



计算机组成原理与接口技术 ——基于MIPS架构

Mar, 2022

第3讲 微处理器

杨明

华中科技大学电子与信息工程系

myang@hust.edu.cn



► 内容

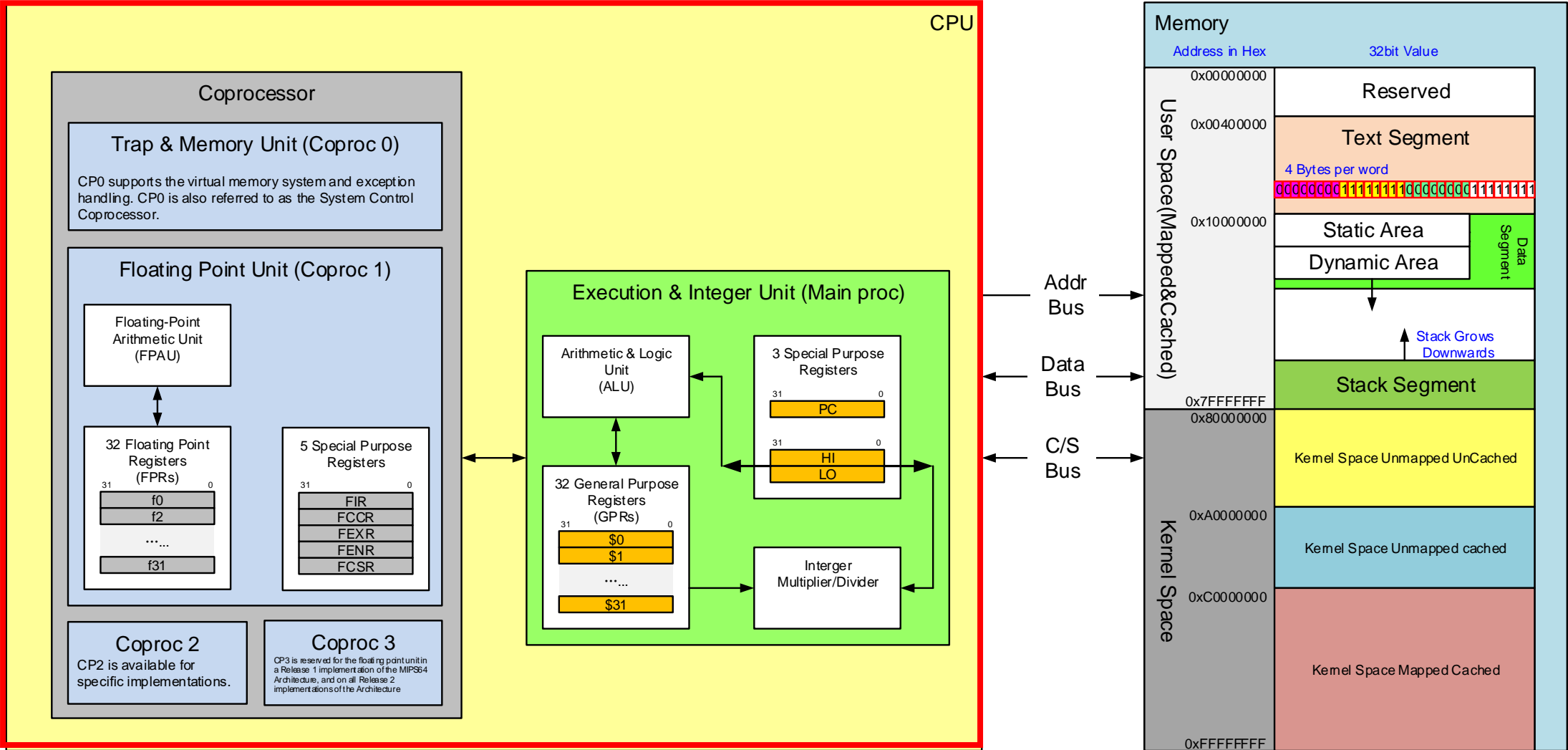
- 微处理器的基本构成
- 简单MIPS微处理器各部件原理及设计
- 现代微处理器的流水线技术原理
- 现代微处理器的超标量技术原理
- 微处理器异常处理机制和外部接口
- MicroBlaze微处理器简介

► 目标

- 理解处理器的基本操作、基本构成部件
- 能用Verilog语言设计简单MIPS微处理器
- 理解解现代微处理器设计的新技术
- 了解微处理器异常处理机制和外部接口
- 了解MicroBlaze微处理器特点

