



MIPS32架构CPU设计实例

2025-8-22



信息与软件工程学院
School of Information and Software Engineering



主要内容

- 1 模型机的总体设计
- 2 算术逻辑运算部件
- 3 运算方法
- 4 模型机的组合逻辑控制器
- 5 模型机的微程序控制器
- 6 MIPS32架构CPU设计实例



主要内容

- 1 MIPS32指令架构
- 2 基本组成部件
- 3 单周期MIPS32处理器

(1)存储器按字节编址

(2)可用寄存器32个，宽度32位

(3)RISC架构

结合高级语言编程，考虑处理器应该有哪些类型的指令？

运算？ 访存？ 转移？ ...

※可提供的寄存器列表

寄存器名	地址编号		用途说明
\$zero	0	✓	保存固定的常数0
\$at	1	✓	汇编器专用
\$v0~\$v1	2~3	✓	表达式计算或函数调用的返回结果
\$a0~\$a3	4~7	✓	函数调用参数
\$t0~\$t7	8~15	✓	临时变量，函数调用时不需要保存和恢复
\$s0~\$s7	16~23	✓	函数调用时需要保存和恢复的寄存器变量
\$t8~\$t9	24~25	✓	临时变量，函数调用时不需要保存和恢复
\$k0~\$k1	26~27	✓	操作系统专用
\$gp	28	✓	全局指针变量(Global Pointer)
\$sp	29	✓	堆栈指针变量(Stack Pointer)
\$fp	30	✓	帧指针变量(Frame Pointer)
\$ra	31	✓	返回地址(Return Address)

指令字长固定为32位，寄存器型寻址，指令中给出寄存器号。

指令类型	指令长度（32位定长）					
	31 ~ 26	25~21	20~16	15~11	10 ~ 6	5 ~ 0
R型	op(6)	rs(5)	rt(5)	rd(5)	shamt	func
I型	op(6)	rs(5)	rt(5)	imm (16)		
J型	op(6)	address(26)				

一、指令格式与指令集

■ R型指令(Register) 典型R型指令见表3-22

指令长度 (32位定长)					
31 ~ 26	25~21	20~16	15~11	10 ~ 6	5 ~ 0
op(6)	rs(5)	rt(5)	rd(5)	sa	func

操作数和保存结果均通过寄存器进行；

- ◆ op: 操作码，所有R型指令中都全为0；
- ◆ rs: 寄存器编号，对应第1个源操作数；
- ◆ rt: 寄存器编号，对应第2个源操作数；
- ◆ rd: 寄存器编号，据此保存结果；
- ◆ sa: 常数，在移位指令中使用；
- ◆ func: 功能码，指定指令的具体功能；

一、指令格式与指令集

■ I型指令(Immediate) 典型I型指令见表3-23

指令长度 (32位定长)					
31 ~ 26	25~21	20~16	15~11	10 ~ 6	5 ~ 0
op(6)	rs(5)	rt(5)	imm (16)		

操作数中涉及立即数，结果保存到寄存器；

- ◆ op: 标识指令的操作功能；
- ◆ rs: 第1个源操作数，是寄存器操作数；
- ◆ rt: 目的寄存器编号，用来保存运算结果；
- ◆ imm: 第2个源操作数，立即数；

■ J型指令(Jump) 典型J型指令见表3-24

指令长度 (32位定长)					
31 ~ 26	25~21	20~16	15~11	10 ~ 6	5 ~ 0
op(6)	address(26)				

实现无条件转移；

◆ op: 确定指令的功能；

◆ address: 转移目标地址的偏移量字段；

在MIPS32指令集中，寻址方式通过`op`字段和`func`字段(`func`针对R型指令)隐含说明。

R型指令：

由`op`和`func`字段共同隐含说明当前的寻址方式；

I型和J型指令：


由`op`字段隐含说明当前指令使用的寻址方式。

● 立即数寻址(Immediate addressing)

操作数在指令中的立即数字段。



addi s1, s2, **10**

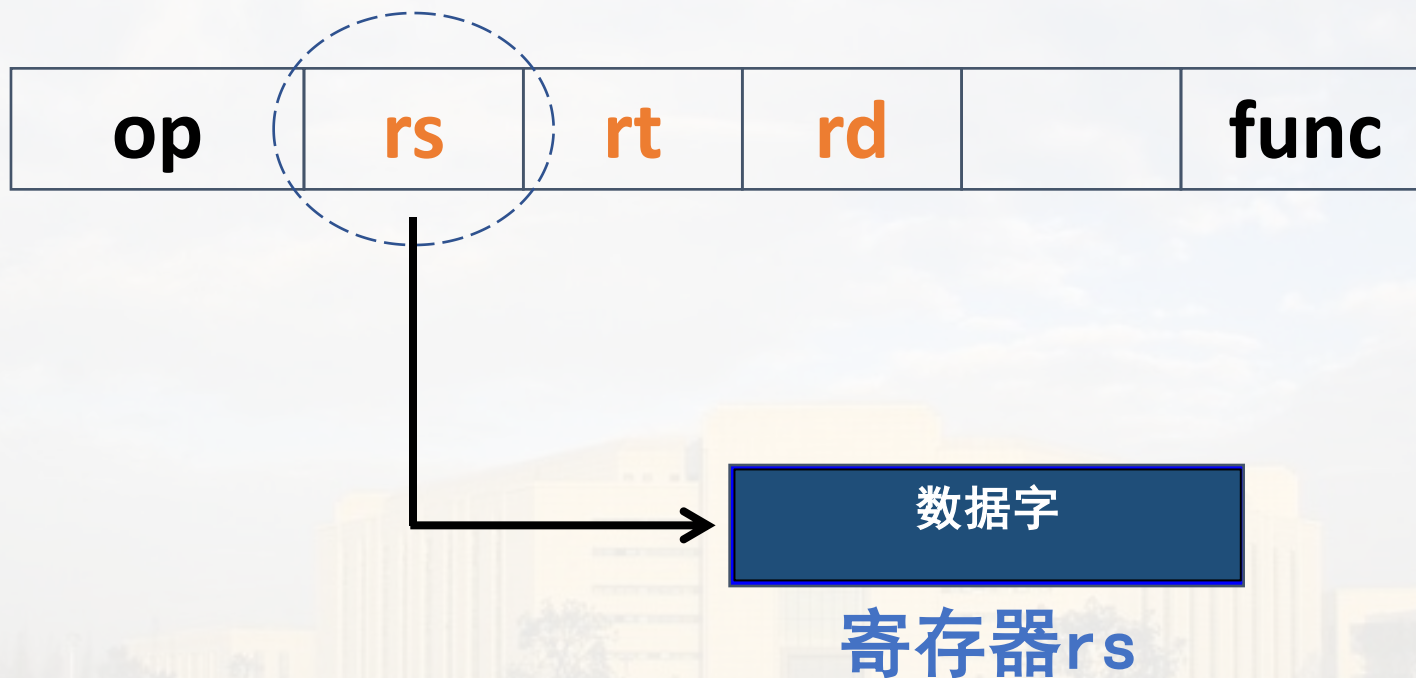


\$s2+10→\$s1

注意：汇编格式和编码格式段的对应关系。

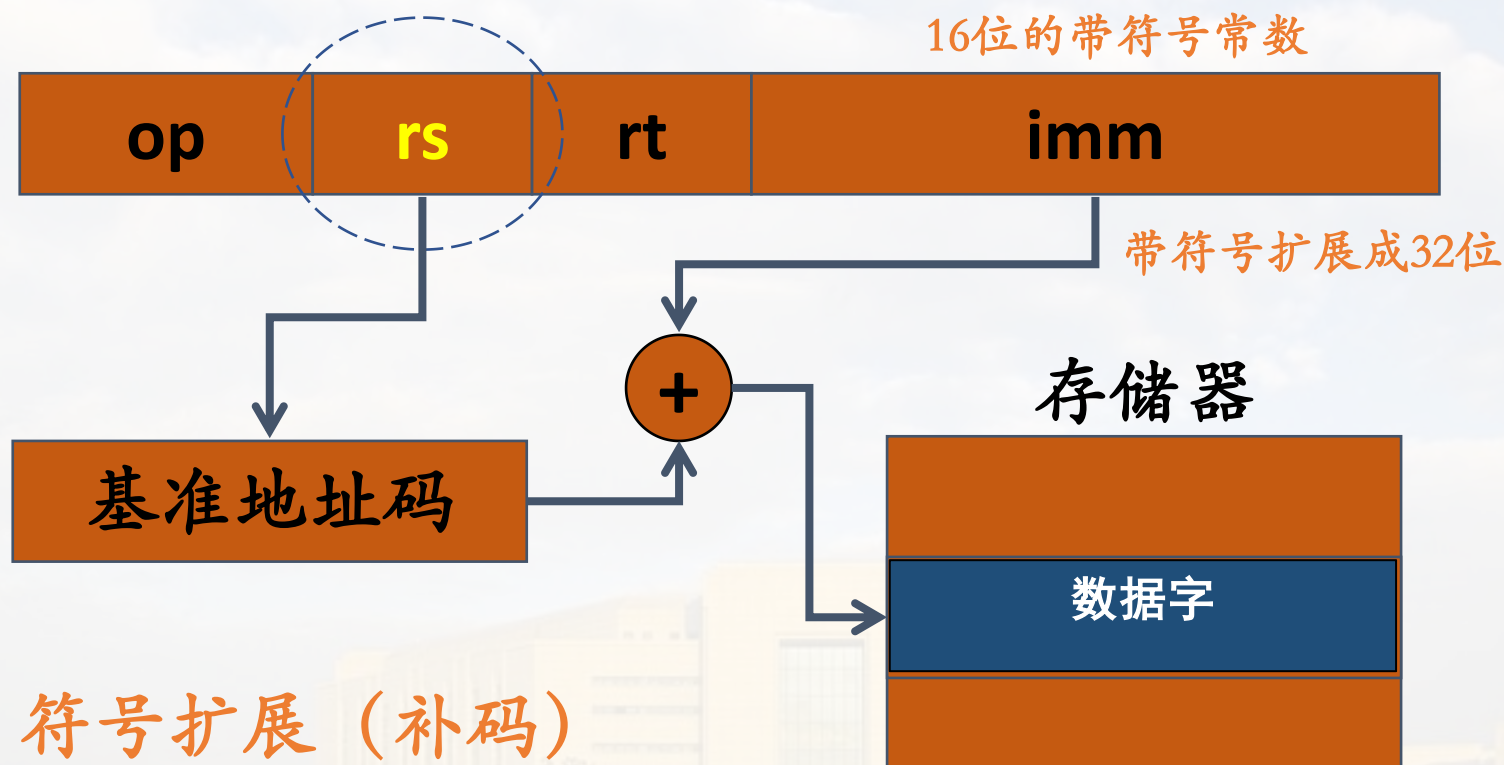
● 寄存器直接寻址(Register Addressing)

操作数直接在寄存器中。



● 基址寻址(Basic Addressing)

操作数地址由寄存器和立即数字段联合产生



符号扩展 (补码)

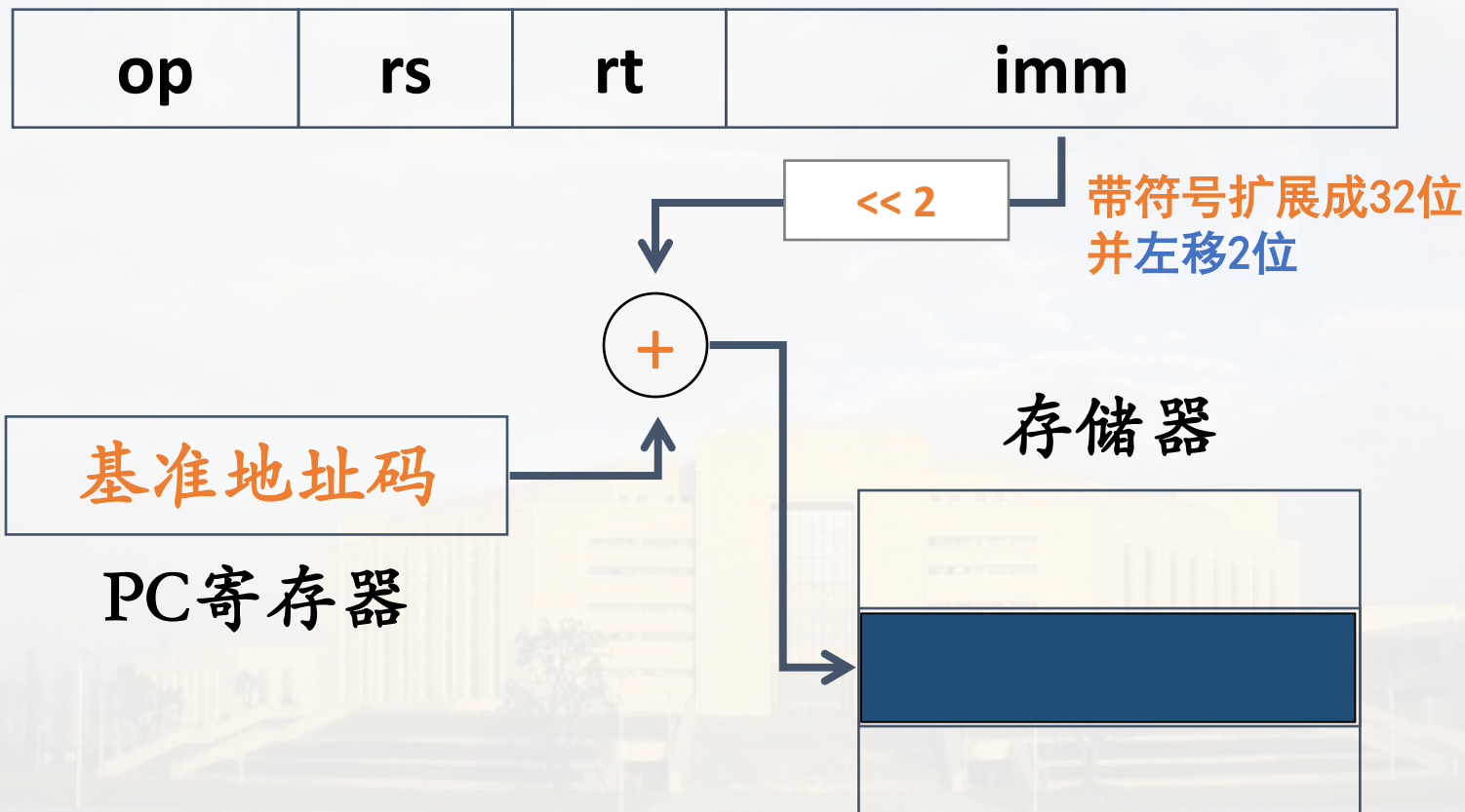
正数, 在高位补上16个0; $012A_H \rightarrow \underline{0000}012A_H$

负数, 在高位补上16个1; $812A_H \rightarrow 0\underline{FFFF}812A_H$

● PC相对寻址(PC-relative Addressing)

