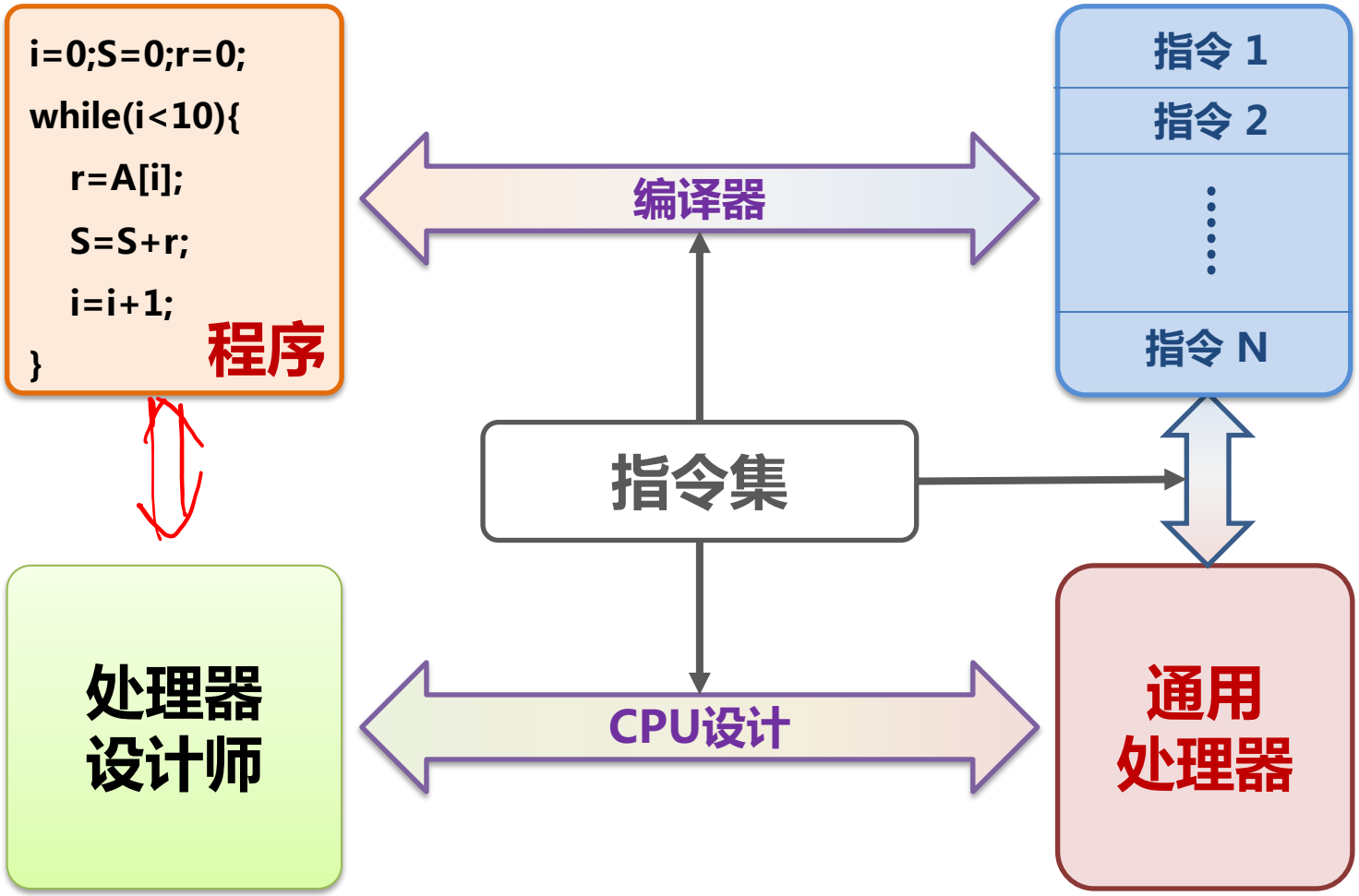
①ASIC
不同算法 →
不同电路

②通用硬件
不同算法 →
相同电路



回顾：如何支持通用算法?

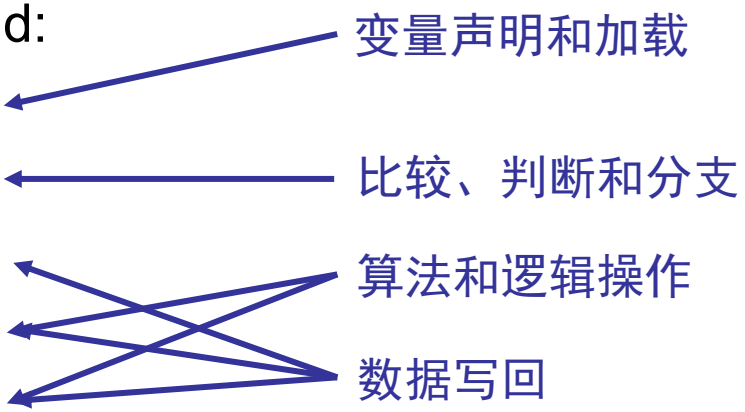
- 指令集的角色
  - ✓ 软件和通用处理器之间的桥梁



从指令集到处理器

E.g. vector add:

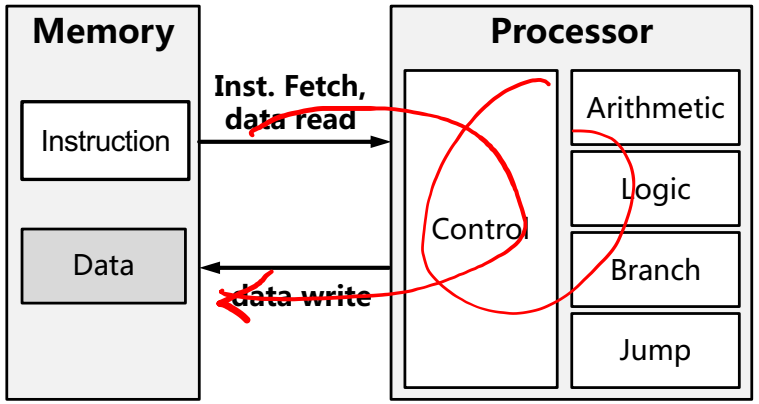
```
i=0;S=0;r=0;
While(i<10){
    r=A[i];
    S=S+r;
    i=i+1;
}
```

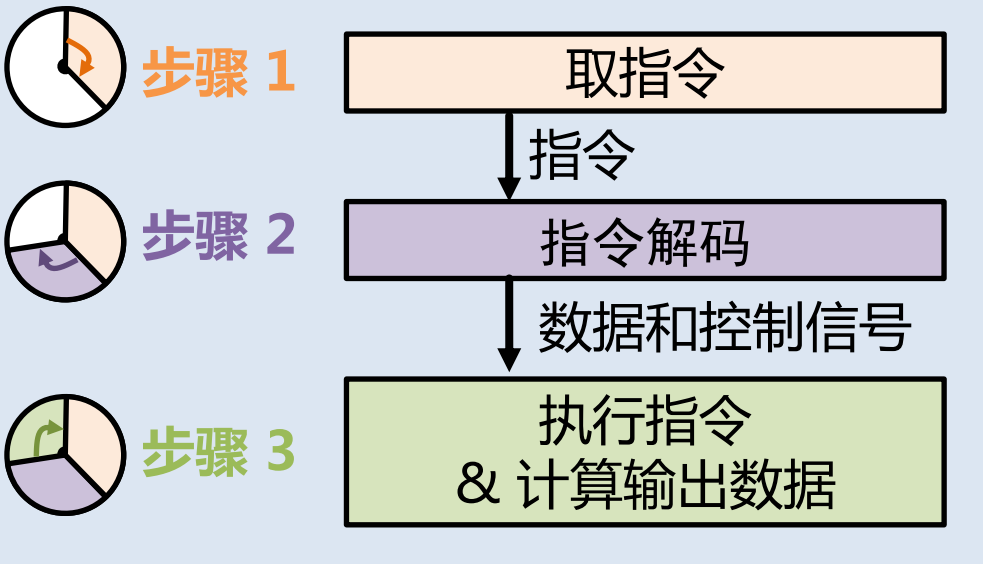


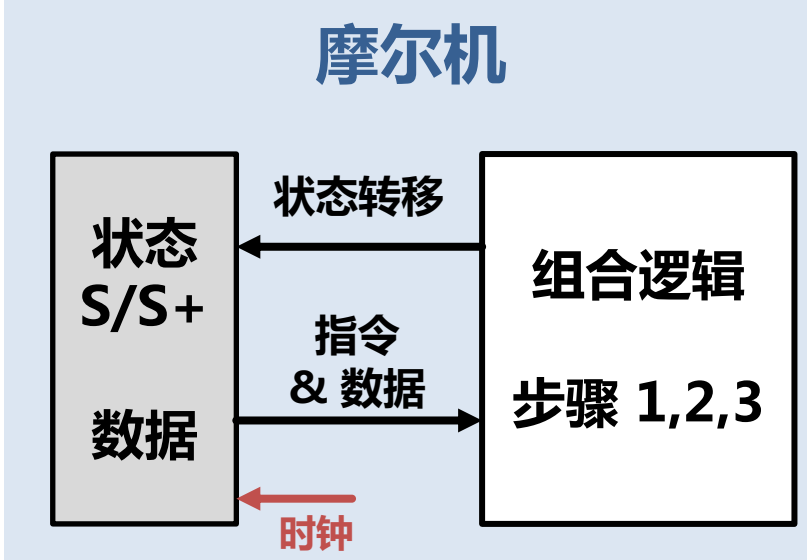
处理器应当具有哪些功能？

回顾：从专用芯片到处理器

- 如何让CPU支持指令集？
  - 明确指令操作步骤
  - 构造摩尔机

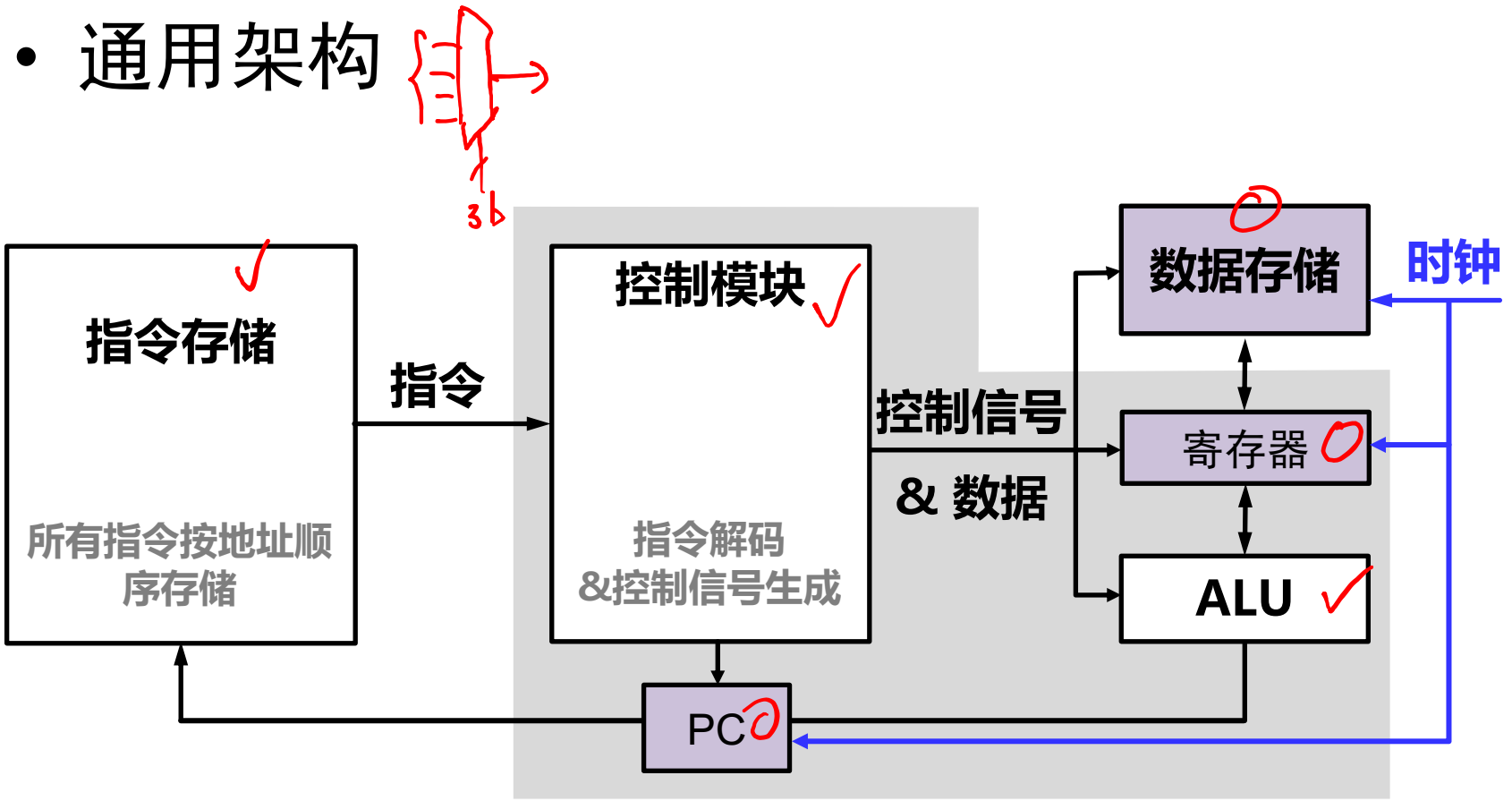






回顾：处理器硬件

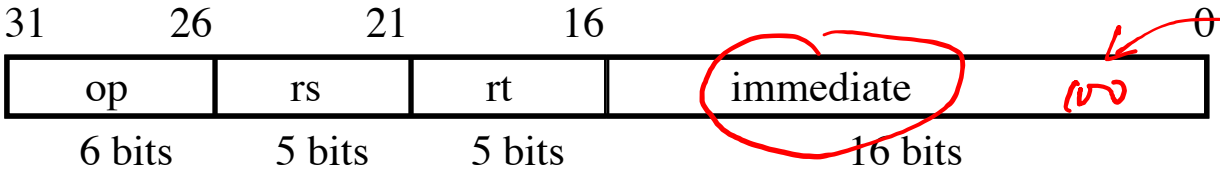
- 通用架构

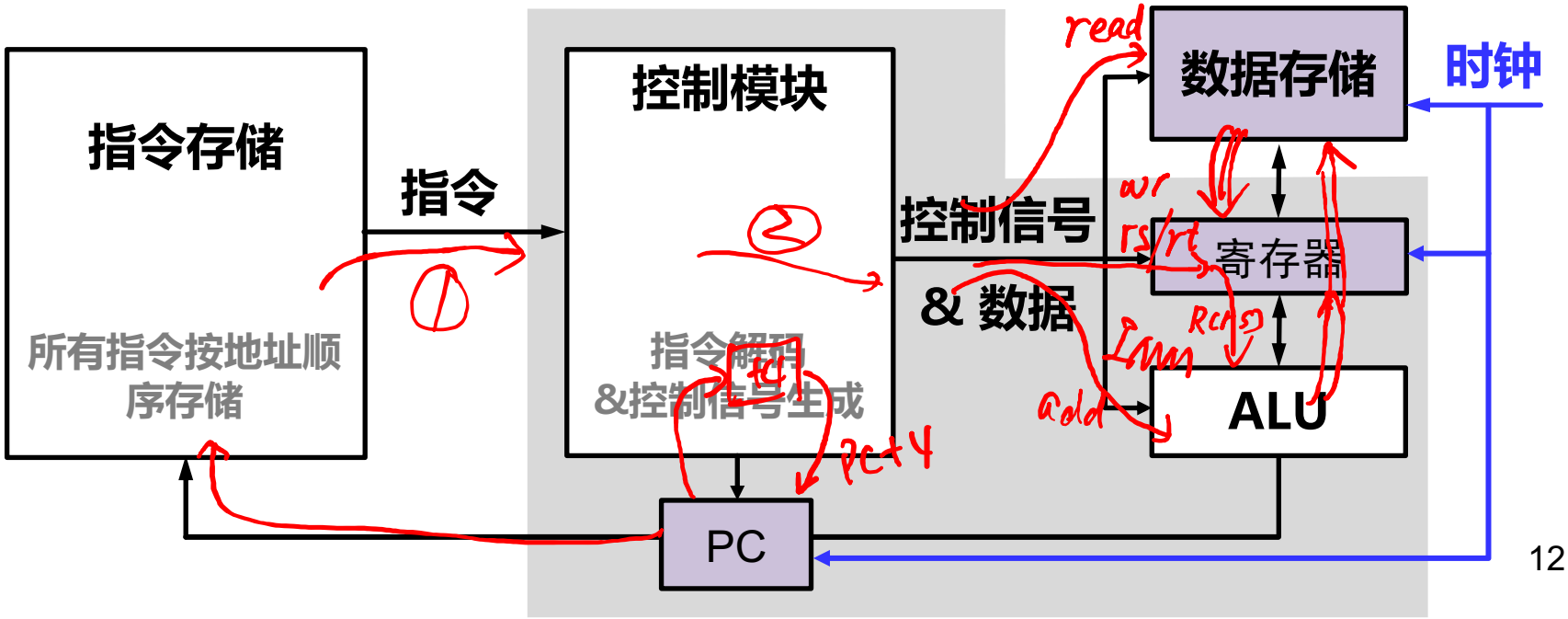


回顾：处理器硬件

- Load 指令
  - R[rt] <= Mem[R[rs] + SignExt[imm16]]

e.g. B=A[25]

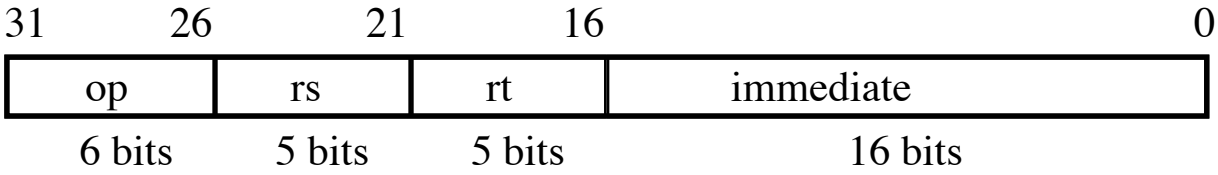


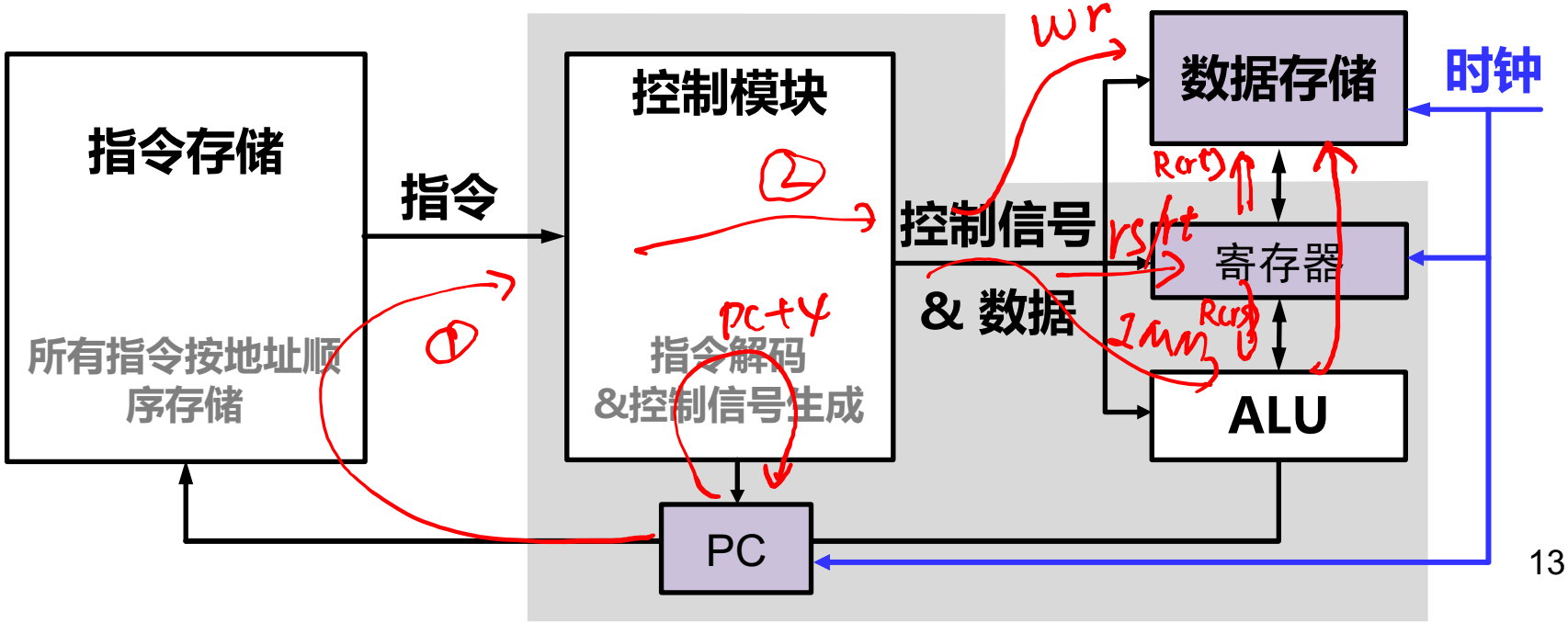


回顾：处理器硬件

- Store 指令
  - Mem[R[rs] + SignExt[imm16]] <= R[rt]

e.g. A[25]=B

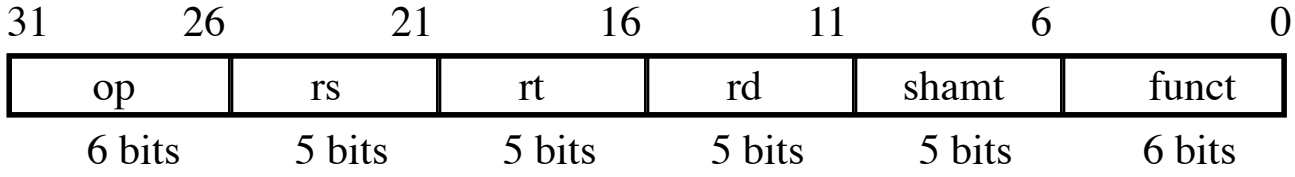


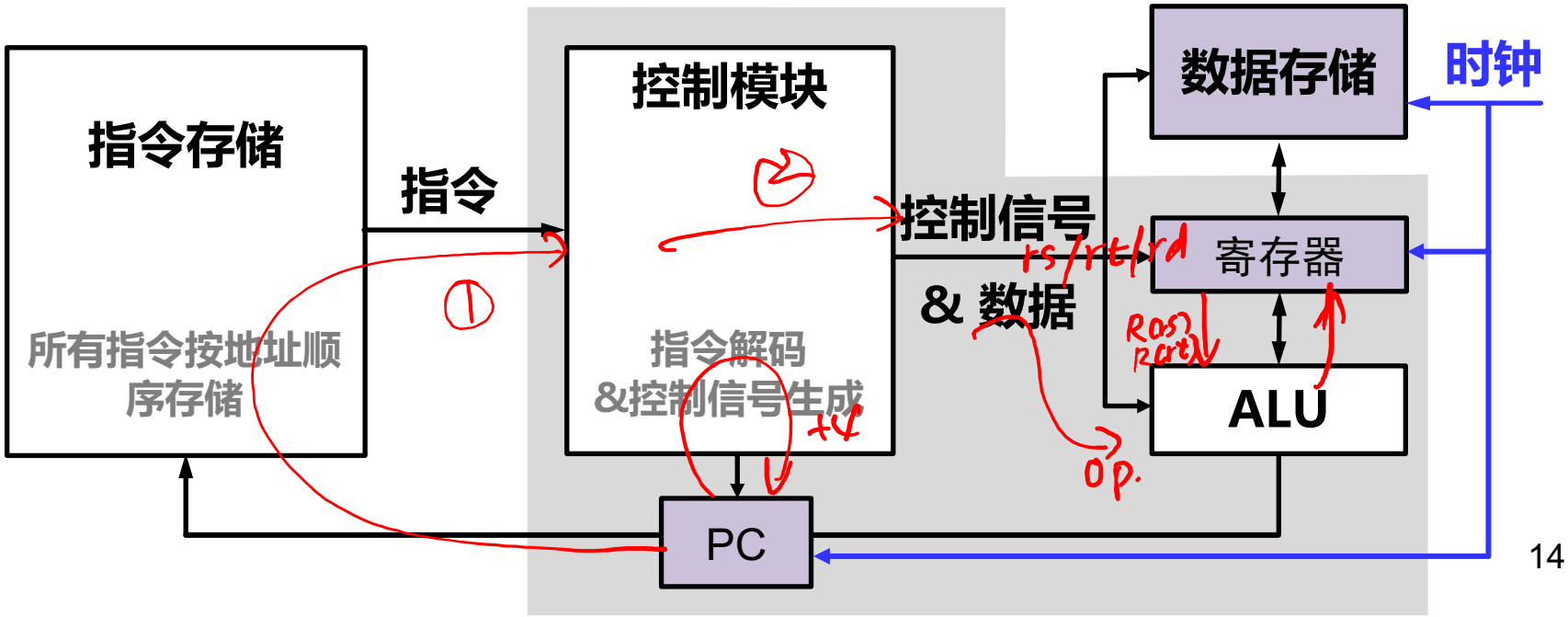


回顾：处理器硬件

- 算数&逻辑运算
  - R[rd] <= R[rs] op R[rt]

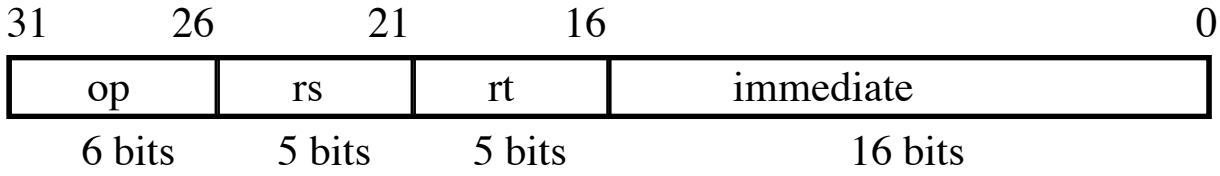
e.g. A=B+C

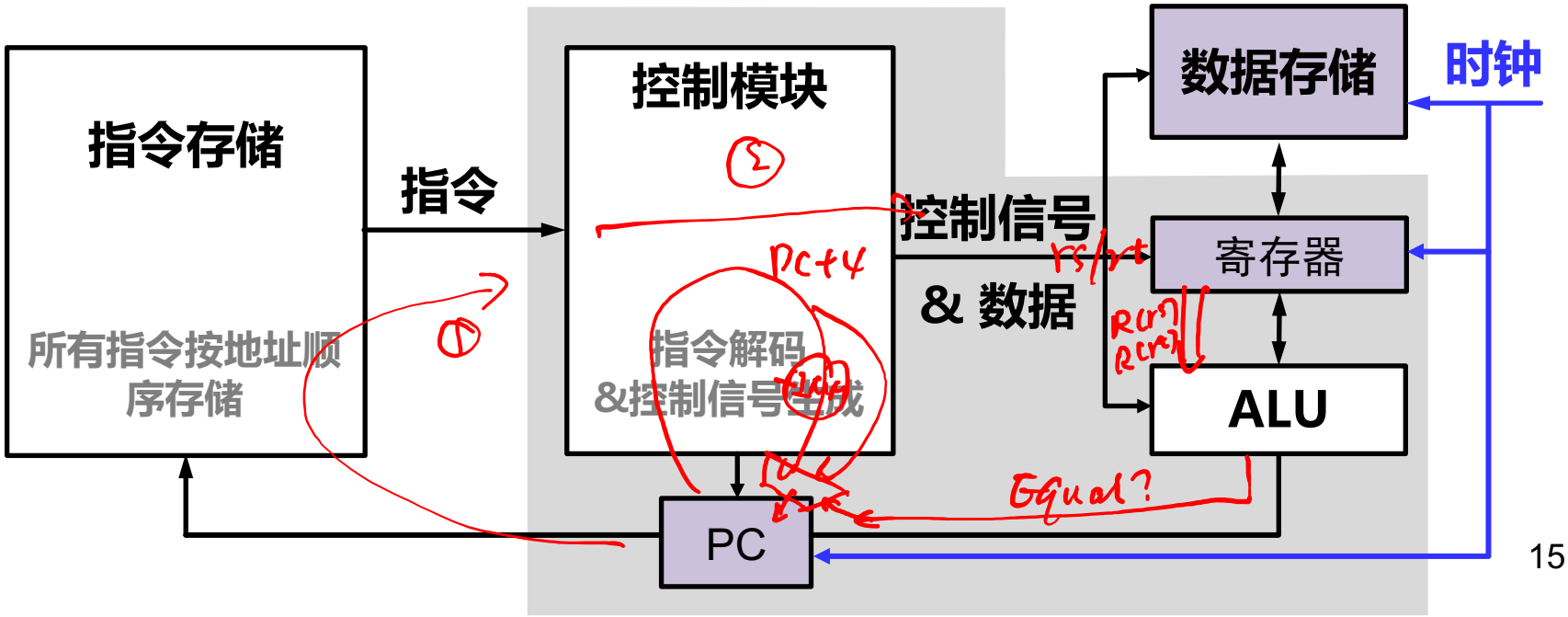




回顾：处理器硬件

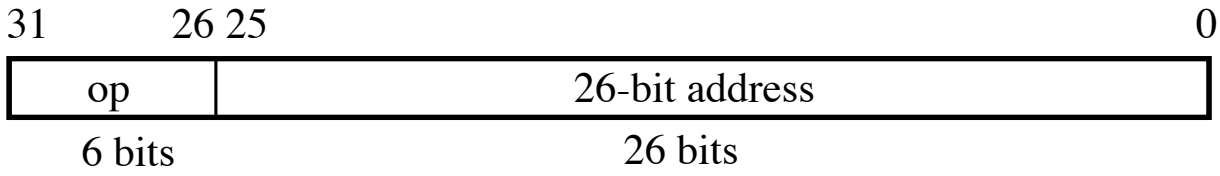
- Branch 分支: beq rs, rt, imm16
  - PC=PC+4 (rs==rt) or PC+4+{SignExtImm << 2} (rs!=rt)

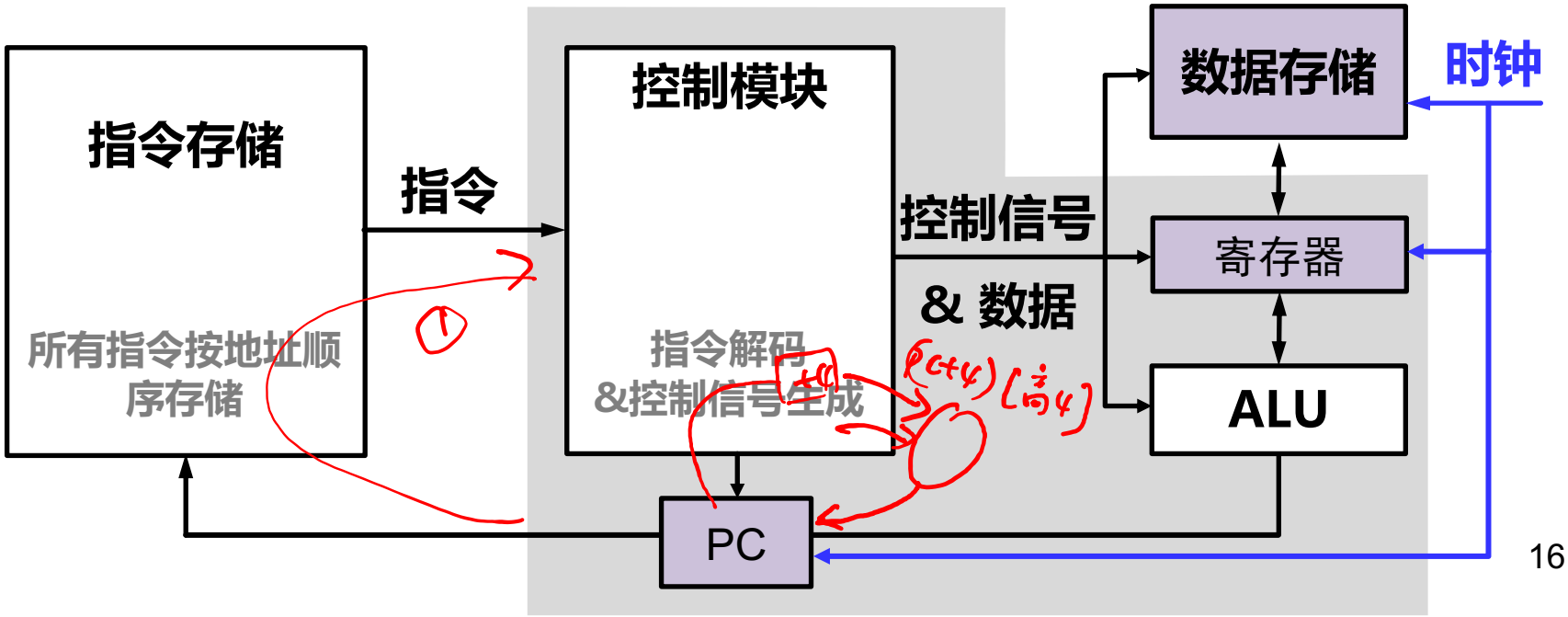




回顾：处理器硬件

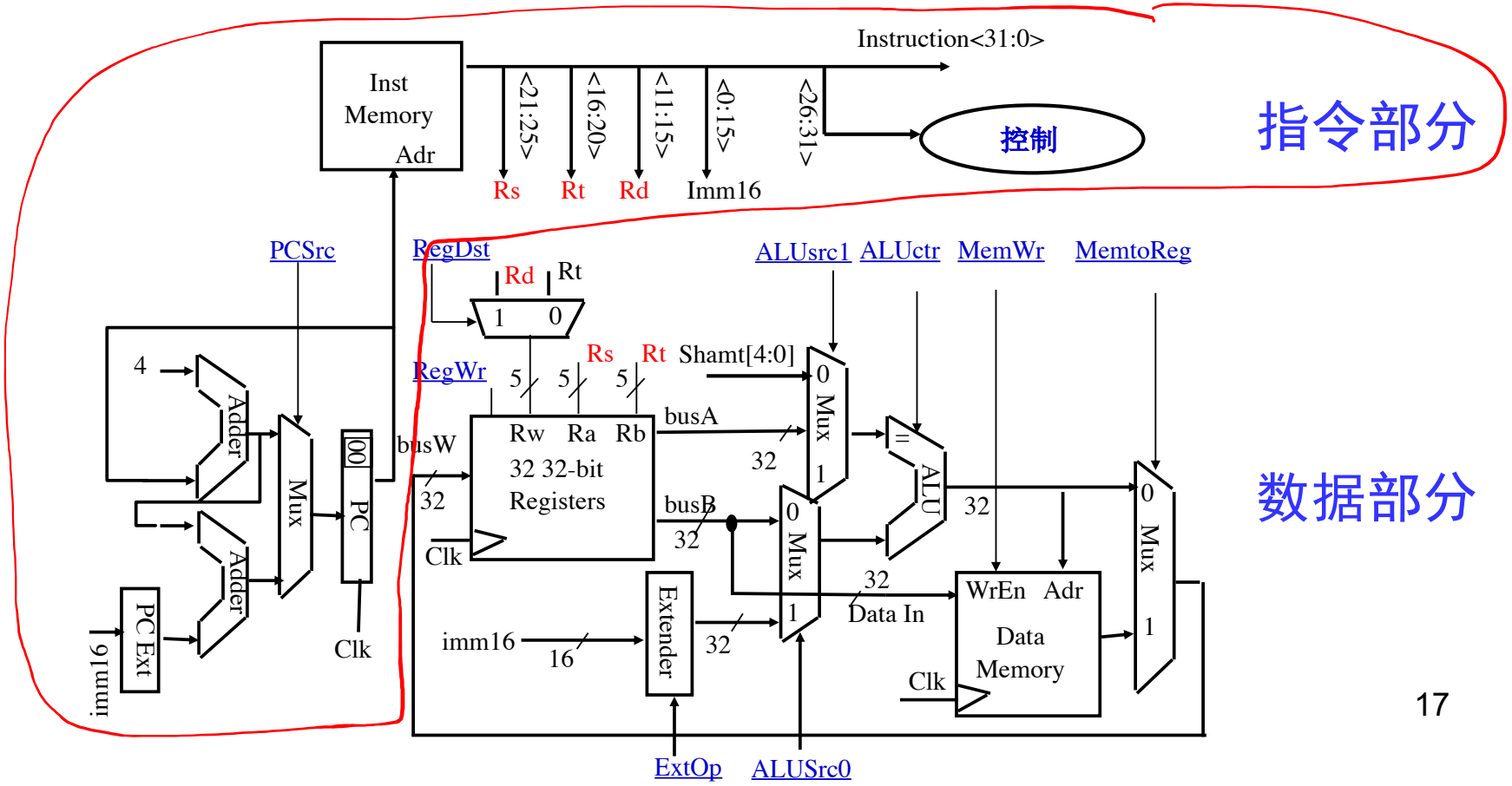
- Jump 跳转
  - 跳转地址: PC = { (PC+4)[31..28], target, 00 }