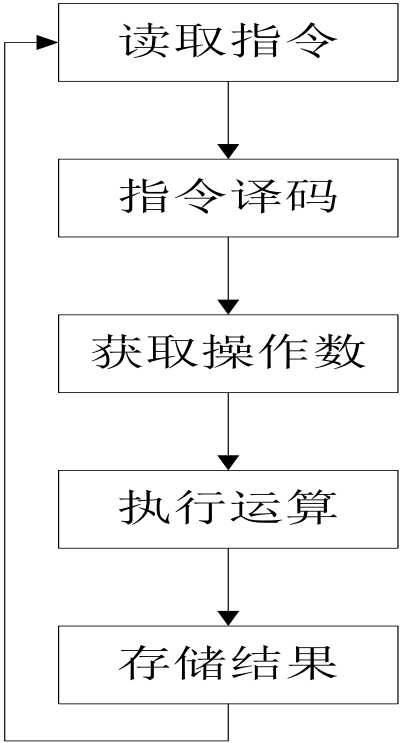
3.1 微处理器的基本构成

► CPU：通过执行指令，完成运算、控制

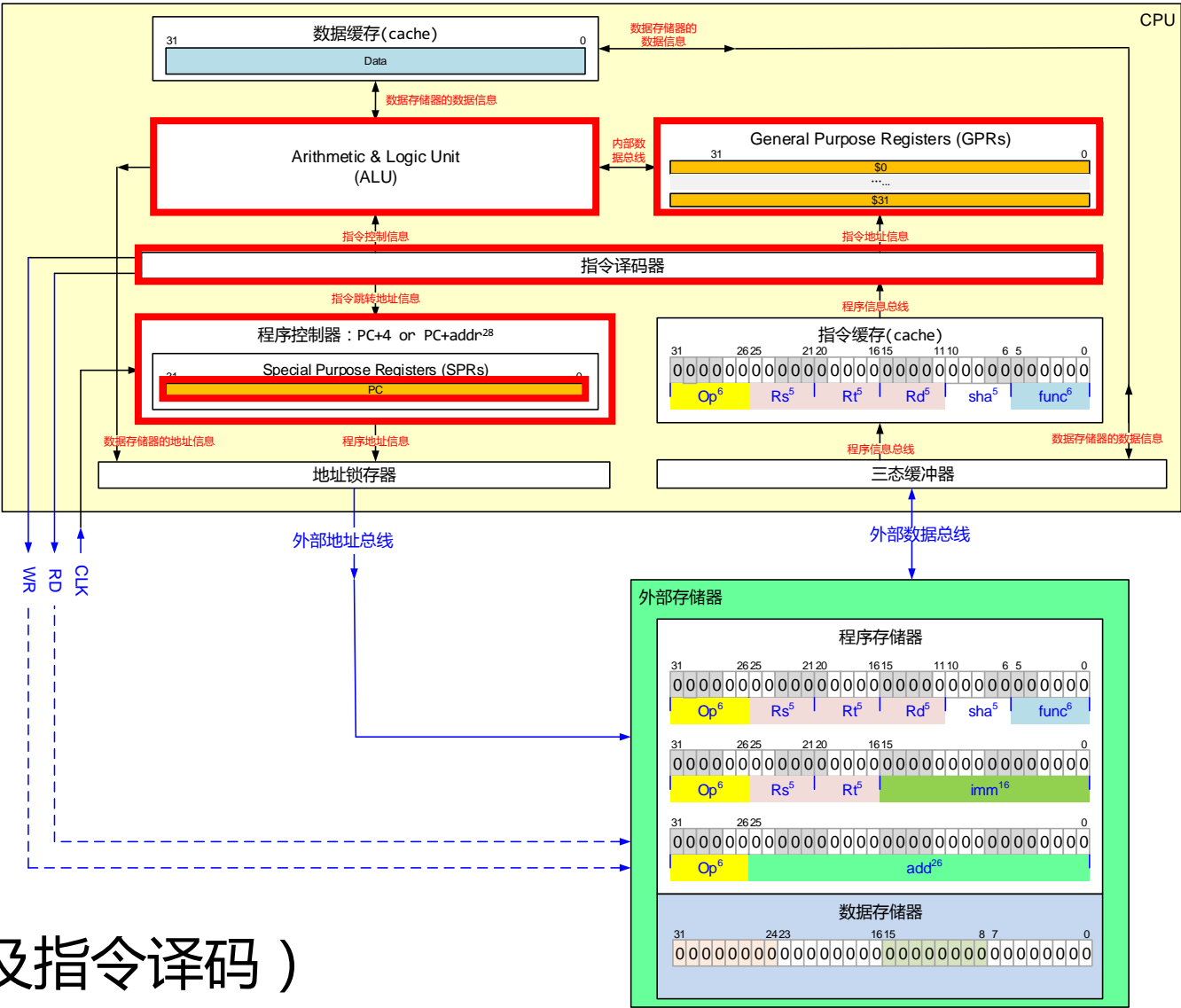


3.1 微处理器的基本构成

- 计算机工作原理
 - 存储程序和程序控制”