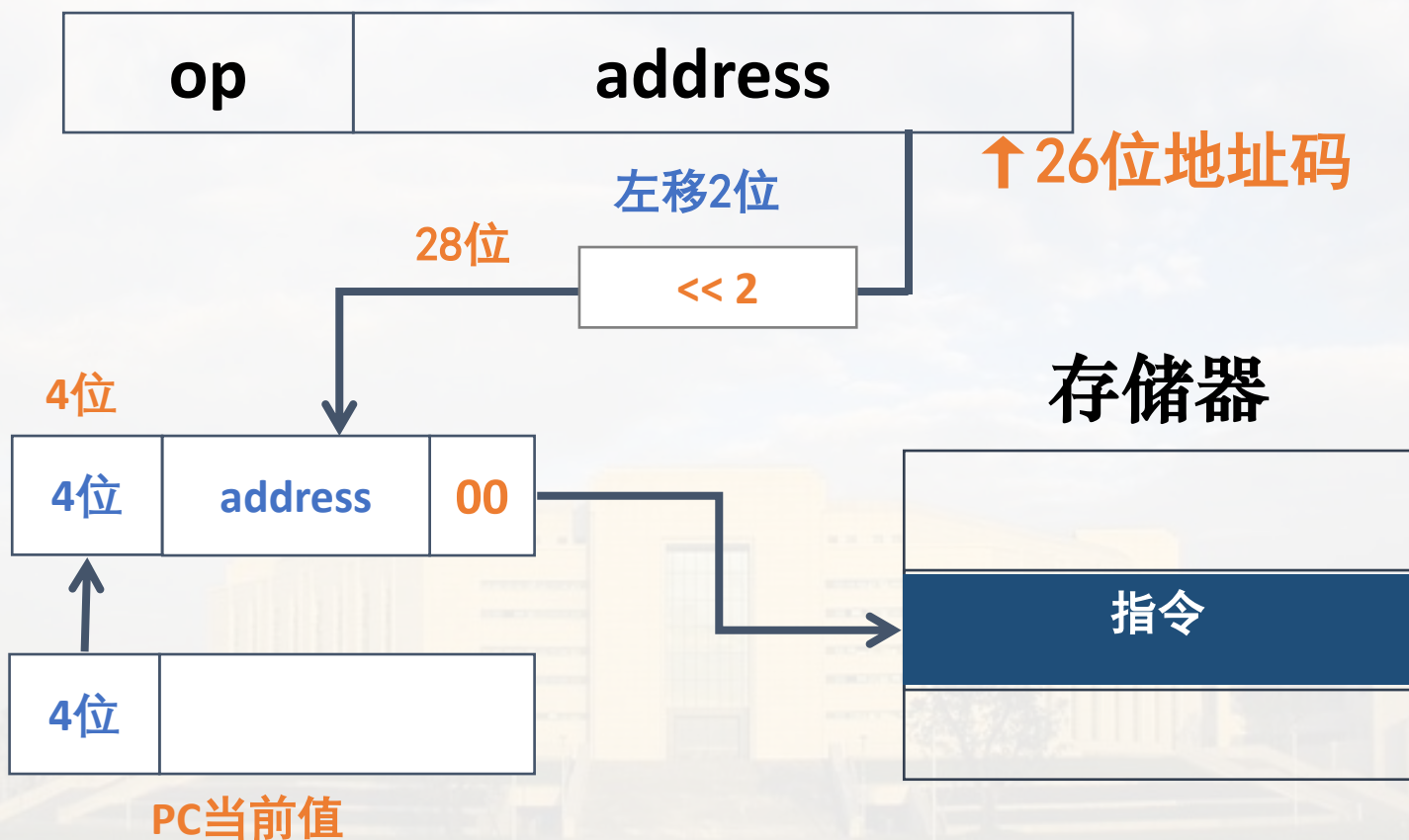
隐含使用PC作为基址寄存器

16位的带符号常数



●伪直接寻址 (Pseudo-direct Addressing)

也叫**页面寻址**，由PC高4位与指令中的地址段组合产生有效地址。



✘ R型指令(只列出了10条)

指令	[31:26]	[25:21]	[20:16]	[15:11]	[10:6]	[5:0]	指令功能
add	000000	rs	rt	rd	<u>00000</u>	100000	寄存器加
Sub	000000	rs	rt	rd	<u>00000</u>	100010	寄存器减
and	000000	rs	rt	rd	<u>00000</u>	100100	寄存器与
or	000000	rs	rt	rd	<u>00000</u>	100101	寄存器或
xor	000000	rs	rt	rd	<u>00000</u>	100110	寄存器异或
slt	000000	rs	rt	rd	<u>00000</u>	101010	比较并置1
sll	000000	<u>00000</u>	rt	rd	sa	000000	逻辑左移
srl	000000	<u>00000</u>	rt	rd	sa	000010	逻辑右移
sra	000000	<u>00000</u>	rt	rd	sa	000011	算术右移
jr	000000	rs	<u>00000</u>	<u>00000</u>	<u>00000</u>	001000	寄存器跳转

三、指令功能

R型指令由操作码`op`配合`func`字段，确定具体的操作；

R型指令有三类：

① 3寄存器R型指令

指令	[31:26]	[25:21]	[20:16]	[15:11]	[10:6]	[5:0]	指令功能
<code>add</code>	000000	rs	rt	rd	<u>00000</u>	100000	寄存器加
<code>sub</code>	000000	rs	rt	rd	<u>00000</u>	100010	寄存器减
<code>and</code>	000000	rs	rt	rd	<u>00000</u>	100100	寄存器与
<code>or</code>	000000	rs	rt	rd	<u>00000</u>	100101	寄存器或
<code>xor</code>	000000	rs	rt	rd	<u>00000</u>	100110	寄存器异或
<code>slt</code>	000000	rs	rt	rd	<u>00000</u>	101010	比较并置1

`add/sub/and/or/xor rd, rs, rt; slt rd, rs, rt`

指令功能： $\$rd \leftarrow \$rs \text{ op } \$rt$; if $\$rs < \rt $\$rd = 1$, else $\$rd = 0$

② 2寄存器R型指令

指令	[31:26]	[25:21]	[20:16]	[15:11]	[10:6]	[5:0]	指令功能
sll	000000	<u>00000</u>	rt	rd	sa	000000	逻辑左移
srl	000000	<u>00000</u>	rt	rd	sa	000010	逻辑右移
sra	000000	<u>00000</u>	rt	rd	sa	000011	算术右移

sll/srl/sra rd, rt, sa;

指令功能: $\$rd \leftarrow \$rt \text{ shift } sa;$

③ 1寄存器R型指令

指令	[31:26]	[25:21]	[20:16]	[15:11]	[10:6]	[5:0]	指令功能
jr	000000	rs	<u>00000</u>	<u>00000</u>	<u>00000</u>	001000	寄存器跳转

jr rs; 指令功能: $PC \leftarrow \$rs;$

※I型指令(只列出9条)

指令	[31:26]	[25:21]	[20:16]	[15:0]	指令功能
addi	001000	rs	rt	imm	寄存器和立即数“加”
andi	001100	rs	rt	imm	寄存器和立即数“与”
ori	001101	rs	rt	imm	寄存器和立即数“或”
xori	001110	rs	rt	imm	寄存器和立即数“异或”
lw	100011	rs	rt	imm	从存储器中读取数据
sw	101011	rs	rt	imm	把数据保存到存储器
beq	000100	rs	rt	imm	寄存器相等则转移
bne	000101	rs	rt	imm	寄存器不等则转移
lui	001111	<u>00000</u>	rt	imm	设置寄存器的高16位

I型指令有4类:

①面向运算的I型指令

指令	[31:26]	[25:21]	[20:16]	[15:0]	指令功能
addi	001000	rs	rt	imm	寄存器和立即数“加”
andi	001100	rs	rt	imm	寄存器和立即数“与”
ori	001101	rs	rt	imm	寄存器和立即数“或”
xori	001110	rs	rt	imm	寄存器和立即数“异或”

addi rt, rs, **imm**; # \$rt \leftarrow \$rs + E(imm)

这里的**imm**为数值型数据，故“带符号扩展”。

andi/ori/xori rt, rs, **imm**; # \$rt \leftarrow \$rs op E(imm)

这里的**imm**为逻辑型数据，故“无符号扩展”。

②面向访存的I型指令

指令	[31:26]	[25:21]	[20:16]	[15:0]	指令功能
lw	100011	rs	rt	imm	从存储器中读取数据
sw	101011	rs	rt	imm	把数据保存到存储器

MIPS32中唯一两条访问存储器的指令(RISC)

lw rt, imm(rs) # $\$rt \leftarrow \text{mem}[\$rs + E(\text{imm})]$

sw rt, imm(rs) # $\text{mem}[\$rs + E(\text{imm})] \leftarrow \rt

符号
扩展

③面向数位设置的I型指令

指令	[31:26]	[25:21]	[20:16]	[15:0]	指令功能
lui	001111	<u>00000</u>	rt	imm	设置寄存器的高16位

lui rt, imm # $\$rt \leftarrow \text{imm} \ll 16$ (空位补0)

④面向条件转移(分支)的I型指令

指令	[31:26]	[25:21]	[20:16]	[15:0]	指令功能
beq	000100	rs	rt	imm	寄存器相等则转移
bne	000101	rs	rt	imm	寄存器不等则转移

beq rs, rt, imm

#if(\$rs==\$rt) $PC \leftarrow PC + 4 + E(imm) \ll 2$

bne rs, rt, imm

#if(\$rs!=\$rt) $PC \leftarrow PC + 4 + E(imm) \ll 2$

符号
扩展

PC相对寻址;

其中imm先带符号扩展成32位, 再左移2位。

※J型指令(列出2条)

指令	[31:26]	[25:0]	指令功能
j	000010	address	无条件跳转
jal	001100	address	子程序调用

j address;

指令功能: $\$PC \leftarrow (\$PC+4)_{H4} \cup (\text{address} \ll 2)$

jal address;

指令功能:

$\$ra \leftarrow \$PC+4$ (保存返回地址)

$\$PC \leftarrow (\$PC+4)_{H4} \cup (\text{address} \ll 2)$

采用页面寻址方式(伪直接寻址)。

基本部件

- ※ 存储相关部件
- ※ 数据预处理部件
- ※ 运算部件
- ※ 数据通路选择部件
- ※ 控制单元（控制器）

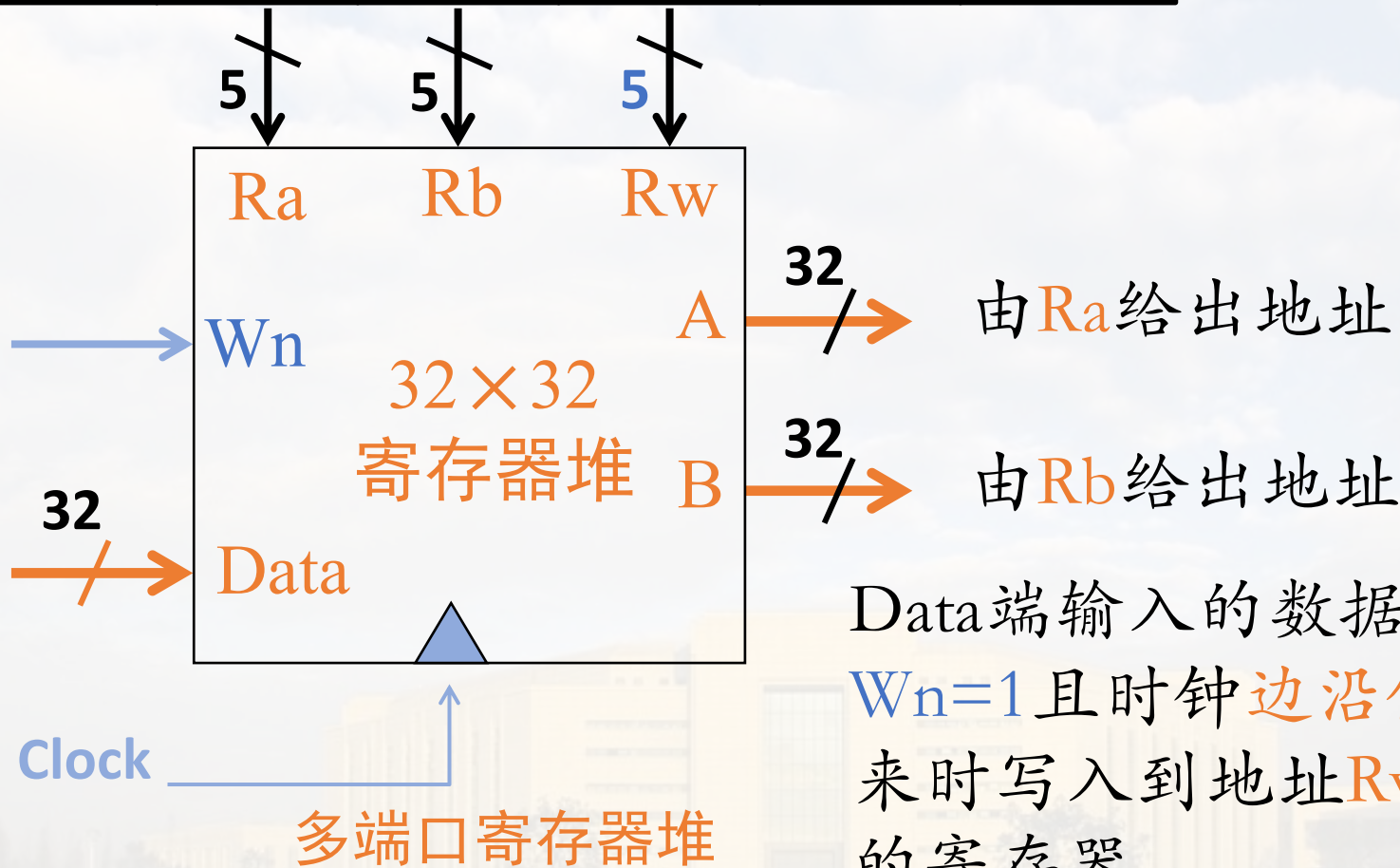
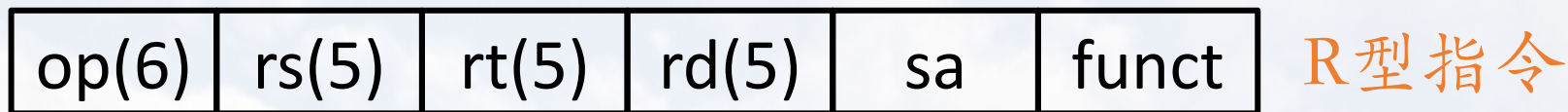
(1) 寄存器堆 (组)

- 读数据(根据指令中的rs或rt)
- 写数据(根据指令中的rt或rd)