MIPS单周期处理器架构

- 根据上述数据通路的分析，可以设计出具体的MIPS单周期处理器（本节课讲授重点）

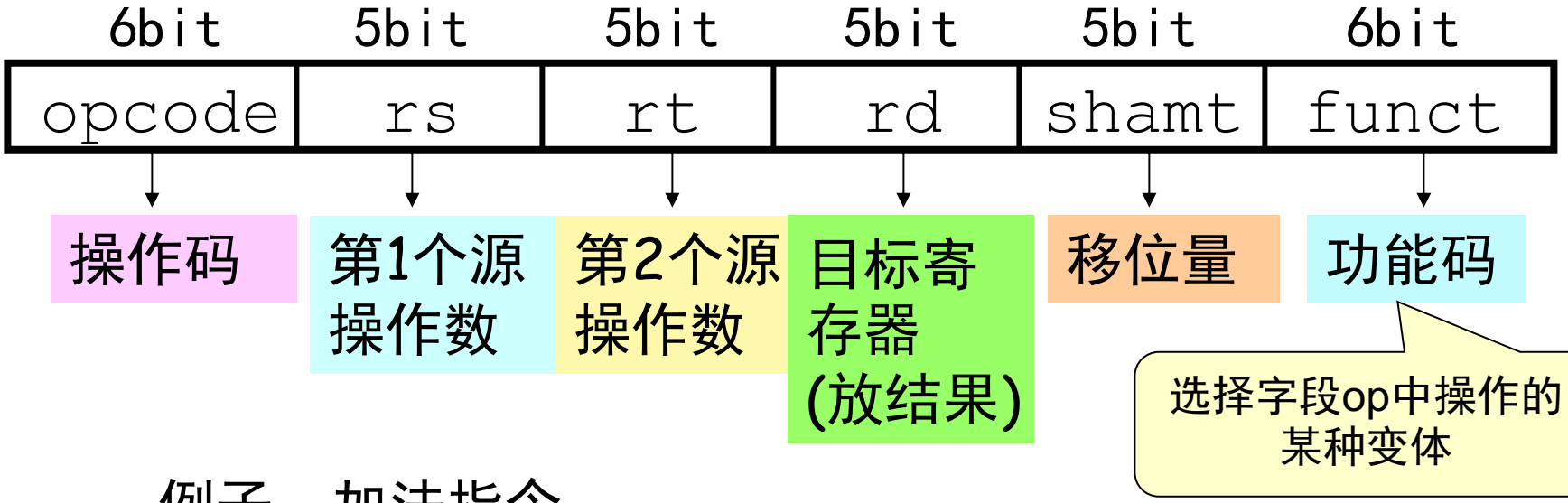


回顾：MIPS指令集架构

- 完备指令集设计应支持计算、访存与决策三类功能
- 在MIPS架构中
  - 计算与访存相关指令根据操作数类型分为两类
    - 操作数以寄存器存储的数据为主：R型指令
    - 操作数涉及到指令字段表示的立即数：I型指令
  - 分支循环等与跳转相关的指令：J型指令

回顾：MIPS指令集架构

- R型指令
  - rs: source, rt: target, rd: destination
  - opcode与funct共同确定该R型指令的具体功能

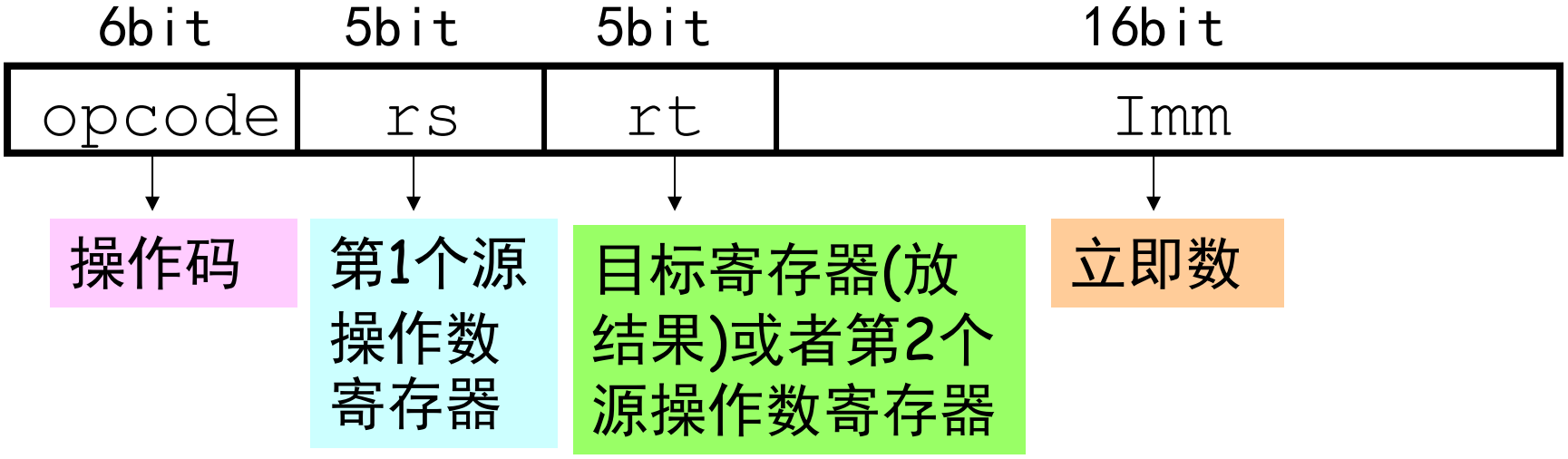


  - 例子：加法指令

add $rd, $rs, $rt　　如add $8, $9, $10

回顾：MIPS指令集架构

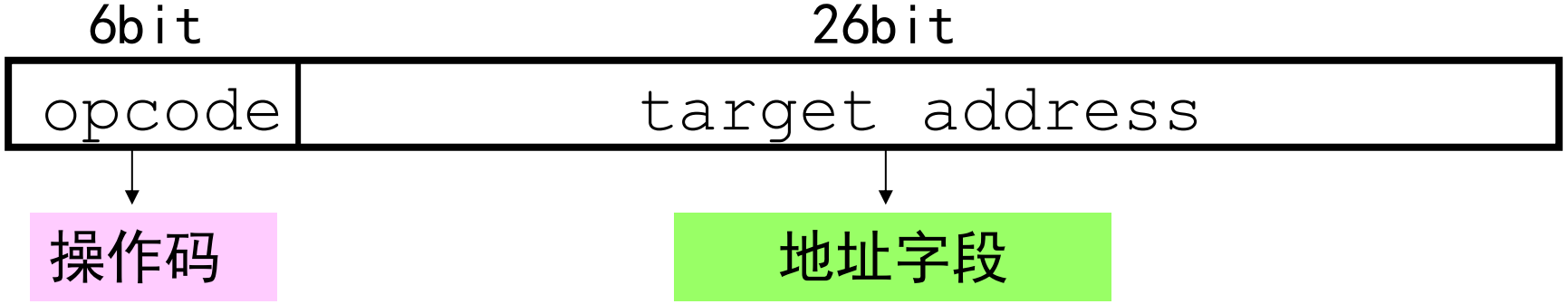
- I型指令
  - rt同时包含源操作数和目标寄存器的含义（根据具体指令而定）
  - 立即数字段可表示16bit常数，也可表示地址偏移量



  - 例子：装入/存储指令: sw/lw $rt, immediate($rs)
  - 与立即数做计算: addi $rt, $rs, 50

回顾：MIPS指令集架构

- J型指令
  - opcode与R/I型指令类似
  - 其他字段：给出跳转的目的地址，可以实现很大的跳转范围



例子：跳转指令，j 10000

回顾:MIPS指令格式总结

- I型：用于有立即数的指令：lw sw beq bne
  - lw/sw: load word; store word
  - beq/bne: branch on equal; branch on not equal
  - opcode: 除000000, 00001x, 0100xx外
- J型：用于跳转，j, jal
  - j: jump; jal: jump and link;
  - opcode: 000010, 000011
- R型：所有其他指令
  - opcode: 000000, jr: jump register

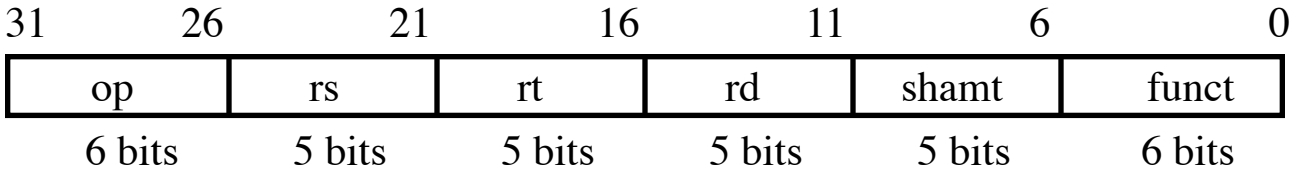
opcode: 区分不同指令的关键

知识回顾

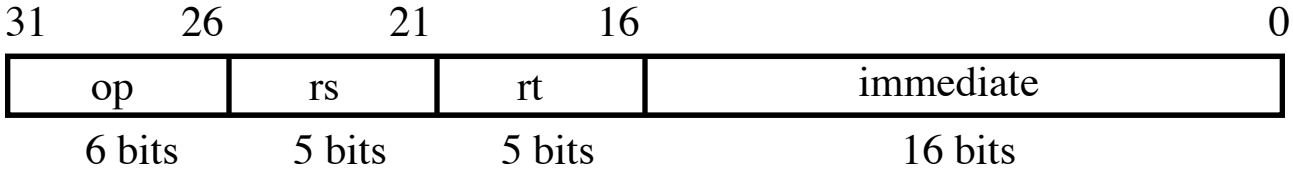
- MIPS指令功能
  - 数据运算指令：add, sub, and, not, …
  - 数据传送指令：lw, sw, lui, …
  - 分支与跳转指令：beq, bne, slt, j, jr, jal

回顾：MIPS指令集架构

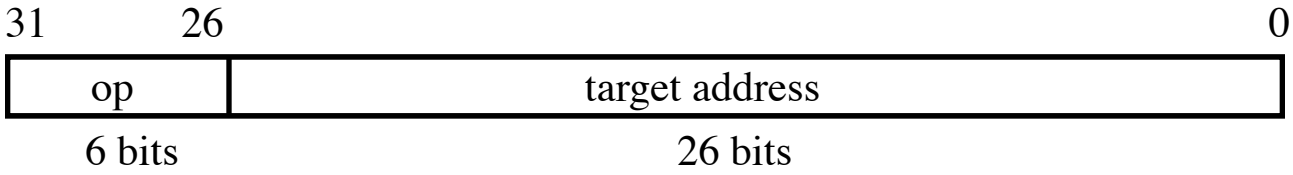
- R-type



- I-type



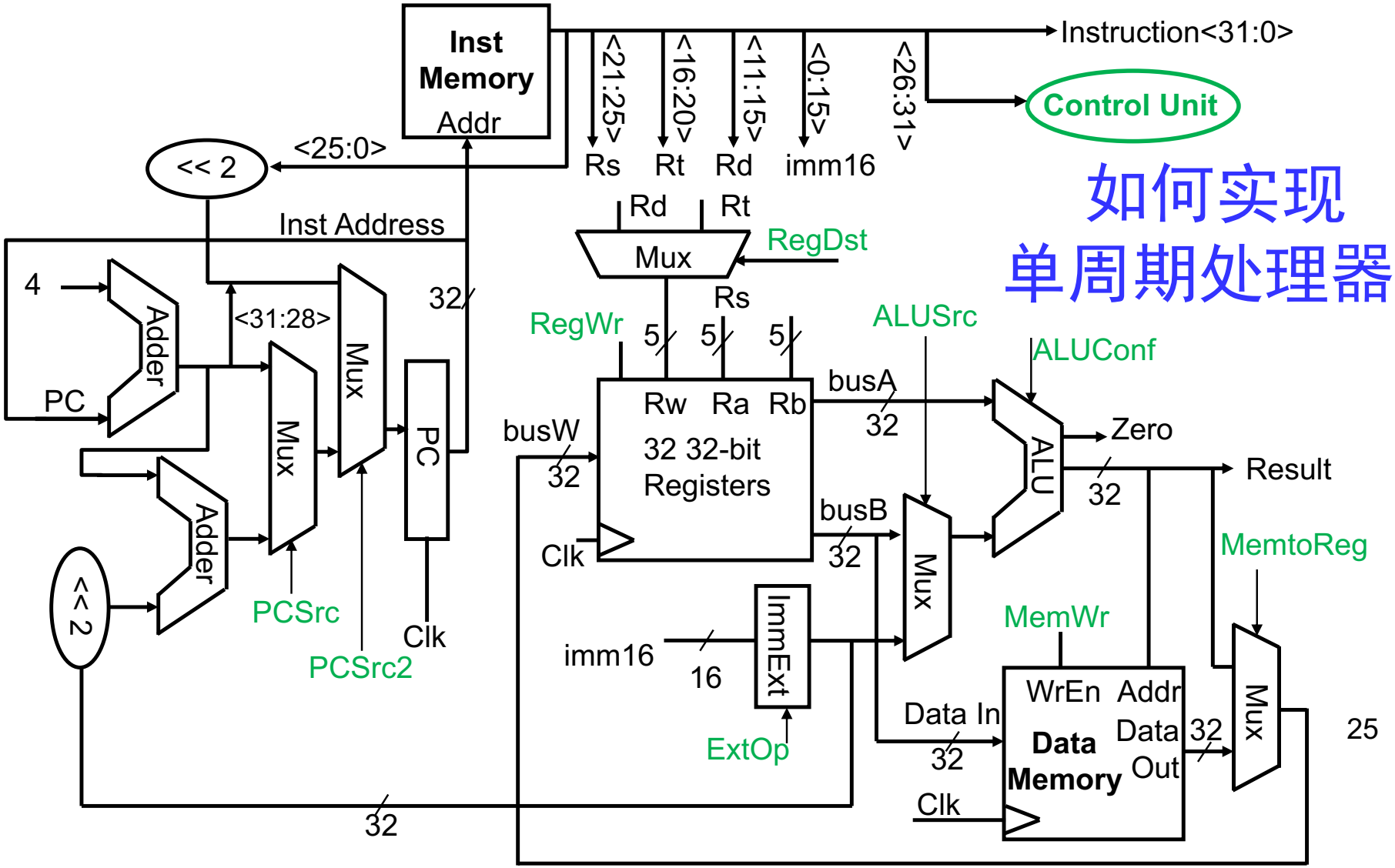
- J-type



各个域分别是

op	指令操作标识
rs, rt, rd	源,目的寄存器地址
shamt	移位量
funct	对op类操作，确定详细的操作类型
immediate	立即数或者地址
target address	目标地址

单周期处理器构成



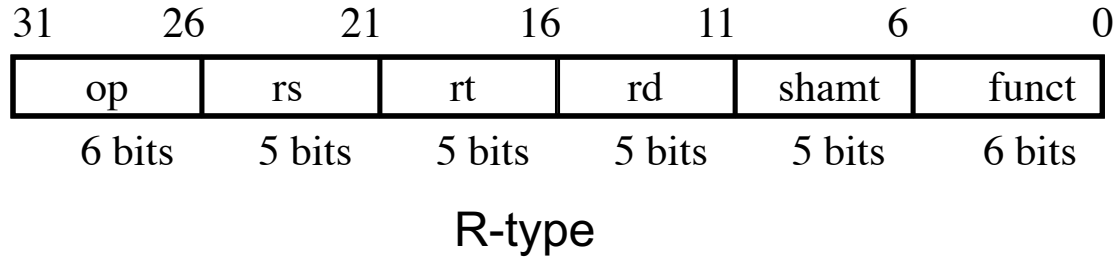
如何实现
单周期处理器？

目录

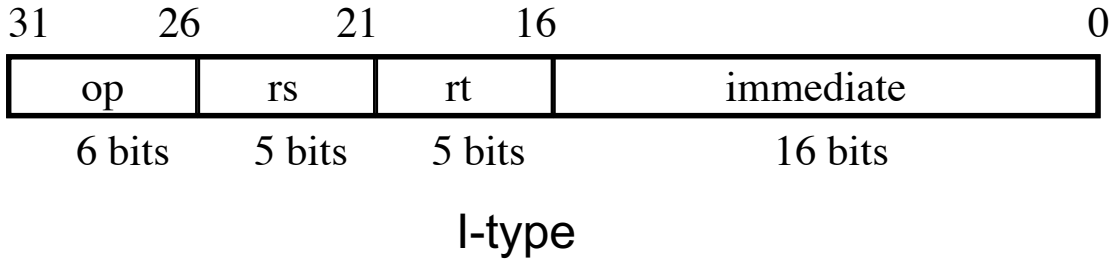
- 复习与回顾
- 单周期数据通路设计流程
  - 指令集和设计需求分析
  - 功能模块设计
  - 数据通路模块组装
  - 控制信号分析与逻辑设计
- 单周期数据通路延时分析
- 多周期数据通路
- 异常与中断

设计目标：MIPS指令子集

- ADD, SUB, AND, SLT
  - add rd, rs, rt
  - sub rd, rs, rt
  - and rd, rs, rt
  - slt rd, rs, rt
- OR Immediate:
  - ori rt, rs, imm16
- LOAD and STORE Word
  - lw rt, imm16(rs)
  - sw rt, imm16(rs)
- BRANCH and JUMP:
  - beq rs, rt, imm16
  - j Lable



R-type



I-type

J-type

根据指令集分析数据通路组成

存储单元：输入、输出和内部状态

- 存储器(Memory)
  - 一个指令存储器、一个数据存储器
- 寄存器(32 x 32bit Register) RF
  - 可以为 rs 提供数据读取
  - 可以为 rt 提供数据读取
  - 可以为 rt 或者 rd 提供数据写入
- 程序计数器(Program Counter, PC)

- ADD , SUB, AND, SLT
  - add rd, rs, rt
- OR Immediate:
  - ori rt, rs, imm16
- LOAD and STORE Word
  - lw rt, imm16(rs)
- BRANCH, JUMP
  - beq rs, rt, imm16
  - j Lable

根据指令集分析数据通路组成

计算单元（组合逻辑部分）

- 算术/逻辑运算
  - 两个寄存器之间或者是一个寄存器和一个经过扩展后的立即数之间
- 扩展电路
  - 为立即数提供扩展支持
- PC更新电路
  - 可以加4或者加上一个立即数扩展
  - 或者j跳转更新

- ADD , SUB, AND, SLT
  - add rd, rs, rt
- OR Immediate:
  - ori rt, rs, imm16
- LOAD and STORE Word
  - lw rt, imm16(rs)
- BRANCH, JUMP
  - beq rs, rt, imm16
  - j Lable

目录

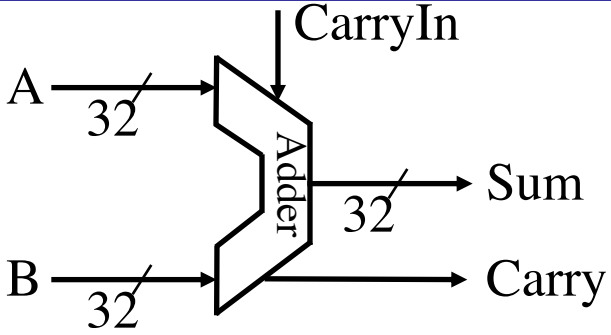
- 复习与回顾
- 单周期数据通路设计流程
  - 指令集和设计需求分析
  - 功能模块设计
  - 数据通路模块组装
  - 控制信号分析与逻辑设计
- 单周期数据通路延时分析
- 多周期数据通路
- 异常与中断

确定数据通路的组成模块

- 组合逻辑电路部分
  - “计算”为主，不同于单纯的“逻辑”
- 存储单元
  - “数据”为主，不同于单纯的“状态”