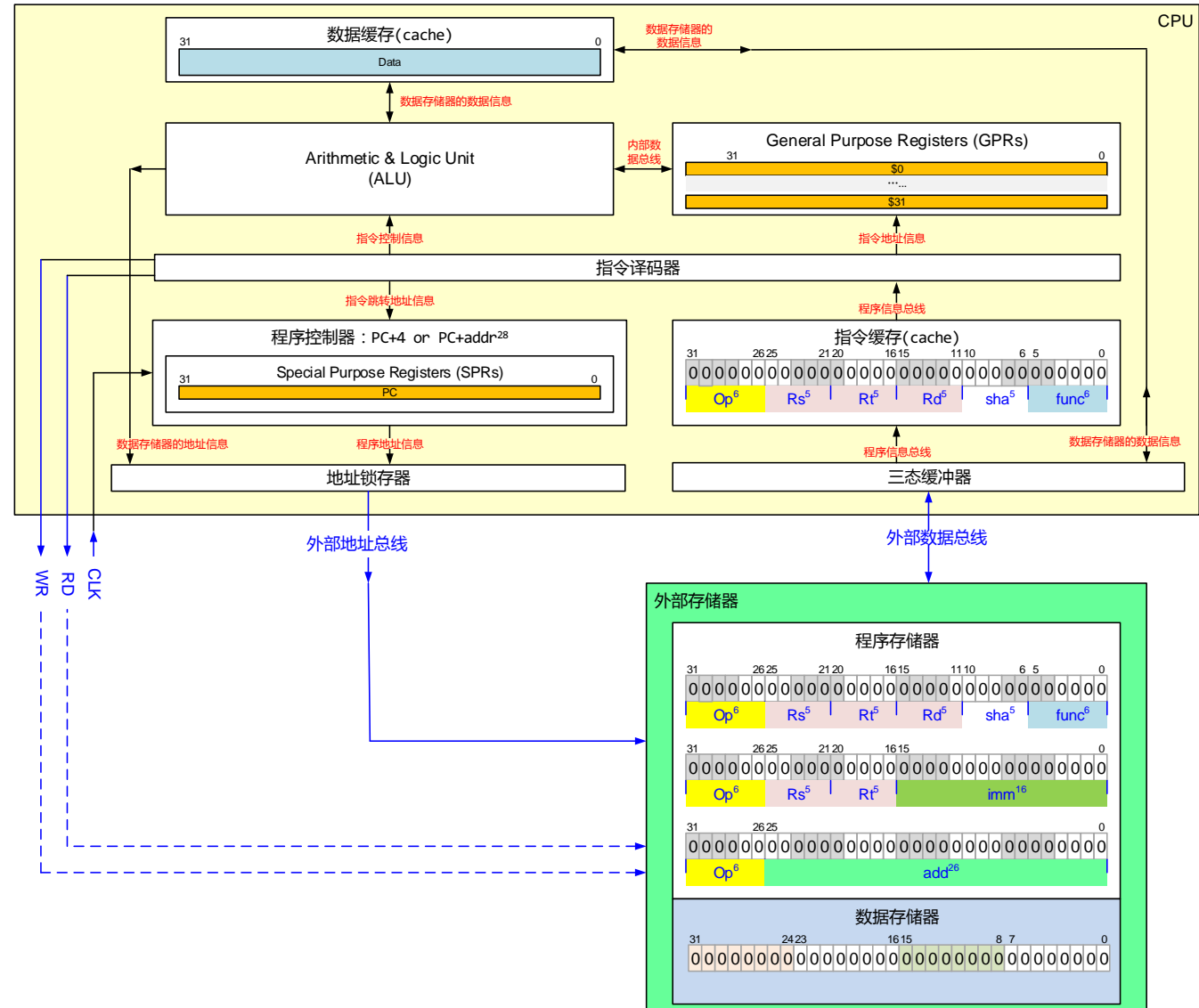
- 简单微处理器的基本构成
 - 运算器、Regs、控制器（程序控制及指令译码）



3.1 微处理器的基本构成

► 微处理器一般执行三种基本操作

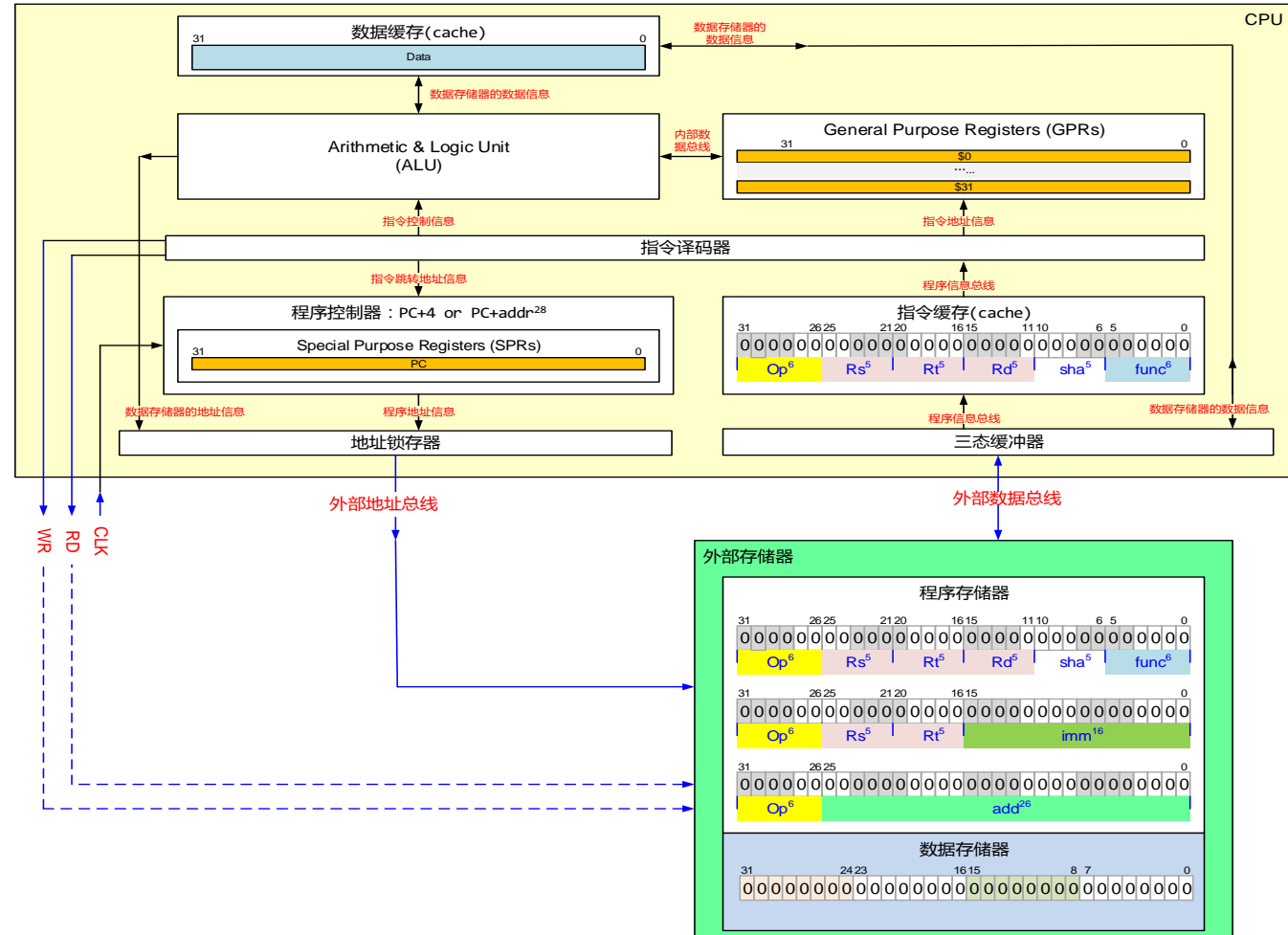
- 算术逻辑运算
 - 通过使用ALU（算术/逻辑单元），微处理器执行数学计算。例如：加法、减法、乘法和除法。现代微处理器包含完整的浮点处理器，它可以对很大的浮点数执行非常复杂的浮点运算。
- 数据搬移（存储器和CPU之间的、存储器存储器之间）
 - 微处理器可以将数据从一个内存位置移动到另一个位置
- 程序控制
 - 微处理器可以做出决定，并根据这些决定跳转到一组新指令。