32个32位寄存器

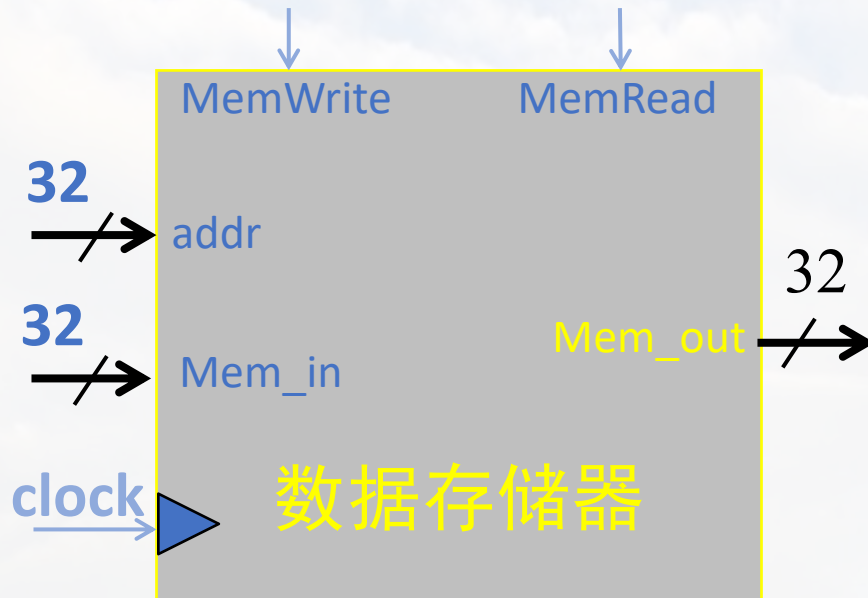
一般通过多端口小型存储器构成寄存器堆

一、主要功能部件



Data端输入的数据在
W_n=1且时钟边沿信号到
来时写入到地址R_w指定
的寄存器。

(2) 存储器



MemWrite	MemRead	模式
1	0	写
0	1	读
0	0	锁定
1	1	无效

(3) 特殊功能寄存器

- PC (程序计数器), IR (指令寄存器)
- FR (标志寄存器, PSW, 与运算器相关)

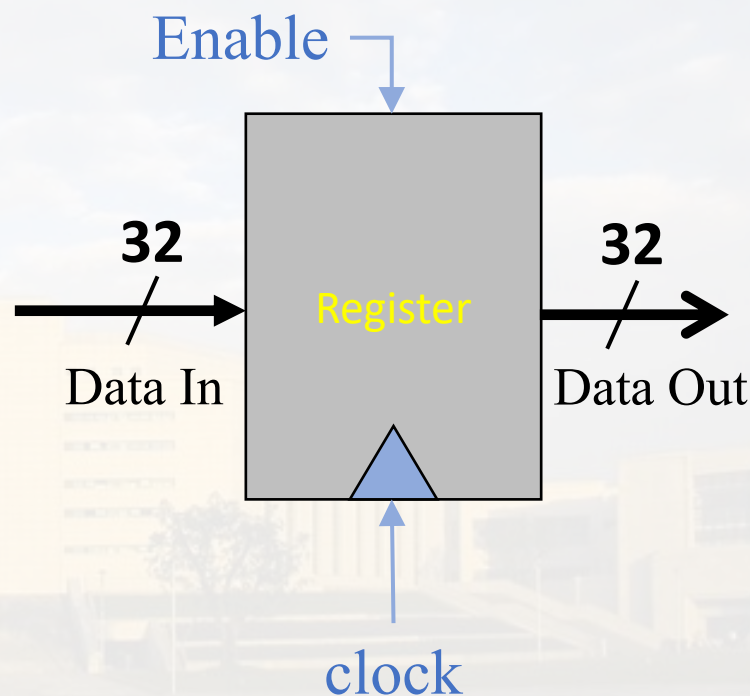
■ Enable: 写使能信号

- Enable=1

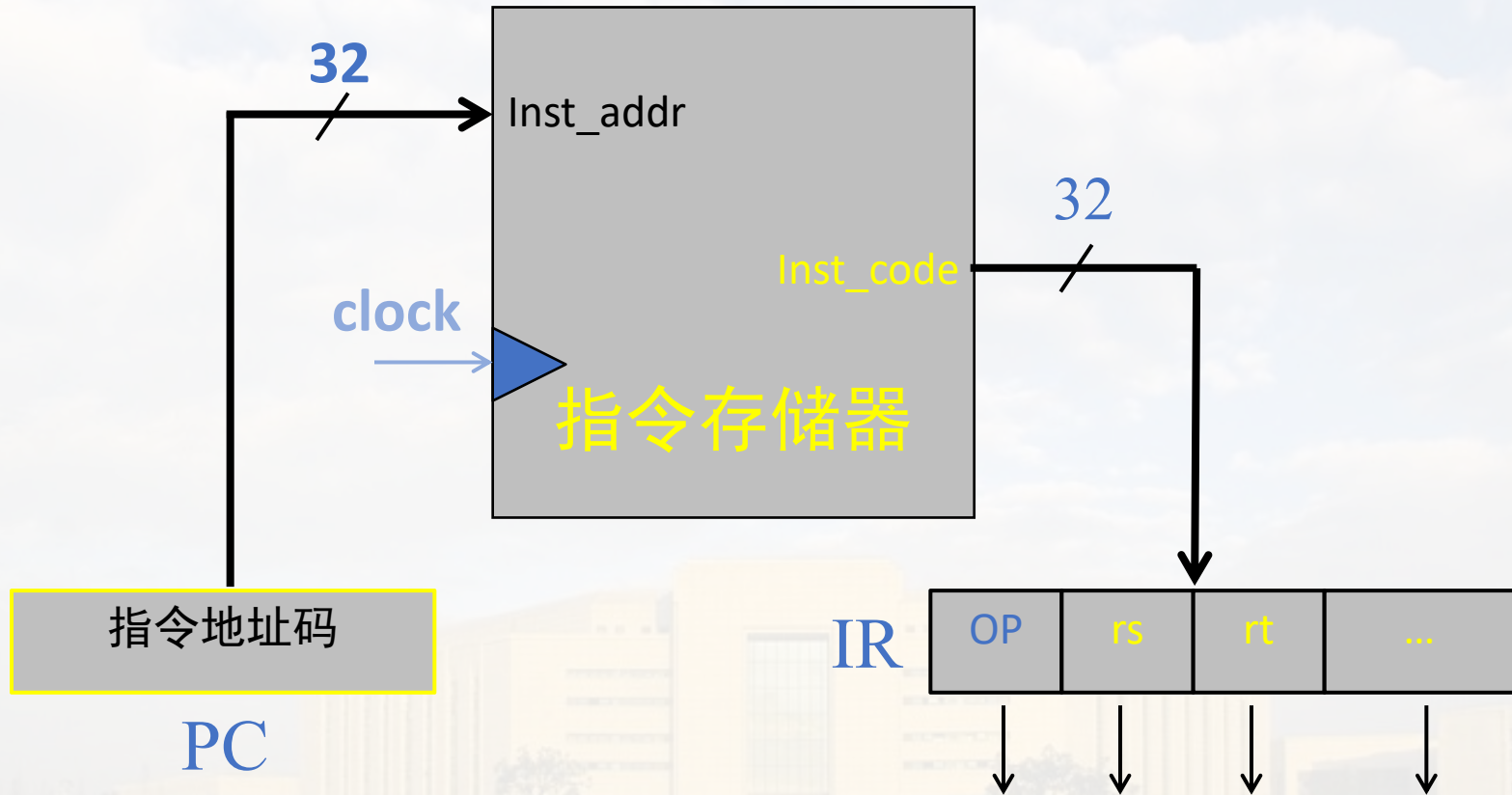
时钟边沿到来时, 写R

- Enable=0

读R

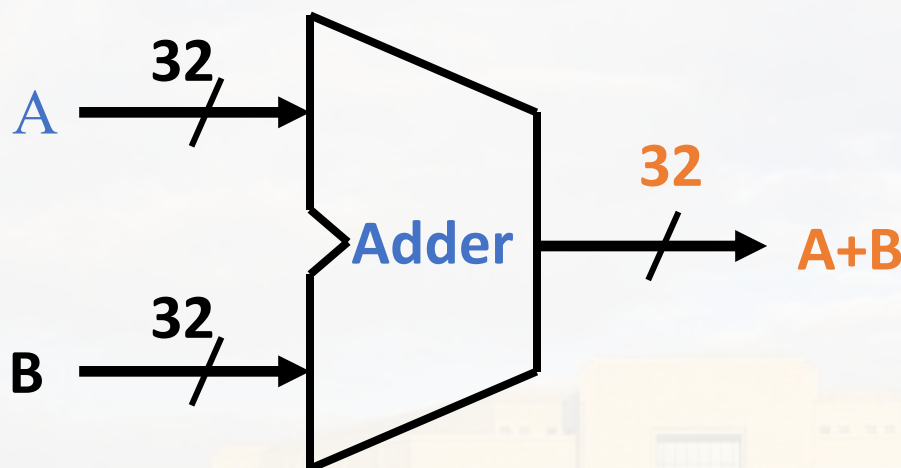


→ PC和IR在数据通路中的作用



包括单功能加法器 (Adder) 和多功能算术逻辑运算单元 (ALU)

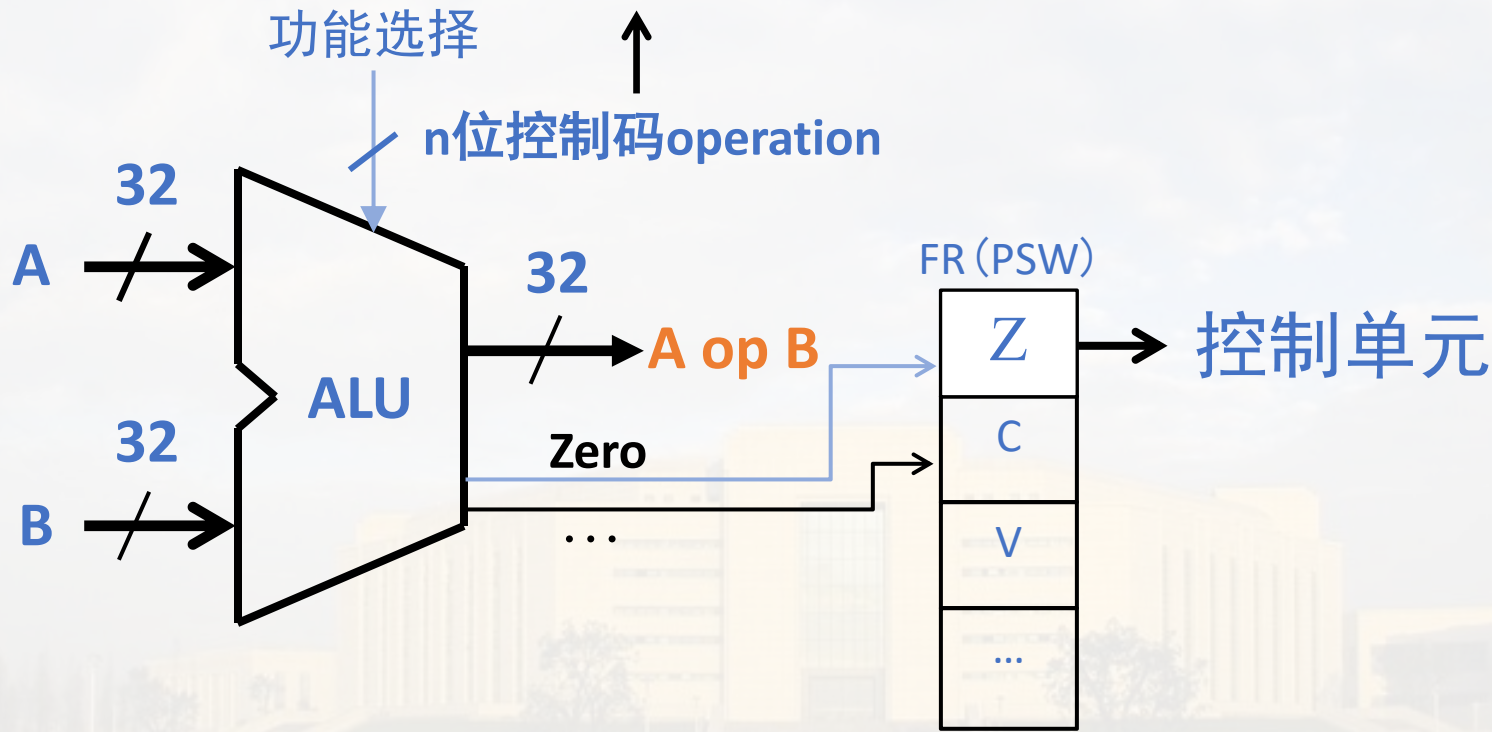
① 单功能32位加法器—Adder



加法器的输入端口A可以固定输入常数4，
即可用作PC自增单元（固定加4）

② 多功能32位运算器—ALU

n 与ALU的运算功能数 m 有关,
 $n \geq \log_2 m$



ALU功能与控制码operation示例

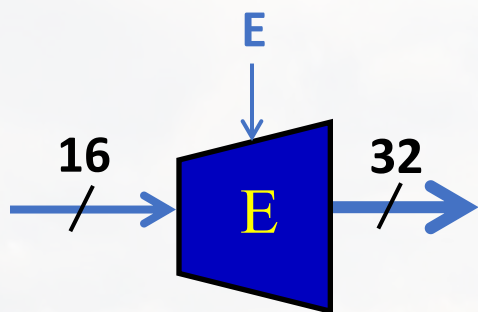
operation	ALU功能	ALU输出
0000	AND	A AND B
0001	OR	A OR B
0010	ADD	A ADD B
0110	SUB	A SUB B
0111	小于则置1	A < B?: 1
1100	NOR	Not (A OR B)
...