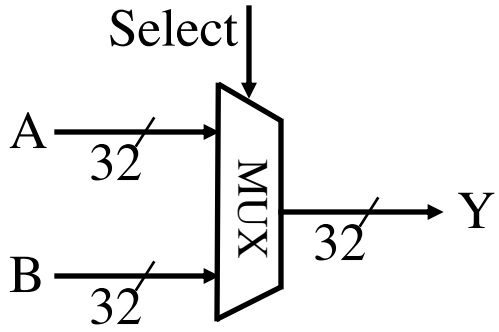
组合逻辑部分

- 加法器



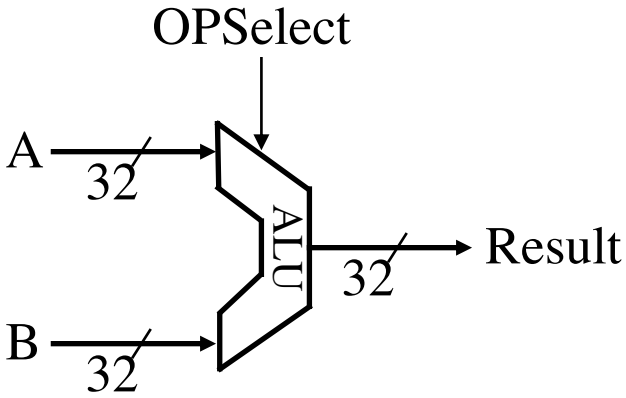
提供32位加法运算

- 多路选择器



提供多路输入选择

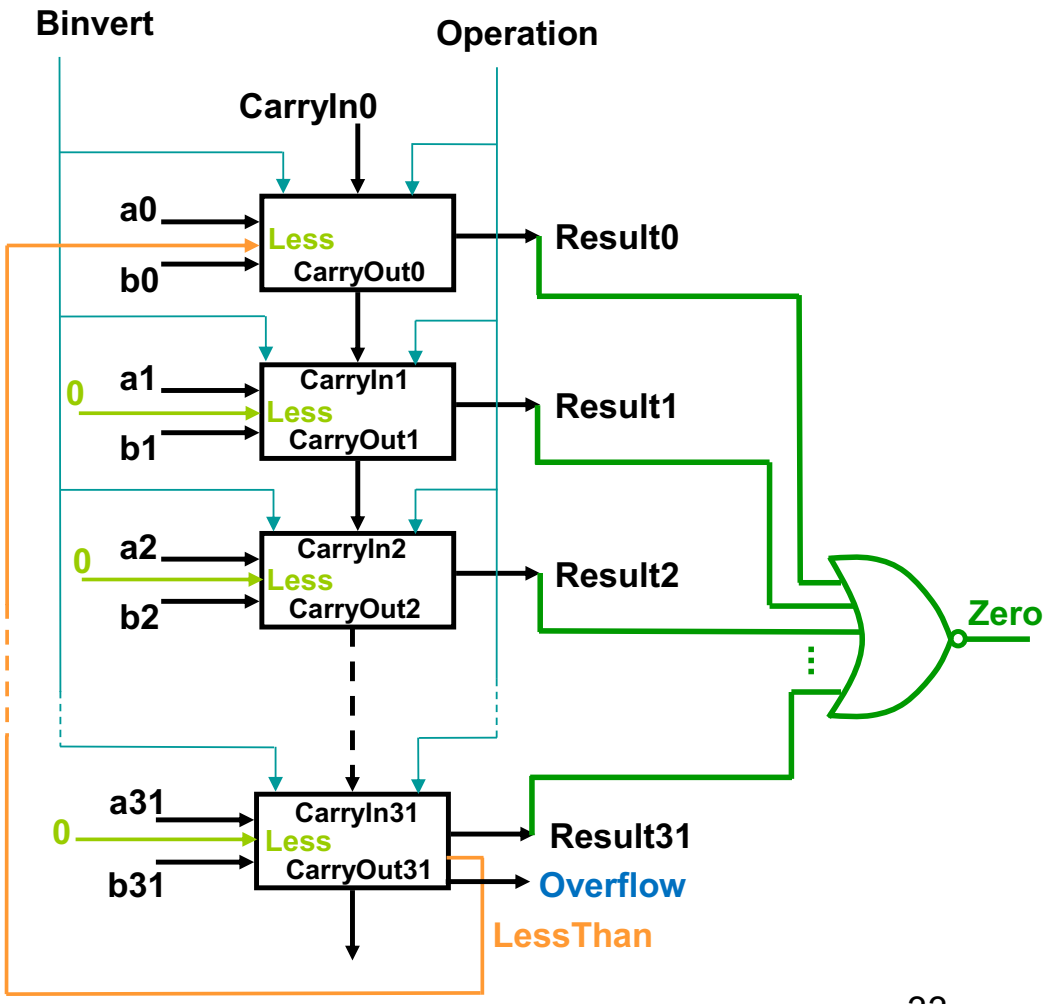
- 算术逻辑单元



实现算术逻辑运算

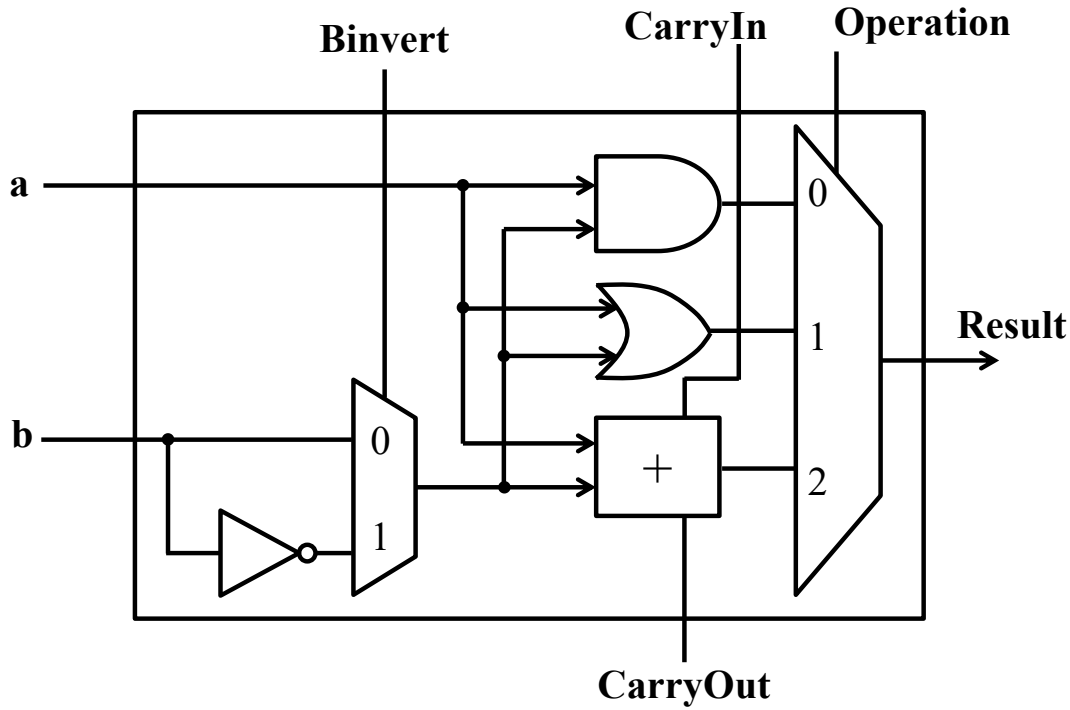
32位ALU设计

- 应具备的功能
  - 支持add/sub/and/or
  - 支持slt/beq
  - 支持溢出检测
  - LSB和MSB需要进行特殊处理



1位ALU电路

按位AND/OR



可实现与、或

1位ALU电路

按位AND/OR

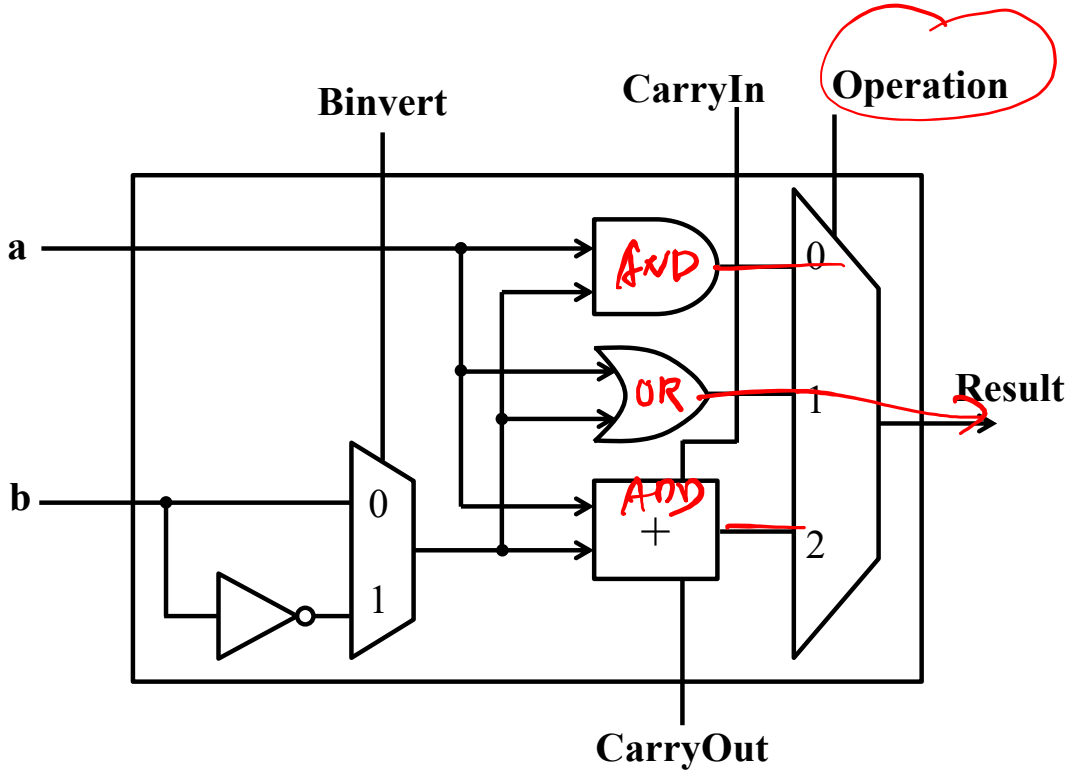
做加法：

Binvert=CarryIn=0

做减法：

Binvert=？ CarryIn=？

回顾补码的概念！



可实现与、或、加法

1位ALU电路

按位AND/OR

做加法：

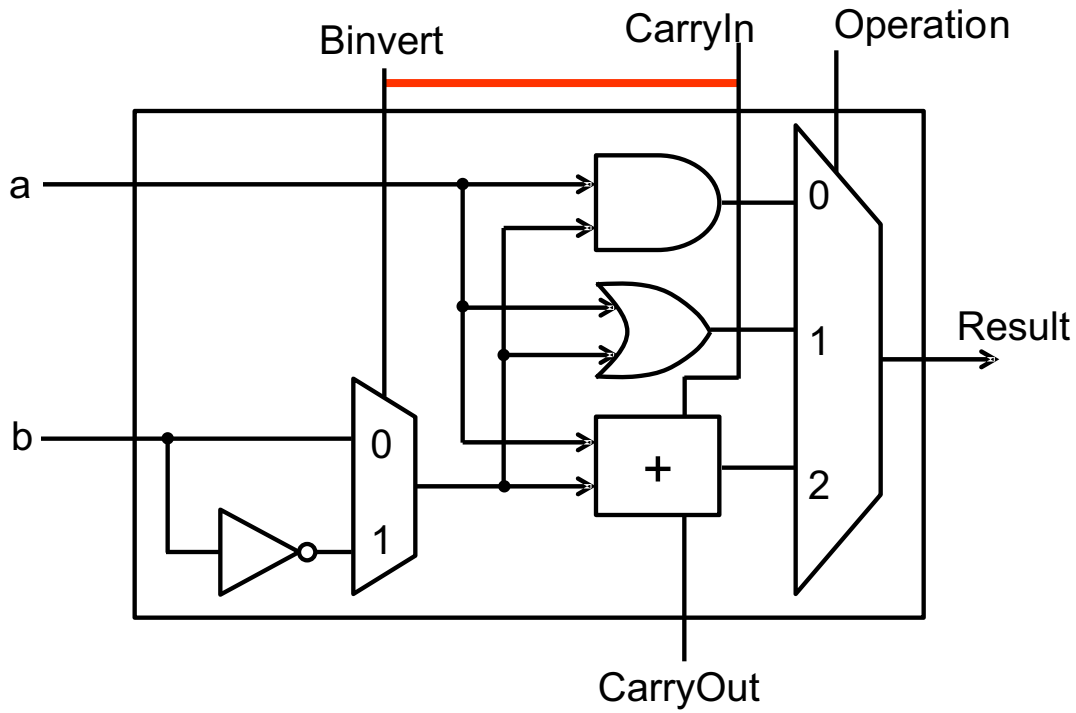
Binvert=CarryIn=0

做减法：

Binvert=？ CarryIn=？

Binvert=CarryIn=1

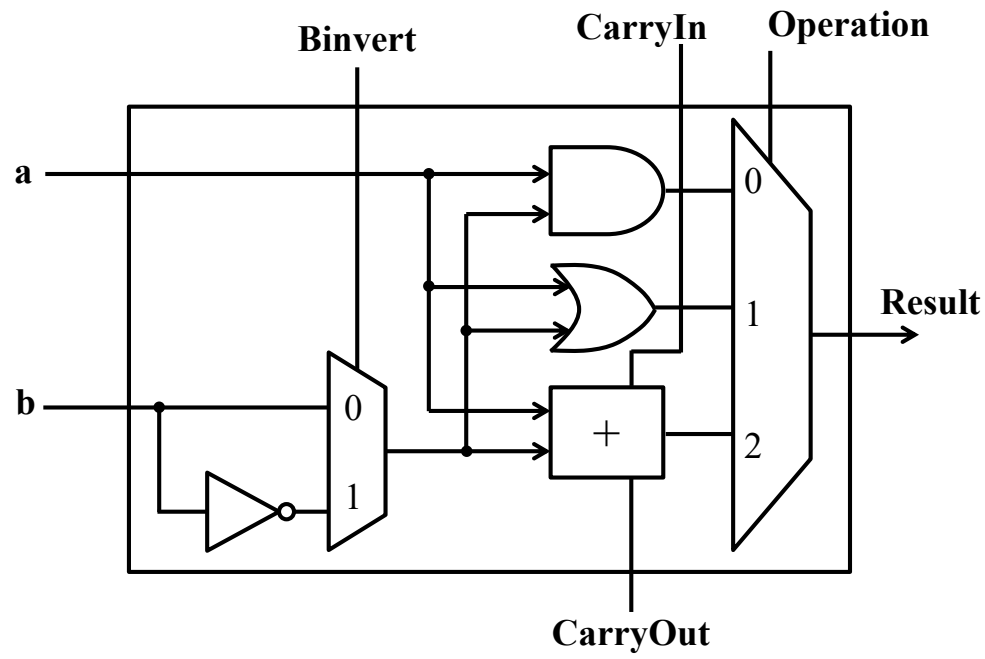
Bnegate：控制简化



可实现与、或、加/减法

多选题

1位ALU如何实现：a异或b操作？（多选题）



a. Binvert=1, CarryIn=0, Operation = 1

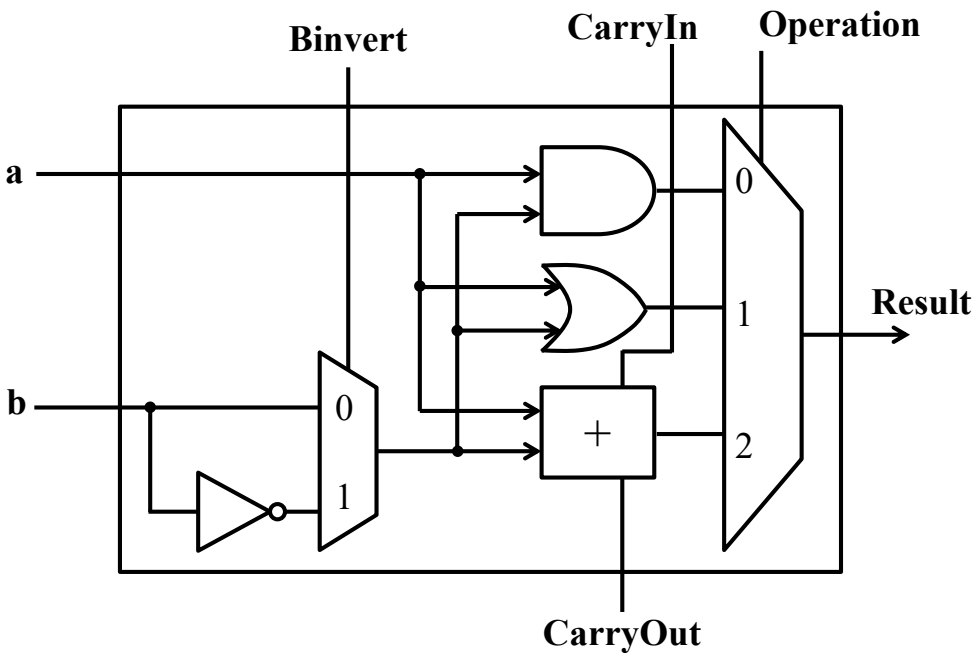
b. Binvert=1, CarryIn=1, Operation = 2

c. Binvert=0, CarryIn=0, Operation = 2

d. Binvert=0, CarryIn=X, Operation = 0

答案

1位ALU如何实现：a异或b操作？（多选题）



a. Binvert=1, CarryIn=0, Operation = 1

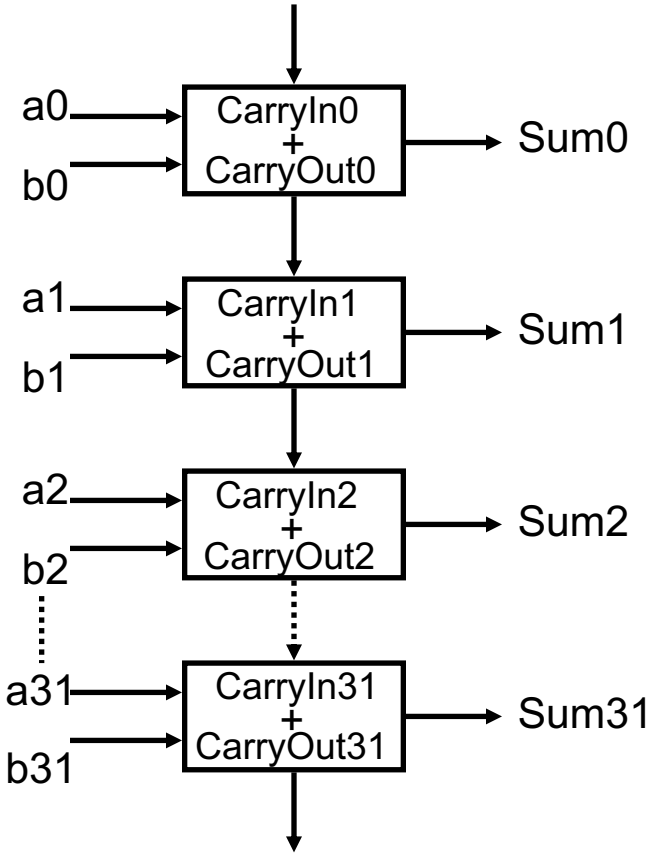
b. （正确）Binvert=1, CarryIn=1, Operation = 2

c. （正确） Binvert=0, CarryIn=0, Operation = 2

d. Binvert=0, CarryIn=X, Operation = 0

分析：a⊕b即(a±b)去掉符号位

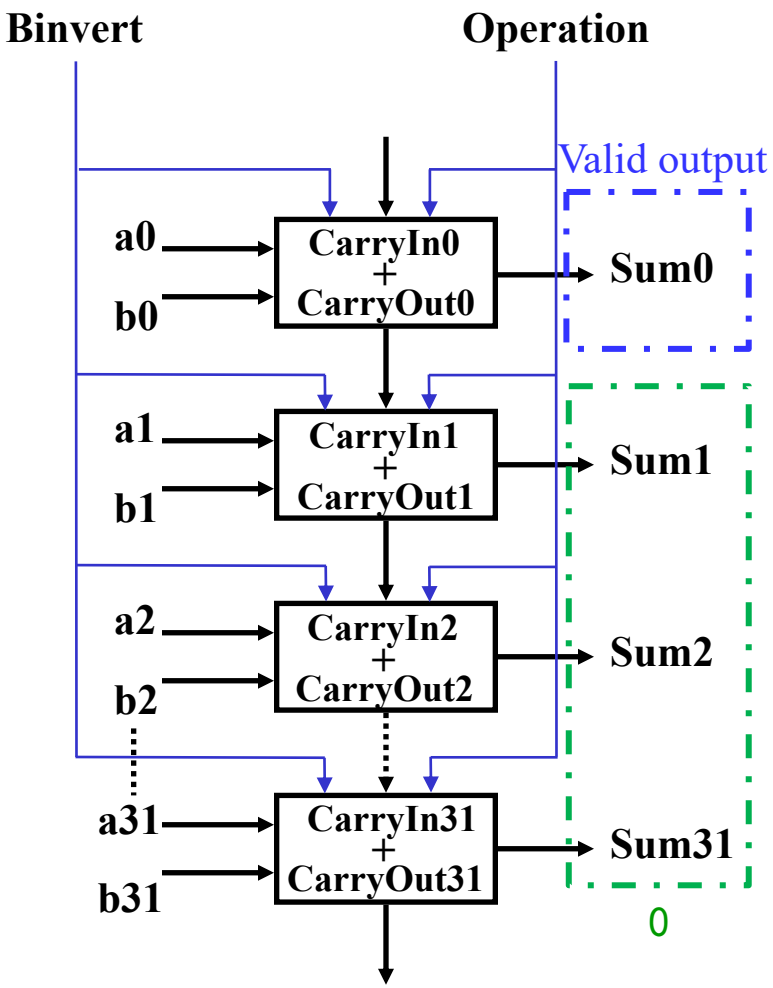
N位加法器



行波进位加法器
(ripple carry adder)

- 大多数MIPS指令都可以用这个ALU实现
- 能实现Slt，Beq指令吗？

32-bit ALU: Slt

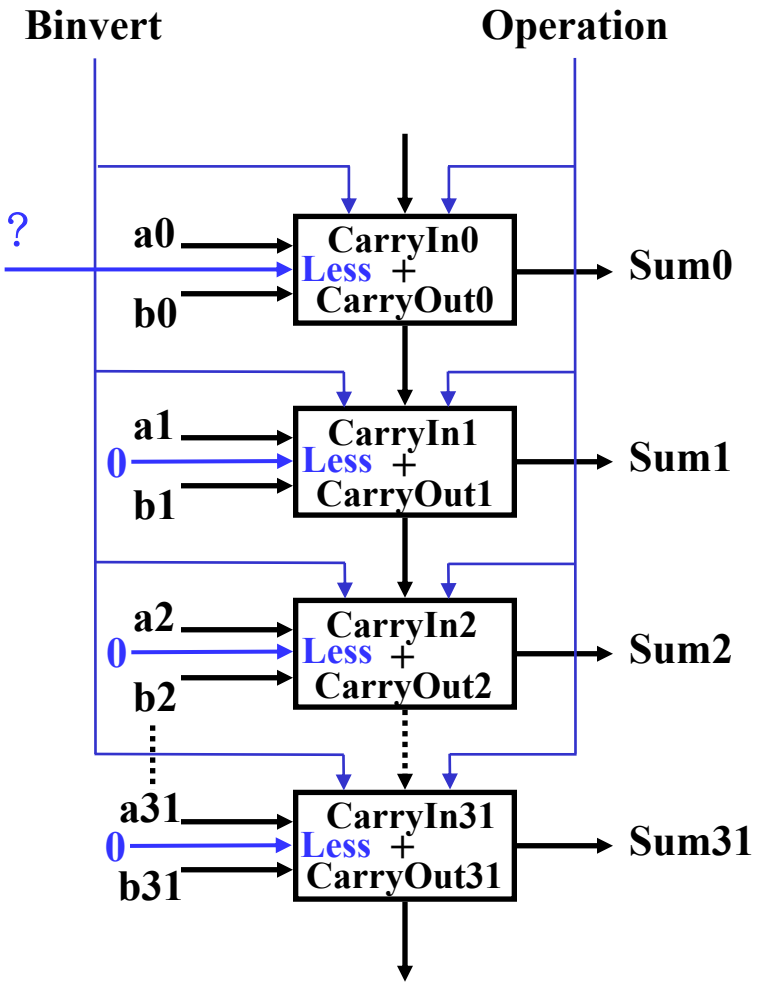


Slt: rd = rs < rt ? 1 : 0;

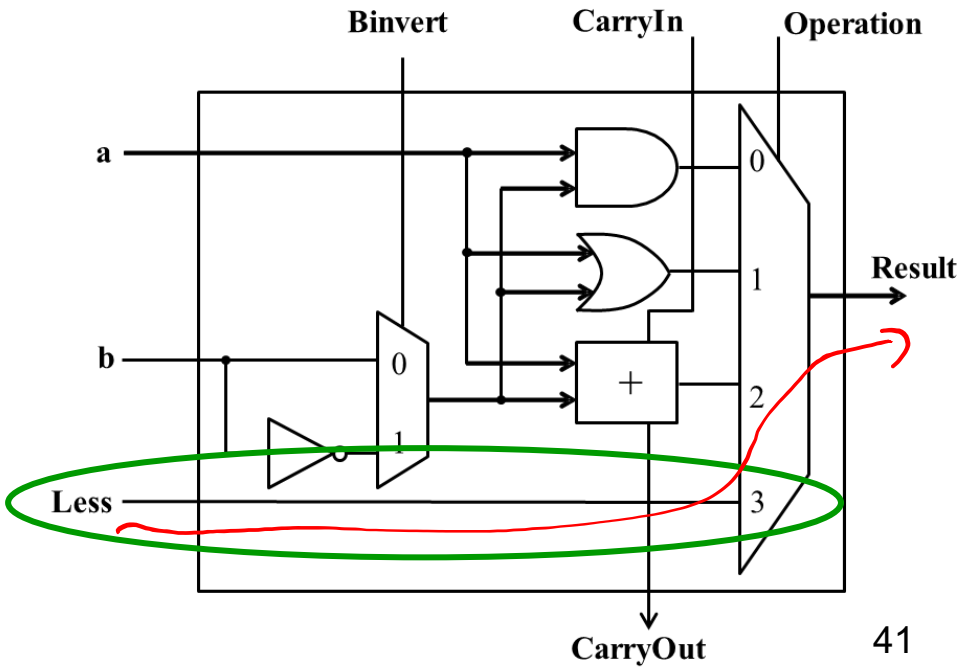
assign a=rs, b=rt
若 a-b<0, 则符号位为 1
若 a-b>0, 则符号位为 0

只有比特0需要计算，其他位(1-31)置为0;
ALU需要额外的Slt输出: a-b的符号位

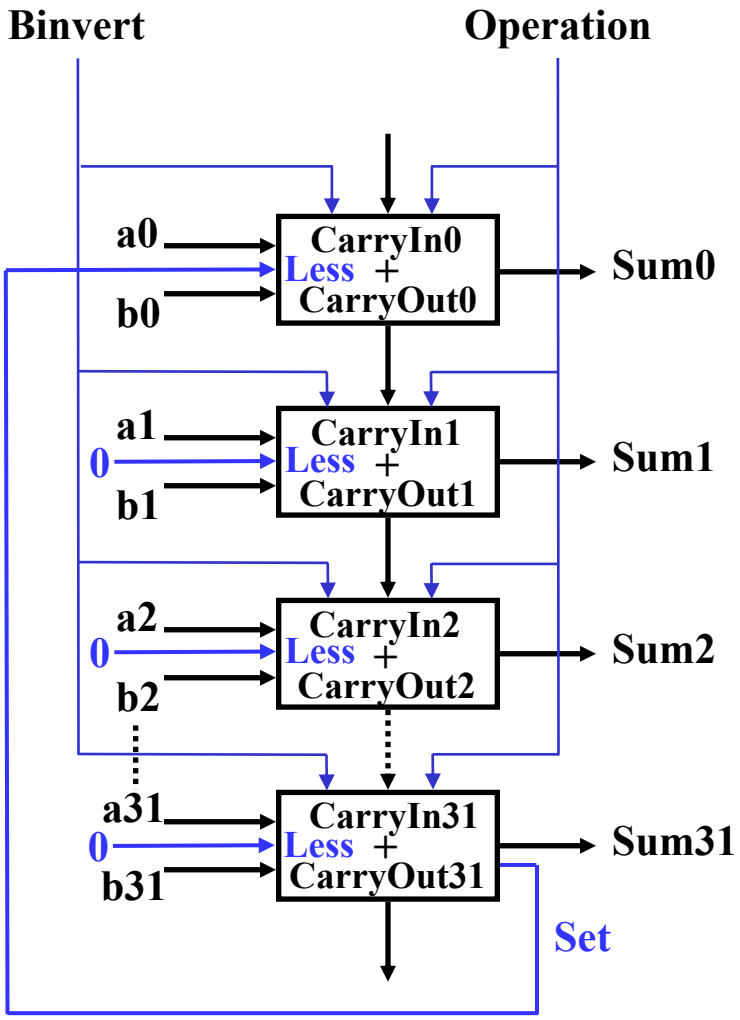
32-bit ALU: Slt



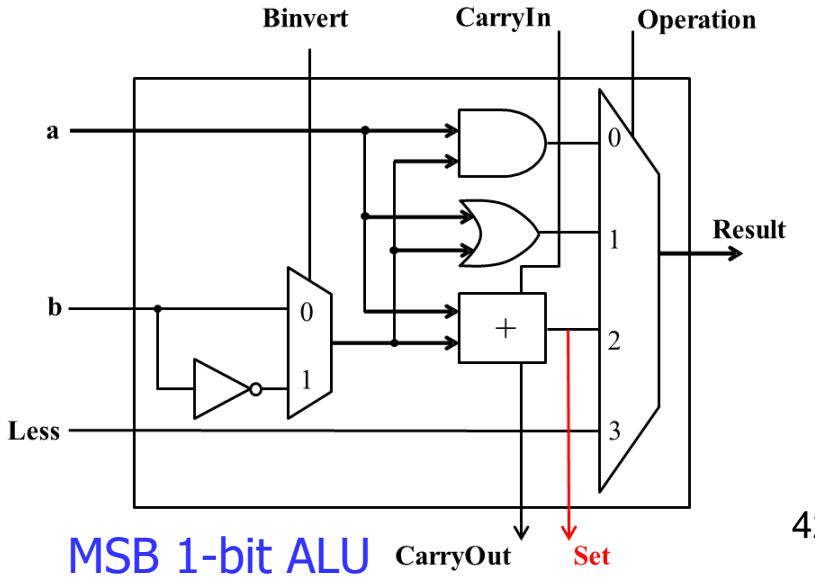
只有比特位0需要计算;
其他位(1-31)置为0



32-bit ALU: Slt



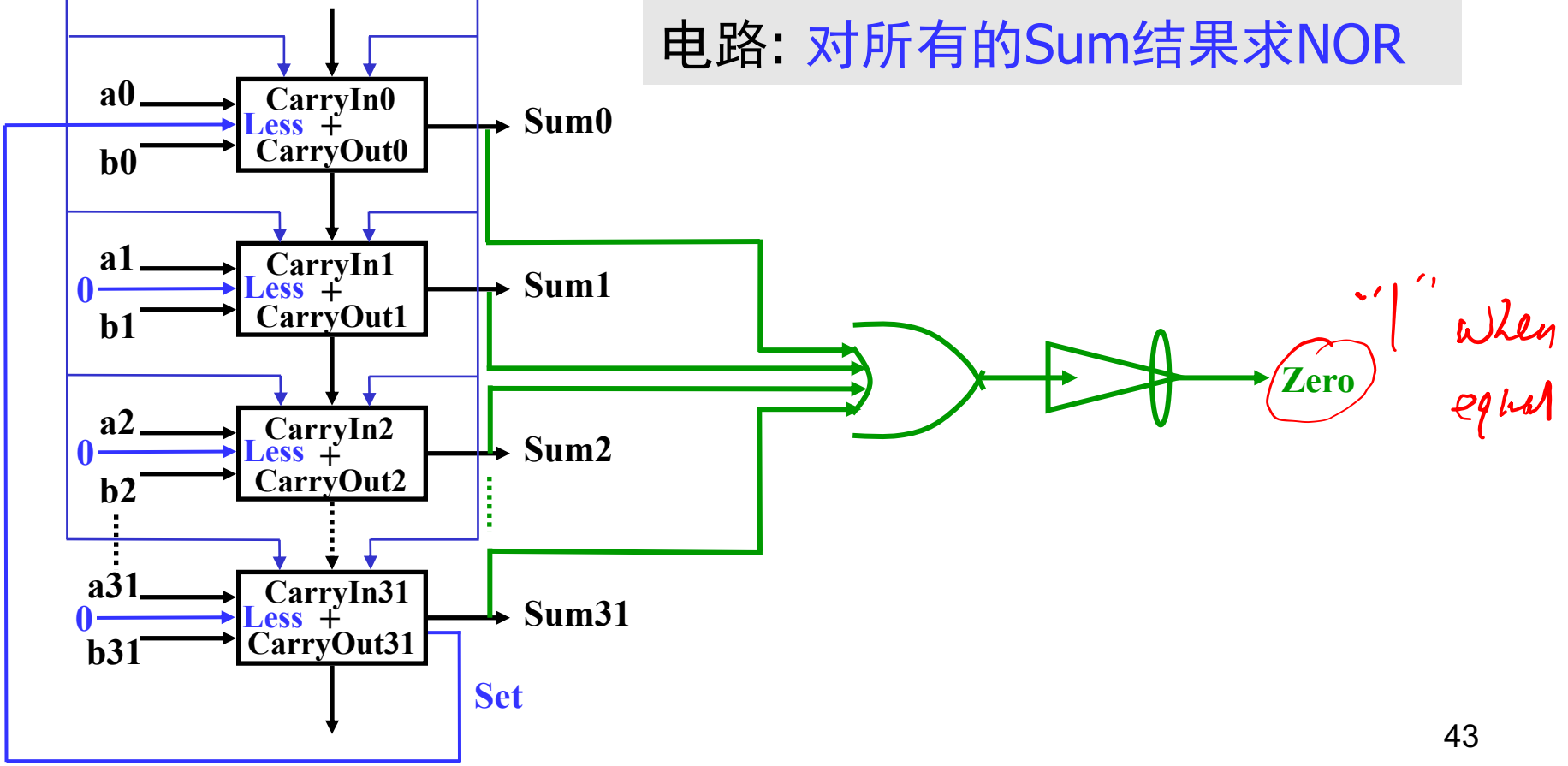
Slt的输出是a-b的符号位:
使用输出符号位来重置最终输出比特



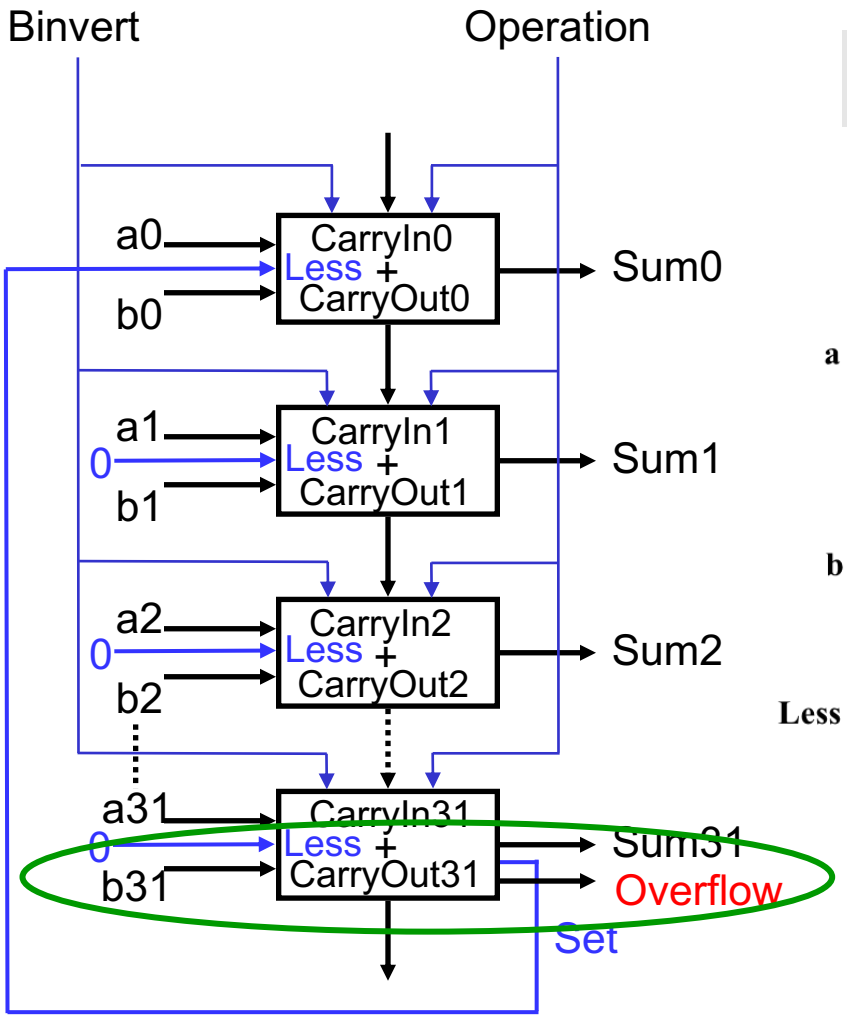
32-bit ALU: Beq

beq: Zero = (a-b == 0) ? 1 : 0;

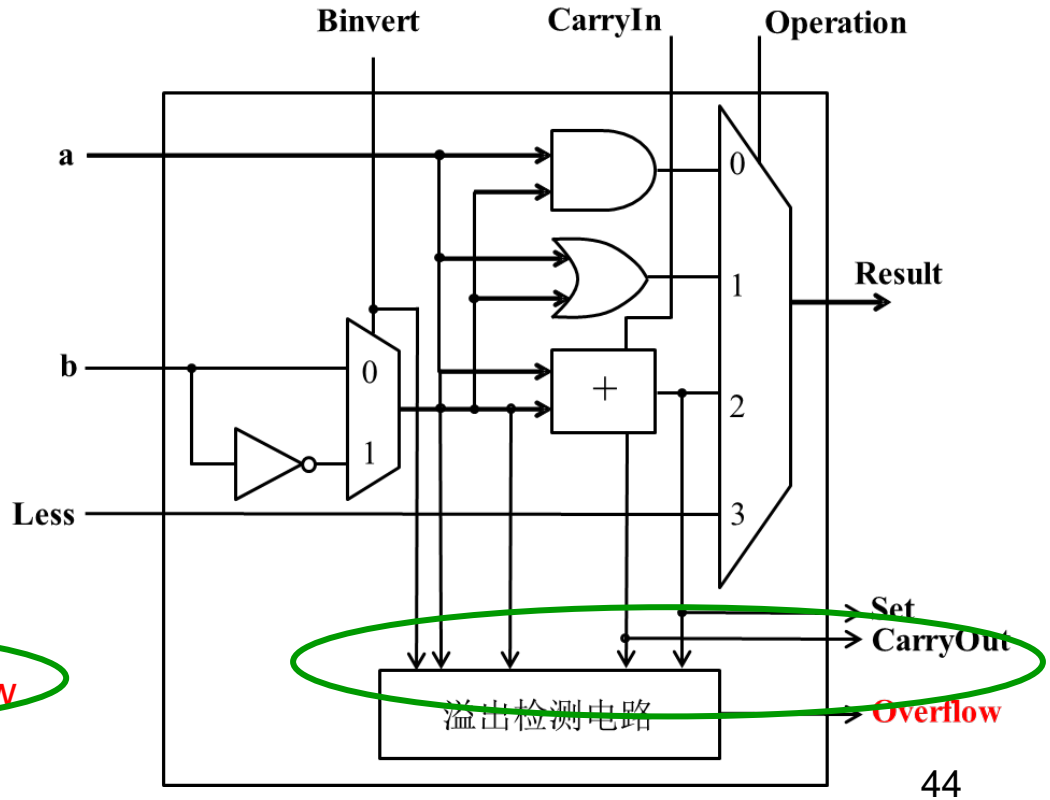
电路: 对所有的Sum结果求NOR



32位ALU：溢出检测



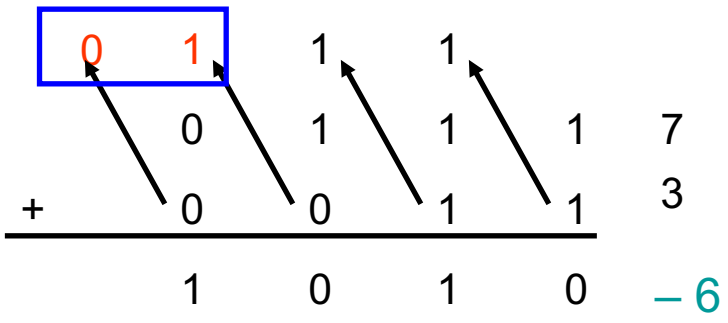
add,sub中的溢出检测（异常）

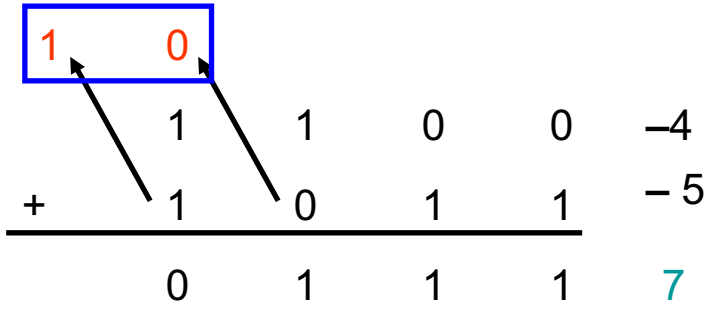


ALU电路—— 溢出检测

- Add, Sub 应支持溢出检测
- 溢出可以用如下方法检测:

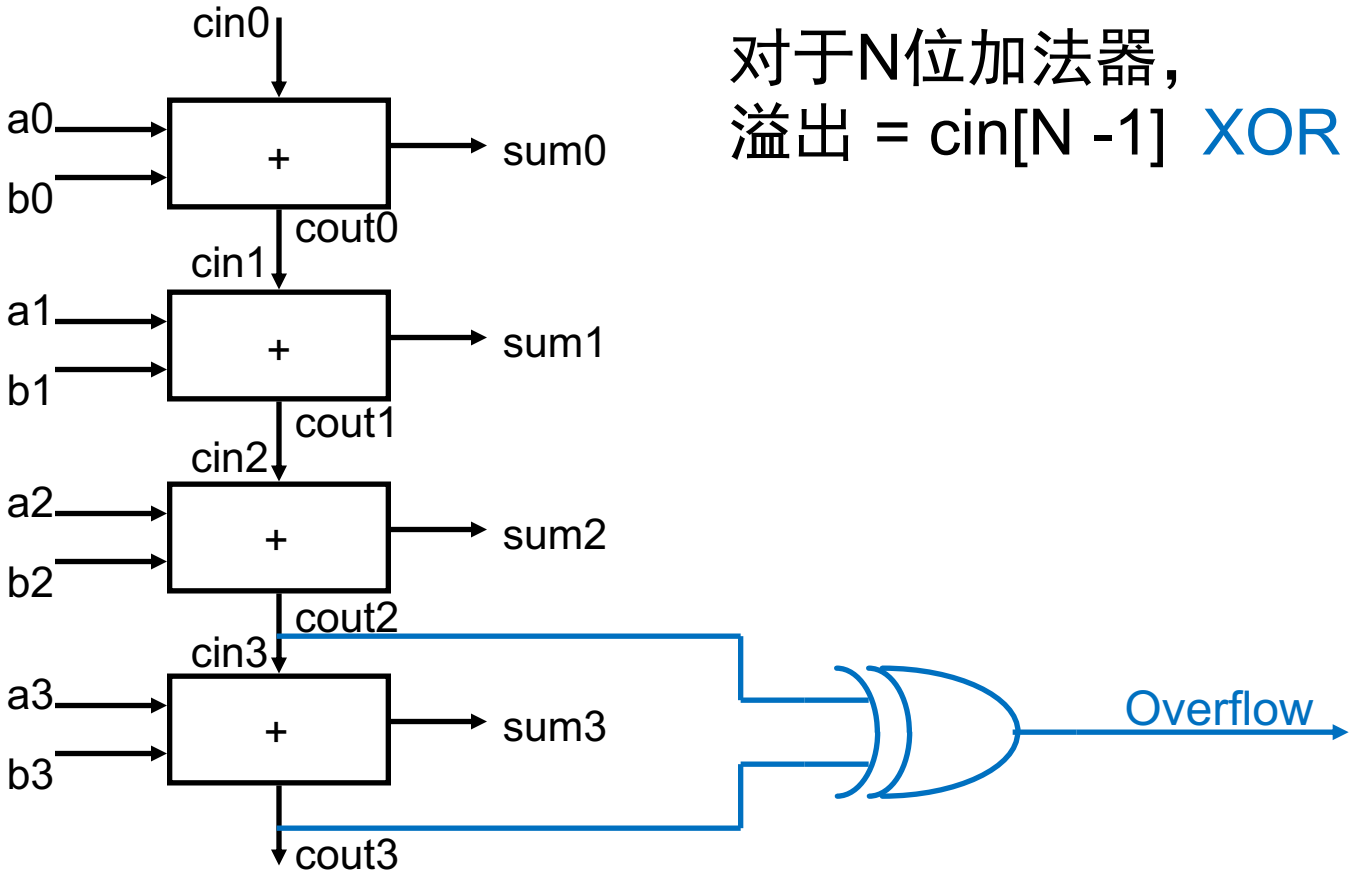
  输入到最高位的进位!= 从最高位输出的进位
  （Carry into MSB != Carry out of MSB ）





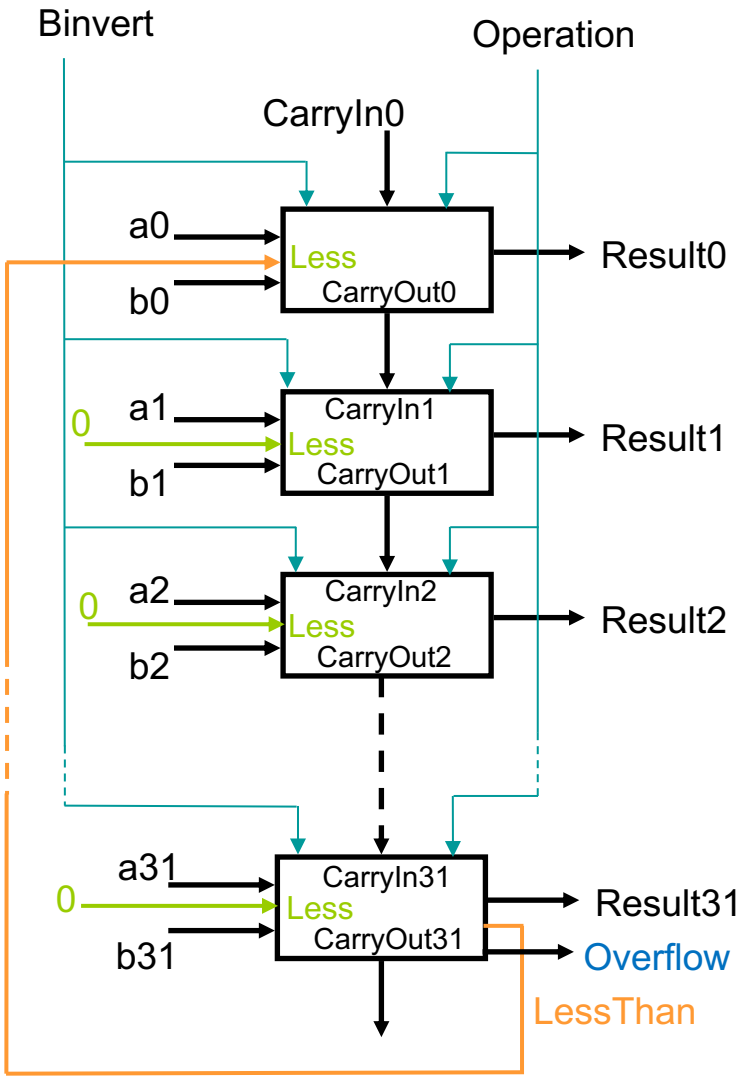
ALU电路—— 溢出检测

溢出检测逻辑

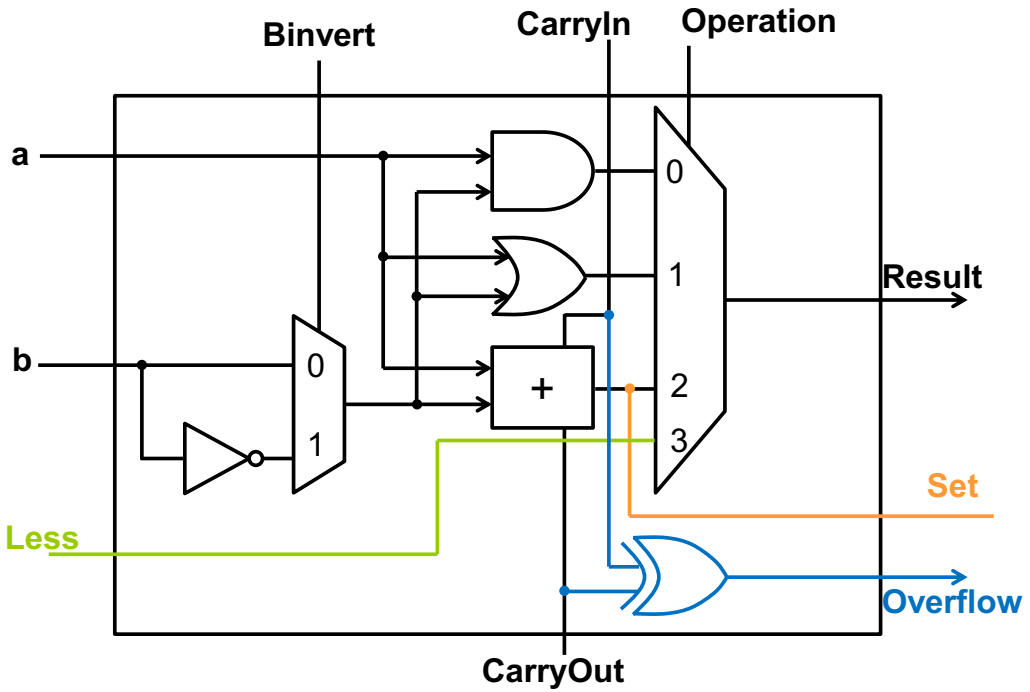


对于N位加法器，
溢出 = cin[N -1] XOR cout[N -1]