3.1 微处理器的基本构成

► 微处理器与外部组件的基本接口包括

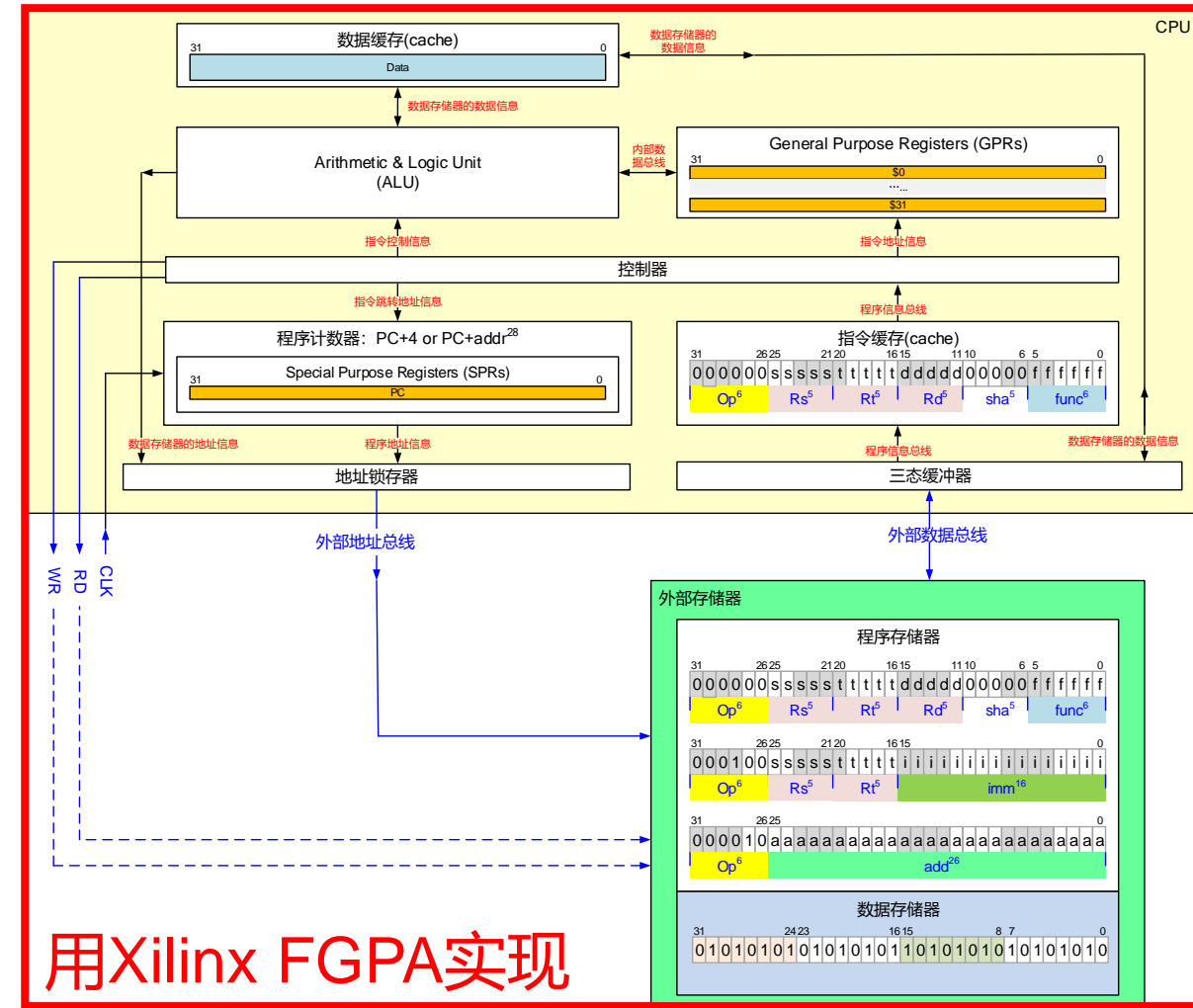
- 一组地址总线（总线宽度可以8位、16位或32位），用于向内存发送地址。
- 一组数据总线（总线宽度可以是8位、16位或32位），能够将数据发送到内存或从内存取得数据。
- 一条RD（读）和WR（写）控制信号，告诉内存它是希望将数据写入某个地址位置还是从某个地址位置获得内容。
- 时钟信号，将时钟脉冲序列发送到处理器，控制微处理器进行工作。
- 复位信号，用于将程序计数器重置为零（或者其他内容）并重新开始执行。