MIPS架构CPU涉及的预处理操作有三类：

- 位数扩展：带符号扩展、无符号扩展
- 左移两位
- 数位拼接：4位-28位拼接

① 数位扩展器—Extender

将16位数扩展为32位数，E为扩展模式控制端



16→32扩展器

① E=1时，数值型(补码)符号扩展

000A → 0000000A

800A → FFFF800A

正数高位全补0，负数高位均补1

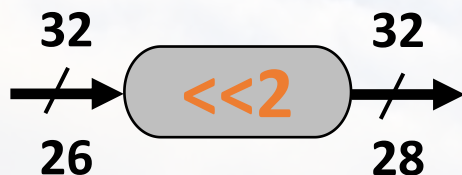
② E=0时，逻辑型扩展(零扩展)：

002A → 0000002A

F12C → 0000F12C

无正负性，高位均全补0

② 左移2位器

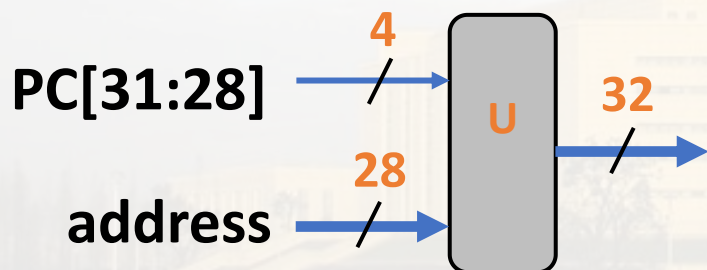


左移时空位自动补0

左移2位

[例] XXXX...XXXX \longrightarrow XX...XX00 (等效于乘4)

③ 2路拼接器



例如：

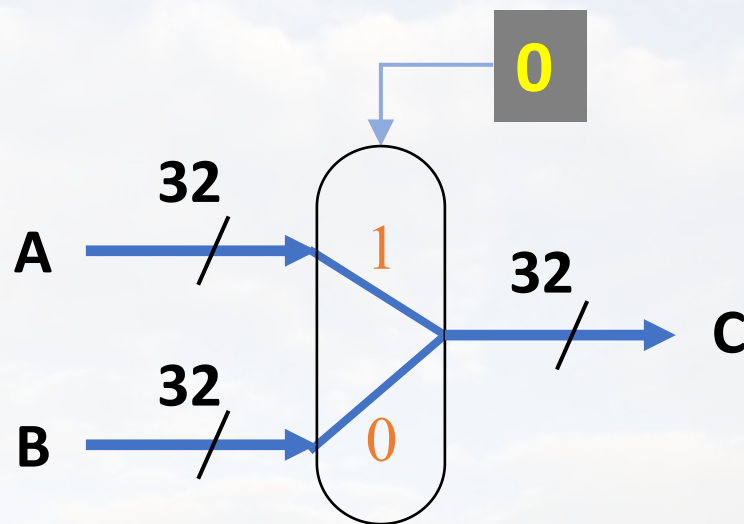
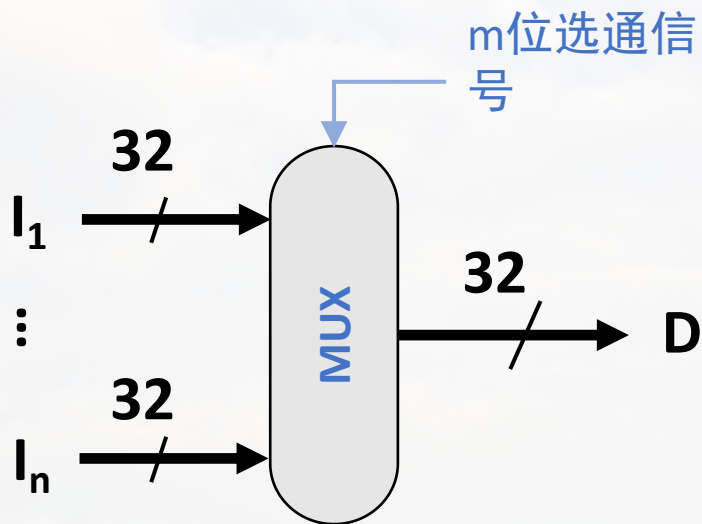
PC=A0000000

address=FFFFBB4

\rightarrow PC[31:28] U address
=AFFFFBB4

四、数据通路选择部件

多路选择器—MUX



其中, $2^m \geq n$

2路选择器

① 单周期CPU

指令固定在1个时钟周期内完成。

- ✓ 时间效率低，时钟宽度由单指令最长时间决定。
- ✓ 在指令周期内，功能部件不能共享，冗余度大；

② 多周期CPU

指令分散在多个时钟周期内完成。

- ✓ 时间效率高，时钟的宽度由单步最长时间决定。
- ✓ 不同的时钟周期之间，部件可共享，冗余降低。

一、待实现的目标指令子集 (11条)

序号	类型	指令	功能操作	寻址方式
1	R型 运算	add rd, rs, rt	$\$rs + \$rt \rightarrow \$rd$	R直接寻址
2		sub rd, rs, rt	$\$rs - \$rt \rightarrow \$rd$	
3		and rd, rs, rt	$\$rs \text{ and } \$rt \rightarrow \$rd$	
4		or rd, rs, rt	$\$rs \text{ or } \$rt \rightarrow \$rd$	
5	I型 运算	addi rt, rs, imm	$\$rs + E(\text{imm}) \rightarrow \rt	立即数寻址 R直接寻址
6		andi rt, rs, imm	$\$rs \text{ and } E(\text{imm}) \rightarrow \rt	
7		ori rt, rs, imm	$\$rs \text{ or } E(\text{imm}) \rightarrow \rt	
8	I型 访存	lw rt, imm(rs)	$\text{Mem}[\$rs + E(\text{imm})] \rightarrow \rt	立即寻址 R直接寻址 R基址寻址
9		sw rt, imm(rs)	$\$rt \rightarrow \text{Mem}[\$rs + E(\text{imm})]$	
10	I型 分支	beq rs, rt, imm	$\$rs = \$rt: PC + 4 + E(\text{imm}) \ll 2 \rightarrow PC$ $\$rs \neq \$rt: PC + 4 \rightarrow PC$	立即数寻址 PC相对寻址
11	J型 跳转	j address	$(PC + 4)[31:28] \cup (\text{address} \ll 2)$	立即数寻址 伪直接寻址

二、单周期数据通路设计

【基本思路】 面向指令功能，逐步扩展、融合

- 分析三类指令的格式和功能
- 选择功能部件，确定部件之间的连接通路
- 整合冗余的部件连线

取指令→

R型→

I型→

J型→

通路整合→

通路整合

完整的数据通路

经过多次扩展整合，得到最终数据通路。

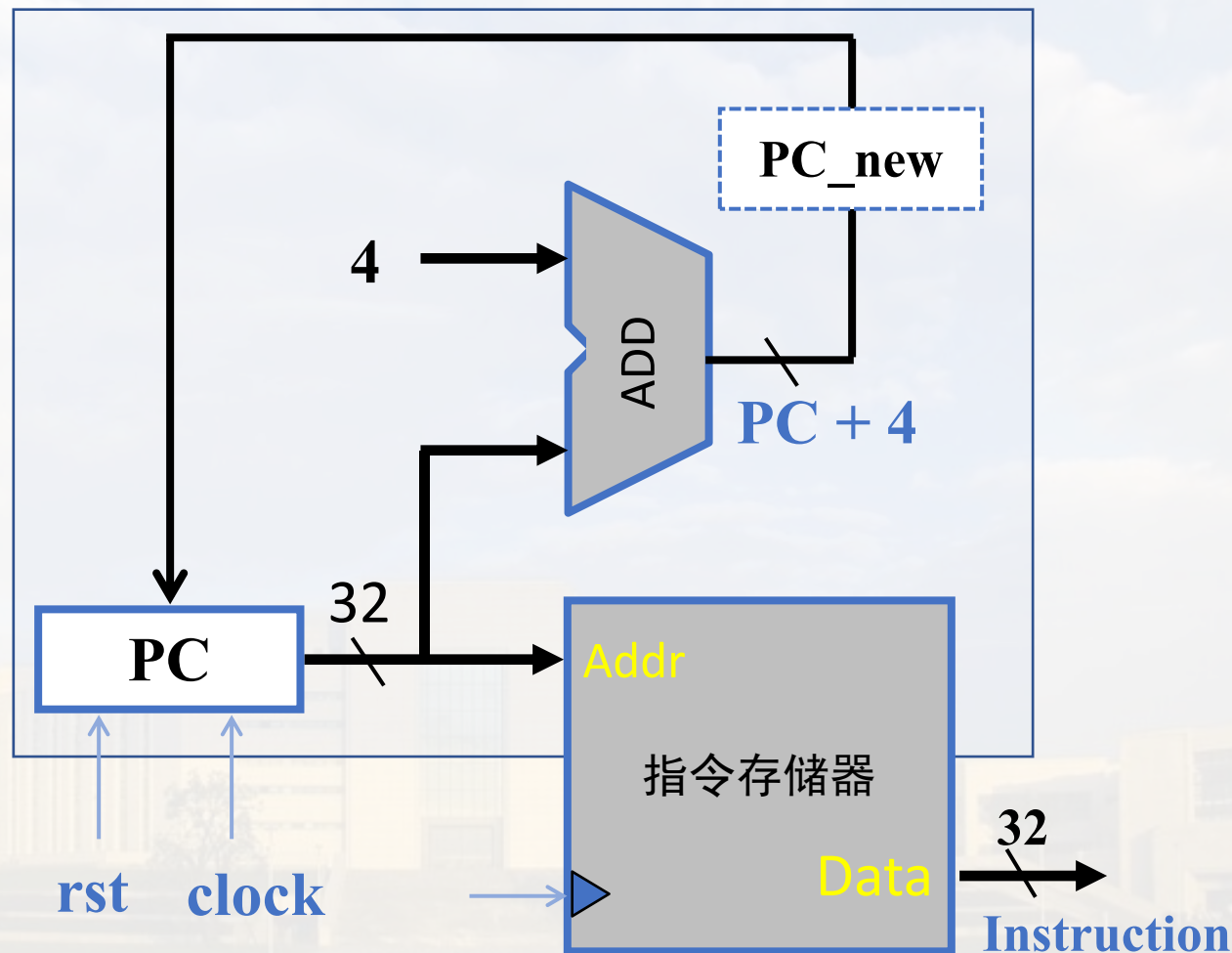
二、单周期数据通路设计



(1) 取指功能的数据通路（公共）

$\text{Instruction} \leftarrow \text{Mem}[\text{PC}]$

$\text{PC} \leftarrow \text{PC} + 4$



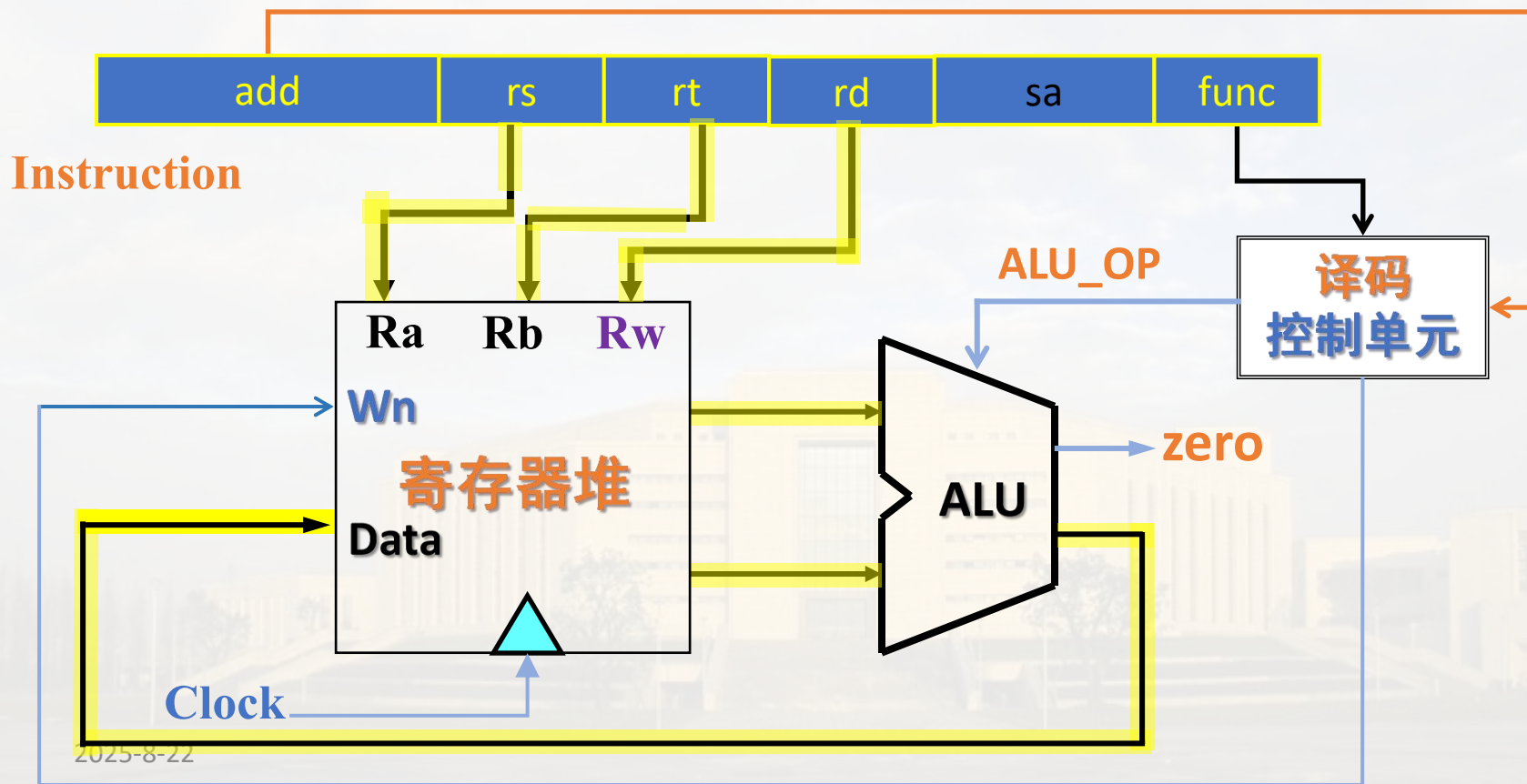
二、单周期数据通路设计

(2) R型运算指令

OP字段6个0，需靠func段确定操作类型；

最多有3个寄存器参与工作；

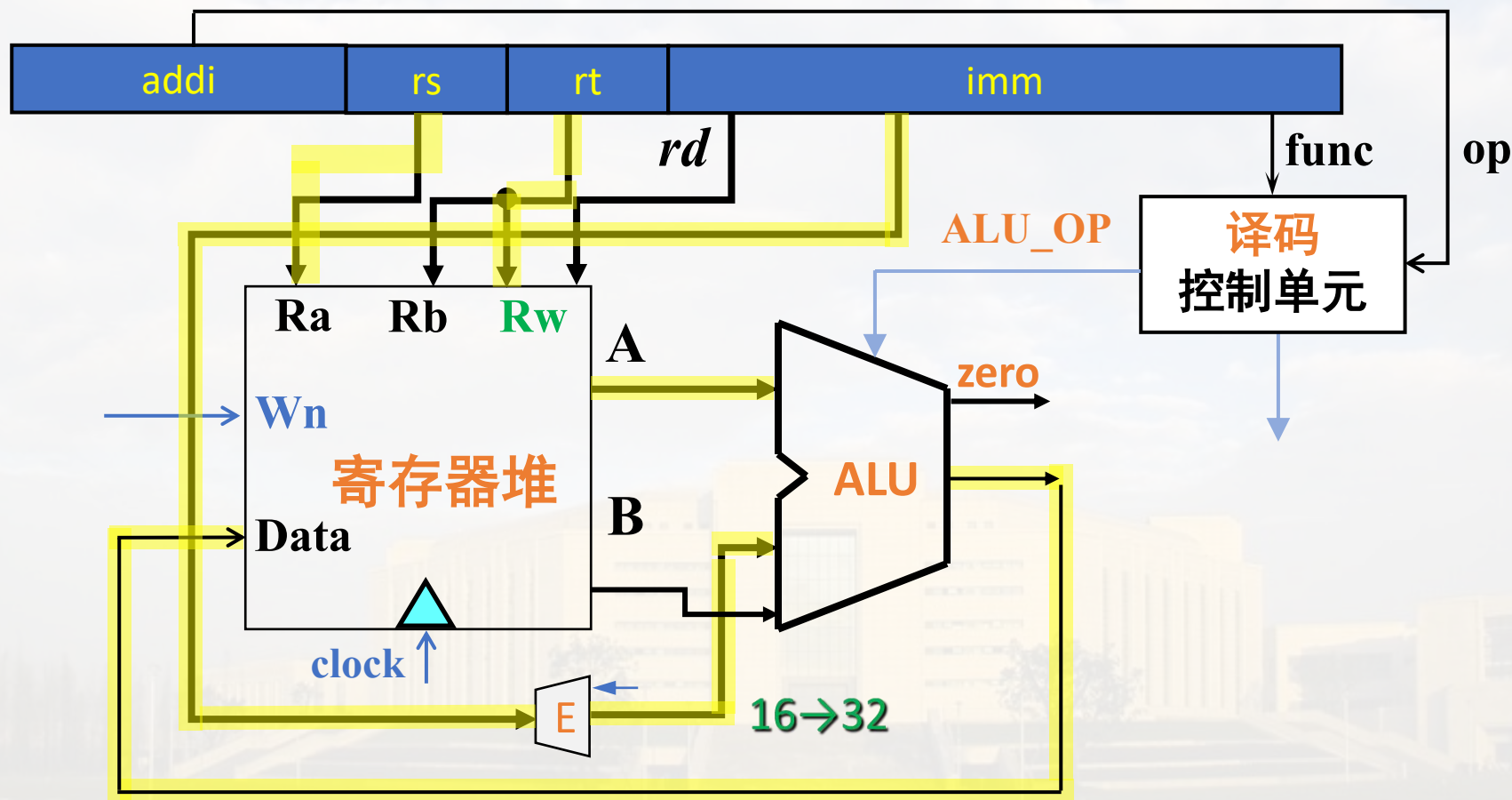
[例] `add rd, rs, rt` # $R[rd] \leftarrow R[rs] + R[rt]$