ALU电路—— 溢出检测



Add, sub应支持溢出检测，输出overflow



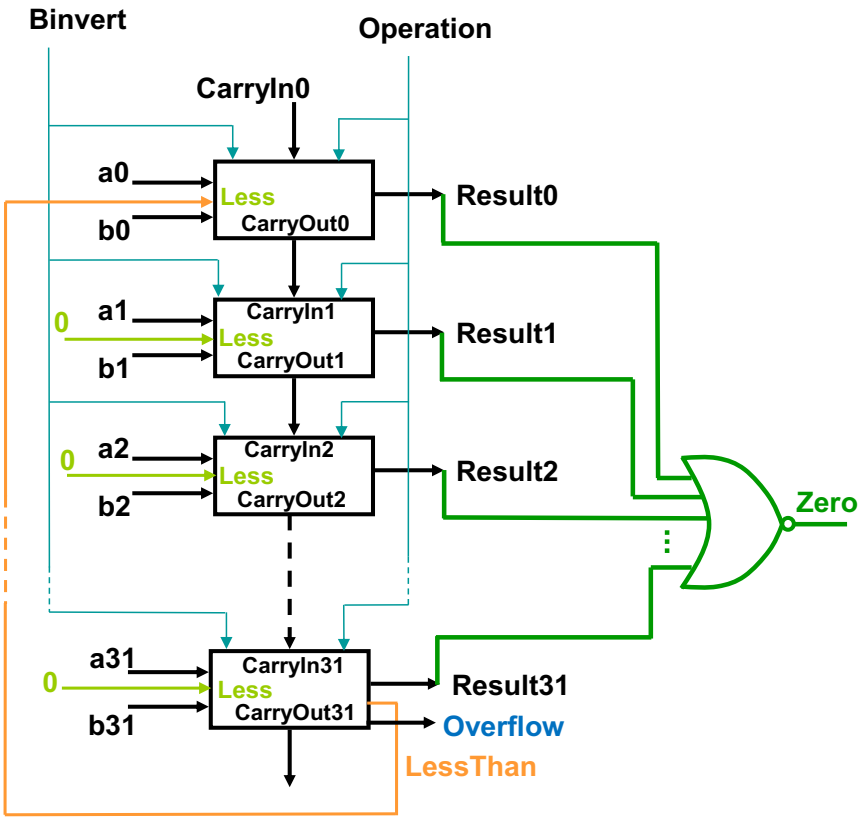
ALU电路—— 溢出检测

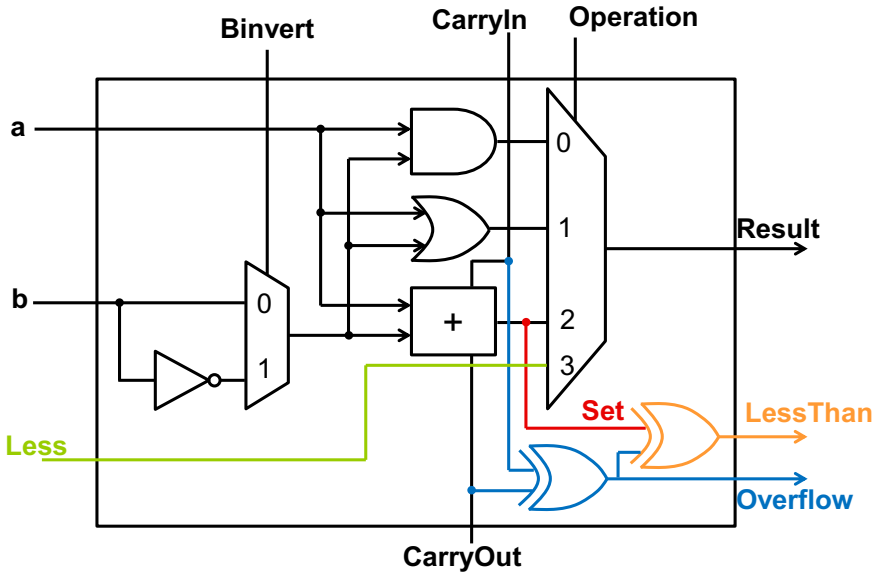
Overflow对Slt的影响

Overflow	Result31	LessThan
0	0	0
0	1	1
1	0	1
1	1	0

$$LessThan = Overflow \oplus Result31$$

ALU电路—— 溢出检测


Binvert
Operation
CarryIn0
a0
b0
Less
CarryOut0
Result0
0
a1
b1
CarryIn1
Less
CarryOut1
Result1
0
a2
b2
CarryIn2
Less
CarryOut2
Result2
Zero
a31
b31
0
CarryIn31
Less
CarryOut31
Result31
Overflow
LessThan


Binvert
CarryIn
Operation
a
b
Less
0
1
+
0
1
2
3
Result
Set
LessThan
Overflow
CarryOut

32位RISC ALU设计

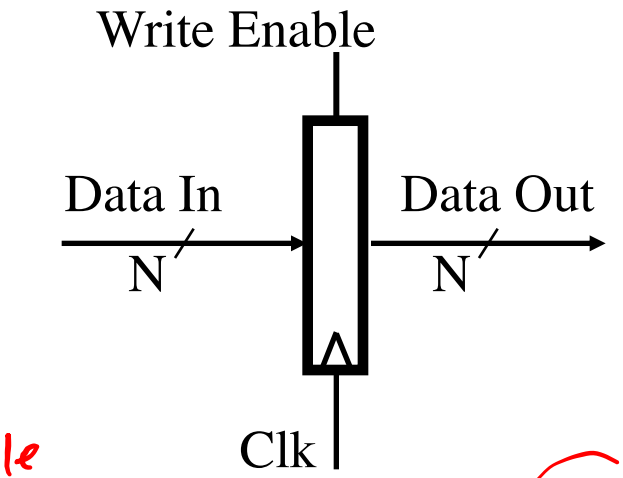
- ALU还有什么？
  - 移位运算…
- 更高效的ALU设计
  - 超前进位加法器…

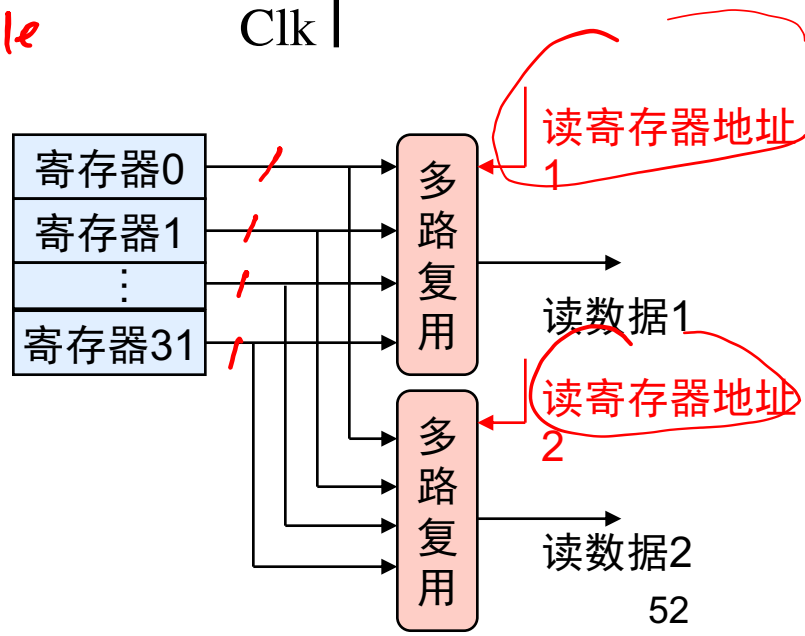
确定数据通路的组成模块

- 组合逻辑电路部分
  - “计算”为主，不同于单纯的“逻辑”
- 存储单元
  - “数据”为主，不同于单纯的“状态”

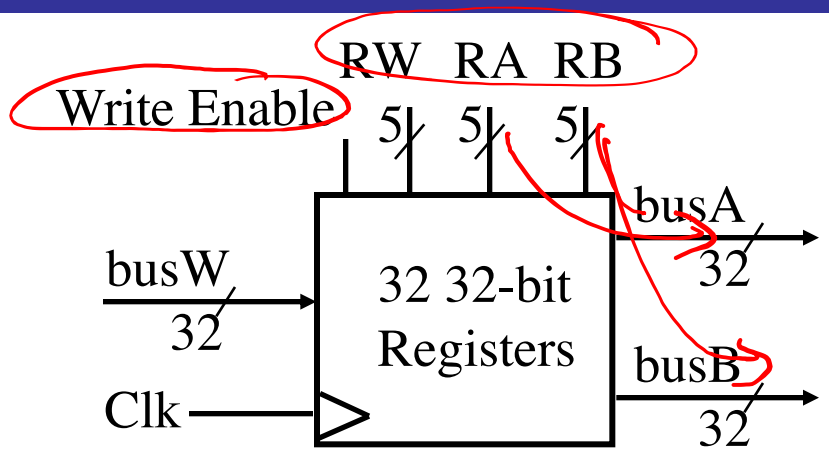
存储单元——寄存器

- 寄存器(register)
  - N个D触发器组成
- 32个寄存器构成寄存器堆 RF file
  - 可以同时从两个口读取
  - 一个写端口，有写入使能
- 写入使能:
  - 禁止时(0)： 寄存器内容不变
  - 使能时(1)： 改变



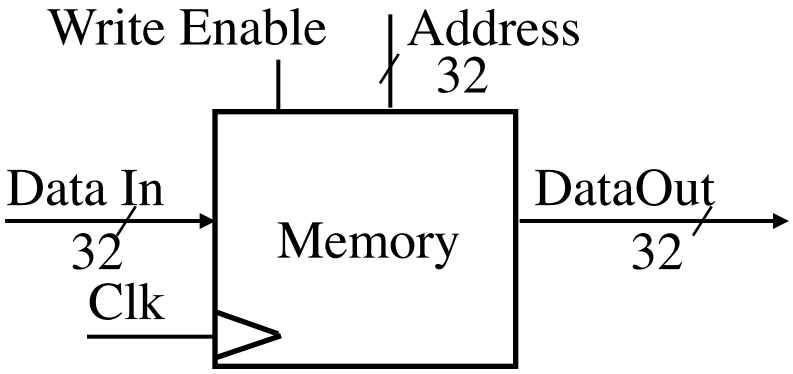


存储单元——寄存器堆



- 寄存器堆包含 32 registers:
  - 两个 32-bit 输出总线:
    busA and busB
  - 一个32-bit 输入总线: busW
- 寄存器堆中的寄存器选择:
  - RA (number) 选出哪个32位寄存器的数据放在busA (data)
  - RB (number)选出哪个32位寄存器的数据放在busB (data)
  - RW (number)选出哪个32位寄存器的数据将通过busW被写入(当WE=1时)
- 时钟 (CLK)
  - CLK 只在写入时发挥作用
  - 读取电路总能读到值
    - RA or RB valid => busA or busB valid after “access time.”

存储单元——存储器



- 存储器(memory): 容量大
  - 一个输入总线: 数据输入
  - 一个输出总线: 数据输出
- 存储器选择:
  - 通过地址线Address 选择哪个32位字被放在数据输出总线上
  - Write Enable = 1: 通过address选择哪个地址的32位字写为输入总线的数据
- Clock input (CLK)
  - CLK input 只在写入有效
  - 在读出时, 类似于组合逻辑:
    - Address valid => Data Out valid after “access time.”

目录

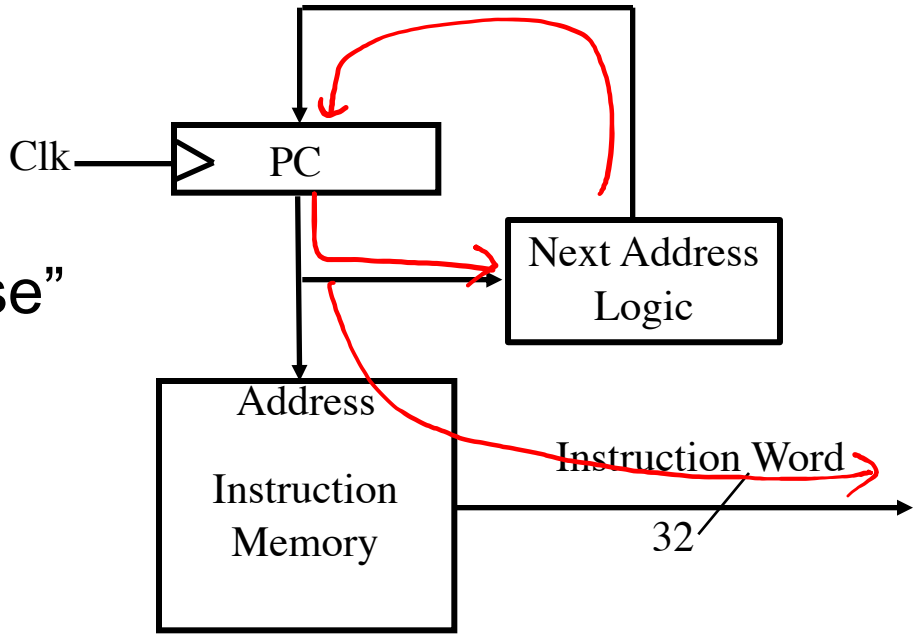
- 复习与回顾
- 单周期数据通路设计流程
  - 指令集和设计需求分析
  - 功能模块设计
  - 数据通路模块组装
  - 控制信号分析与逻辑设计
- 单周期数据通路延时分析
- 多周期数据通路
- 异常与中断

取指令单元IF

- 所有指令都公用的单元
  - Fetch the Instruction: Inst_Mem[PC]
  - 更新PC: Next Address Logic
    - 顺序指令时:

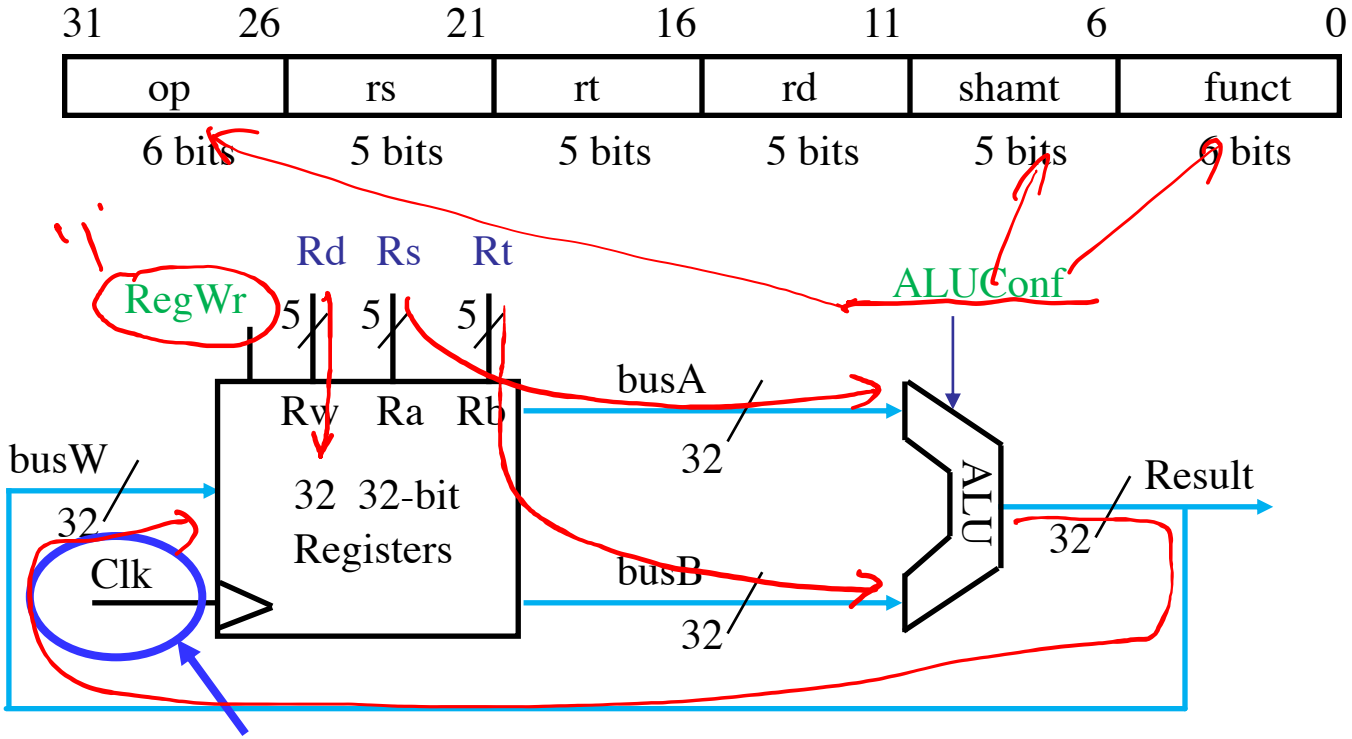
      PC <= PC + 4
    - 分支跳转时:

      PC <= “something else”



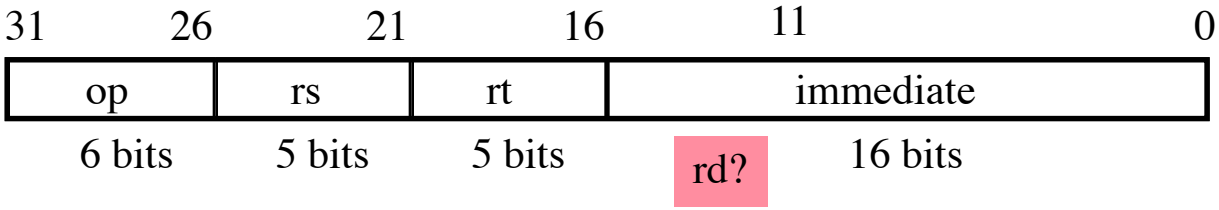
加减法等运算

- R[rd] <= R[rs] op R[rt]
  例子: addu rd, rs, rt
  - Ra, Rb, and Rw 来自指令自中的rs, rt, 和 rd
  - ALUConf and RegWr: 根据指令译码确定的控制信号

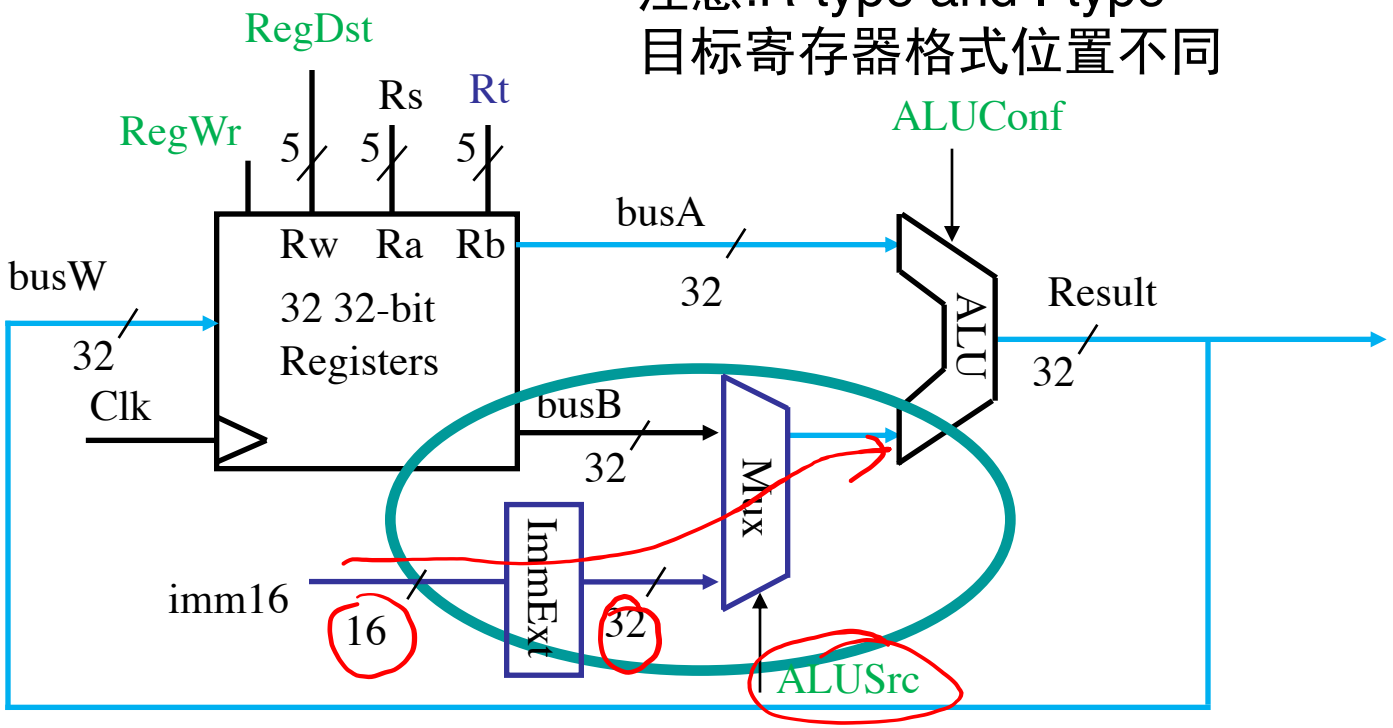


立即数操作

- R[rt] <= R[rs] op ZeroExt[imm16]

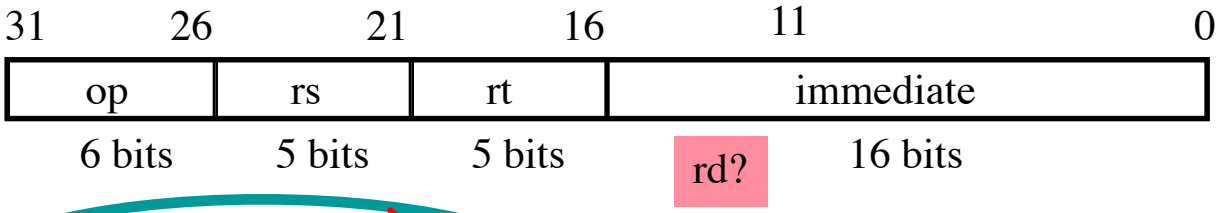


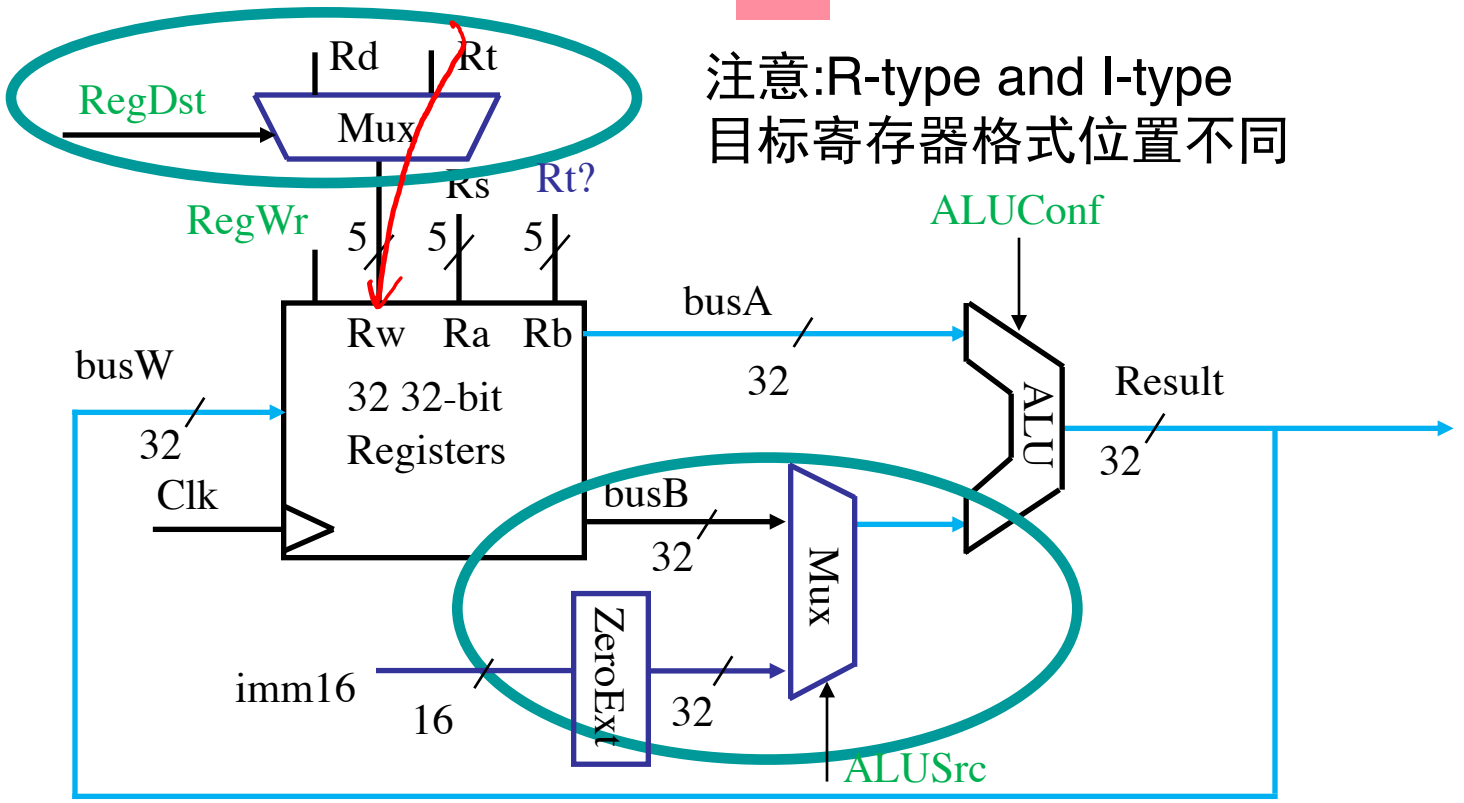
注意:R-type and I-type
目标寄存器格式位置不同



立即数操作

- R[rt] <= R[rs] op ZeroExt[imm16]

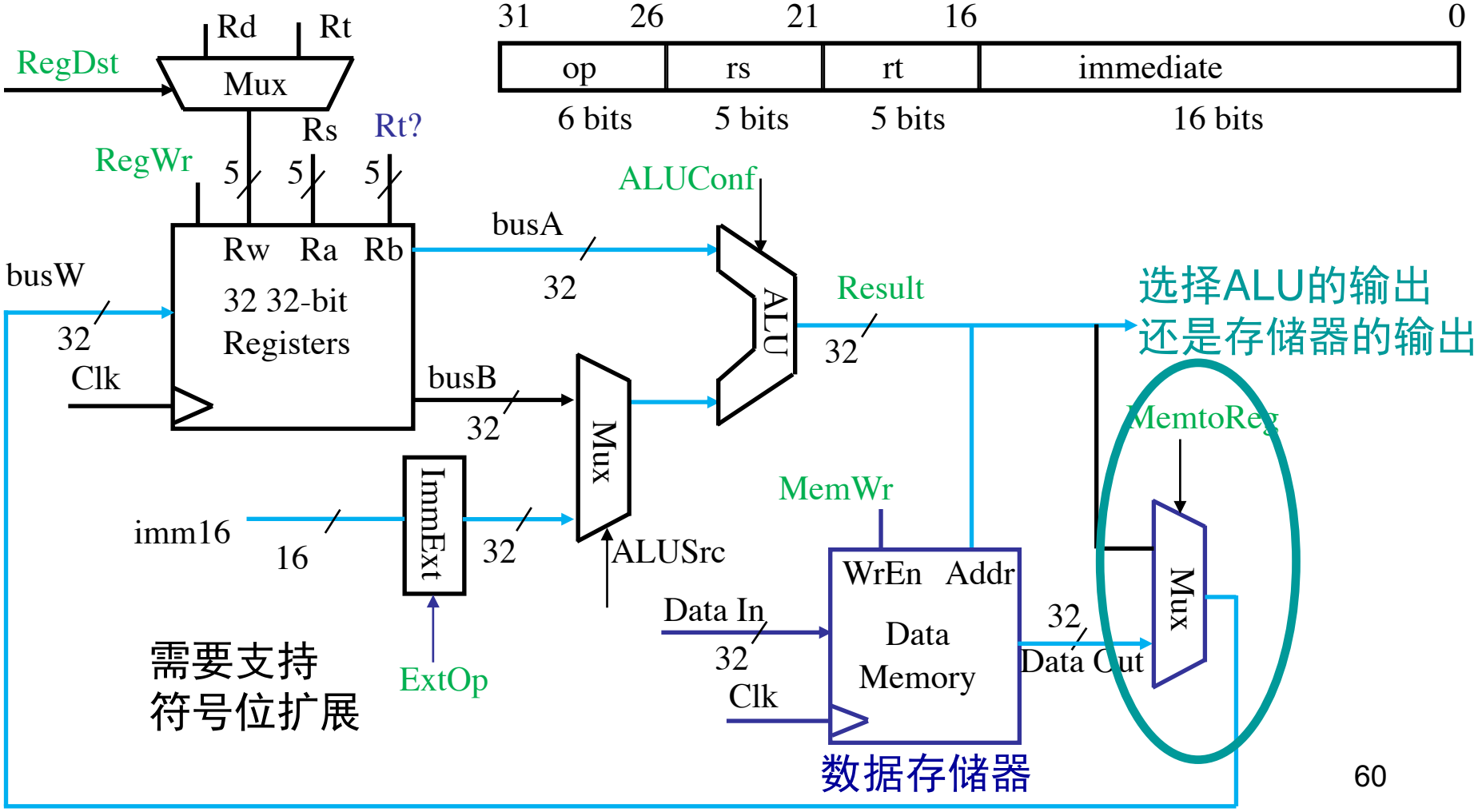




注意:R-type and I-type
目标寄存器格式位置不同

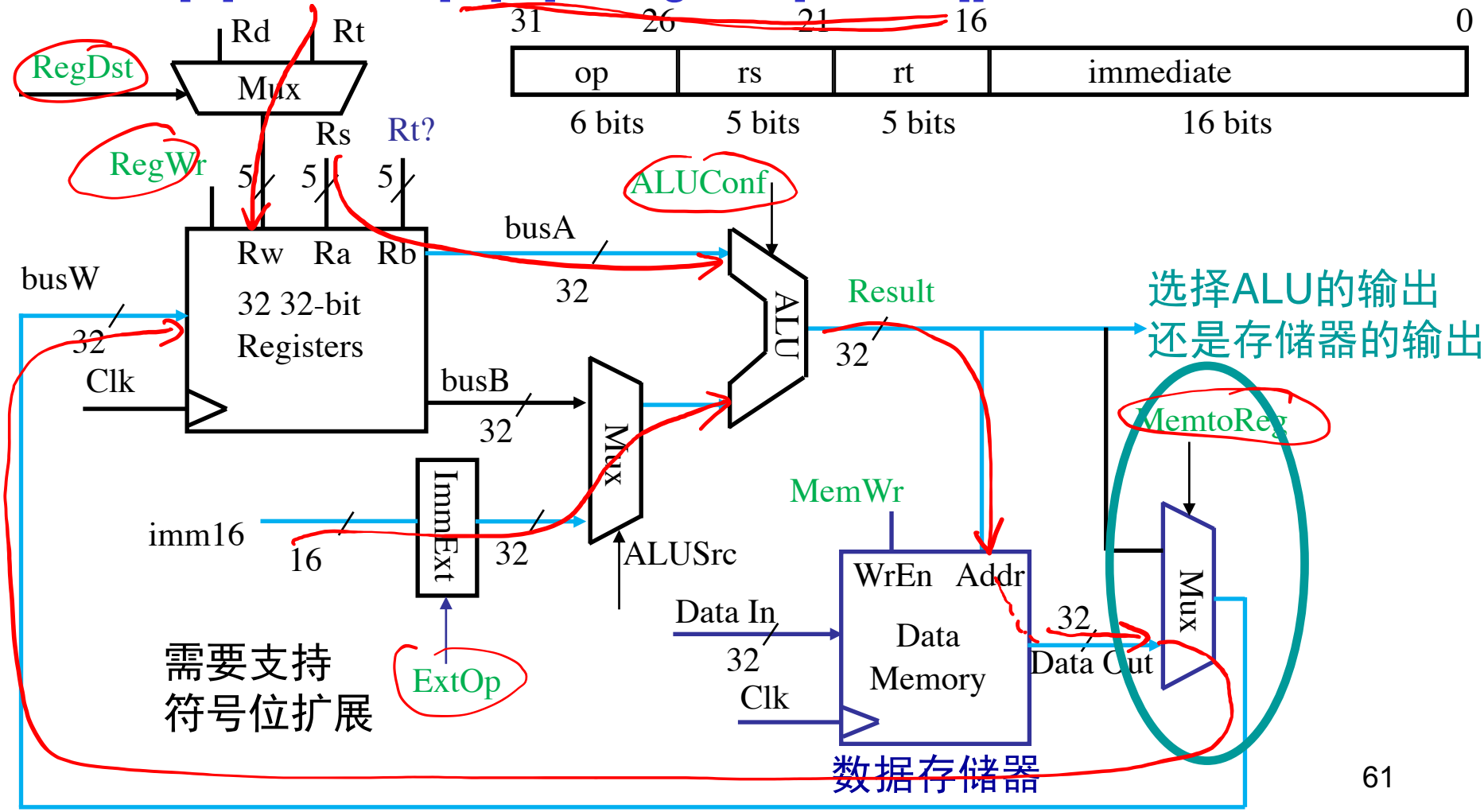
Load Word 操作

- R[rt] <= Mem[R[rs] + SignExt[imm16]]