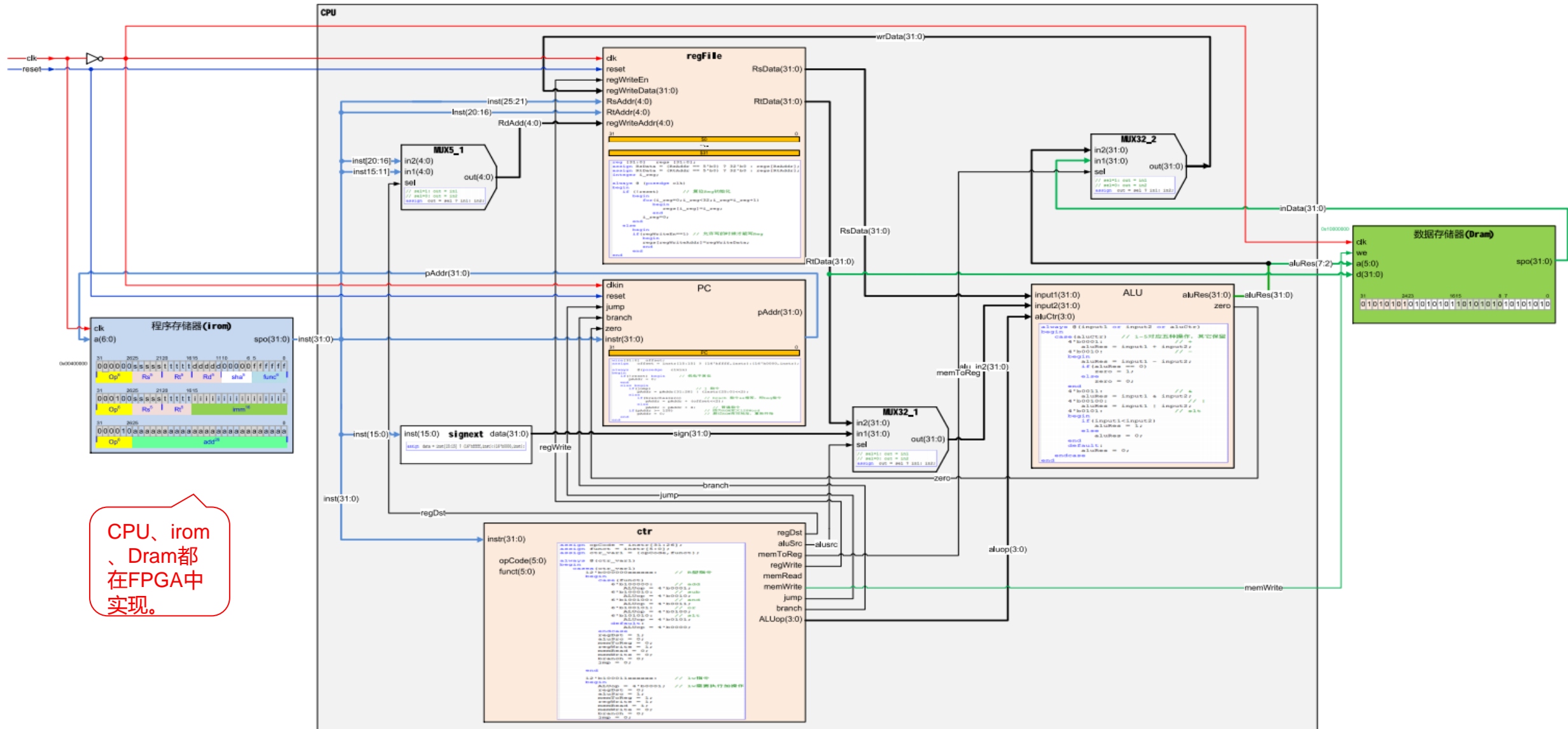
3.2 MIPS指令集微处理器基本构成

► 假设MIPS微处理器支持以下指令

- 算术逻辑运算如add, sub, and, or, slt指令
- 存储器读写: lw, sw指令
- 程序控制如beq, j指令