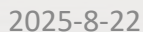
二、单周期数据通路设计

(3) 在R型上扩展I型运算指令

[例1] `addi rt, rs, imm` $\# R[rt] \leftarrow R[rs] + E(imm)$

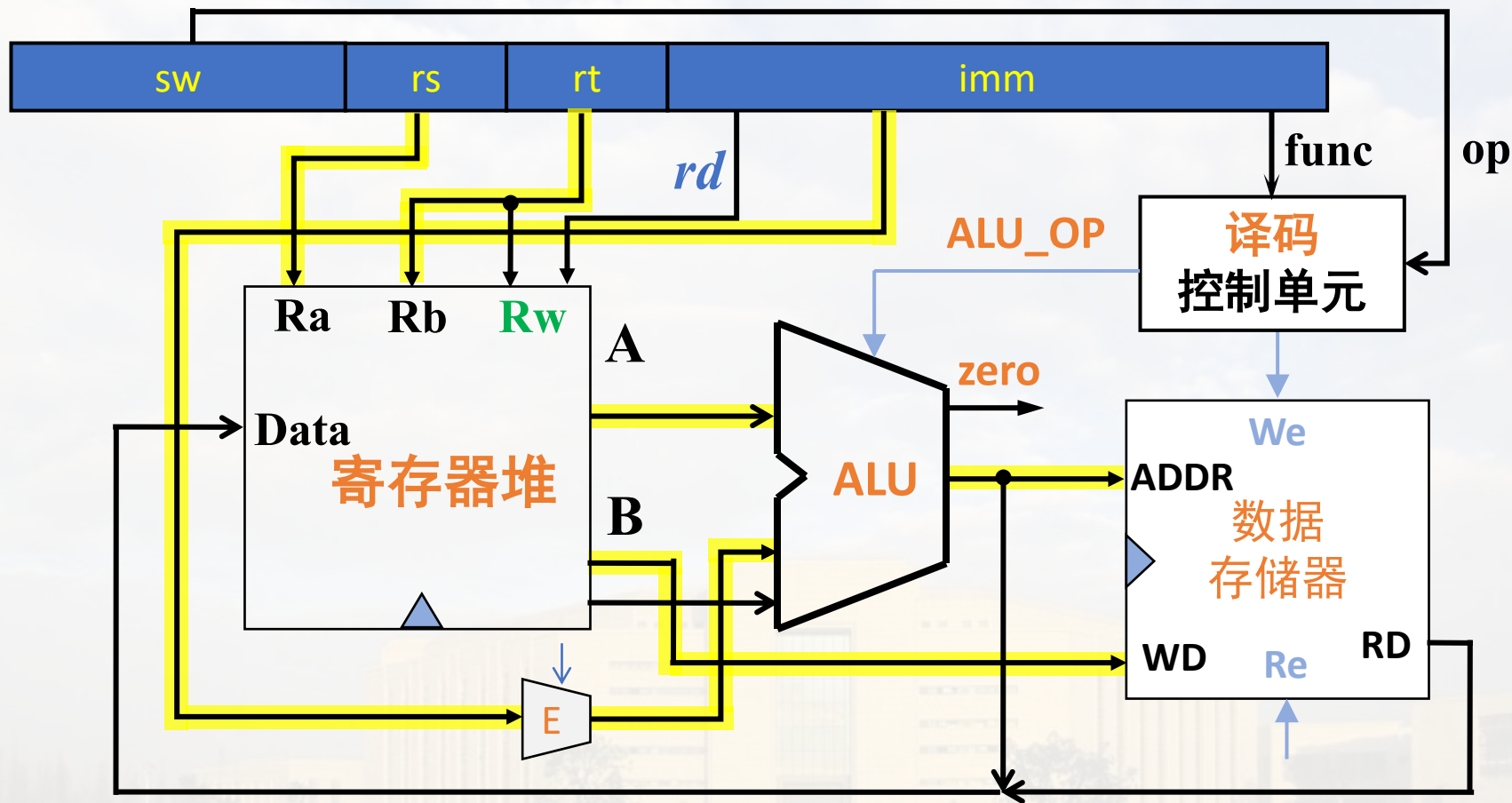


[例2] lw rt, imm(rs) # $R[rt] \leftarrow Mem[R[rs] + E(imm)]$



二、单周期数据通路设计

[例3] `sw rt, imm(rs)` $\# M[R[rs] + E(imm)] \leftarrow R[rt]$

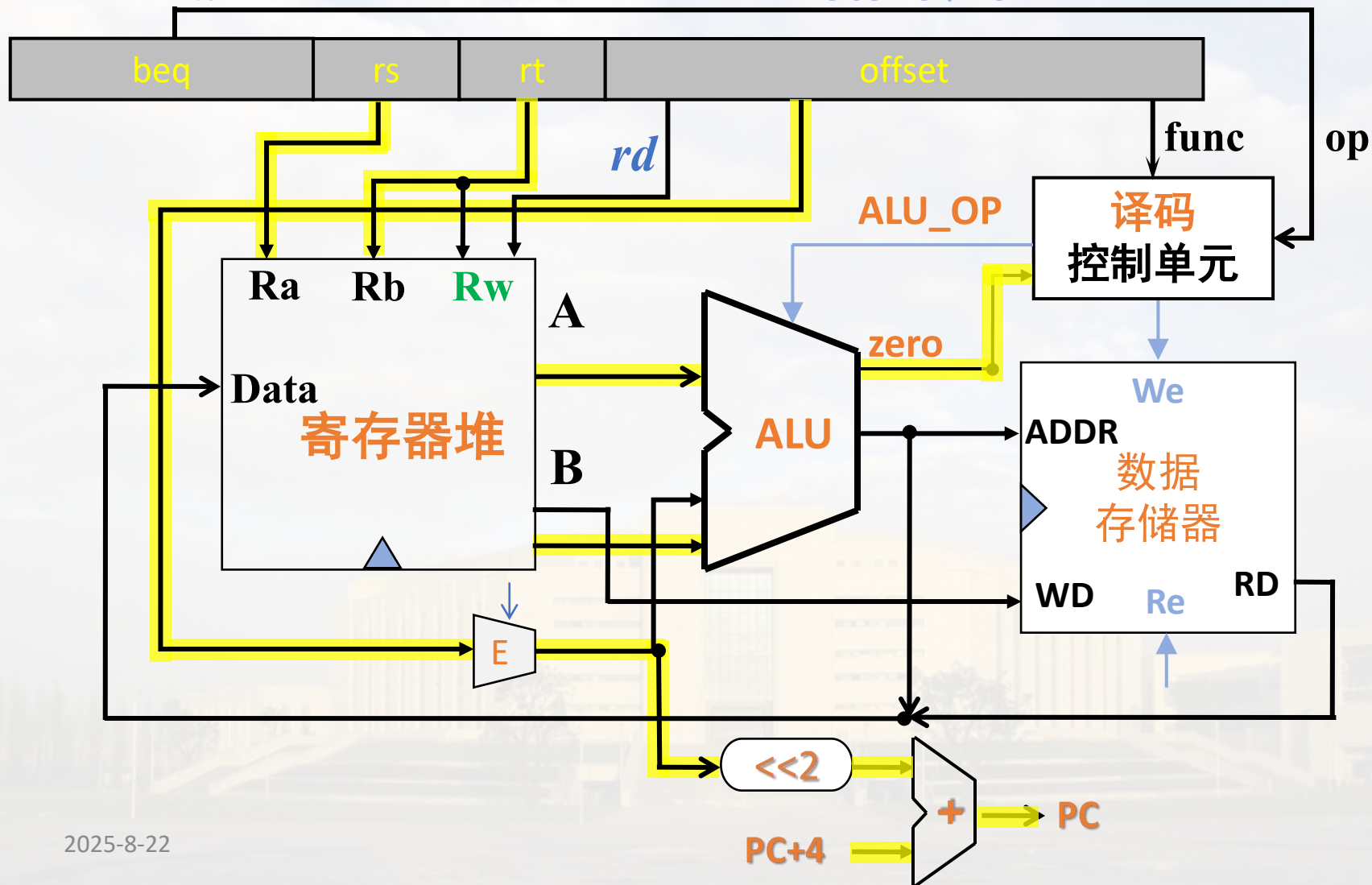


二、单周期数据通路设计

(5) 扩展I型分支指令

[例4] beq, rs, rt, offset

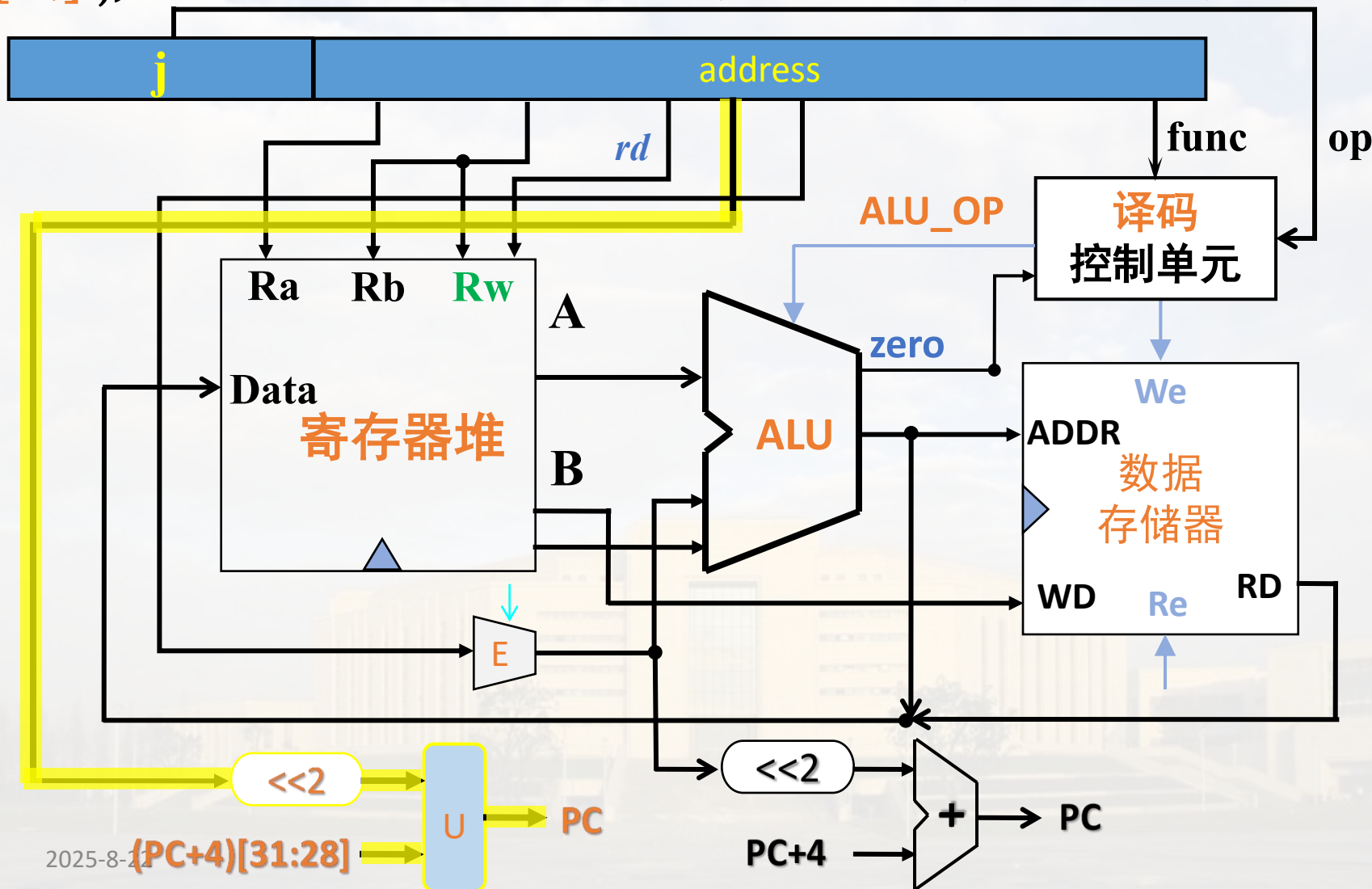
if $R[rs] == R[rt]$
 # then $PC \leftarrow PC + 4 + E(offset) \ll 2$
 # else $PC \leftarrow PC + 4$