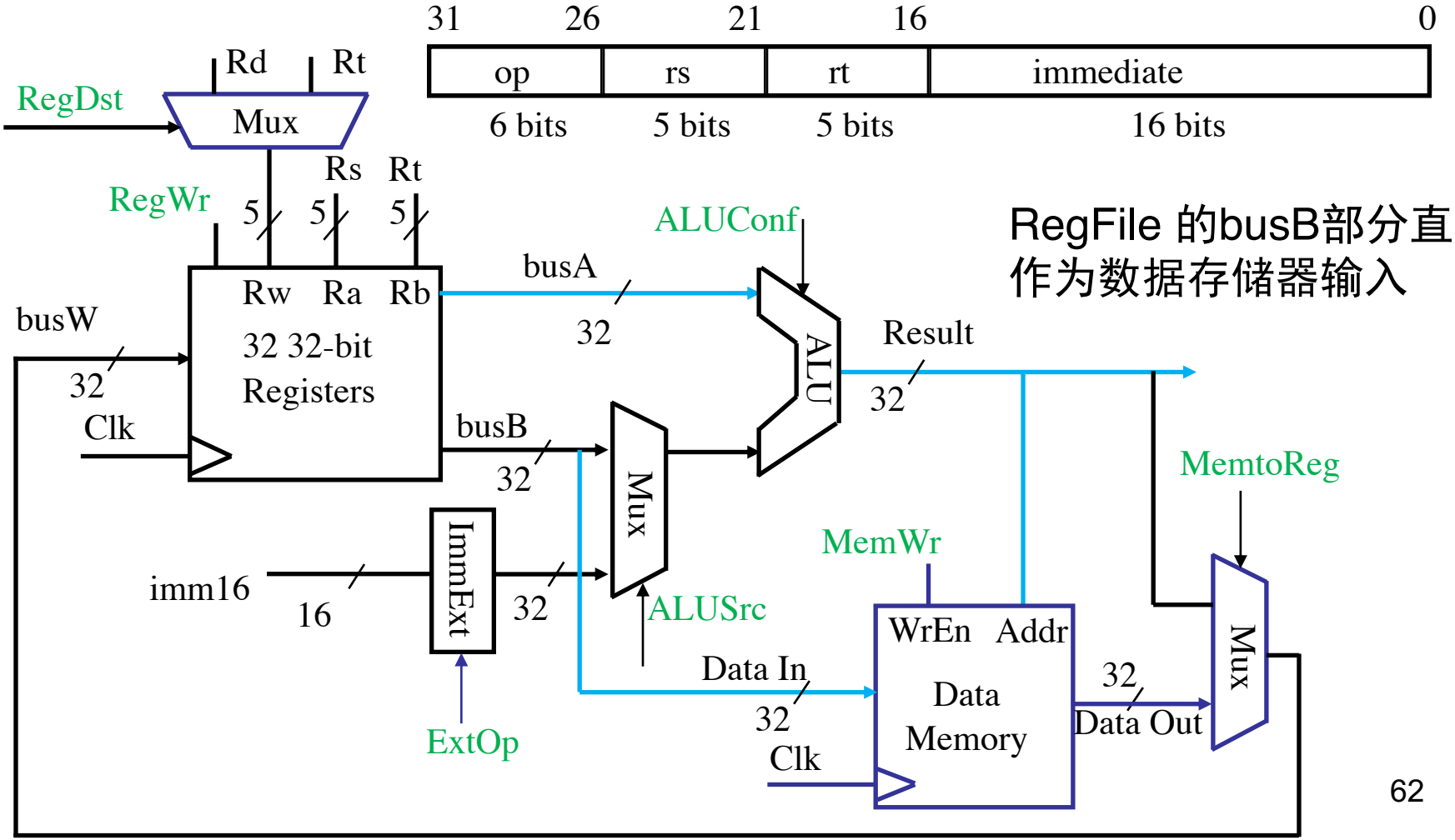
Load Word 操作

- R[rt] <= Mem[R[rs] + SignExt[imm16]]



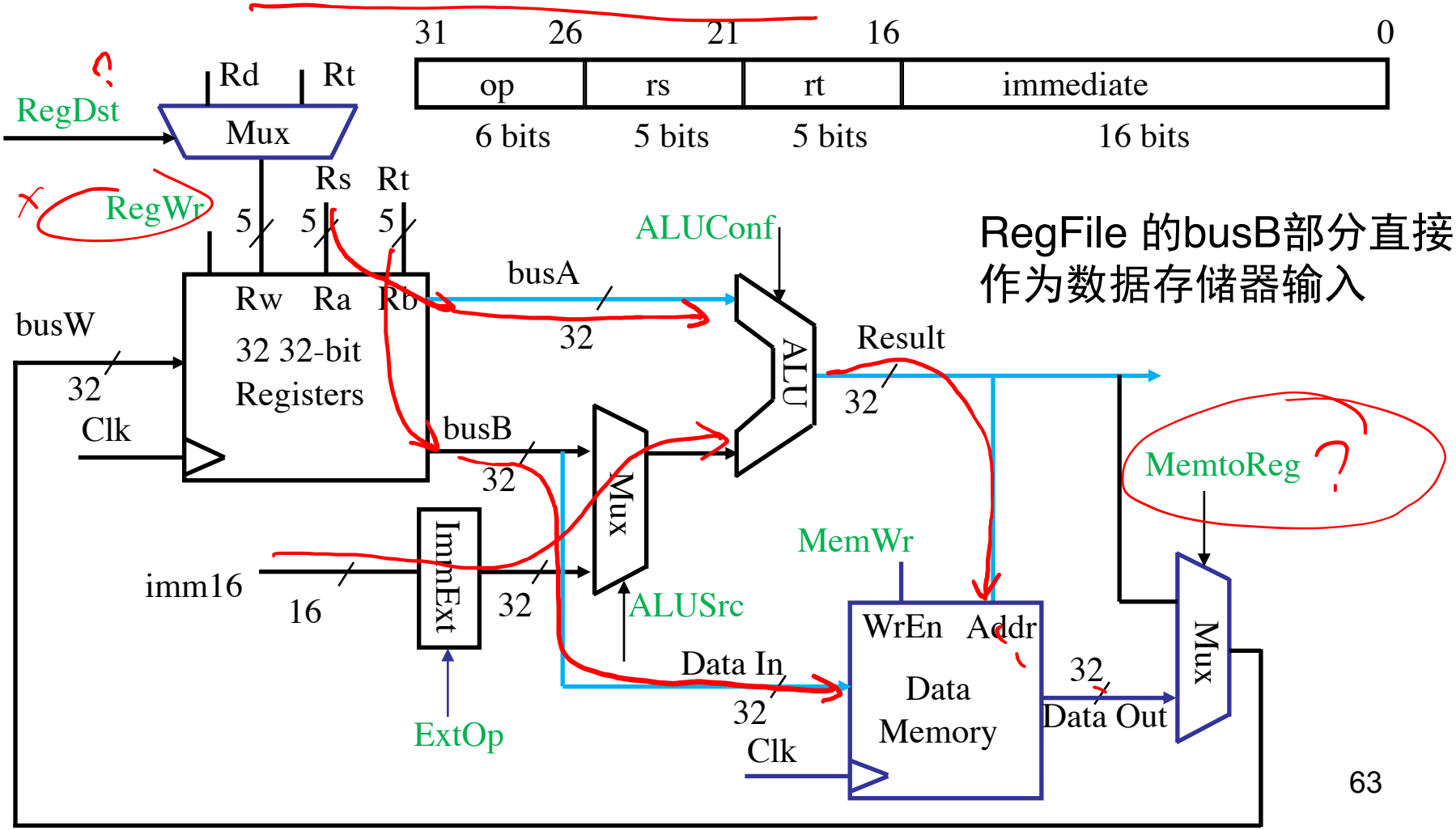
Store Word 操作

- Mem[R[rs] + SignExt[imm16]] <= R[rt]



Store Word 操作

- Mem[R[rs] + SignExt[imm16]] <= R[rt]



判断题

1. add/sub/and/slt共用ALU是为了复用硬件，降低代价

2. 前述微处理架构支持变址寻址，例如：
   lw($rd<=MEM[$rs+$rt])，
   sw(MEM[$rs+$rt]<=$rd)

答案

1. add/sub/and/slt共用ALU是为了复用硬件，降低代价

（√）满足设计要求下，复用硬件单元可降低代价

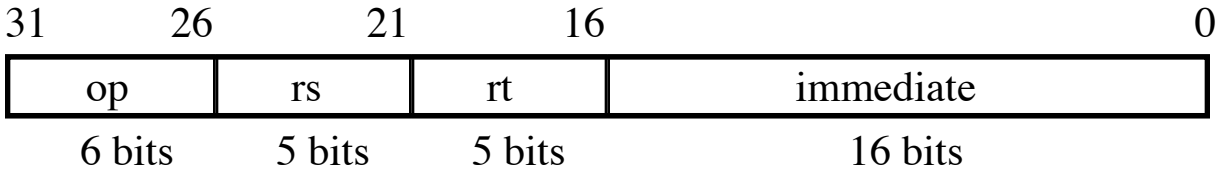
2. 前述微处理架构支持变址寻址（×）

sw(MEM[$rs+$rt]<=$rd)

需要三端口输出的RegFile, 现有

架构RegFile仅有两端口输出

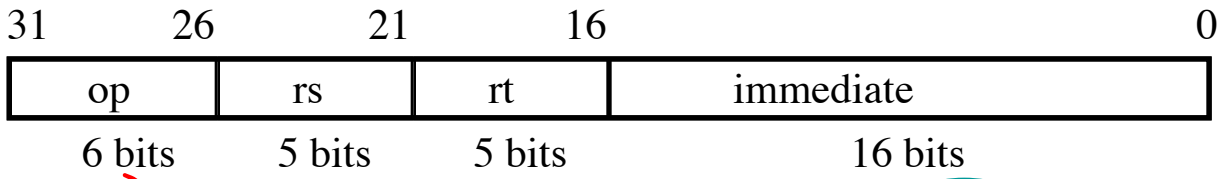
分支指令

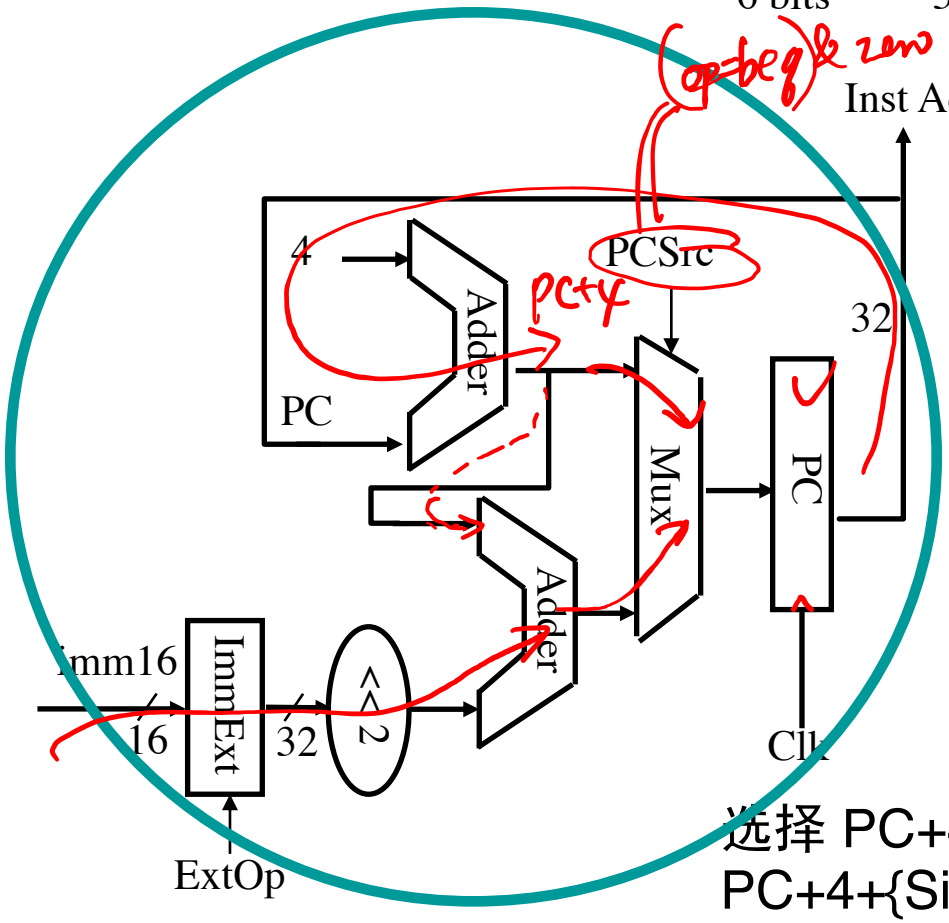


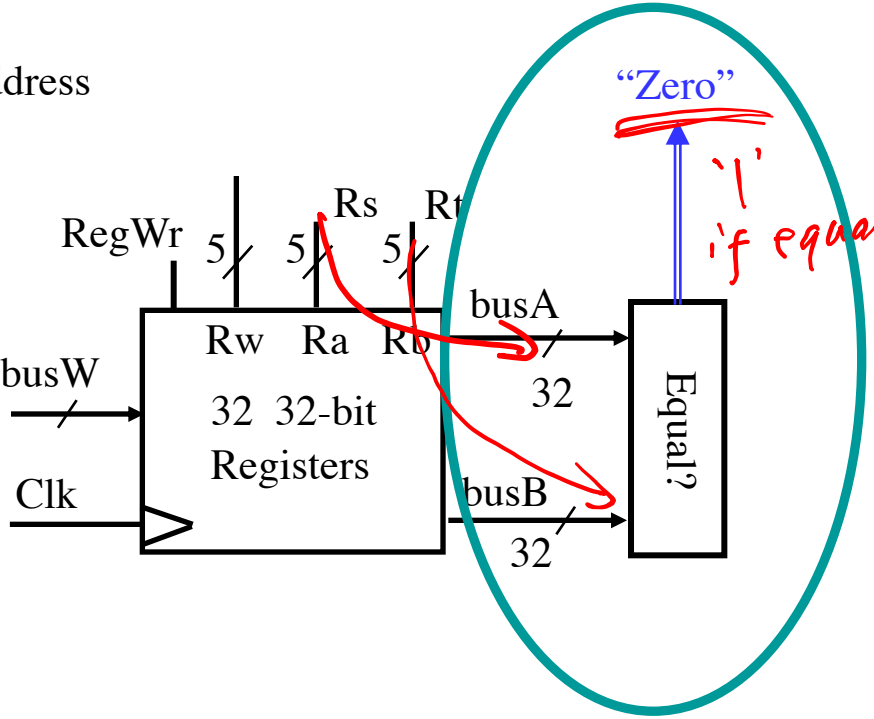
- beq rs, rt, imm16
  - –mem[PC] 从存储器取指令
  - –Equal <= (R[rs] == R[rt]) 计算分支条件
  - –if (Equal) 计算下一个指令的地址
    - PC <= PC + 4 + { SignExt(imm16) , 2b00 }
  - – else
    - PC <= PC + 4

分支指令——取指和条件选择

beq rs, rt, imm16



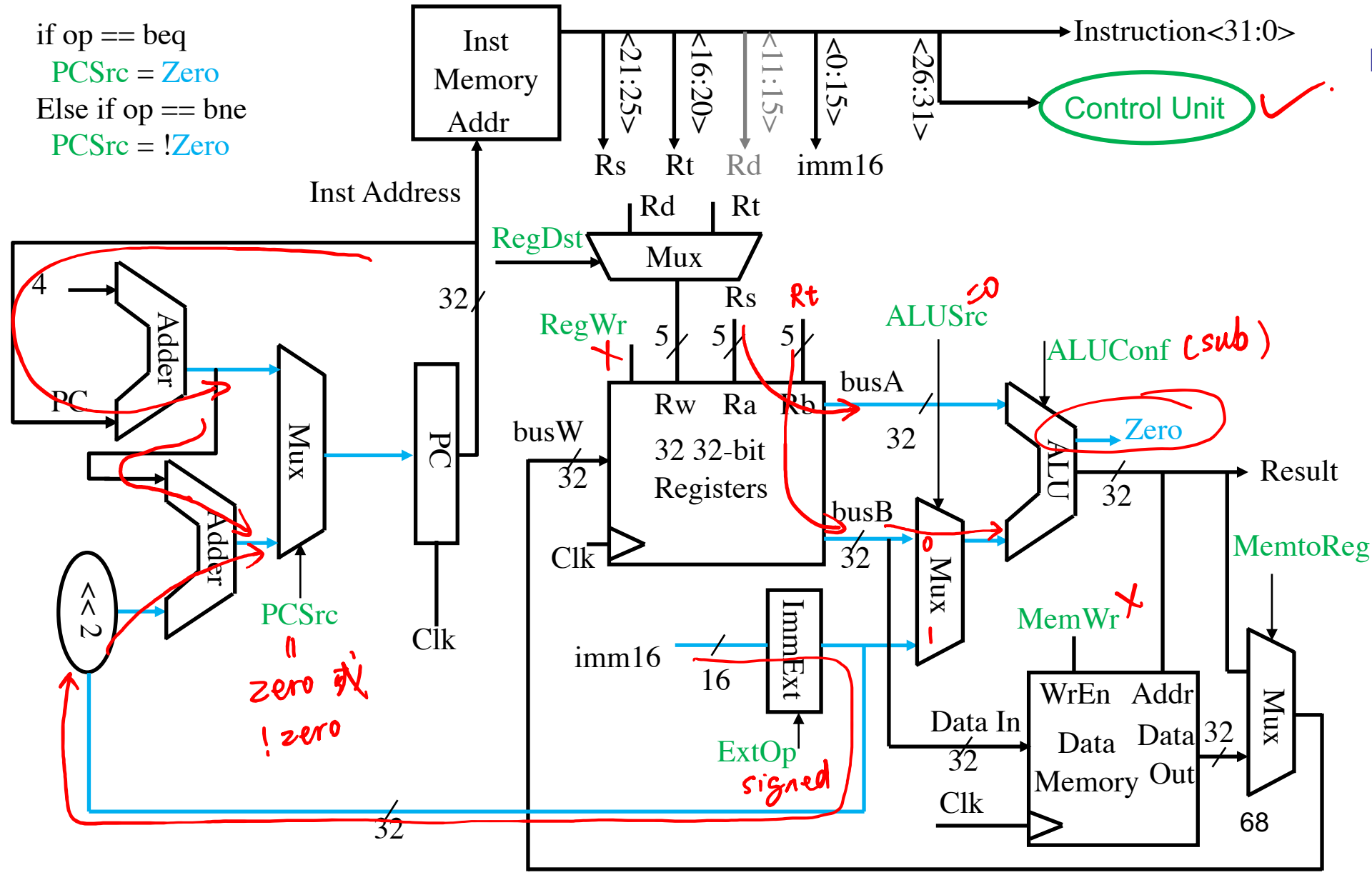




选择 PC+4 或者
PC+4+{SignExtImm << 2}

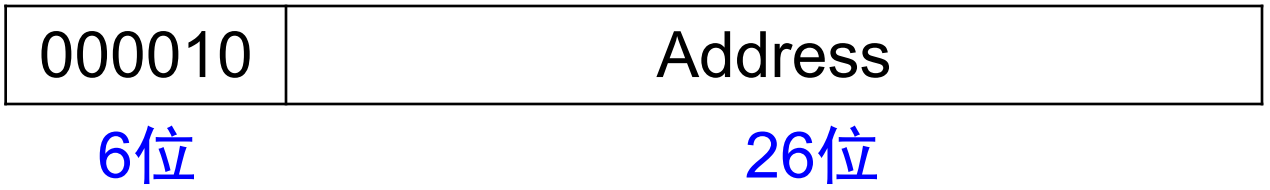
做Ra和Rb是否相等的判断

分支指令——完整datapath



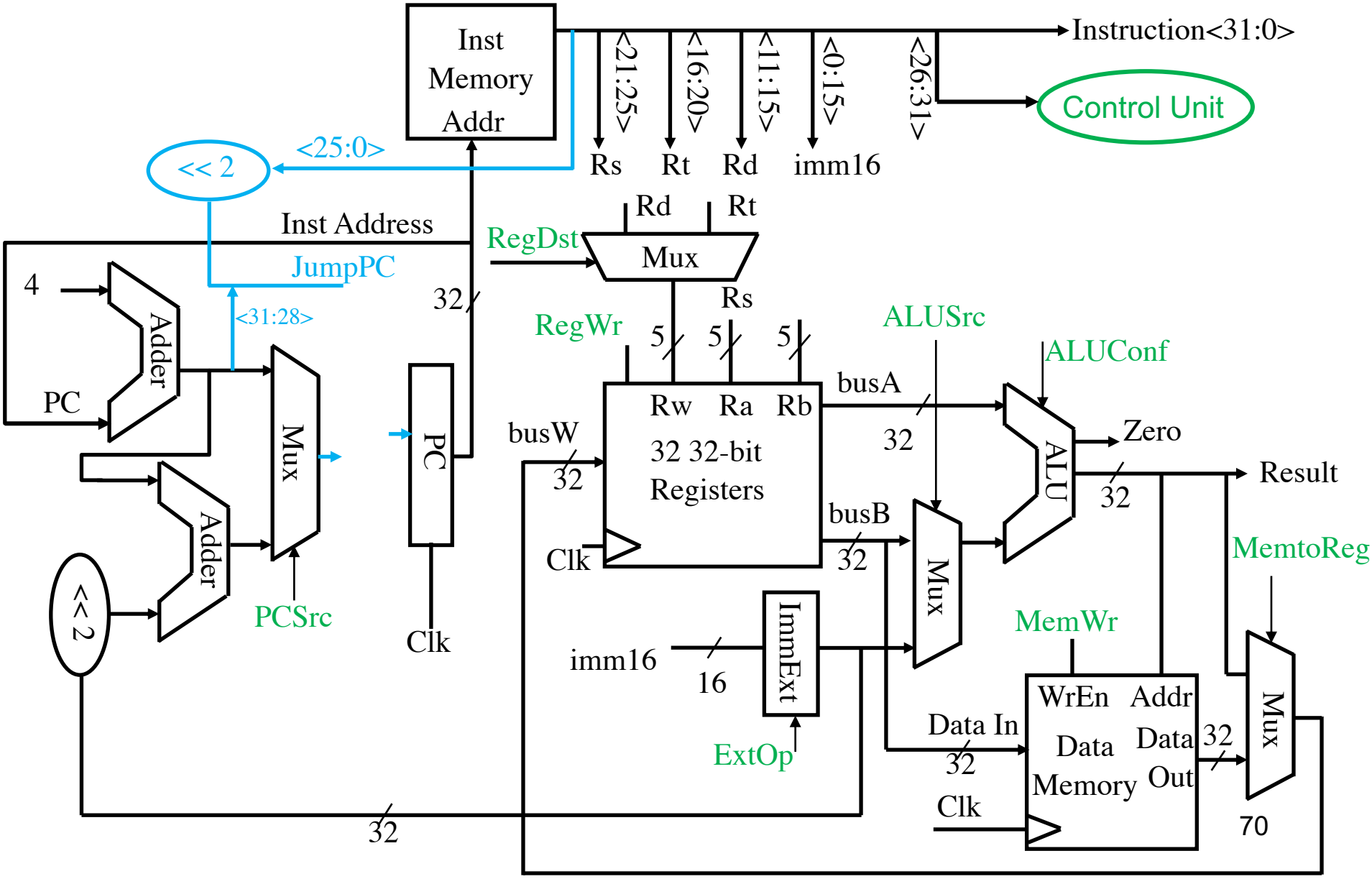
跳转指令

- j Lable

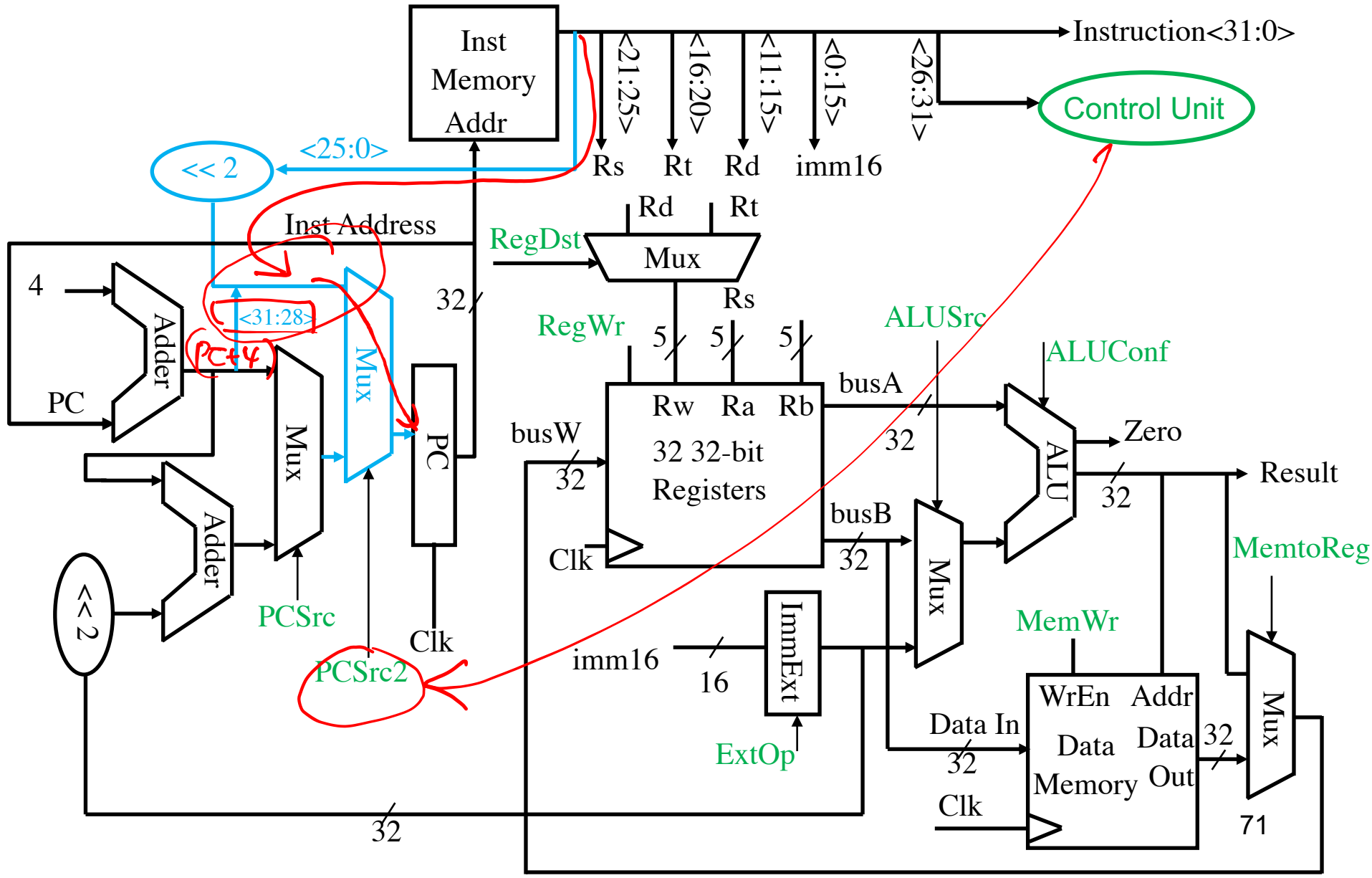


跳转 PC = { (PC+4)[31..28], 目标地址, 00 }

跳转指令


Inst
Memory
Addr
Instruction<31:0>
<21:25>
<16:20>
<11:15>
<0:15>
<26:31>
Control Unit
Rs
Rt
Rd
imm16
<< 2
<25:0>
Inst Address
JumpPC
<31:28>
4
Adder
PC
Mux
Adder
<< 2
PCSrc
PC
Clk
32
Rd
Rt
RegDst
Mux
Rs
RegWr
5
5
5
Rw
Ra
Rb
busW
32
32 32-bit
Registers
Clk
busA
32
busB
32
ALUSrc
ALUConf
ALU
Zero
Result
32
MemtoReg
Mux
imm16
16
ImmExt
ExtOp
MemWr
WrEn
Addr
Data In
Data
Memory
Data
Out
32
Clk
32
Mux
70

跳转指令

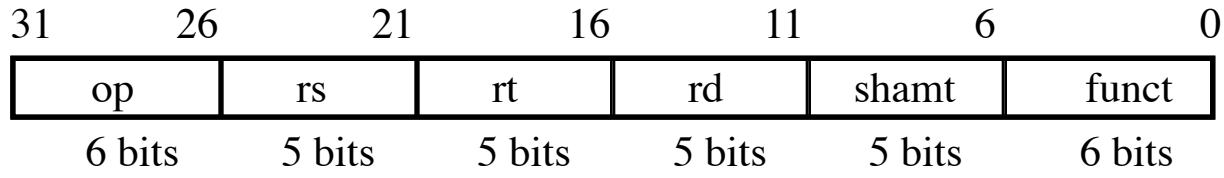

Inst Memory Addr
Instruction<31:0>
<21:25>
<16:20>
<11:15>
<0:15>
<26:31>
Control Unit
Rs
Rt
Rd
imm16
<< 2
<25:0>
Inst Address
RegDst
Rd
Rt
Mux
Rs
4
Adder
PC
<31:28>
PC+4
32
RegWr
5
5
5
ALUSrc
ALUConf
busA
Rw
Ra
Rb
32 32-bit Registers
busW
32
Clk
32
Zero
ALU
Result
busB
32
MemtoReg
Mux
PCSrc
Clk
PCSrc2
imm16
16
ImmExt
ExtOp
MemWr
WrEn
Addr
Data In
Data Memory
Data Out
32
Clk
32
Mux
71

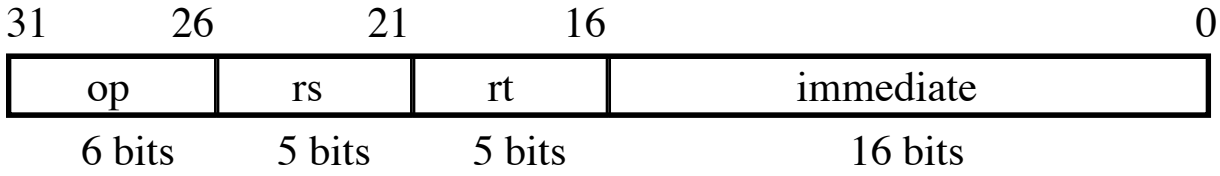
目录

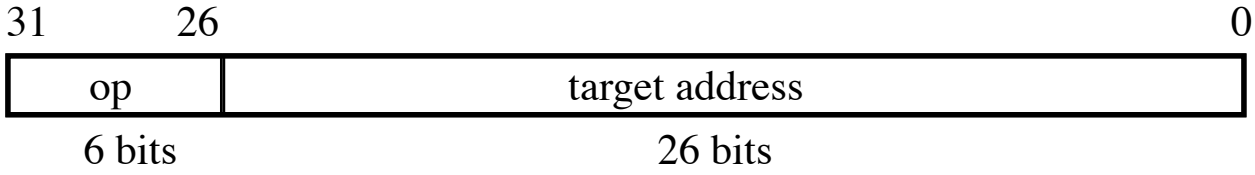
- 复习与回顾
- 单周期数据通路设计流程
  - 指令集和设计需求分析
  - 功能模块设计
  - 数据通路模块组装
  - 控制信号分析与逻辑设计
- 单周期数据通路延时分析
- 多周期数据通路
- 异常与中断

进一步精简的MIPS子集

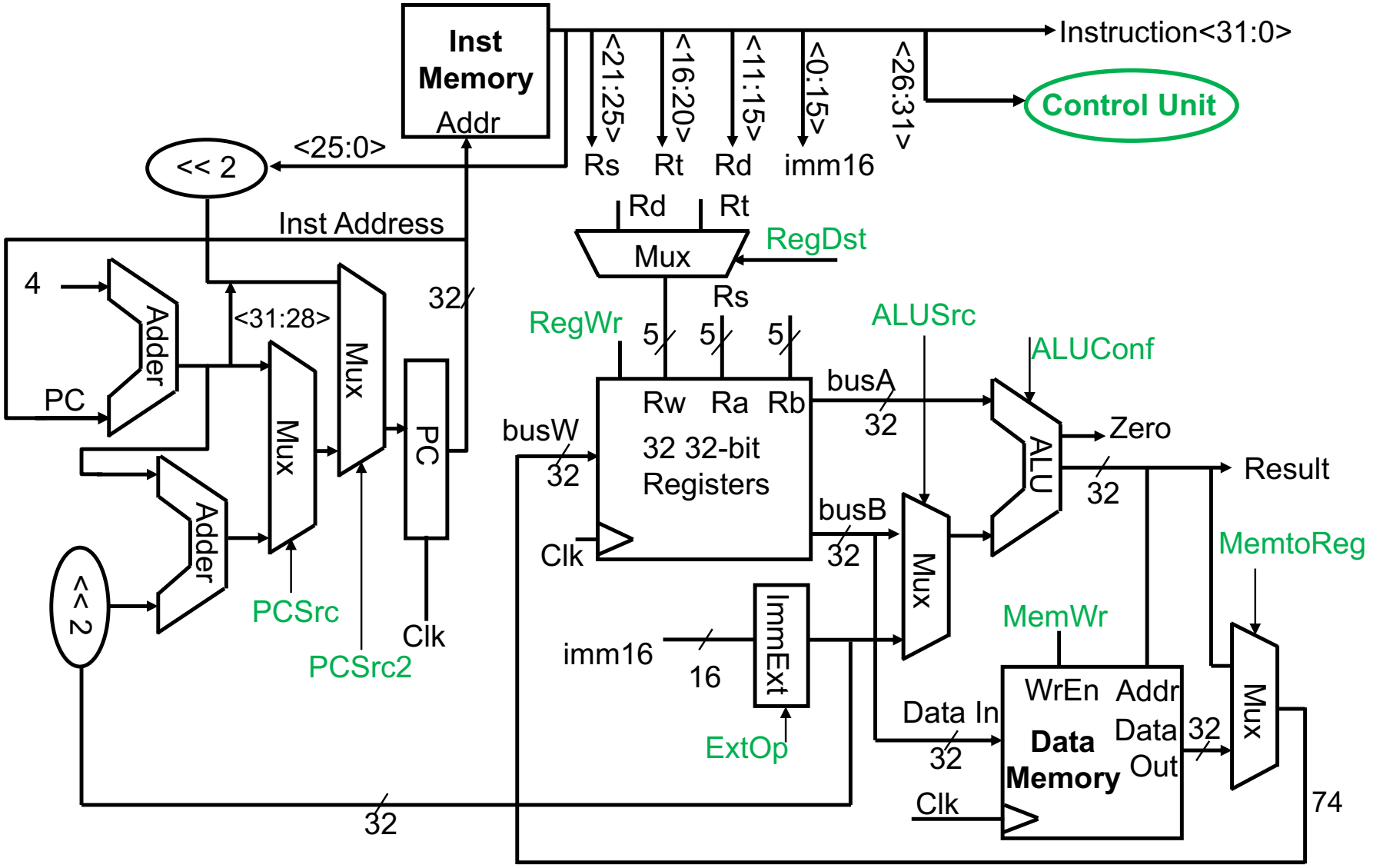
- ADD, SUB, AND, SLT
  - add rd, rs, rt
  - sub rd, rs, rt
  - and rd, rs, rt
  - slt rd, rs, rt
- OR Immediate:
  - ori rt, rs, imm16
- LOAD and STORE Word
  - lw rt, imm16(rs)
  - sw rt, imm16(rs)
- BRANCH and JUMP:
  - beq rs, rt, imm16
  - j Lable