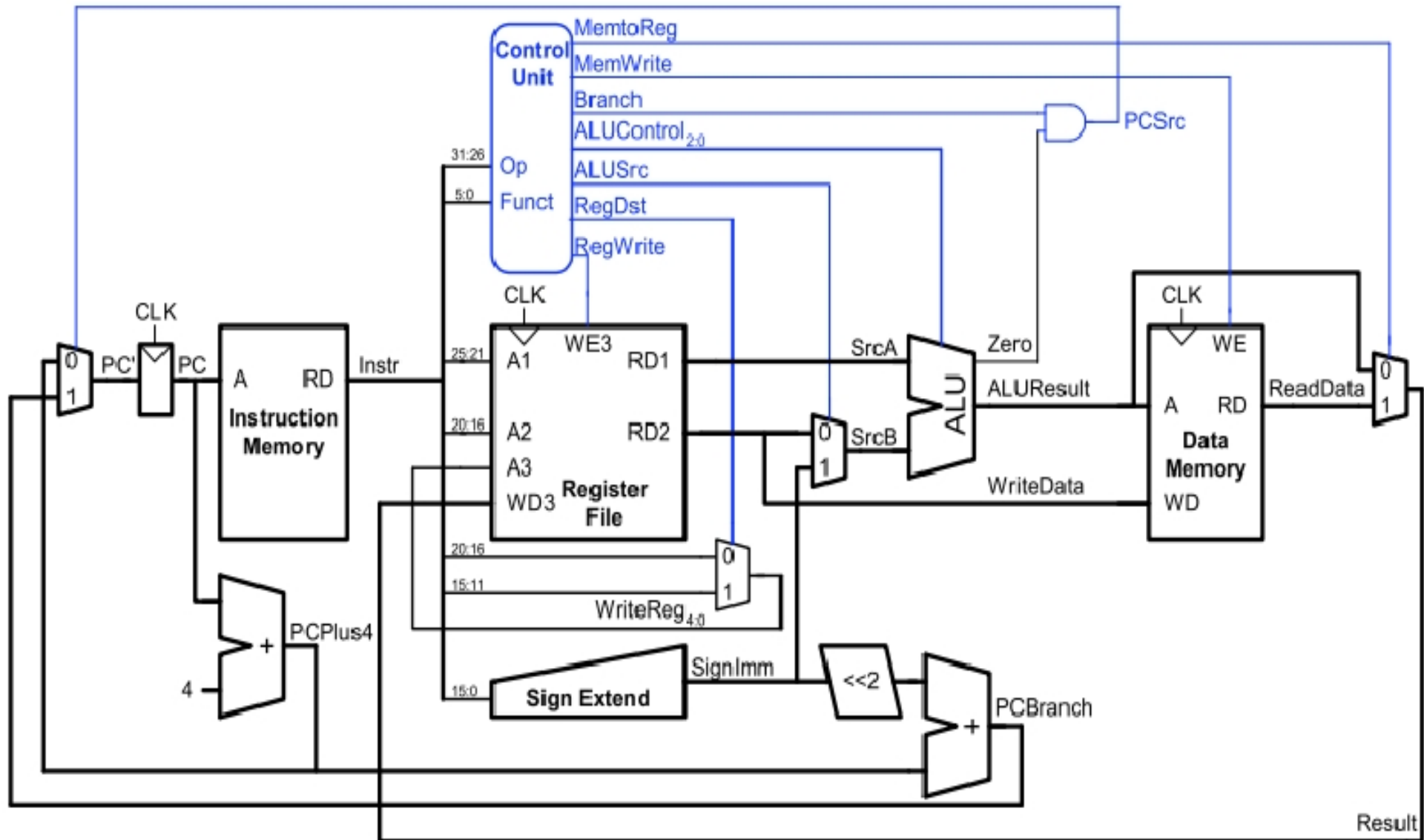
MIPS CPU的实现框图一



CPU、irom、Dram都在FPGA中实现。

MIPS CPU的实现框图二

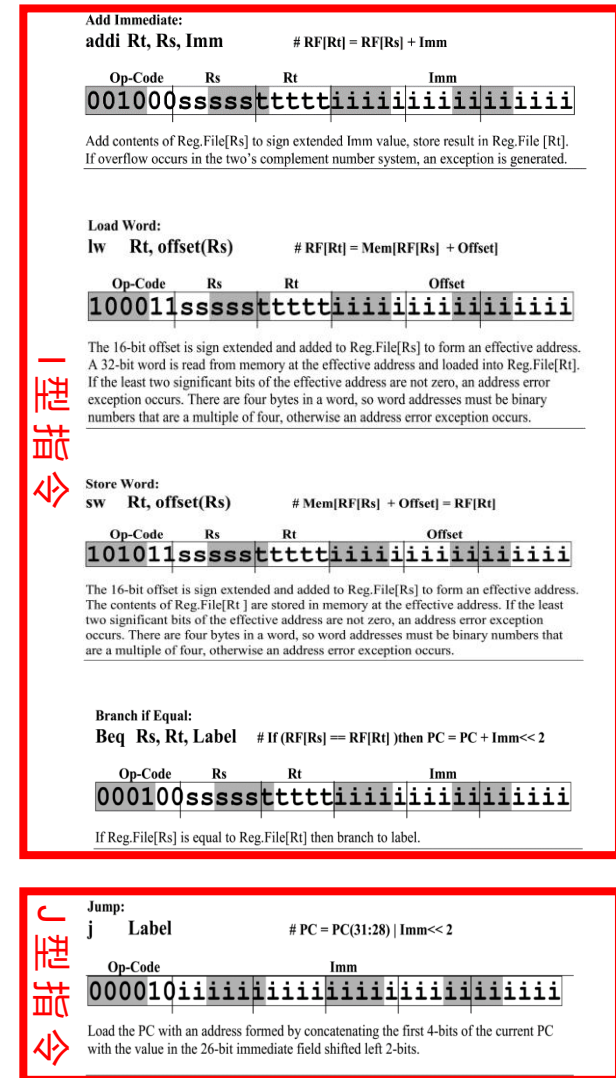
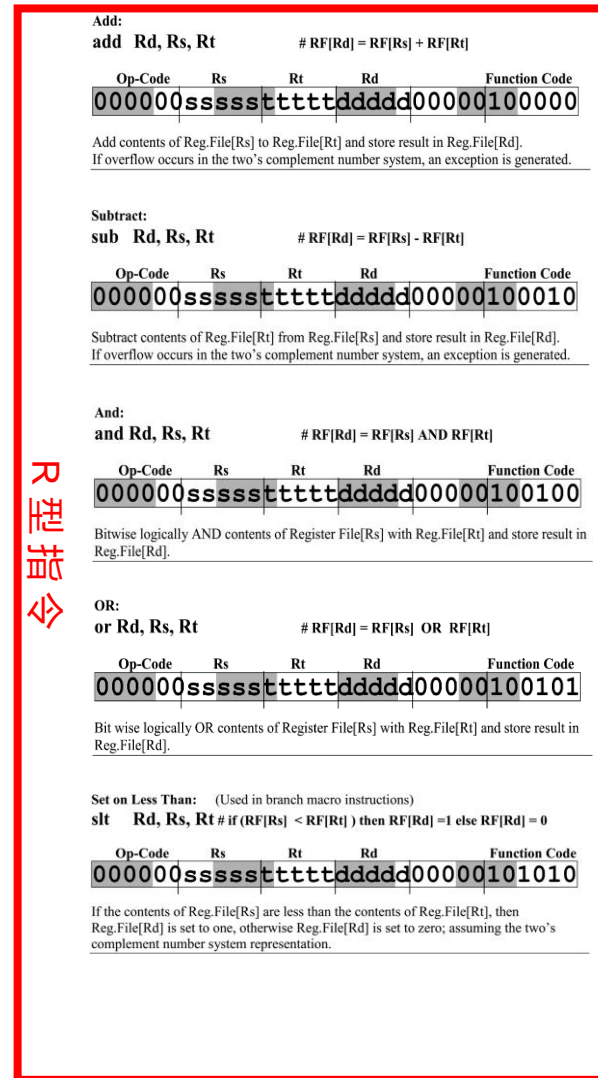
► 框图2