二、单周期数据通路设计

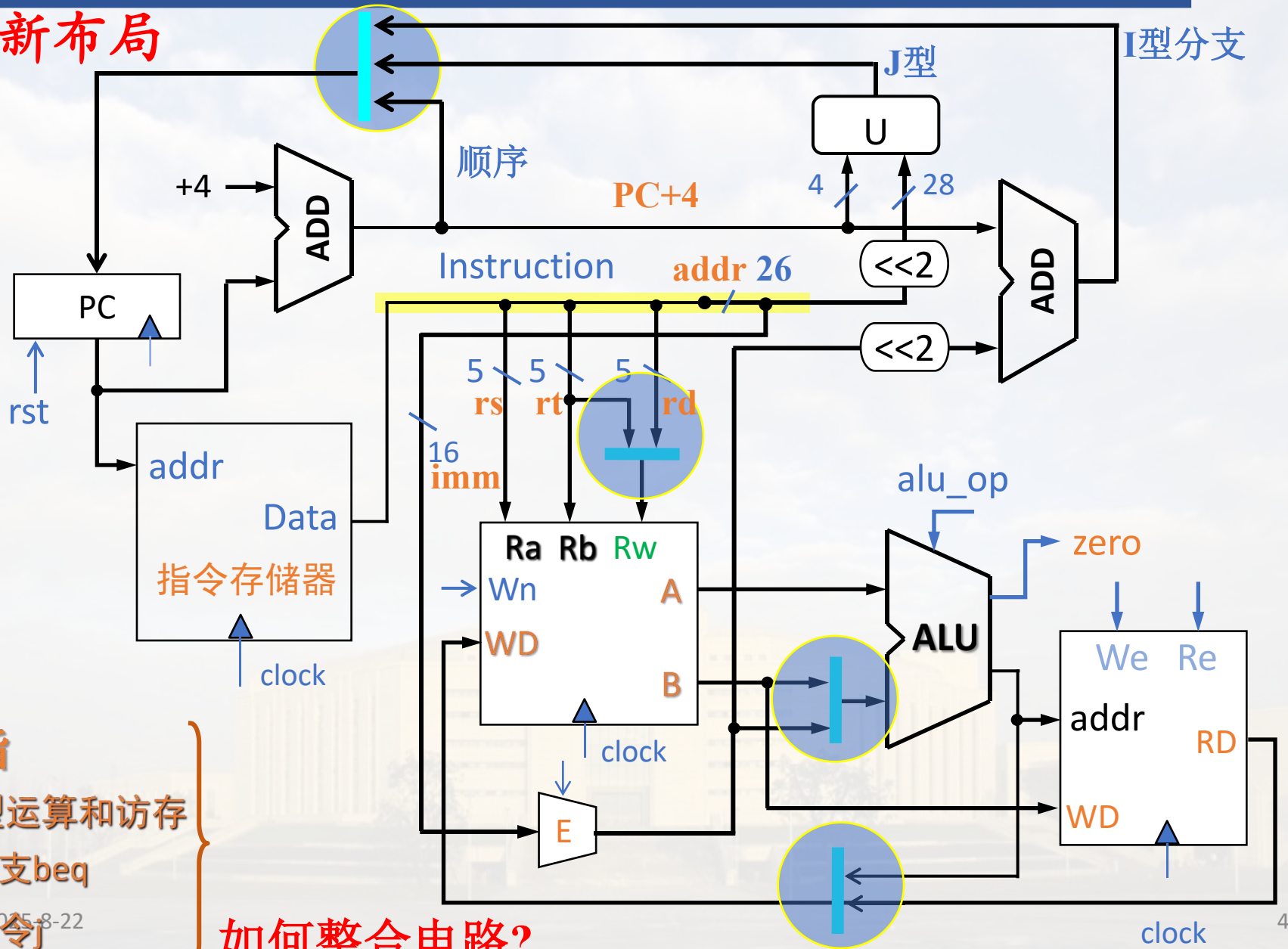
(6) 最后扩展J型j指令

[例] j, address $\# PC \leftarrow (PC+4) [31:28] \cup (address \ll 2)$



二、单周期数据通路设计

#重新布局



取指

R/I型运算和访存
I型分支beq
J型指令

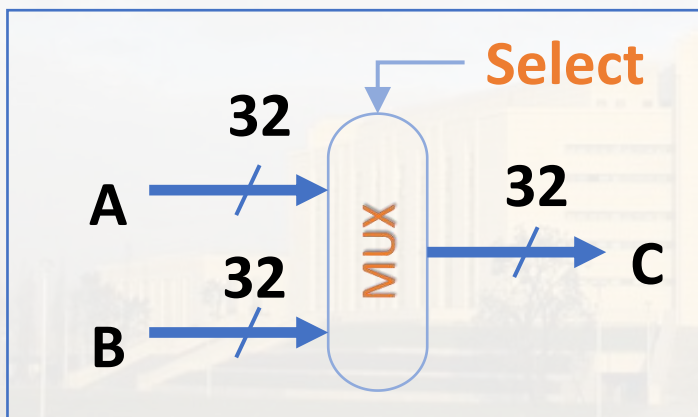
如何整合电路?

#数据通路整合

目标：把各种指令的数据路径合并

- 取指令（各指令共享）
- R型指令
- I型指令（运算、访存、分支）
- J型指令（j指令）

基本思路：用多路选择器，整合冗余通路。

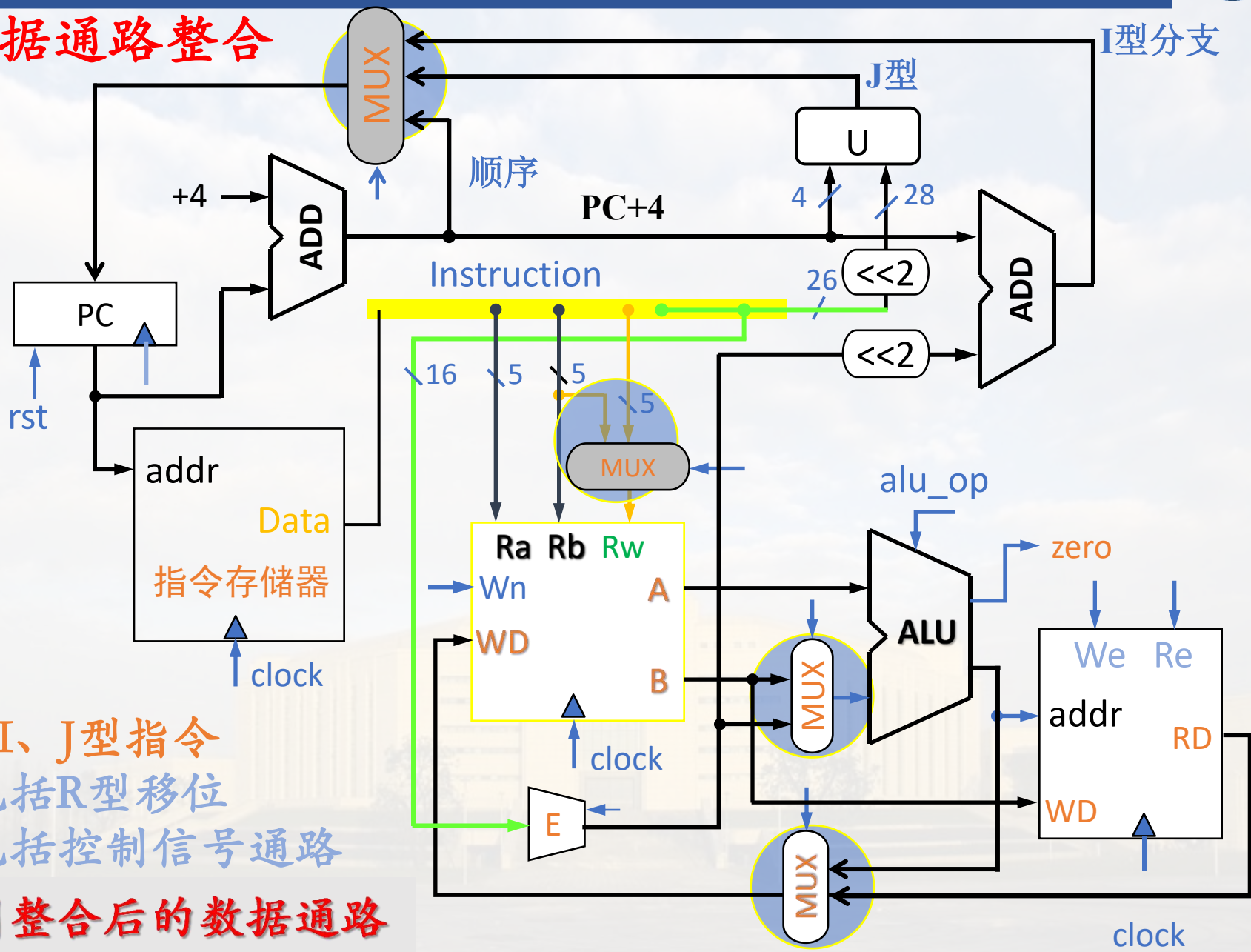


利用不同的选通信号，
控制选通（切换）不同的
数据通路。

二、单周期数据通路设计

#数据通路整合

I型分支



R、I、J型指令
未包括R型移位
未包括控制信号通路
得到整合后的数据通路

※采用组合逻辑方式

设计步骤:

①基于数据通路，确定各指令的控制信号。

➤ 编码: 0、1、X(无关项);

➤ 复杂信号特殊处理，如ALU的控制信号;

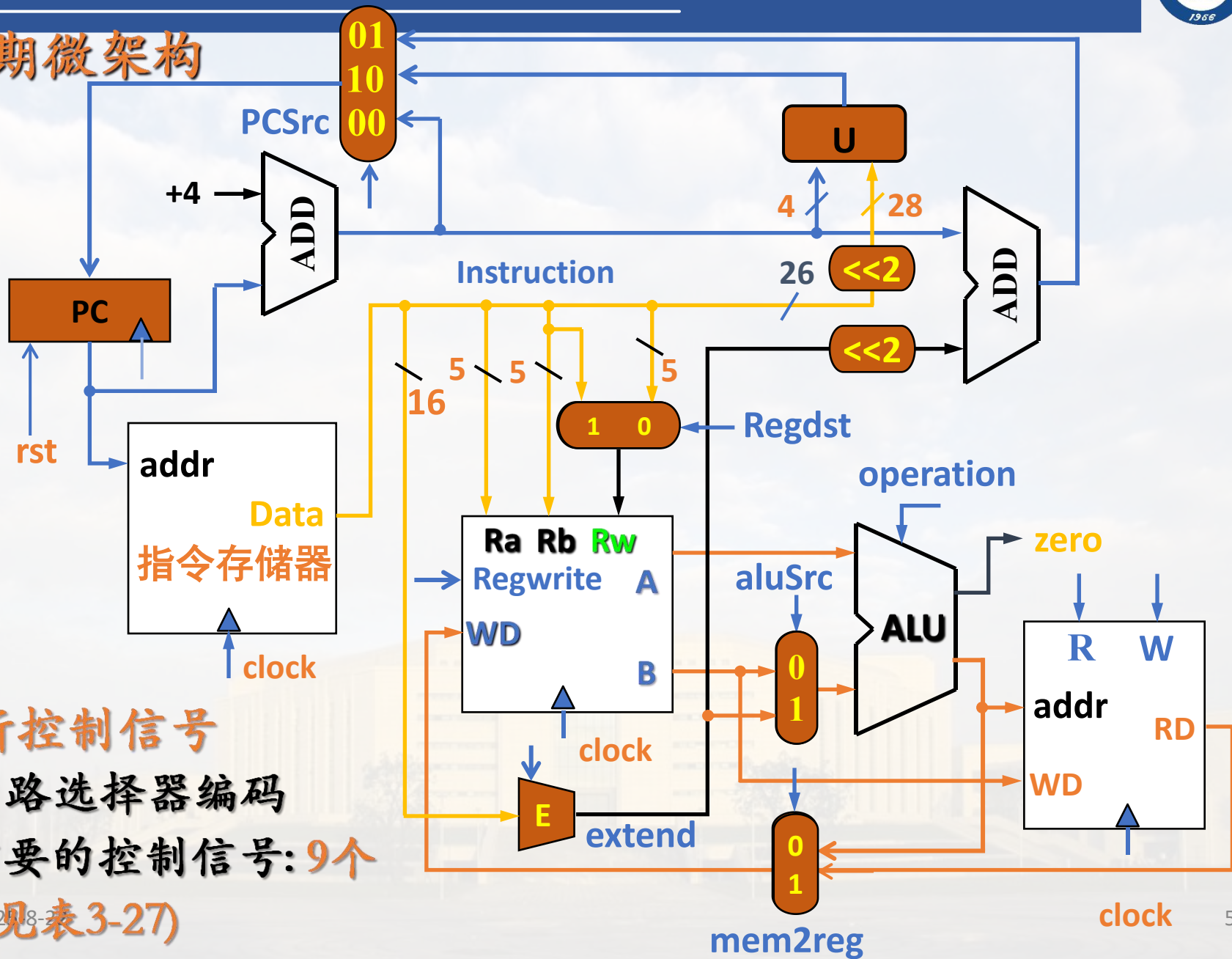
②构建控制信号的真值表。

③将真值表转换成控制信号的产生逻辑。

仿真、硬件实现

三、单周期控制系统设计

单周期微架构



分析控制信号

✓多路选择器编码

✓需要的控制信号: 9个

(参见表3-27)

三、单周期控制系统设计

各控制信号的定义:

控制器输出	无效(=0)时含义	有效(=1)时含义
RegDst	选通rd端	选通rt端
RegWrite	寄堆置于读模式	寄堆置于写模式
AluSrc	选通寄堆的输出B	选通E(offset)
PCSrc[1:0]✓	00选择PC+4, 01选择分支地址, 10选择跳转	
MemRead	两者都为0时, 存储器禁用。	存储器置于读模式
MemWrite		存储器置于写模式
Mem2Reg	选通ALU的输出	选通存储器的输出
extend	执行零扩展	执行符号扩展
operation[3:0]✓	参见ALU的控制代码	

7个一位的控制信号, 2个多位的控制信号。

(1) 控制单元的总体结构



PCSrc(2位)和operation(4位)比较复杂，需单独设计

对于PCSrc (00, 01, 10) :

取决于当前指令是否是beq和j, 如果是beq则还需参考ALU输出的zero标志位

对于operation (4位代码) :

R型指令, 则取决于OP和func字段;

I型指令, 则只取决于OP字段;