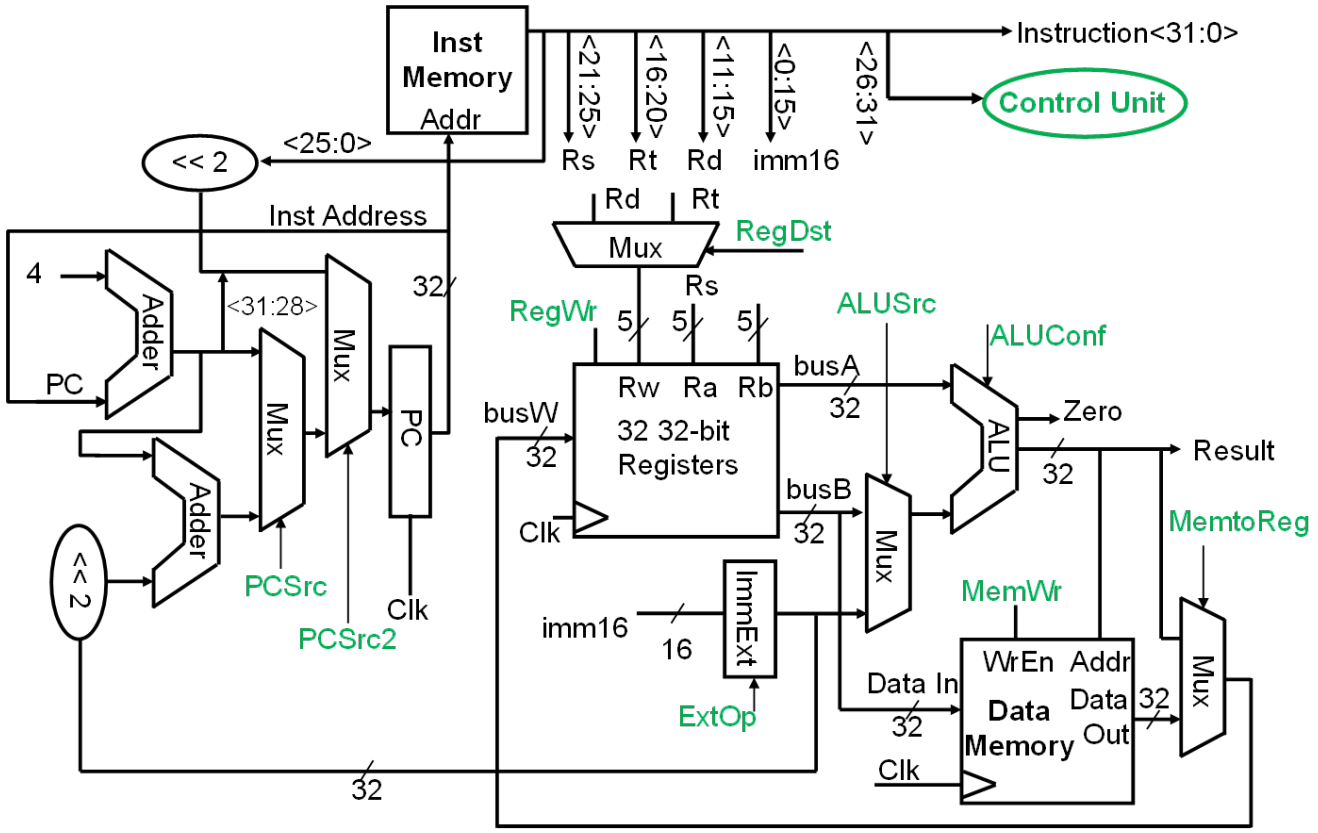


精简MIPS子集的数据通路设计



控制信号分析

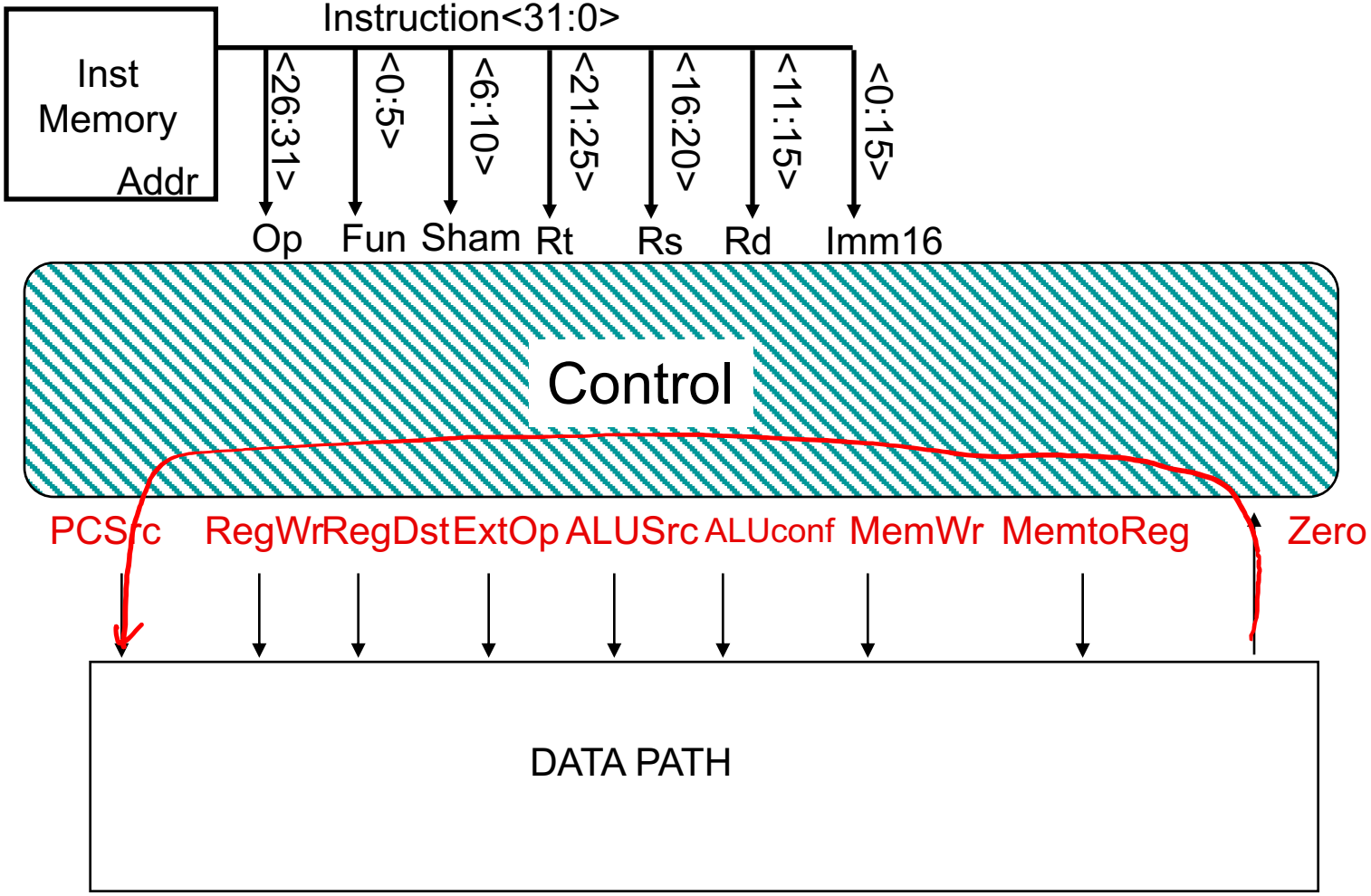
- PCSrc： 0-顺序，1-分支跳转
- ExtOp: “zero”, “sign”
- ALUsrc: 0 ⇒ regB; 1 ⇒ imm32
- ALUconf: “add”, “sub”, “or”
- MemWr： 1 ⇒ write memory
- MemtoReg: 0 ⇒ ALU; 1 ⇒ Mem
- RegDst: 0 ⇒ “Rt”; 1 ⇒ “Rd”
- RegWr: 1 ⇒ write registers



控制信号分析

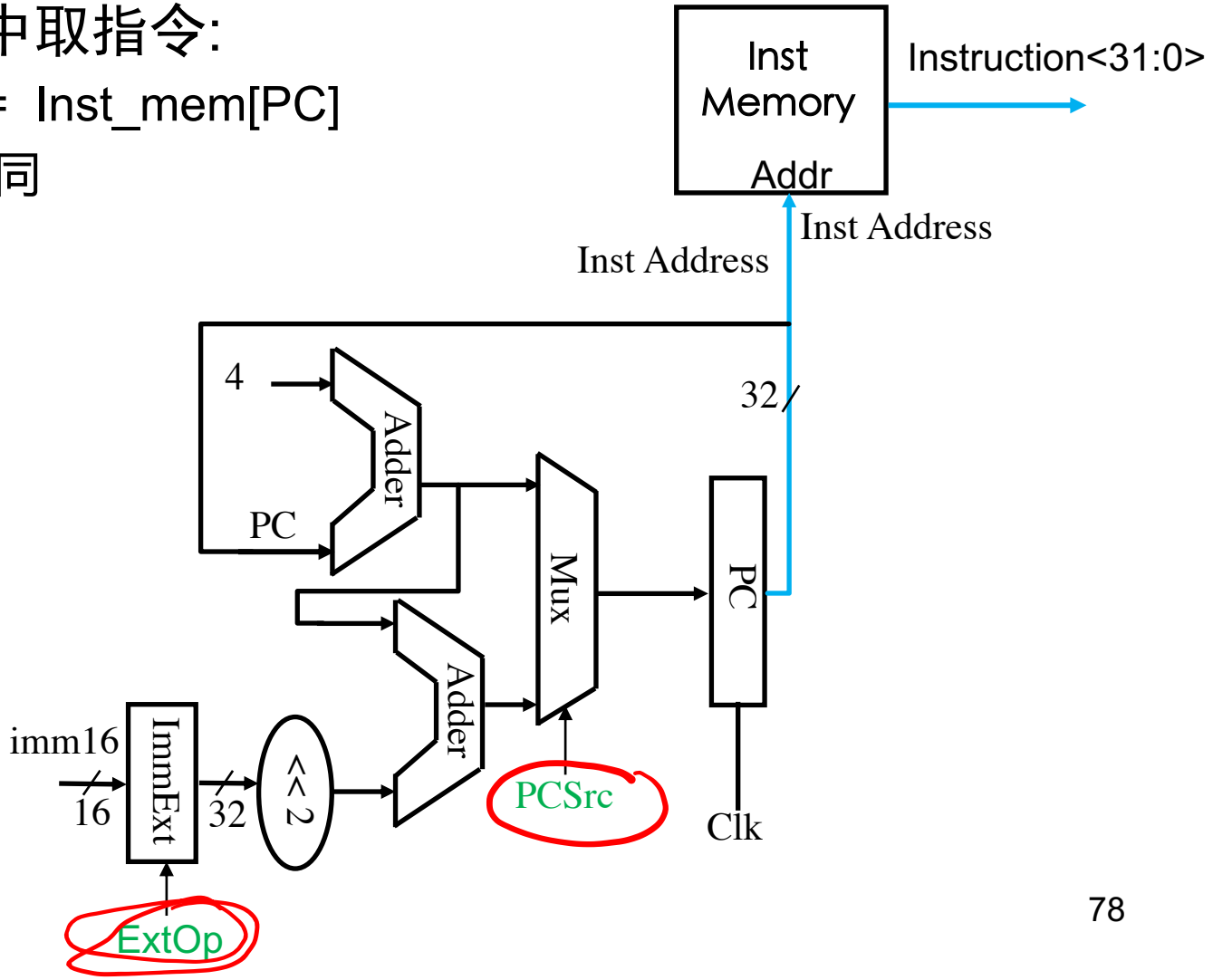
- 选择
  - PCSrc(/2)　　程序计数方式控制
  - ALUSrc　　ALU输入选择控制
  - ExtOp　　立即数扩展方式控制
  - ALUconf　　ALU运算选择
- 写入控制
  - RegWr　　寄存器写入控制
  - RegDst　　写入寄存器选择
  - MemWr　　存储器写入控制
  - MemtoReg　　寄存器数据写入端选择

控制信号和Datapath



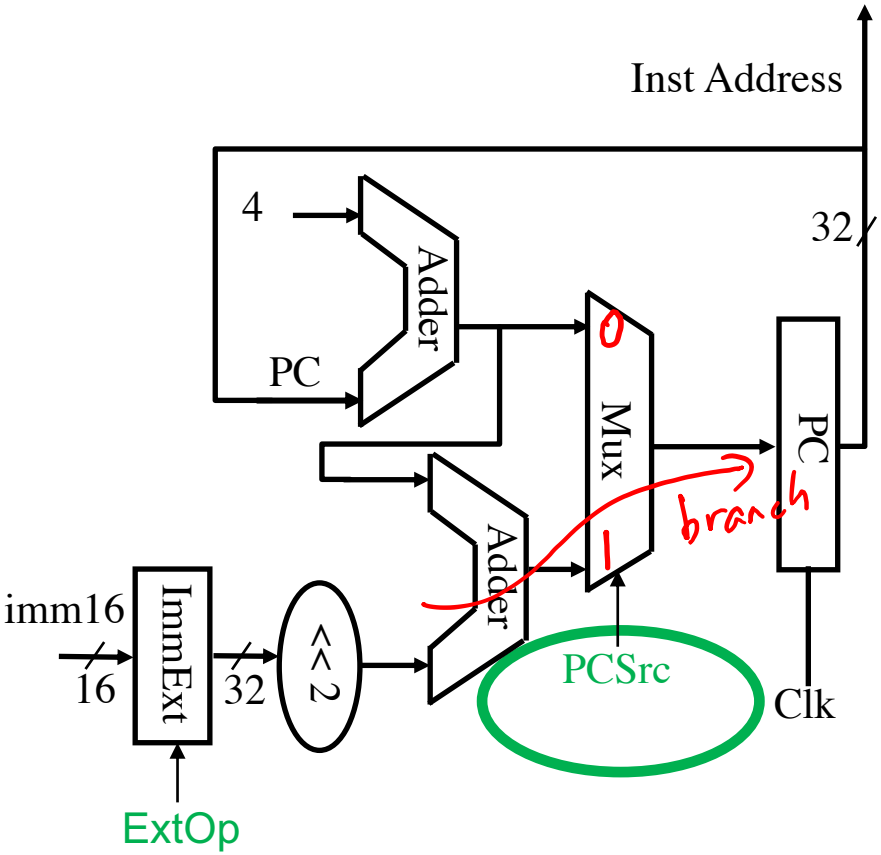
取指令过程

- 从指令存储器中取指令:
  - Instruction <= Inst_mem[PC]
  - 对所有指令相同



PCSrc信号的控制逻辑

- PCSrc: 0 ⇒ PC <= PC + 4
  1 ⇒ PC <= PC + 4 + {SignExt(Im16), 2b00}



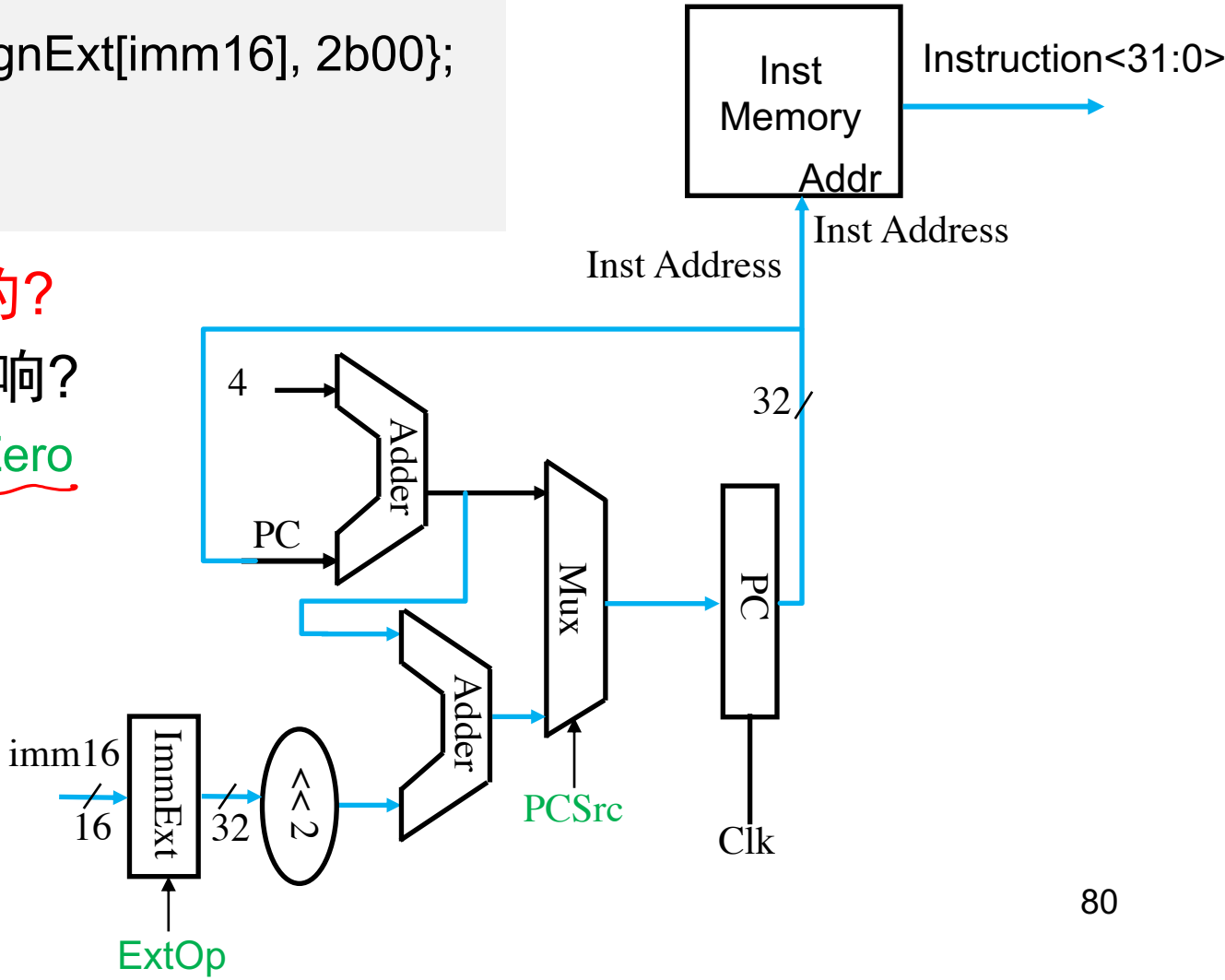
	Addu	SubU	Ori	LW	SW	Beq
PCSrc	0	0	0	0	0	1?

对于Beq的指令地址更新过程

```
if  (Zero == 1)
    PC = PC + 4 + {SignExt[imm16], 2b00};
else
    PC = PC + 4;
```

- PCSrc如何实现的?
  - 受哪些因素影响?

PCSrc = Branch AND Zero



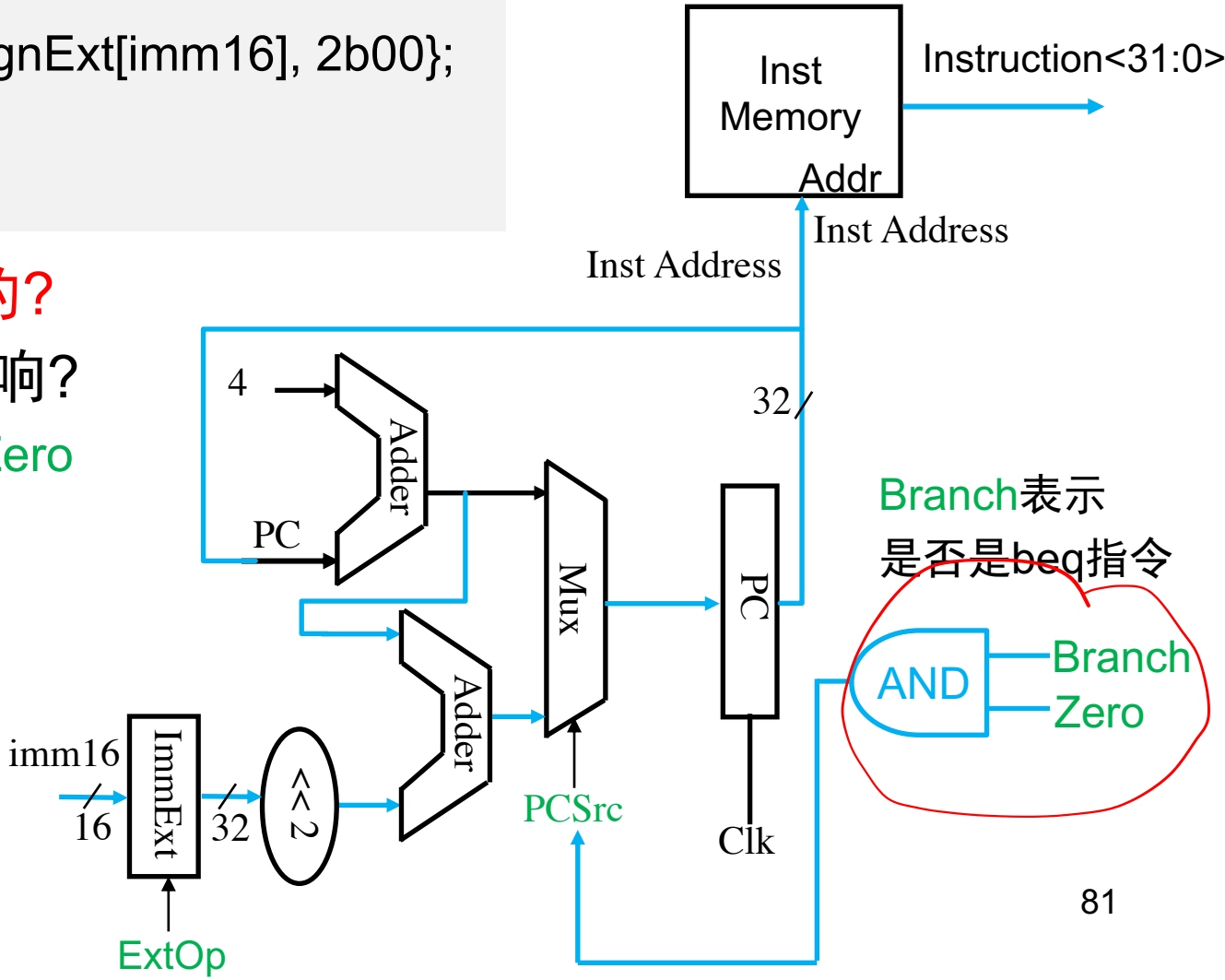
对于Beq的指令地址更新过程

```
if  (Zero == 1)
    PC = PC + 4 + {SignExt[imm16], 2b00};
else
    PC = PC + 4;
```

- PCSrc如何实现的?
  - 受哪些因素影响?

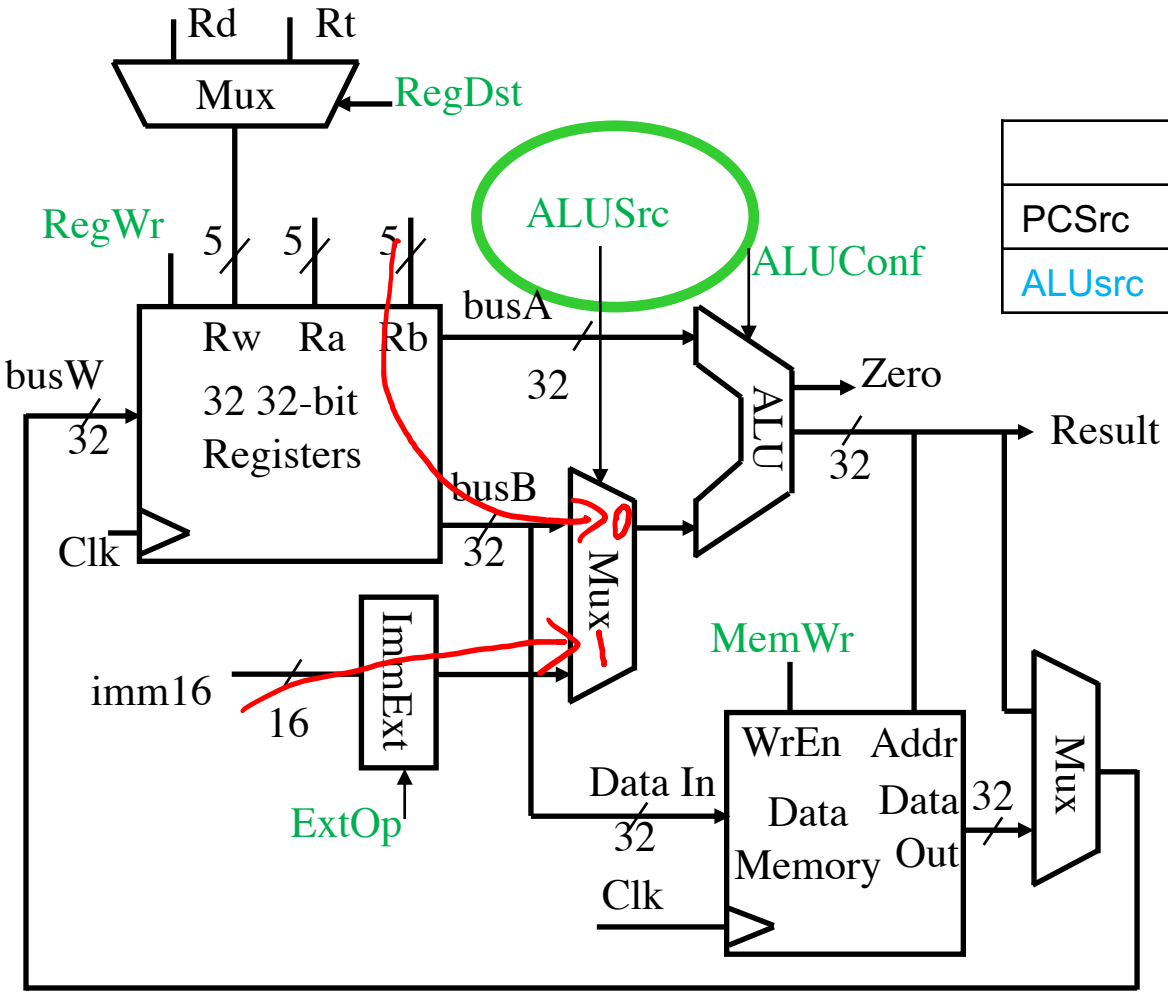
PCSrc = Branch AND Zero

Branch	Zero	MUX
0	X	0
1	0	0
1	1	1



ALUSrc信号的控制逻辑

- ALUsrc: 0 ⇒ reg 为ALU B输入; 1 ⇒ 立即数为ALU B 输入

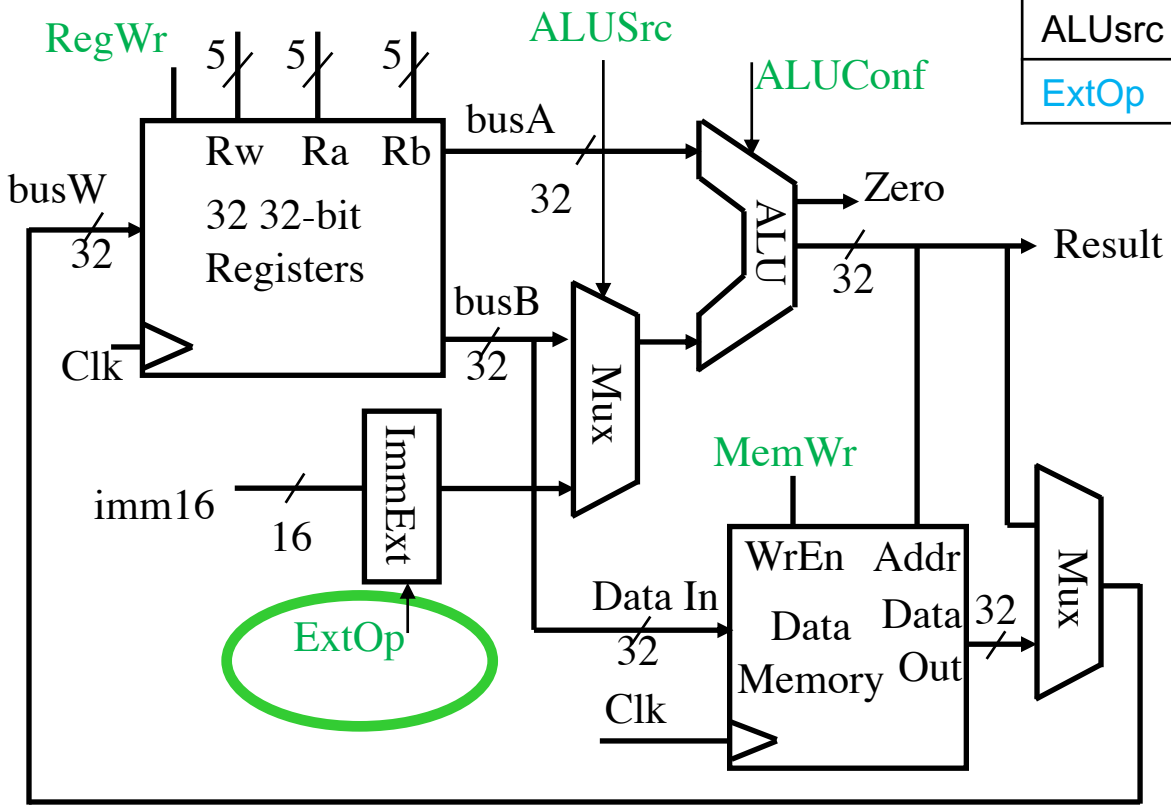


	Addu	SubU	Ori	LW	SW	Beq
PCSrc	0	0	0	0	0	1
ALUsrc	0	0	1	1	1	0

ExtOp:立即数扩展方式控制

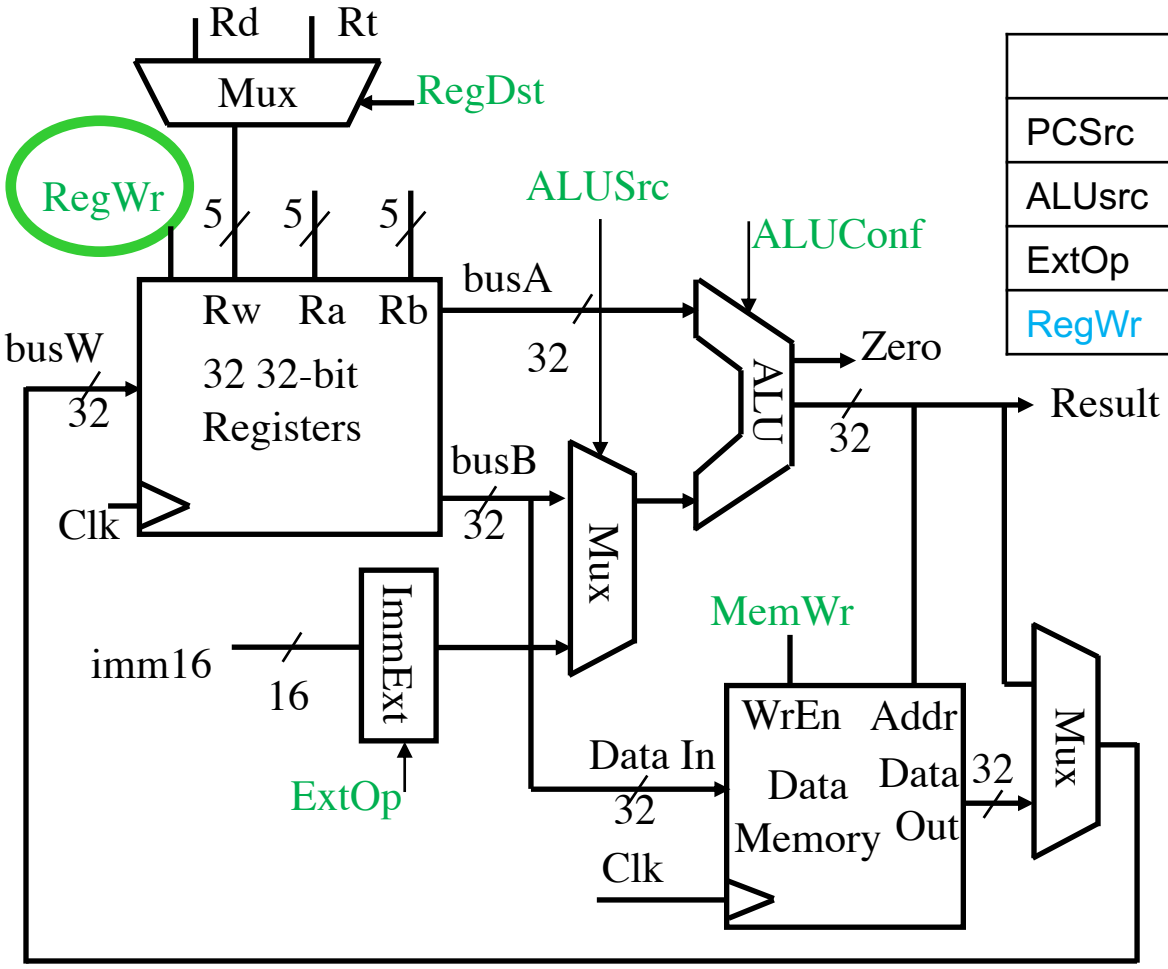
- ExtOp
  - 0 ⇒ “立即数高位填0”
  - 1 ⇒ “立即数符号扩展”