3.2 MIPS指令集微处理器基本构成

► 指令类型及机器码

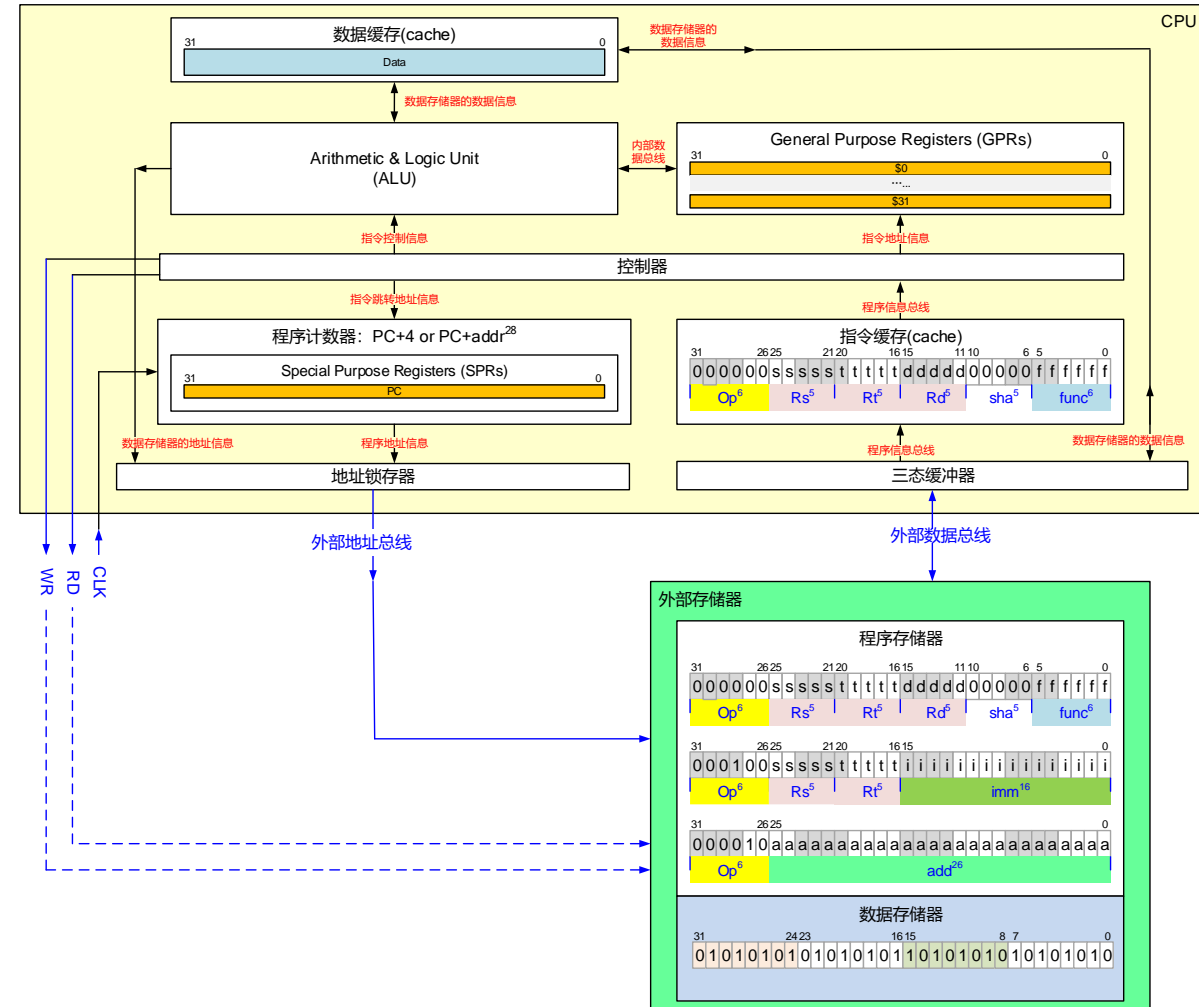
- 支持add, sub, and, or, slt, **andi**, lw, sw, beq, j 等指令
 - 加、减、与、或、slt的op-code均为000000, 通过function code来区分(2级译码):
 - Func = 100000 : add
 - Func = 100010 : sub
 - Func = 100100 : and
 - Func = 100101 : or
 - Func = 101010 : slt