J型指令, 不涉及到operation。

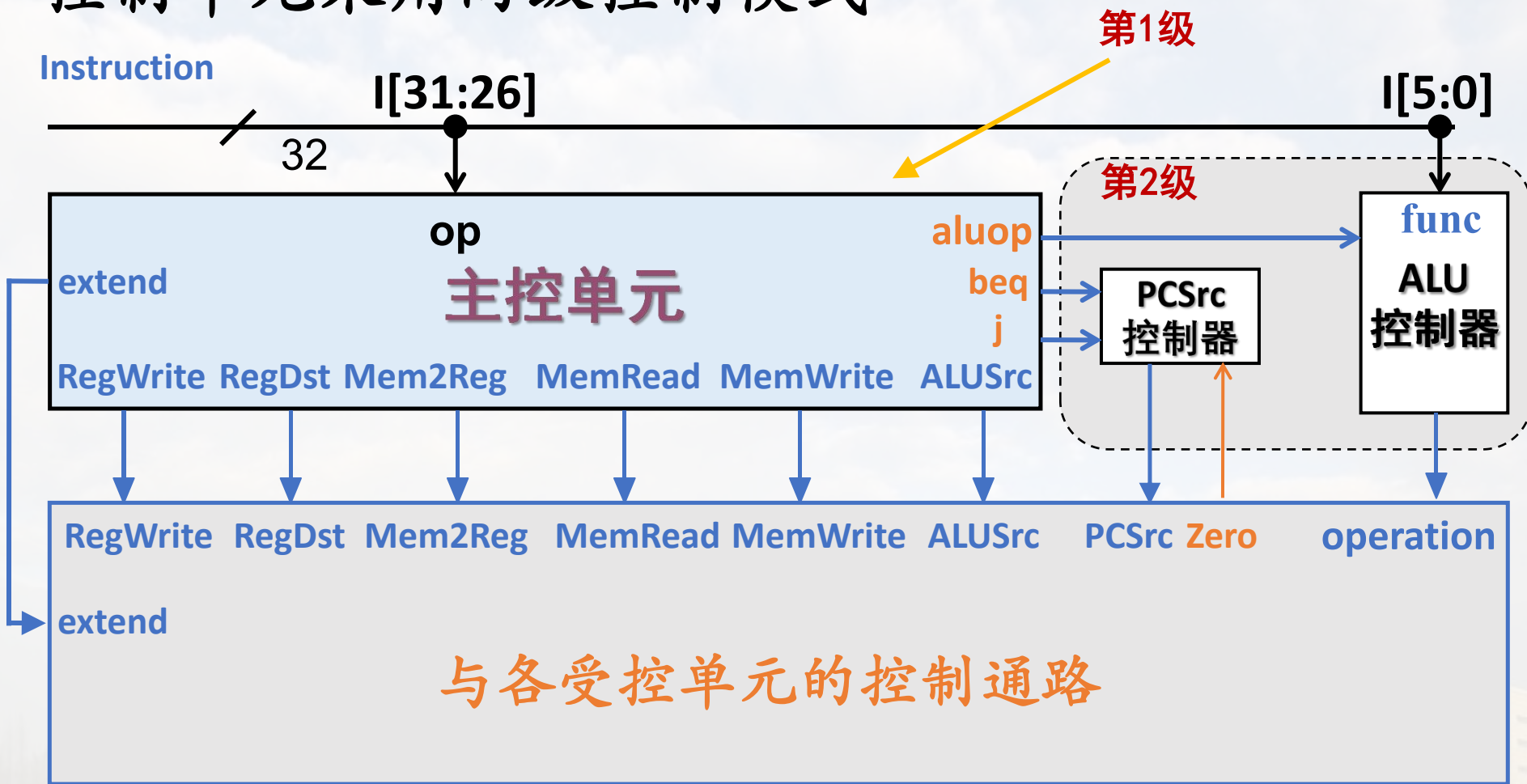
思路: 采用分级控制方案

主控单元→部分简单控制信号

主控单元→中间信号→下级控制器→复杂控制信号

三、单周期控制系统设计

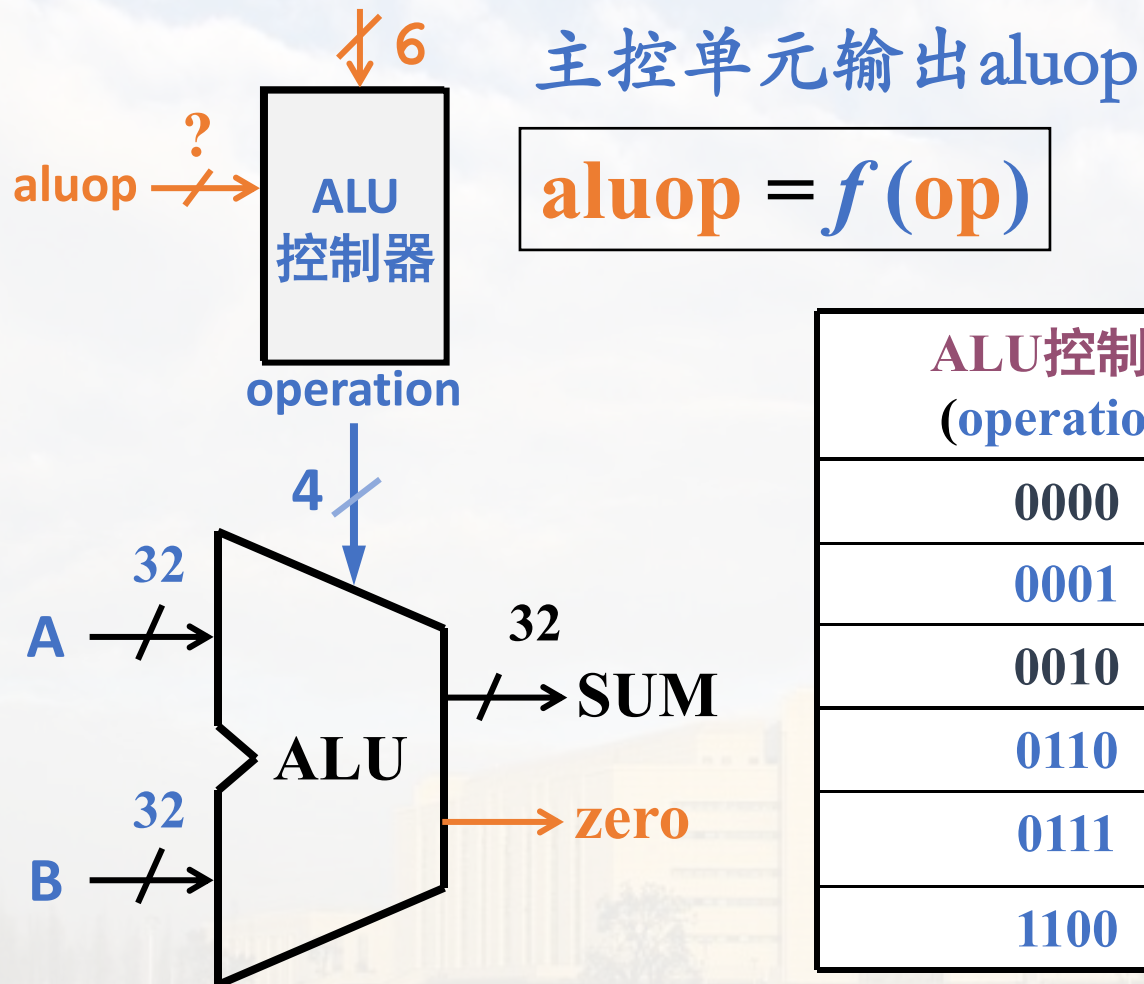
控制单元采用两级控制模式



定义主-从接口: **aluop**, **beq**, **j** 编码? 位数?

三、单周期控制系统设计

(2) 设计ALU控制单元



ALU控制码 (operation)	ALU功能
0000	AND (与)
0001	OR (或)
0010	ADD (加)
0110	SUB (减)
0111	小于则置1
1100	或非

■ ALU控制器的输出依赖于？

- ✓ 指令类型(OP字段)
- ✓ func字段(仅R型指令)

■ 由此可知：

- ✓ func只影响控制器输出的operation控制码

$$\text{operation} = f(\text{aluop}, \text{func})$$

[设计思路]

先根据目标指令涉及到的ALU运算，定义aluop编码
再整理输入输出真值表

最后转换得到operation各位的产生逻辑



三、单周期控制系统设计

分析指令与ALU功能，编码 $aluop = f(op)$ 位数: 3

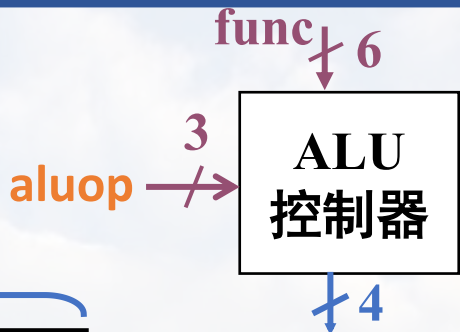
指令类型	指令	func段	ALU功能	ALU控制码 (operation)	<i>aluop</i>
取数	lw ●	XX	<div> R型指令 ↓ 6 ? → func aluop → ALU 控制器 4 ↓ operation </div>	0010	010
存数	sw ●	XX		0010	010
分支	beq ●	XX		0110	110
立即数加	addi ●	XX		0010	010
立即数与	andi ●	XX		0000	000
立即数或	ori ●	XX	或	0001	001
R-型	add ●	100	<div> 执行I型指令时 ? → ALU 控制器 aluop → ALU 控制器 4 ↓ operation </div>	0010	100
R-型	sub ●	100		0110	100
R-型	and ●	100		0000	100
R-型	or ●	100		0001	100
J-型	j	XX		XXX	XXX

三、单周期控制系统设计

ALU控制器的真值表

忽略无关项

输入



指令j
与ALU无关

aluop[2:0]			func[5:0]						operation			
[2]	[1]	[0]	[5]	[4]	[3]	[2]	[1]	[0]	[3]	[2]	[1]	[0]
0	1	0	x	x	x	x	x	x	0	0	1	0
1	1	0	x	x	x	x	x	x	0	1	1	0
0	0	0	x	x	x	x	x	x	0	0	0	0
0	0	1	x	x	x	x	x	x	0	0	0	1
1	0	0	1	0	0	0	0	0	0	0	1	0
1	0	0	1	0	0	0	1	0	0	1	1	0
1	0	0	1	0	0	1	0	0	0	0	0	0
1	0	0	1	0	0	1	0	1	0	0	0	1

(+) lw,sw,addi

(-) beq

(&) andi

(|) ori

(+) add

(-) sub

(&) and

(|) or

三、单周期控制系统设计

根据真值表, 写出输入-输出逻辑式:

$$\text{operation}[3] \equiv 0$$

$$\text{operation}[2] = \text{aluop}[2] \text{aluop}[1] +$$

$$\overline{\text{aluop}[2] \text{aluop}[1]} \cdot \overline{f[2]} f[1] \overline{f[0]}$$

$$\text{operation}[1] = \overline{\text{aluop}[2] \text{aluop}[1]} + \text{aluop}[2] \text{aluop}[1] +$$

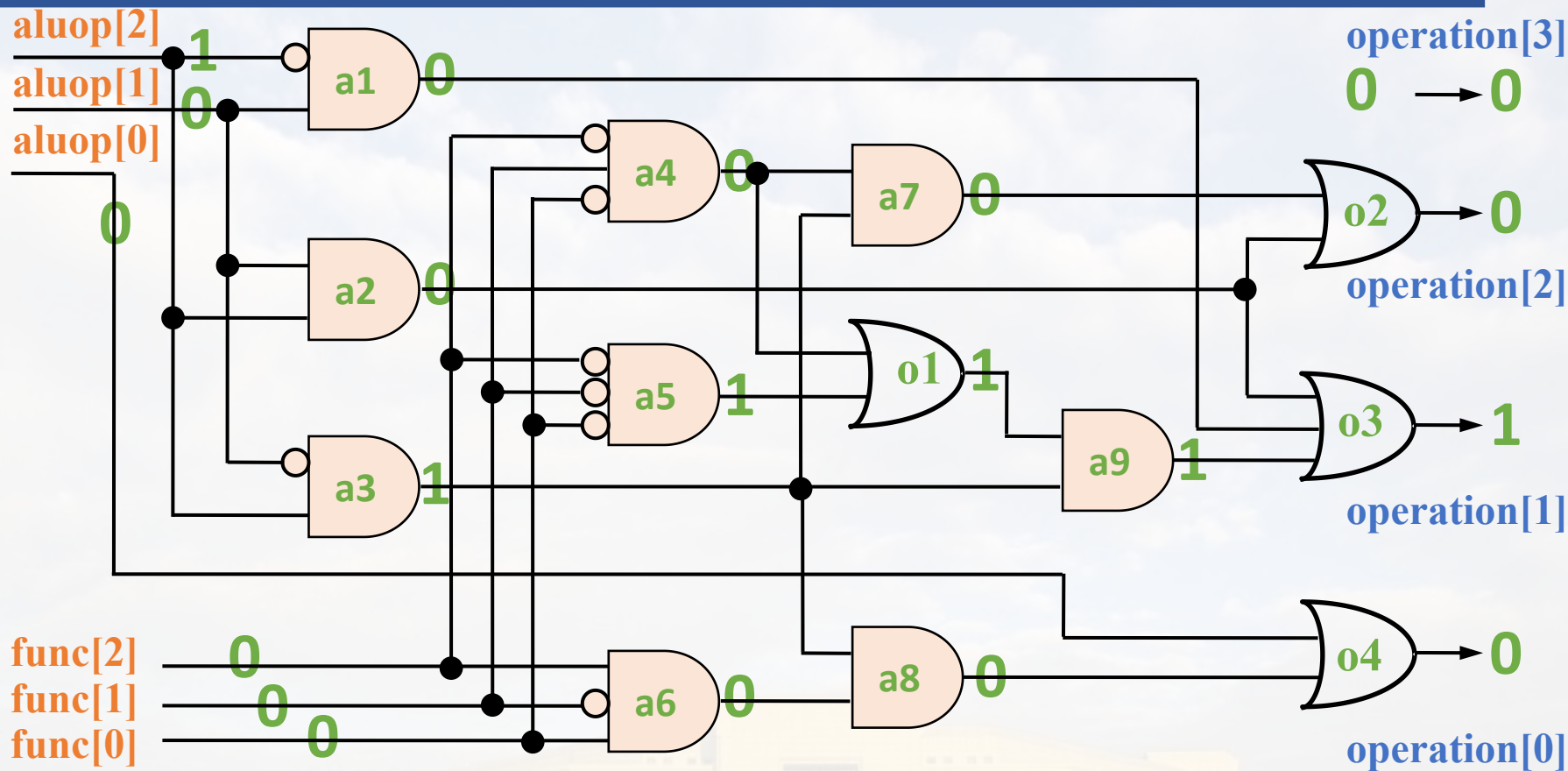
$$\overline{\text{aluop}[2] \text{aluop}[1]} \cdot (\overline{f[2]} \overline{f[1]} \overline{f[0]} + \overline{f[2]} f[1] \overline{f[0]})$$

$$\text{operation}[0] = \text{aluop}[0] + \overline{\text{aluop}[2] \text{aluop}[1]} \cdot f[2] \overline{f[1]} f[0]$$

可直接仿真, 也可转换得到逻辑电路:



三、单周期控制系统设计



`func[5:3]`是无关项，不使用。

[验证] `aluop[2:0]=100`，`func[2:0]=000`时，`operation=?`

`operation`→0010（符合真值表）

其它编码也能验证通过 alu控制器设计完成!