	Addu	SubU	Ori	LW	SW	Beq
PCSrc	0	0	0	0	0	1
ALUsrc	0	0	1	1	1	0
ExtOp	X	X	0	1	1	1



RegWr:寄存器写入控制

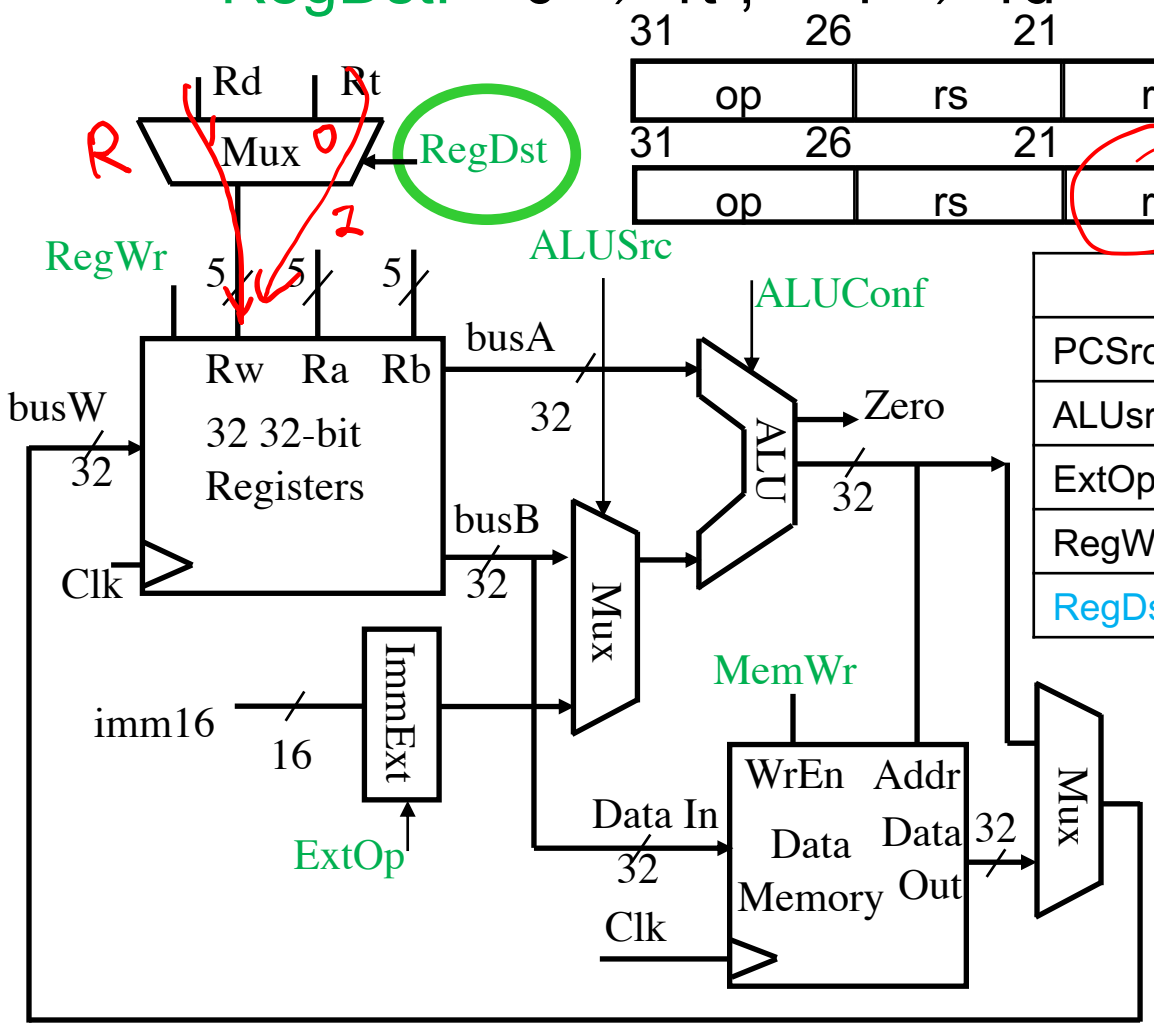
- RegWr: 1 ⇒ 写寄存器使能

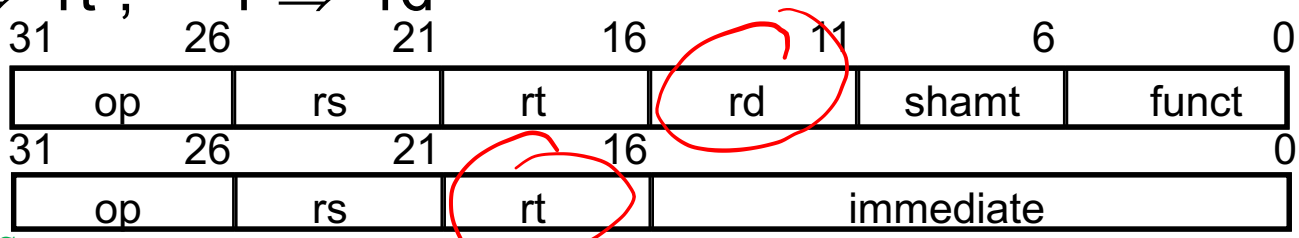


	Addu	SubU	Ori	LW	SW	Beq
PCSrc	0	0	0	0	0	1
ALUsrc	0	0	1	1	1	0
ExtOp	X	X	0	1	1	1
RegWr	1	1	1	1	0	0

RegDst:写入寄存器选择

- RegDst: 0 ⇒ “rt”; 1 ⇒ “rd”

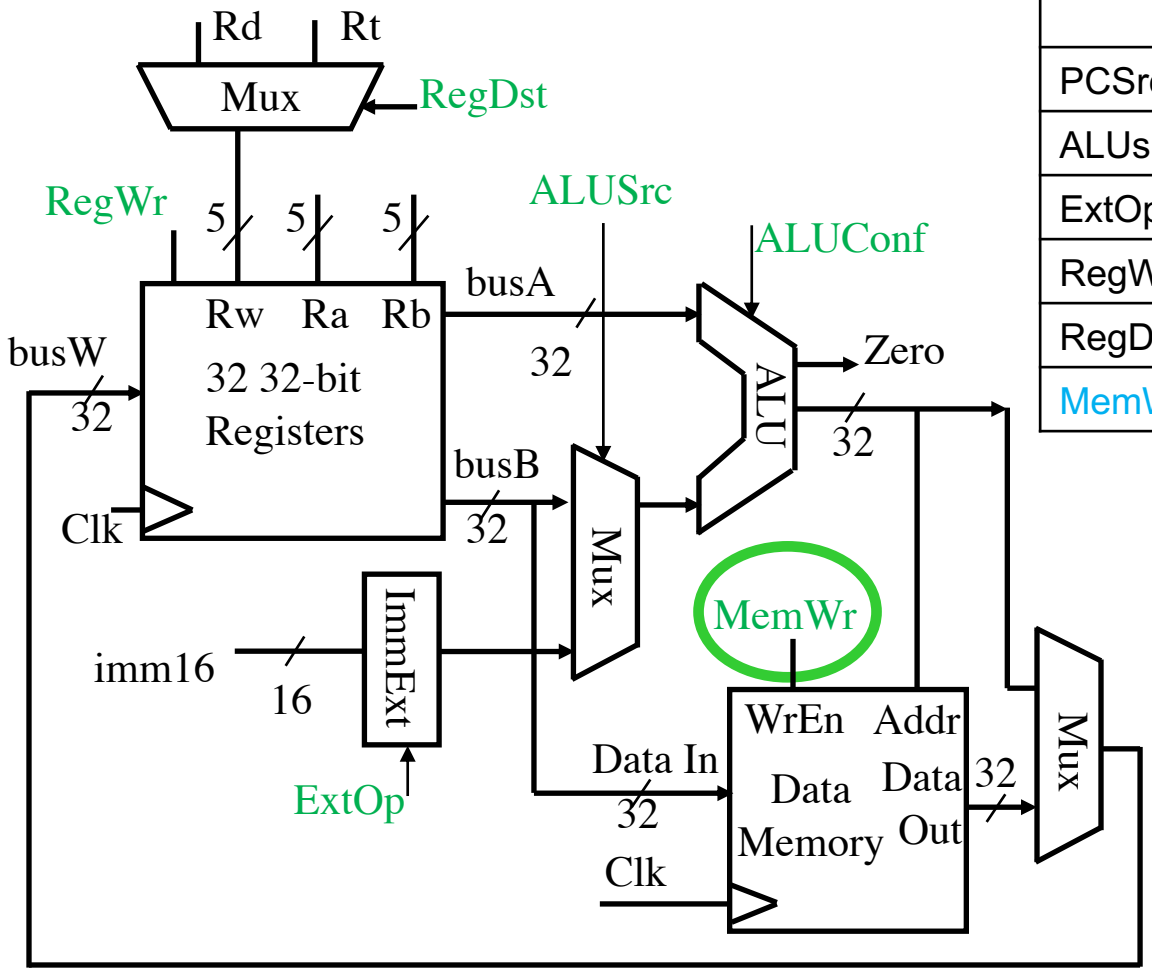




	Addu	SubU	Ori	LW	SW	Beq
PCSrc	0	0	0	0	0	1
ALUsrc	0	0	1	1	1	0
ExtOp	X	X	0	1	1	1
RegWr	1	1	1	1	0	0
RegDst	1	1	0	0	X	X

MemWr存储器写入控制

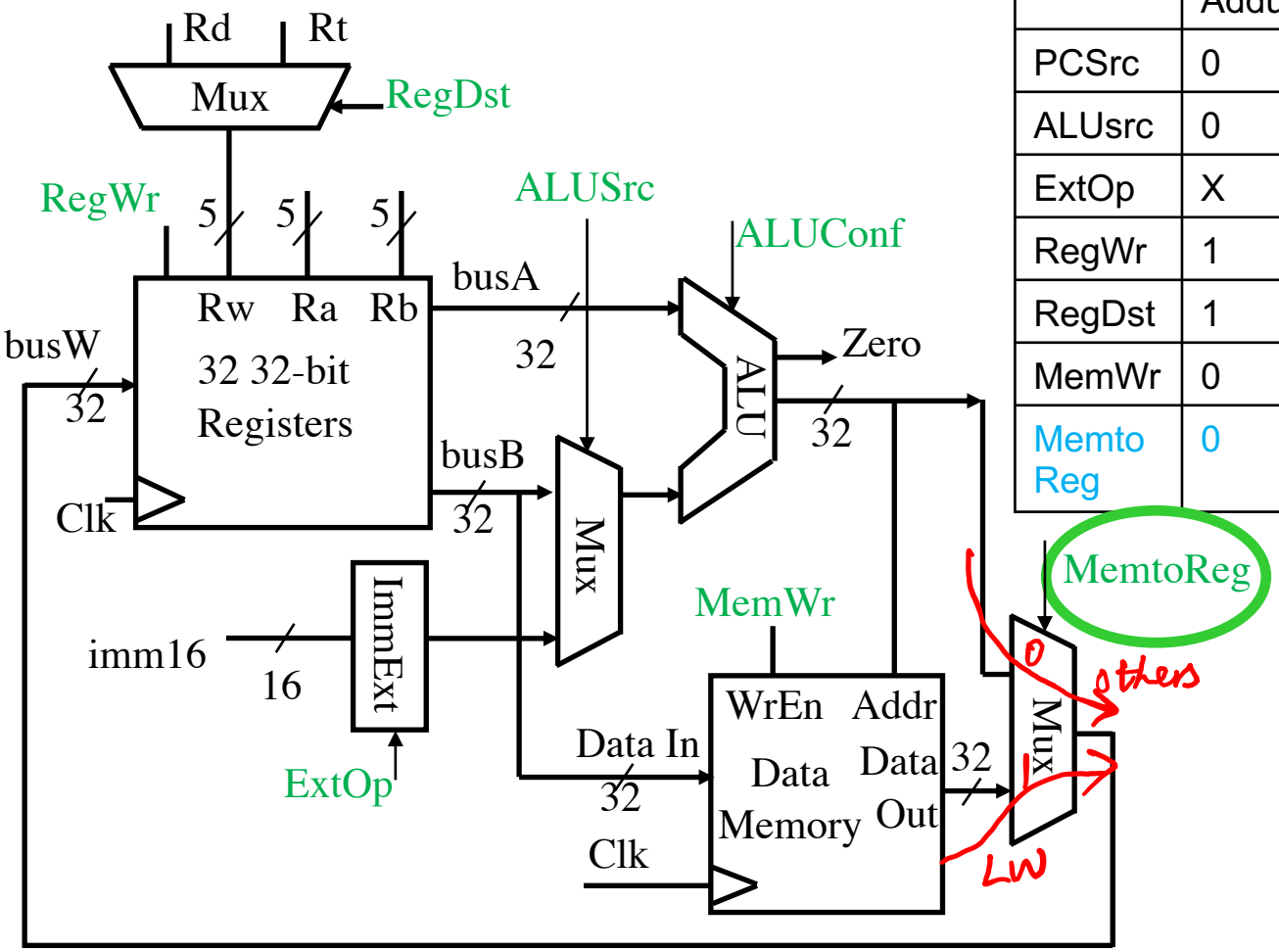
- MemWr: 1 ⇒ 写存储器使能



	Addu	SubU	Ori	LW	SW	Beq
PCSrc	0	0	0	0	0	1
ALUsrc	0	0	1	1	1	0
ExtOp	X	X	0	1	1	1
RegWr	1	1	1	1	0	0
RegDst	1	1	0	0	X	X
MemWr	0	0	0	0	1	0

MemtoReg:寄存器数据写入端选择信号

- MemtoReg: 0 ⇒ ALU; 1 ⇒ Mem

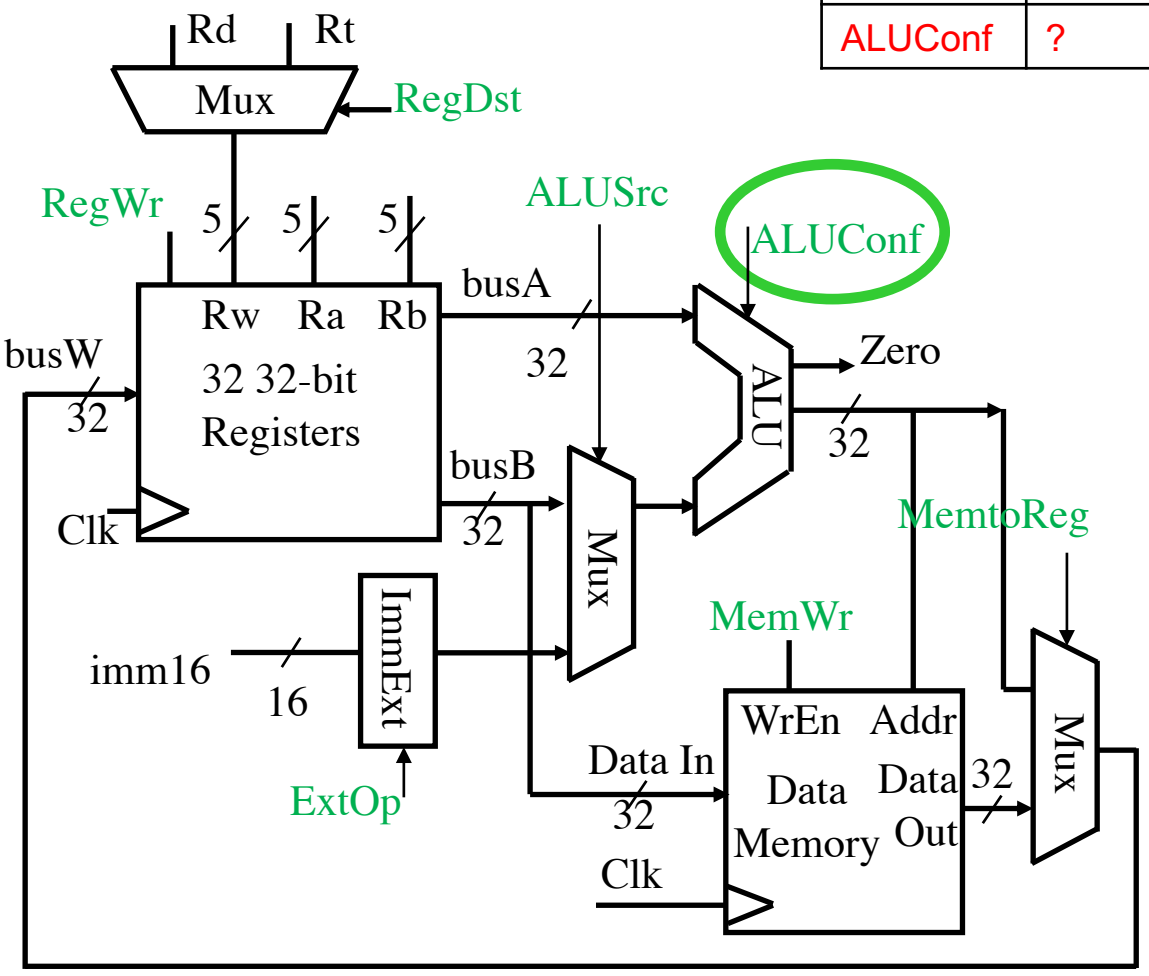


	Addu	SubU	Ori	LW	SW	Beq
PCSrc	0	0	0	0	0	1
ALUsrc	0	0	1	1	1	0
ExtOp	X	X	0	1	1	1
RegWr	1	1	1	1	0	0
RegDst	1	1	0	0	X	X
MemWr	0	0	0	0	1	0
Memto Reg	0	0	0	1	X	X

单选题

- ALUConf: 0 ⇒ “add”, 1 ⇒ “sub”, 2 ⇒ “or”

	Addu	SubU	Ori	LW	SW	Beq
ALUConf	?	?	?	?	?	?



ALUConf 的赋值为哪一项？

A. 0 1 0 0 0 1

B. 0 1 0 0 0 X

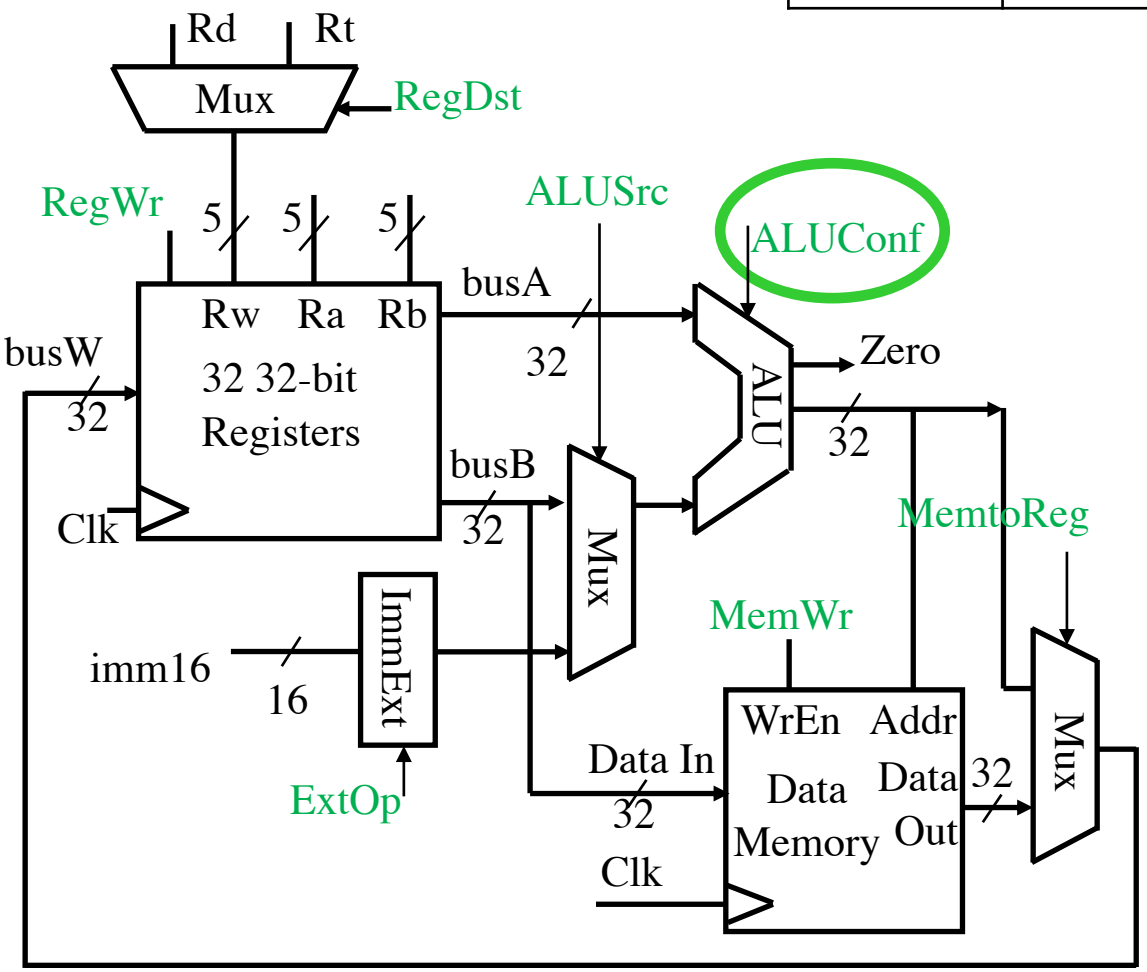
C. 0 1 2 0 0 1

D. 0 1 2 0 0 X

解答

- ALUConf: 0 ⇒ “add”, 1 ⇒ “sub”, 2 ⇒ “or”

	Addu	SubU	Ori	LW	SW	Beq
ALUConf	?	?	?	?	?	?



ALUConf 的赋值为哪一项？

A. 0 1 0 0 0 1

B. 0 1 0 0 0 X

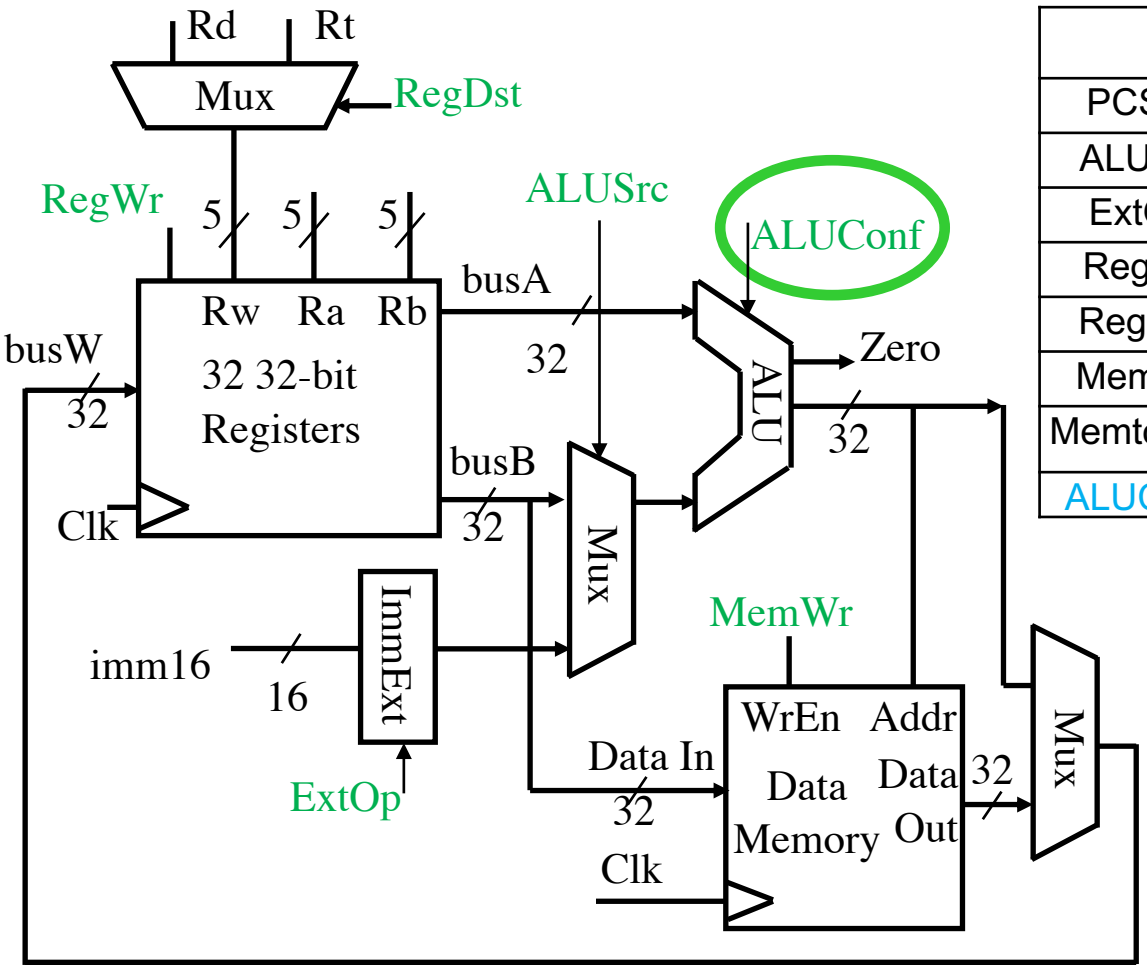
C. 0 1 2 0 0 1

D. 0 1 2 0 0 X

分析：Beq：减法判断 rs==rt

ALUConf: ALU运算选择

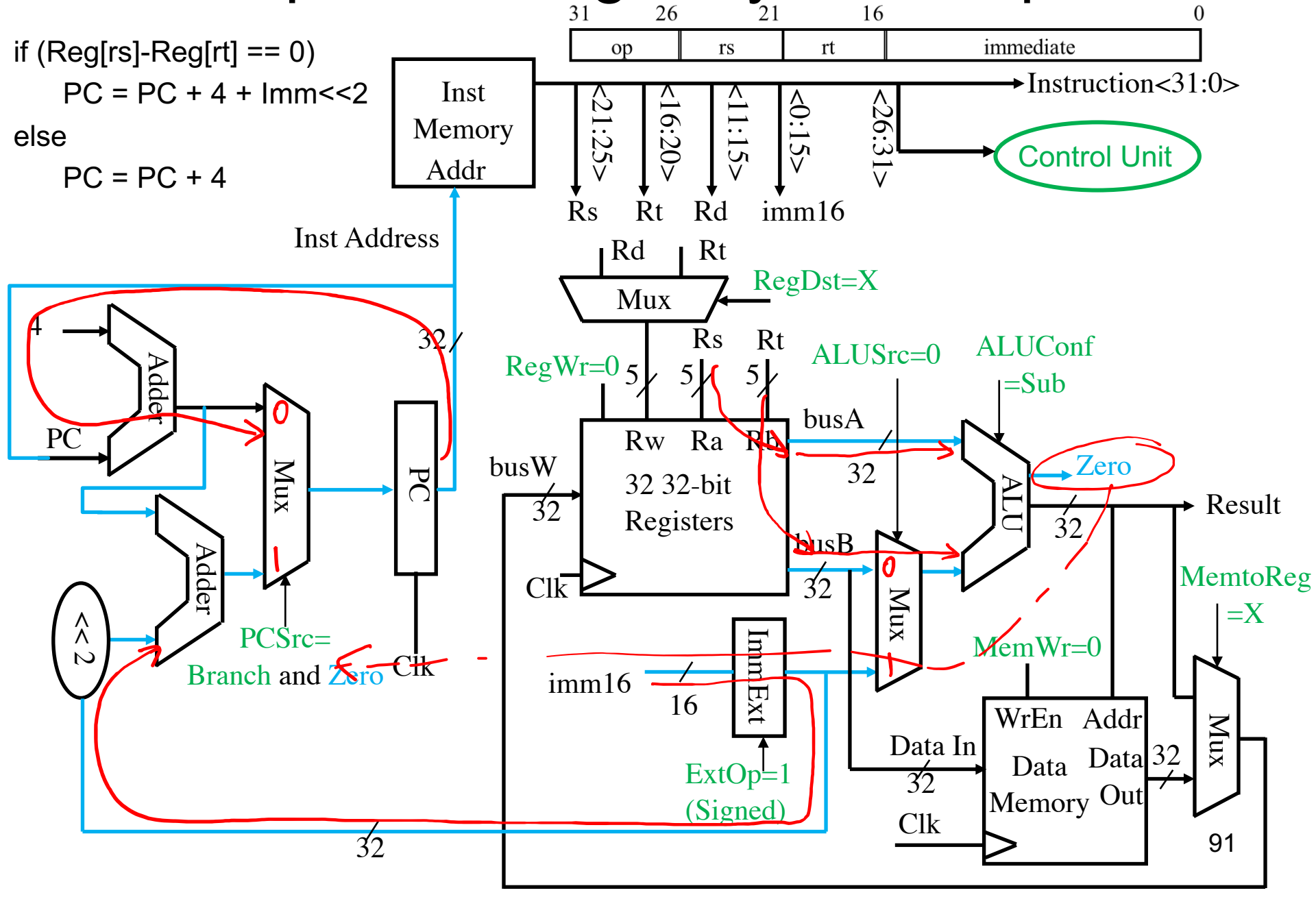
- ALUConf: 0 ⇒ “add”, 1 ⇒ “sub”, 2 ⇒ “or”



	Addu	SubU	Ori	LW	SW	Beq
PCSrc	0	0	0	0	0	1
ALUsrc	0	0	1	1	1	0
ExtOp	X	X	0	1	1	1
RegWr	1	1	1	1	0	0
RegDst	1	1	0	0	X	X
MemWr	0	0	0	0	1	0
MemtoReg	0	0	0	1	X	X
ALUConf	0	1	2	0	0	1

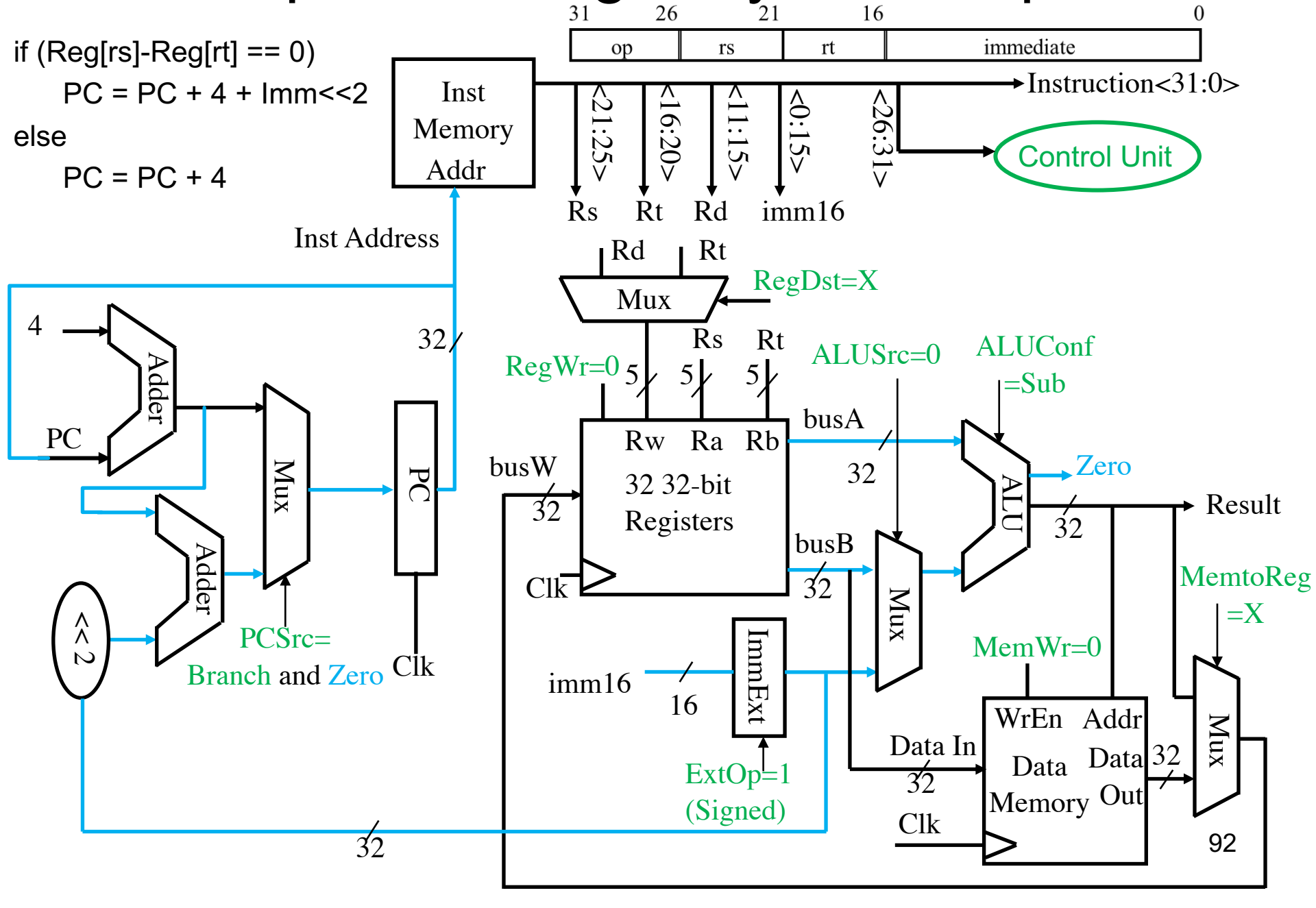
Beq指令的Single Cycle Datapath

if (Reg[rs]-Reg[rt] == 0)
PC = PC + 4 + Imm<<2
else
PC = PC + 4

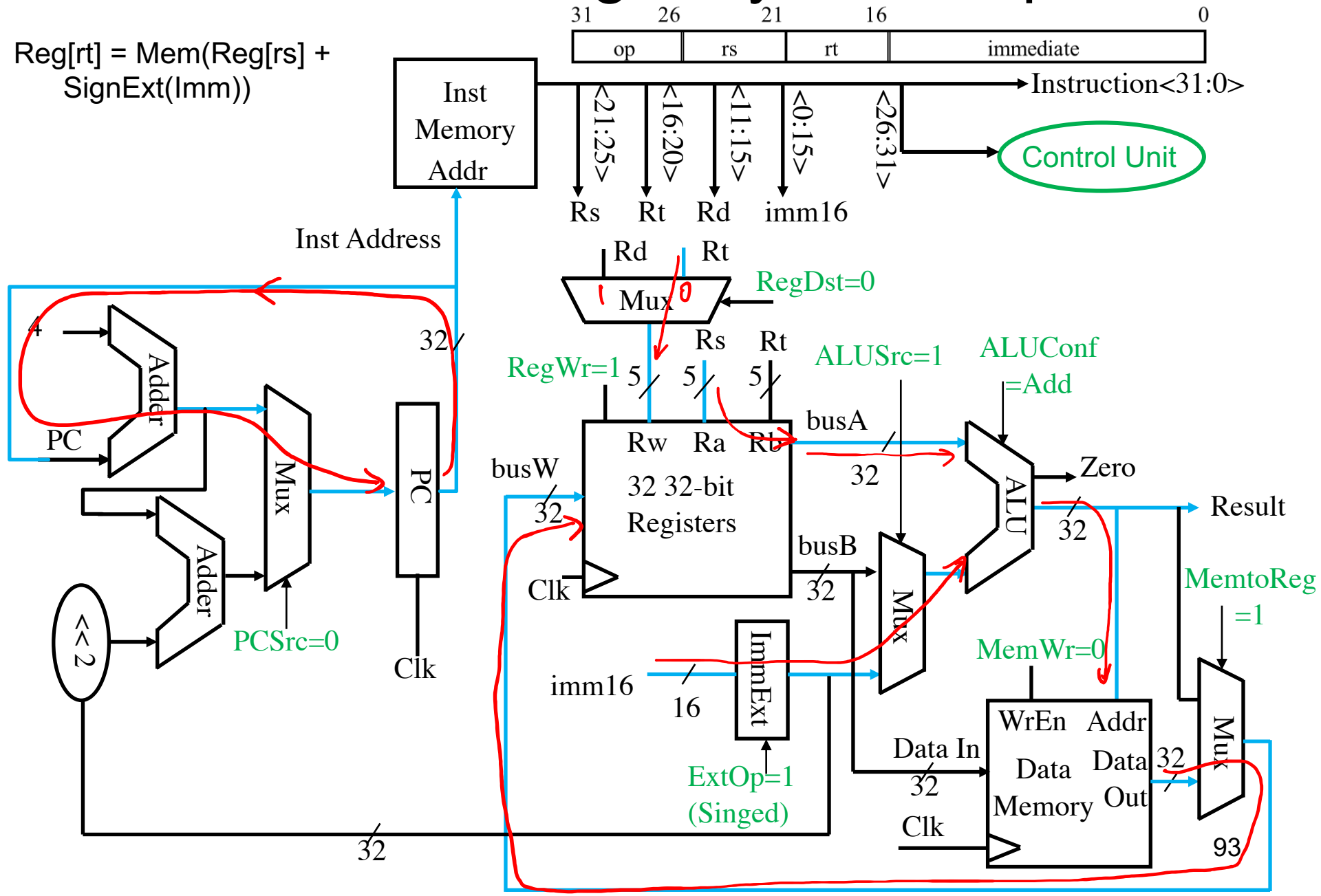


Beq指令的Single Cycle Datapath

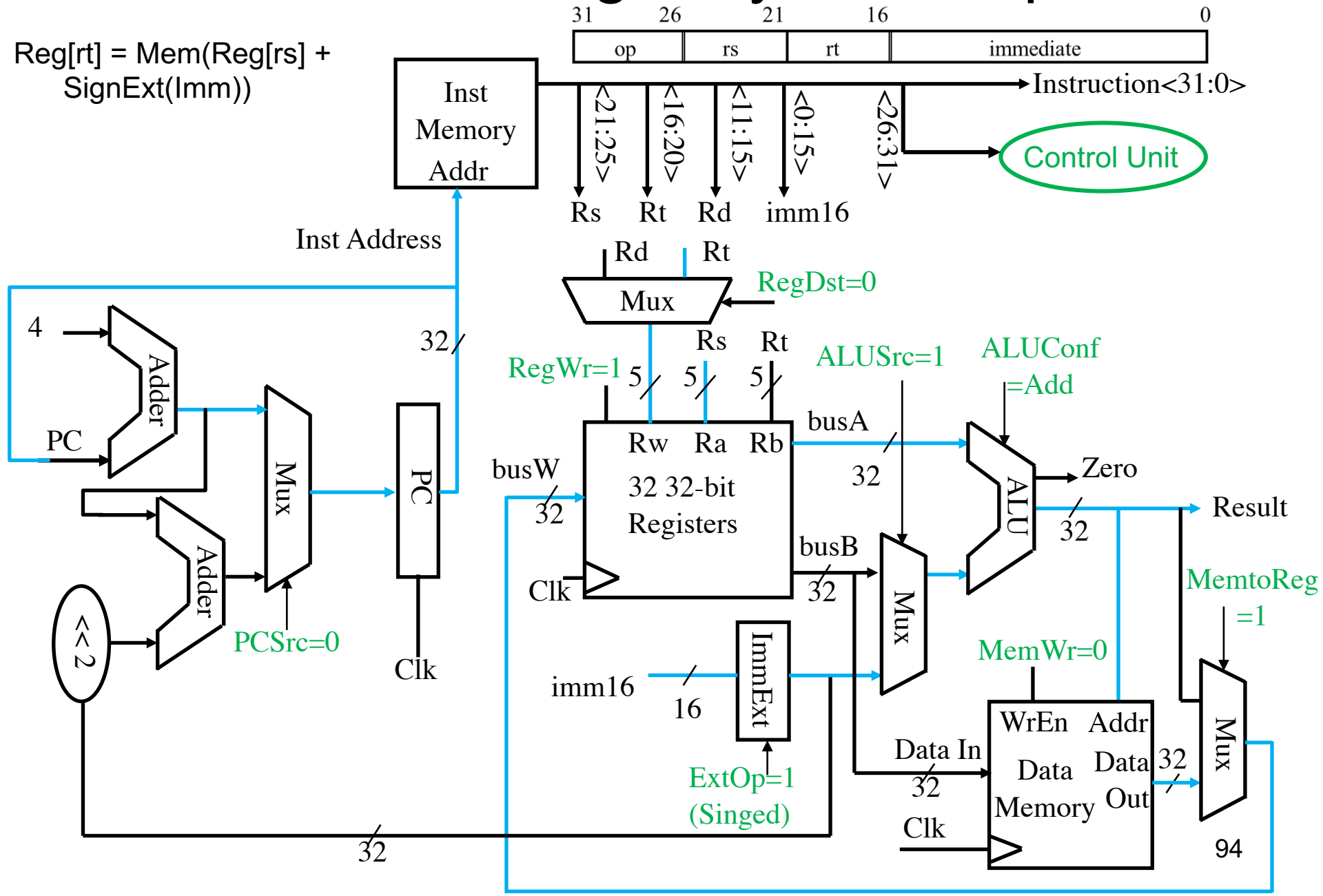
if (Reg[rs]-Reg[rt] == 0)
PC = PC + 4 + Imm<<2
else
PC = PC + 4



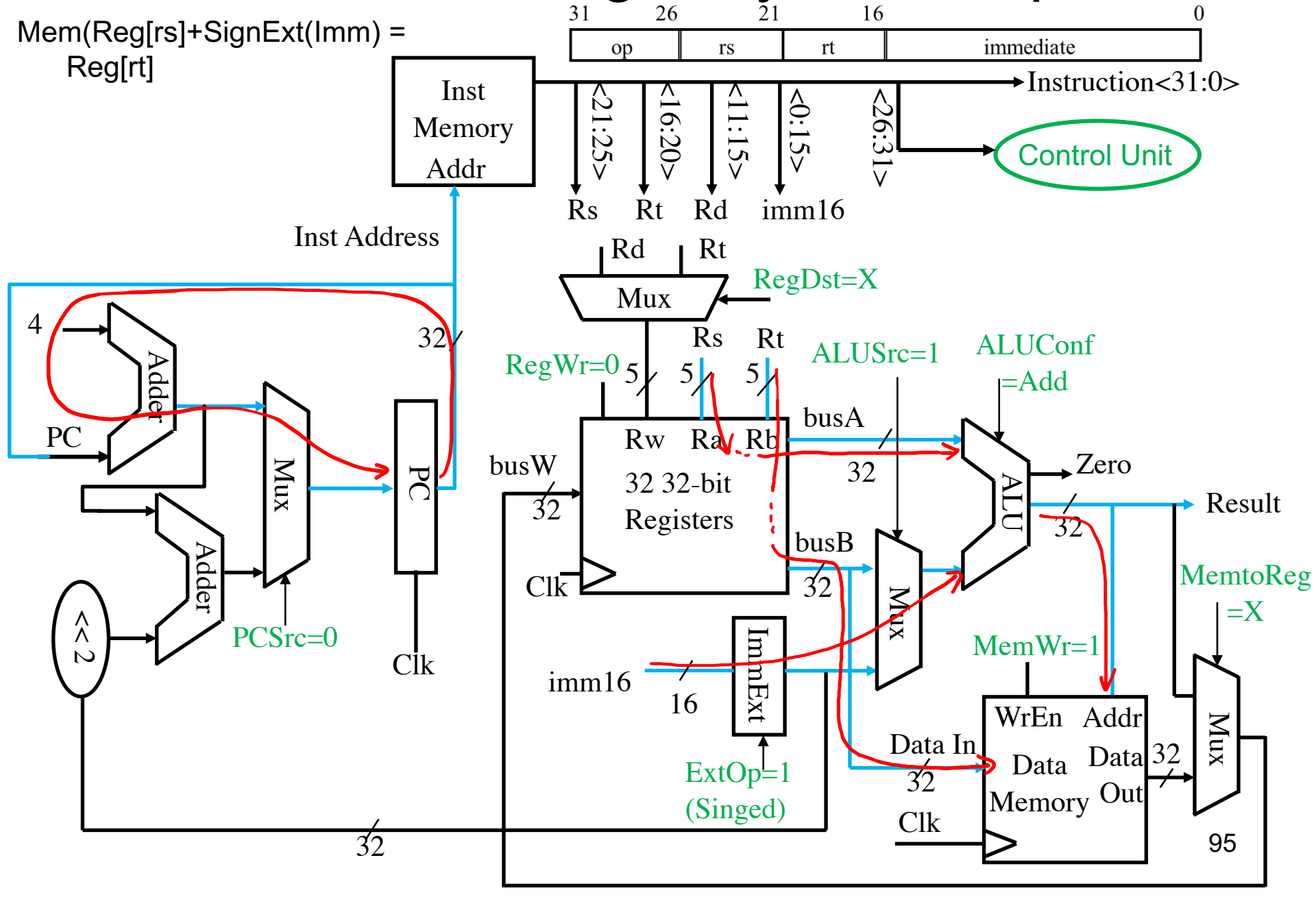
Load指令的Single Cycle Datapath



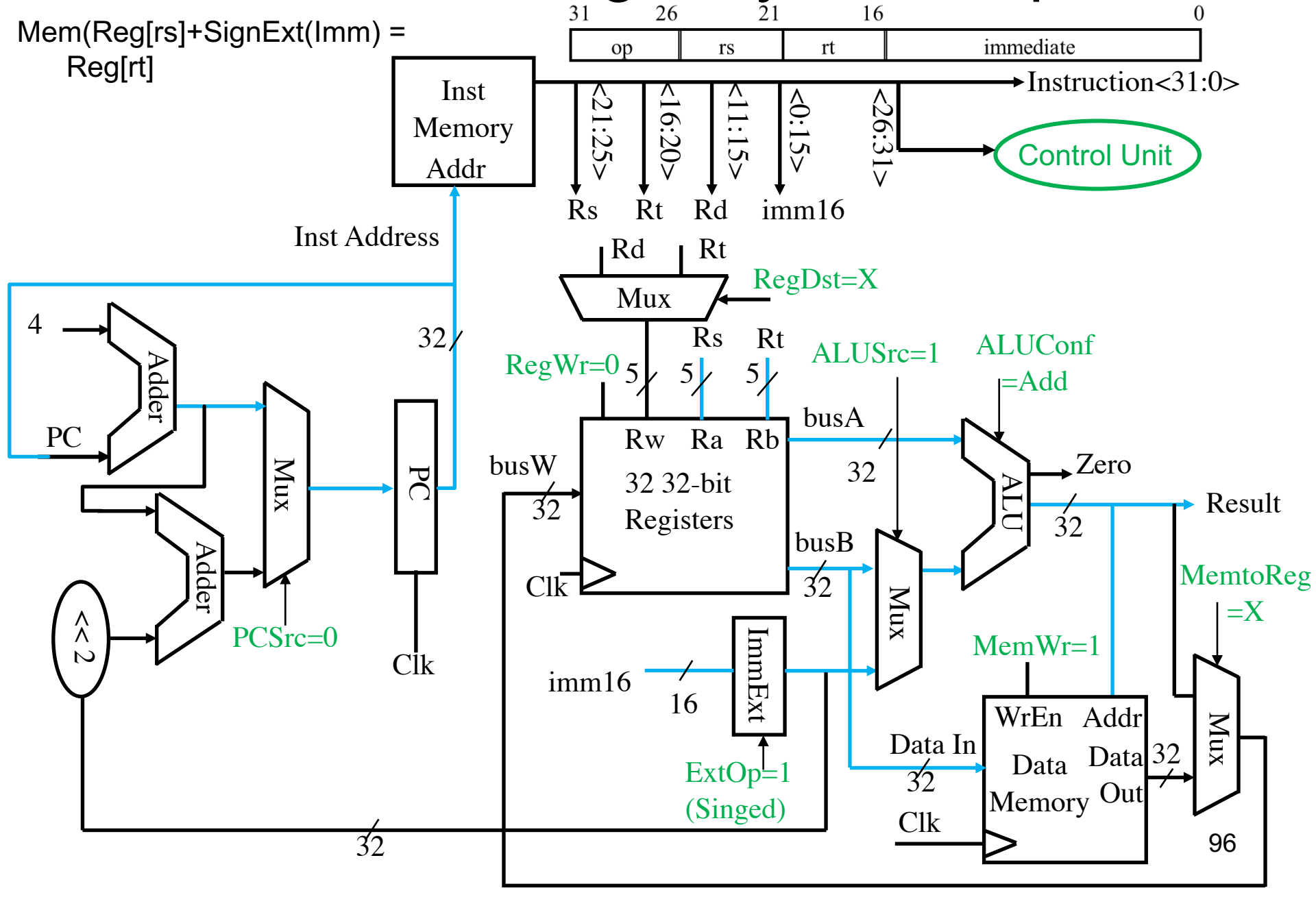
Load指令的Single Cycle Datapath



Store指令的Single Cycle Datapath

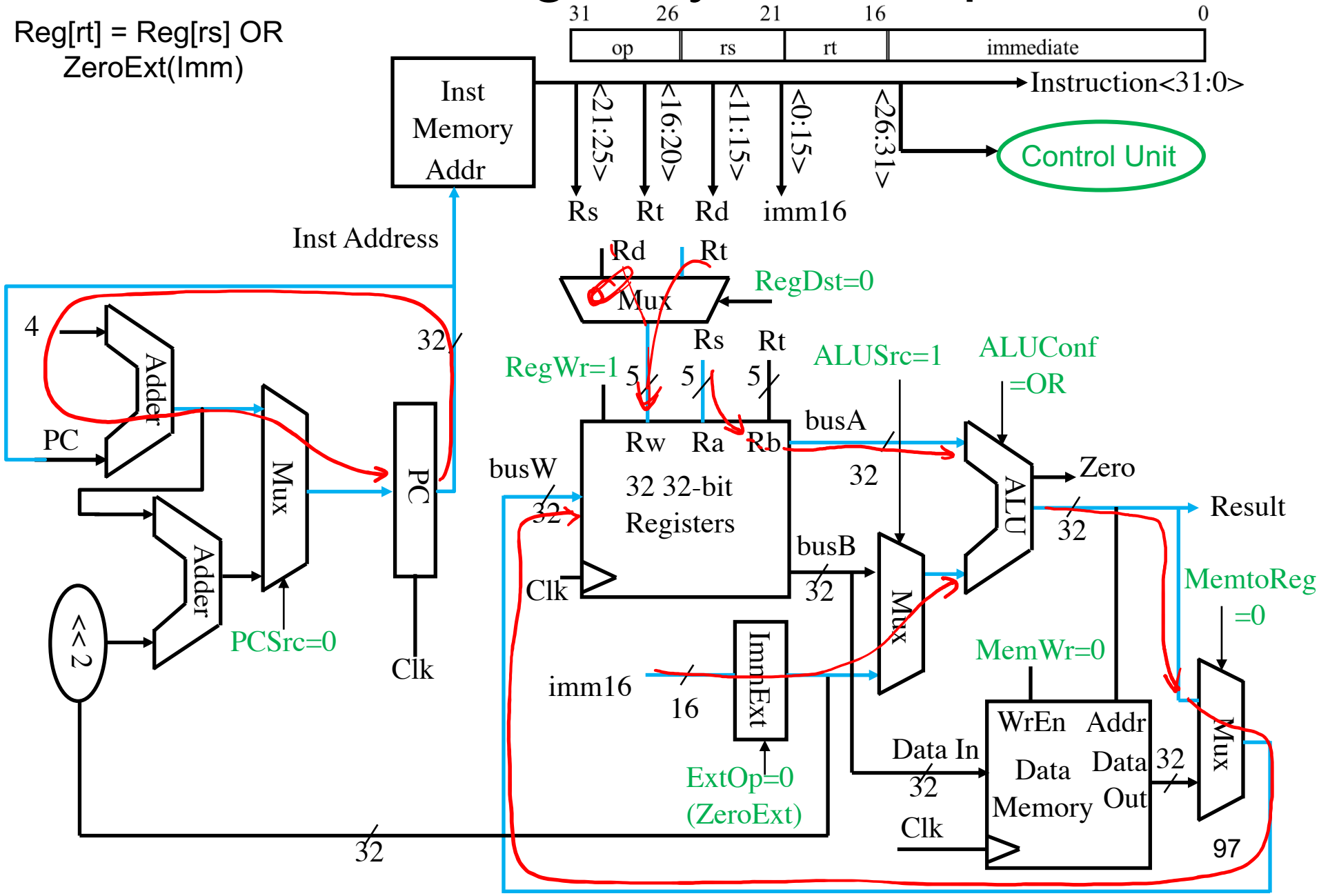


Store指令的Single Cycle Datapath

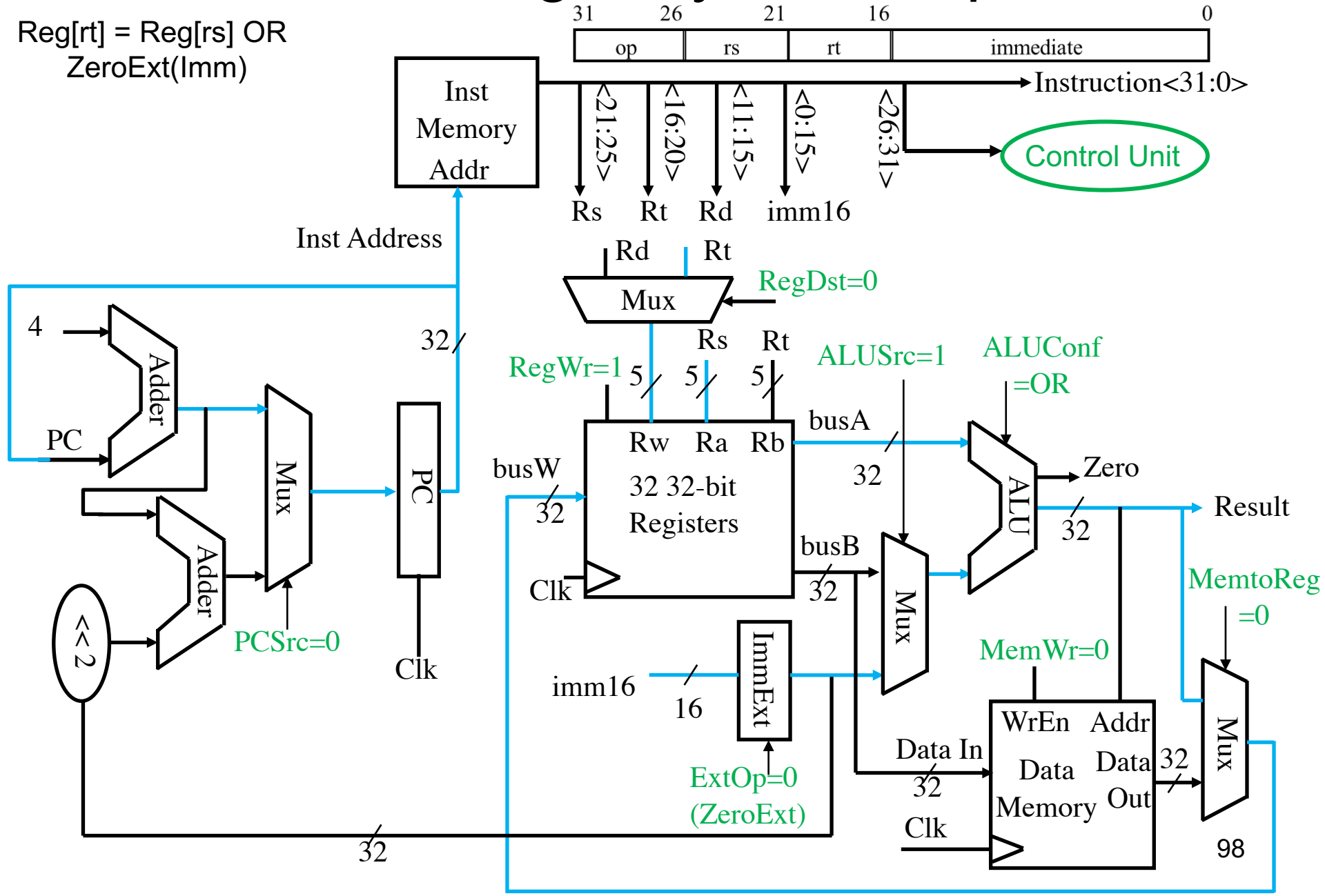


Ori指令的Single Cycle Datapath

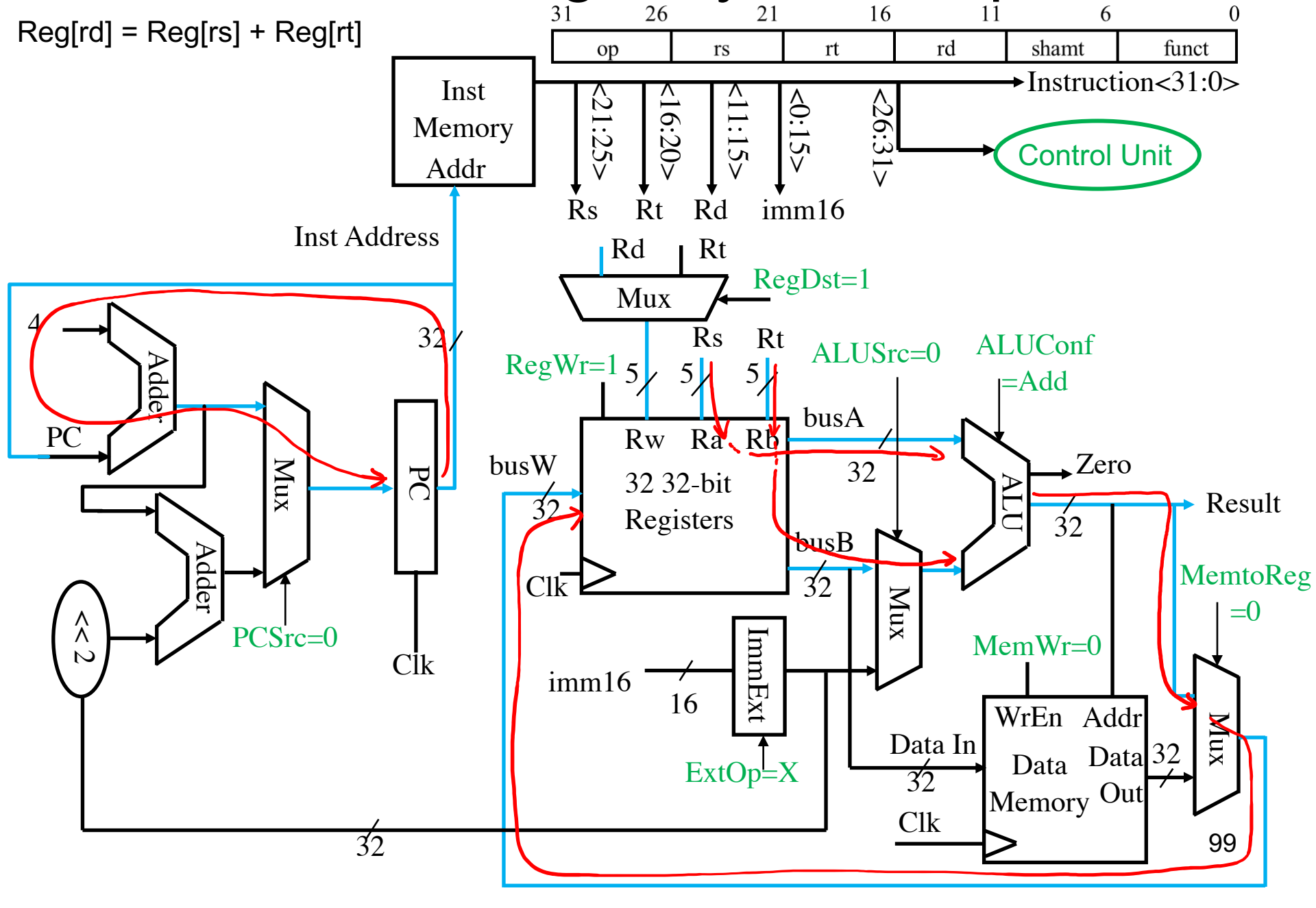
Reg[rt] = Reg[rs] OR ZeroExt(Imm)



Ori指令的Single Cycle Datapath

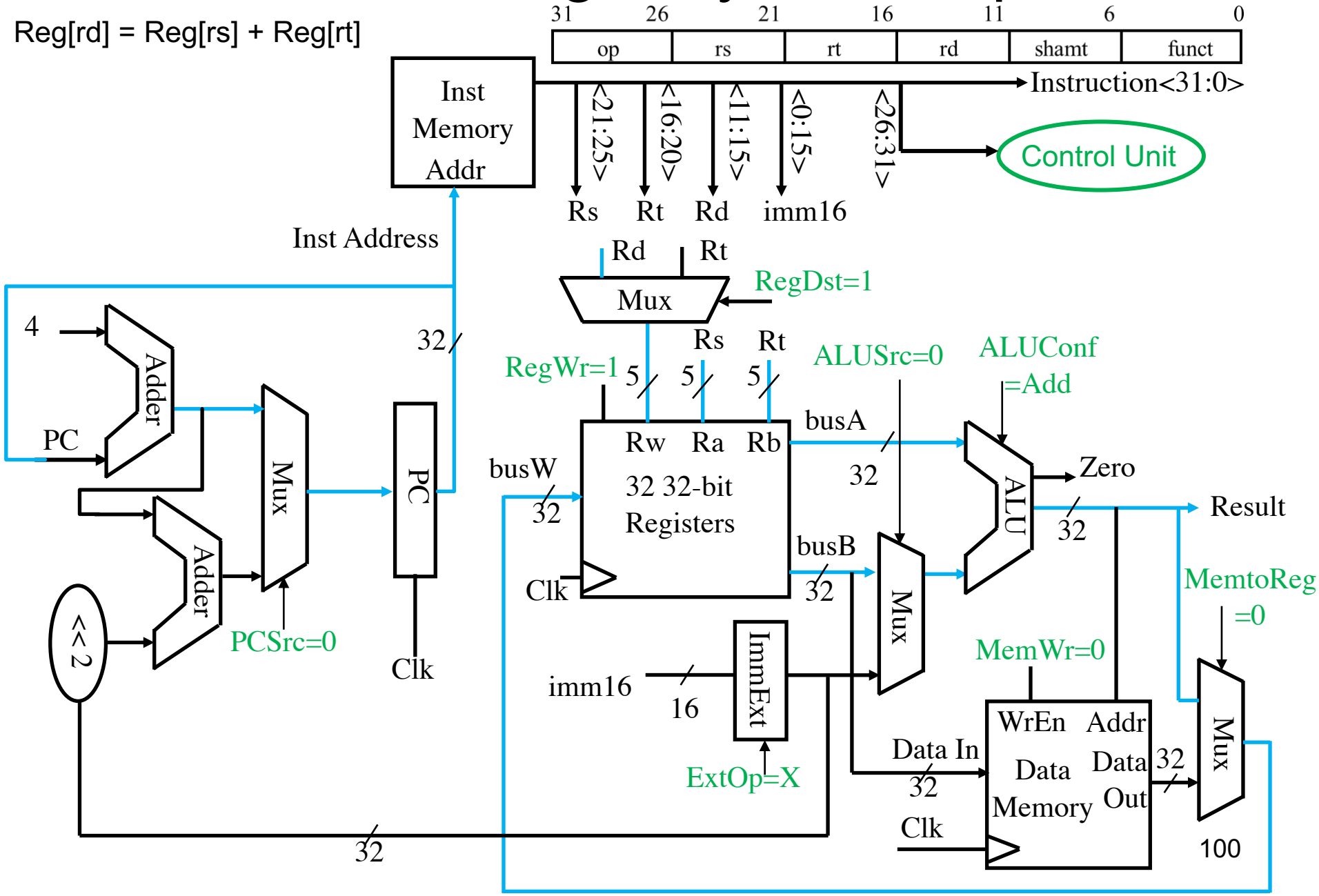


Add指令的Single Cycle Datapath



Add指令的Single Cycle Datapath

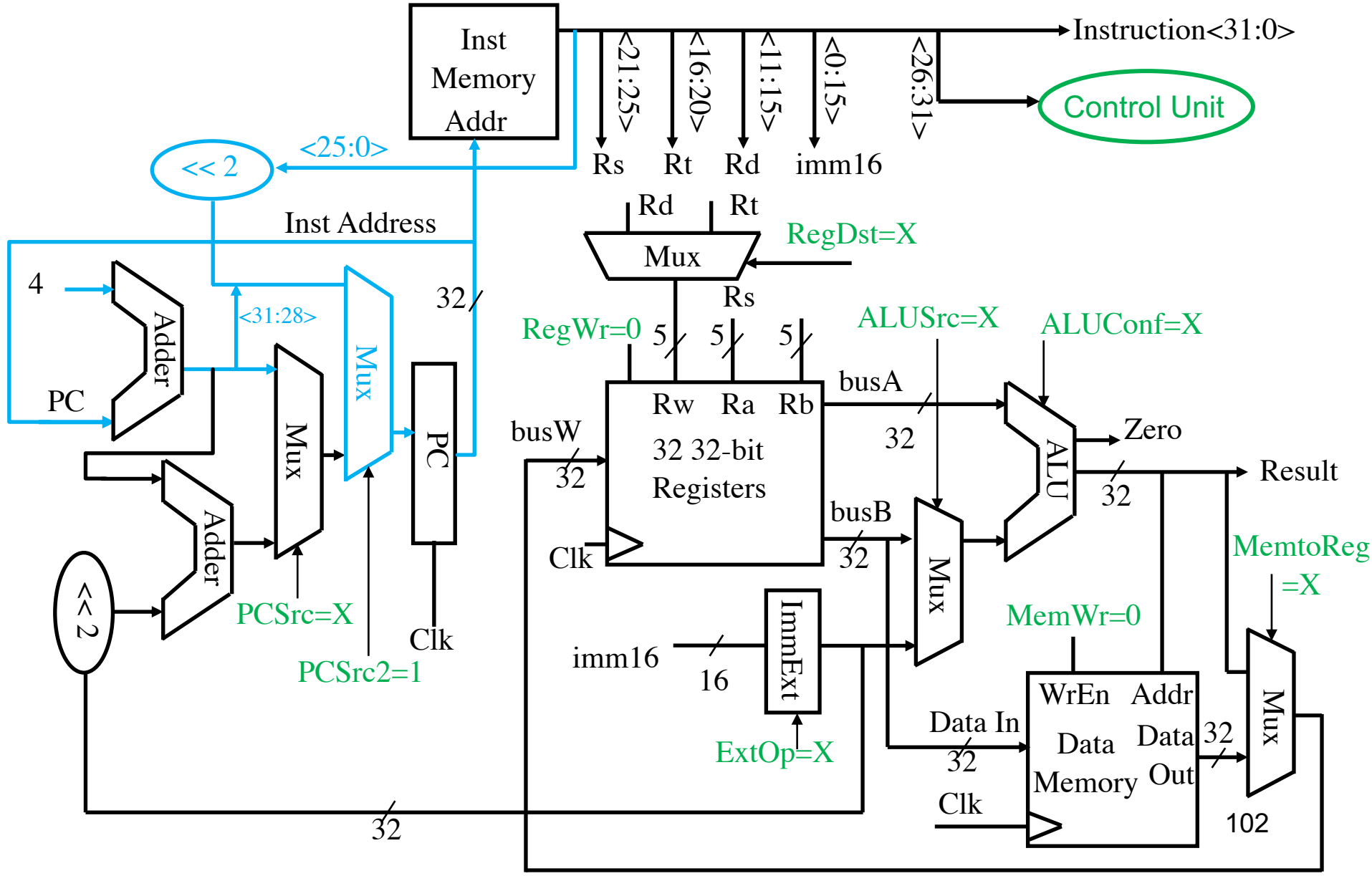
Reg[rd] = Reg[rs] + Reg[rt]



MIPS子集控制信号小结

inst	Register Transfer	
ADD	R[rd] <= R[rs] + R[rt];	PC <= PC + 4
	ALUsrc = RegB, ALUConf = “add”, RegDst = rd, RegWr, PCSrc = “+4”	
SUB	R[rd] <= R[rs] – R[rt];	PC <= PC + 4
	ALUsrc = RegB, ALUConf = “sub”, RegDst = rd, RegWr, PCSrc = “+4”	
ORi	R[rt] <= R[rs] + zero_ext(Imm16);	PC <= PC + 4
	ALUsrc = Im, Extop = “Z”, ALUConf = “or”, RegDst = rt, RegWr, PCSrc = “+4”	
LOAD	R[rt] <= MEM[R[rs] + sign_ext(Imm16)];	PC <= PC + 4
	ALUsrc = Im, Extop = “Sn”, ALUConf = “add”, MemtoReg, RegDst = rt, RegWr,	PCSrc = “+4”
STORE	MEM[R[rs] + sign_ext(Imm16)] <= R[rs];	PC <= PC + 4
	ALUsrc = Im, Extop = “Sn”, ALUConf = “add”, MemWr, PCSrc = “+4”	
BEQ	if (R[rs] == R[rt]) then PC <= PC +4 + {sign_ext(Imm16)], 00’b2} else PC<=PC + 4	
	PCSrc = “Br”, Extop = “Sn”, ALUConf = “sub”, PCSrc = “Br” and “Zero”	

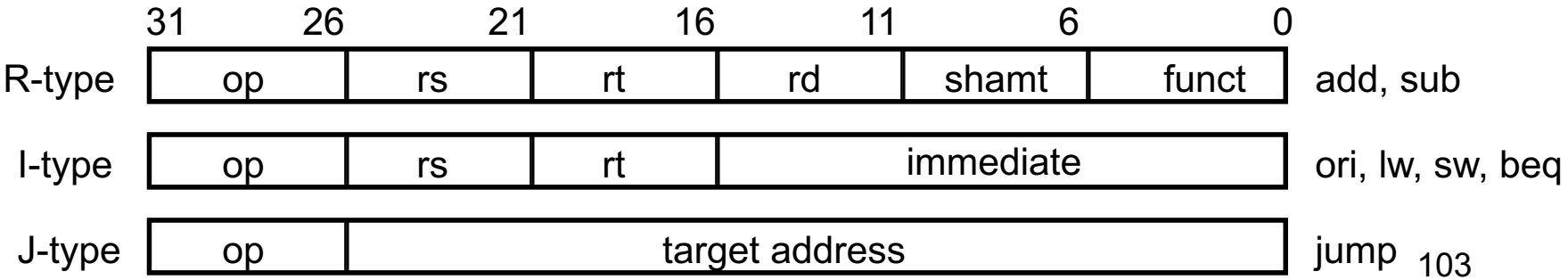
扩展实现：跳转指令


Inst
Memory
Addr
Instruction<31:0>
<21:25>
<16:20>
<11:15>
<0:15>
<26:31>
Control Unit
Rs
Rt
Rd
imm16
<< 2
<25:0>
Inst Address
Rd
Rt
Mux
RegDst=X
Rs
4
Adder
<31:28>
32
RegWr=0
5
5
5
ALUSrc=X
ALUConf=X
busA
Rw
Ra
Rb
busW
32 32-bit
Registers
32
Zero
ALU
Result
32
32
Clk
busB
32
PC
Mux
Mux
Adder
<< 2
PCSrc=X
PCSrc2=1
Clk
MemtoReg
=X
MemWr=0
imm16
16
ImmExt
ExtOp=X
Mux
WrEn
Addr
Data In
Data
Data
32
Memory
Out
32
Clk
Mux
102

MIPS子集控制信号小结

See Appendix A

	add	sub	ori	lw	sw	beq	jump
func	10 0000	10 0010	We Don't Care :-)				
op	00 0000	00 0000	00 1101	10 0011	10 1011	00 0100	00 0010
RegDst	1	1	0	0	x	x	x
ALUSrc	0	0	1	1	1	0	x
MemtoReg	0	0	0	1	x	x	x
RegWrite	1	1	1	1	0	0	0
MemWrite	0	0	0	0	1	0	0
PCSrc	0	0	0	0	0	ZERO	0
ExtOp	x	x	0	1	1	1	x
ALUConf<2:0>	Add	Subtract	Or	Add	Add	Subtract	xxx



小结

- 5个步骤设计单周期处理器
  1. 分析指令集 => datapath 的需求
  2. 确定datapath的组成模块
  3. 组装datapath
  4. 通过分析指令确定影响寄存器转移的控制信号.
  5. 设计控制信号
- MIPS 实现这样一个功能比较容易
  - 指令结构简单,等长; 源寄存器, 立即数总是在相同的位置
  - 只对寄存器内容或者立即数进行操作,操作类型少
- 单周期datapath=> CPI=1, 时钟周期长

小结

- 单周期MIPS处理器执行阶段

1. 取指令 (Instruction Fetch, IF)
2. 指令译码和控制信号生成 (Instruction Decode, ID)
3. 操作执行 (Execution, EX)
4. 内存访问 (Memory Access, MEM)
5. 写回寄存器堆 (Write Back, WB)