三、单周期控制系统设计

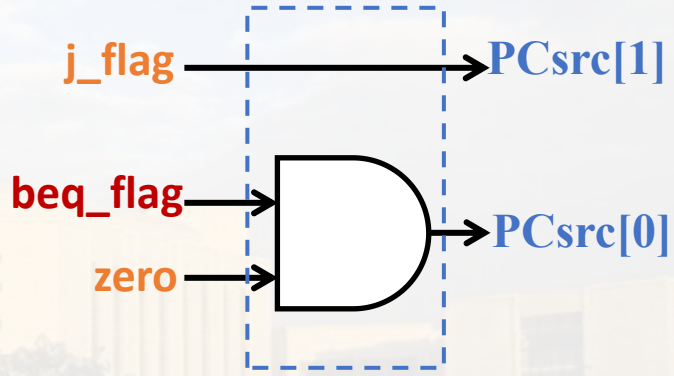
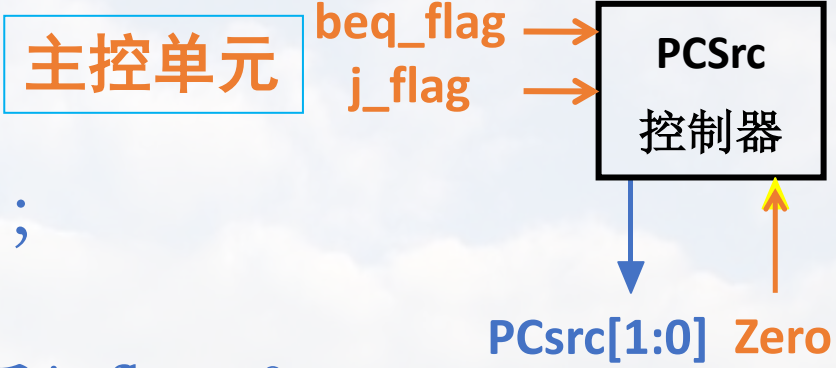
(3) PCSrc控制单元

方案分析:

执行beq时，输出beq_flag=1;
 执行j时，输出j_flag=1;
 其余指令，输出beq_flag=0且j_flag=0;

PCSrc控制器“输入-输出”真值表如下:

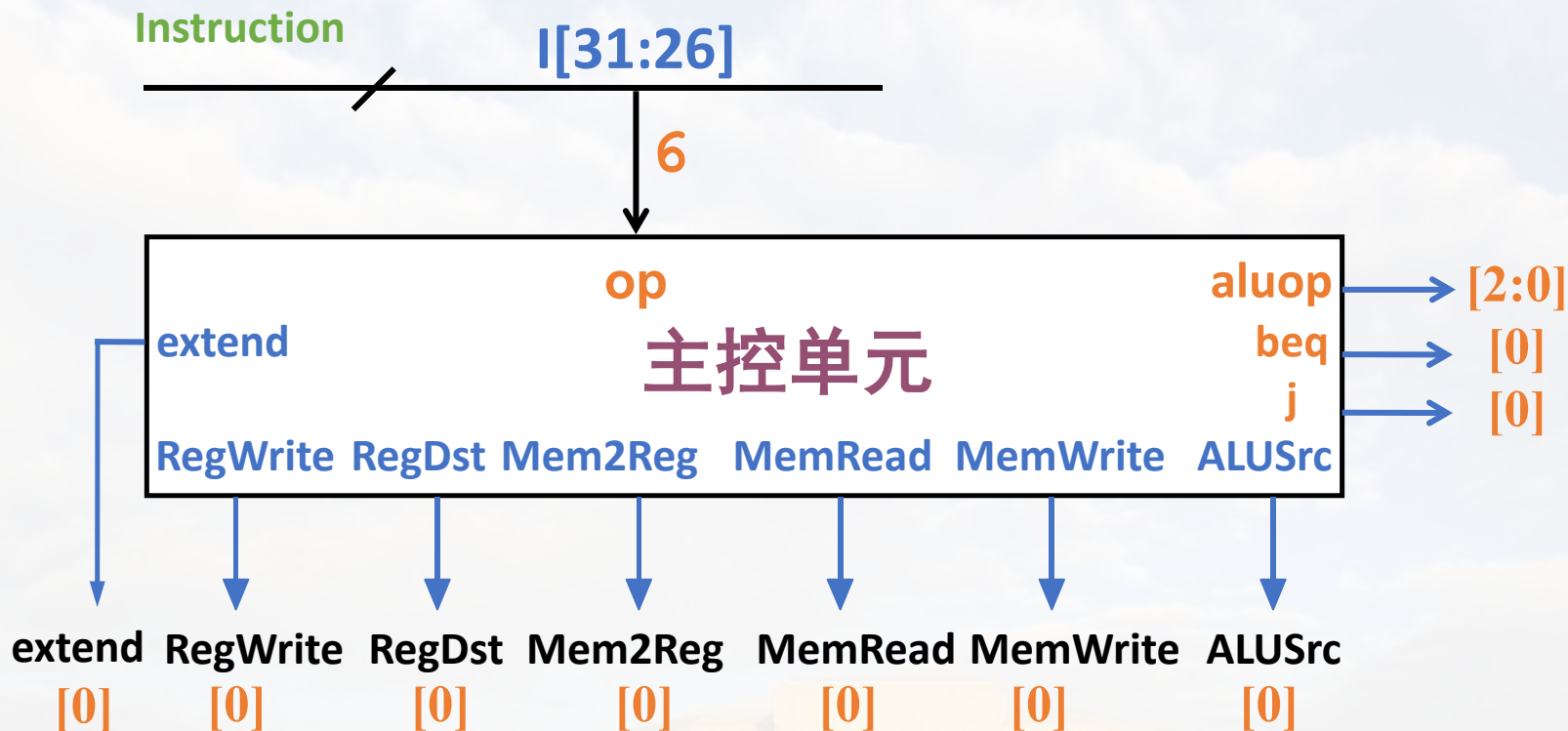
指令	beq_flag	j_flag	zero	PCsrc [1]	PCsrc [0]
beq	1	x	0	0	0
	1	x	1	0	1
j	x	1	x	1	0
其它	0	0	x	0	0



PCsrc[1]=j_flag; PCsrc[0]=beq_flag . zero (图3-75)

三、单周期控制系统设计

(4) 主控单元的设计



主控单元根据1个输入，产生10个输出：

9个1位的控制信号 + 1个3位的控制信号

→ 整理主控单元的“输入-输出”真值表

三、单周期控制系统设计

主控单元的“输入-输出”真值表 (表3-31)

	输入						RegDst	ALUSrc	Mem2Reg	RegWrite	MemWrite	MemRead	extend	aluop[2:0]			beq_flag	j_flag
	OP[5:0]													[2]	[1]	[0]		
	[5]	[4]	[3]	[2]	[1]	[0]												
add	0	0	0	0	0	0	0	0	0	1	0	0	x	1	0	0	0	0
sub	0	0	0	0	0	0	0	0	0	1	0	0	x	1	0	0	0	0
and	0	0	0	0	0	0	0	0	0	1	0	0	x	1	0	0	0	0
or	0	0	0	0	0	0	0	0	0	1	0	0	x	1	0	0	0	0
addi	0	0	1	0	0	0	1	1	0	1	0	0	1	0	1	0	0	0
andi	0	0	1	1	0	0	1	1	0	1	0	0	0	0	0	0	0	0
ori	0	0	1	1	0	1	1	1	0	1	0	0	0	0	0	1	0	0
lw	1	0	0	0	1	1	1	1	1	1	0	1	1	0	1	0	0	0
sw	1	0	1	0	1	1	x	1	x	0	1	0	1	0	1	0	0	0
beq	0	0	0	1	0	0	x	0	x	0	0	0	1	1	1	0	1	0
j	0	0	0	0	1	0	x	x	x	0	0	0	x	x	x	x	0	1

写出各输出信号的逻辑式：

输出10种共12位控制信号 \Rightarrow 共12个逻辑式

$$\text{regdst} = \text{op}[3] \overline{\text{op}[2]} \overline{\text{op}[1]} + \text{op}[3] \overline{\text{op}[2]} \text{op}[0] + \text{op}[3] \text{op}[2] \overline{\text{op}[0]} + \text{op}[5] \overline{\text{op}[3]}$$

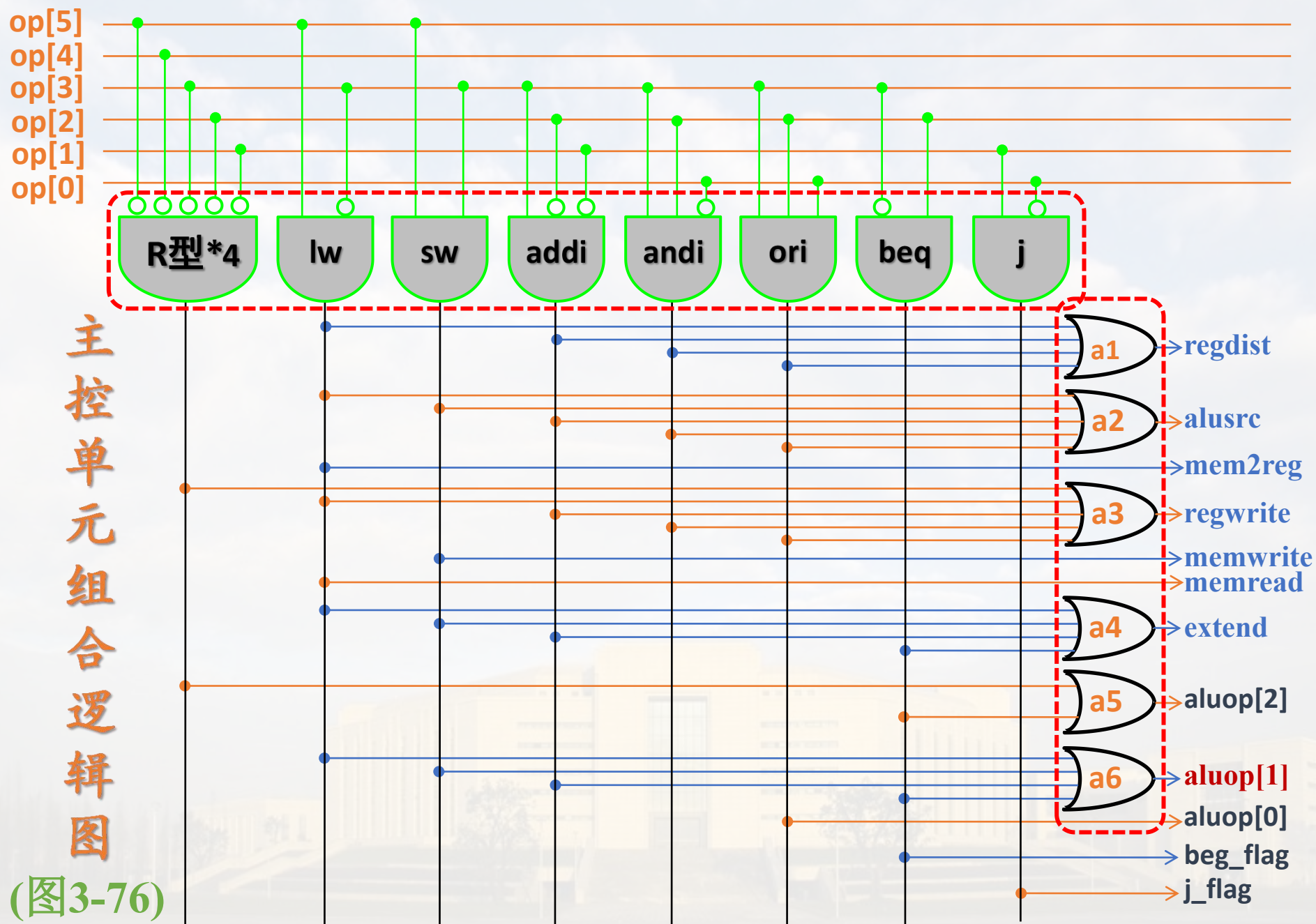
...

$$\text{aluop}[0] = \text{op}[3] \text{op}[2] \text{op}[0]$$

$$\text{beq_flg} = \overline{\text{op}[3]} \text{op}[2]$$

$$\text{j_flag} = \text{op}[1] \overline{\text{op}[0]}$$

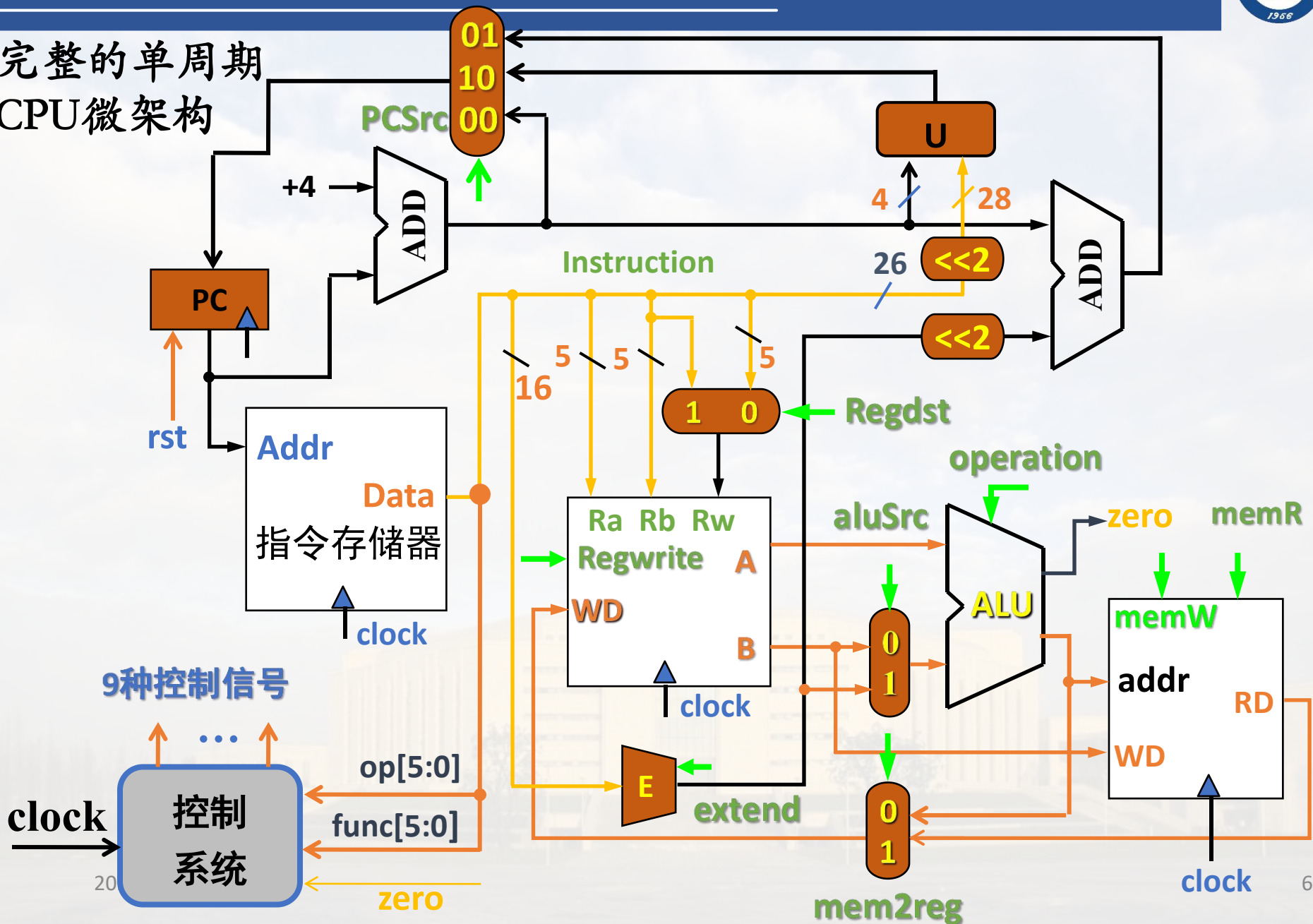
根据各逻辑式，就可得到主控 单元的组合逻辑。
(参见教材图 3-76)



主
控
单
元
组
合
逻
辑
图
(图3-76)

三、单周期控制系统设计

完整的单周期
CPU微架构



- 1、MIP32指令架构
- 2、MIP32处理器基本组成部件
- 3、单周期MIPS32处理器设计



谢谢观看

计算机系统结构

2025-8-22



信息与软件工程学院
School of Information and Software Engineering