3.2 MIPS指令集微处理器基本构成

► 数据通路

- 指令获取部件
- R型指令实现部件
- 存储器数据存取部件
- 控制器部件
- ALU部件



- ▶ 指令获取部件
 - 顺序执行时
 - PC自动加4

Op-Code	Rs	Rt	Rd	Function Code
000000	ssssst	ttttt	ddddd	00000100000

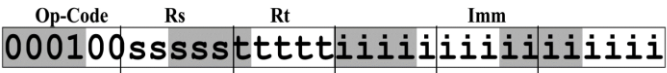
```

graph TD
    IM[指令存储器] --> AB[地址总线]
    AB --> PC[PC]
    PC --> add[add]
    C[4] --> add
    add --> PC
  
```



3.3 数据通路实现原理

Branch if Equal:
Beq Rs, Rt, Label # If (RF[Rs] == RF[Rt]) then PC = PC + Imm<<2

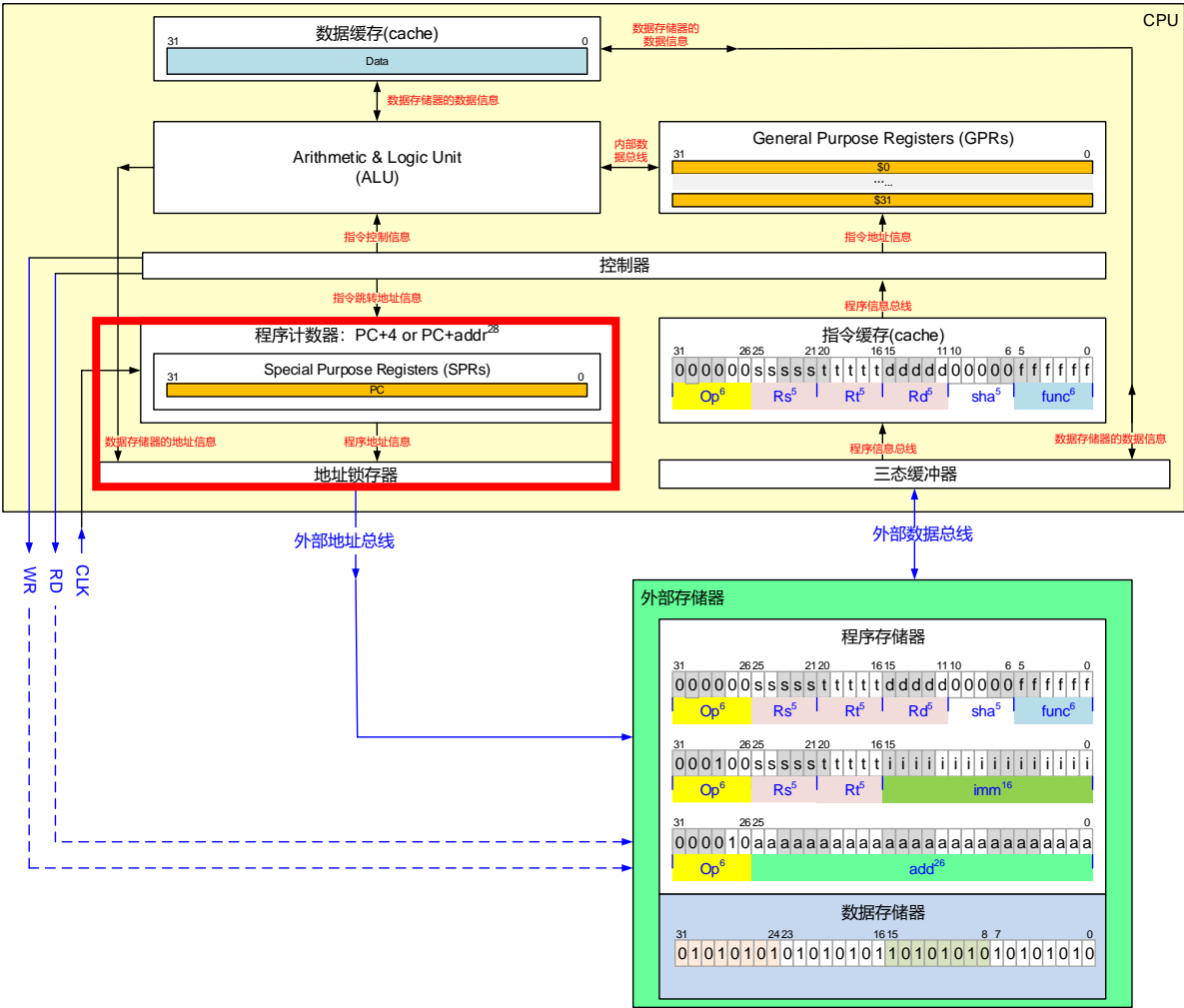
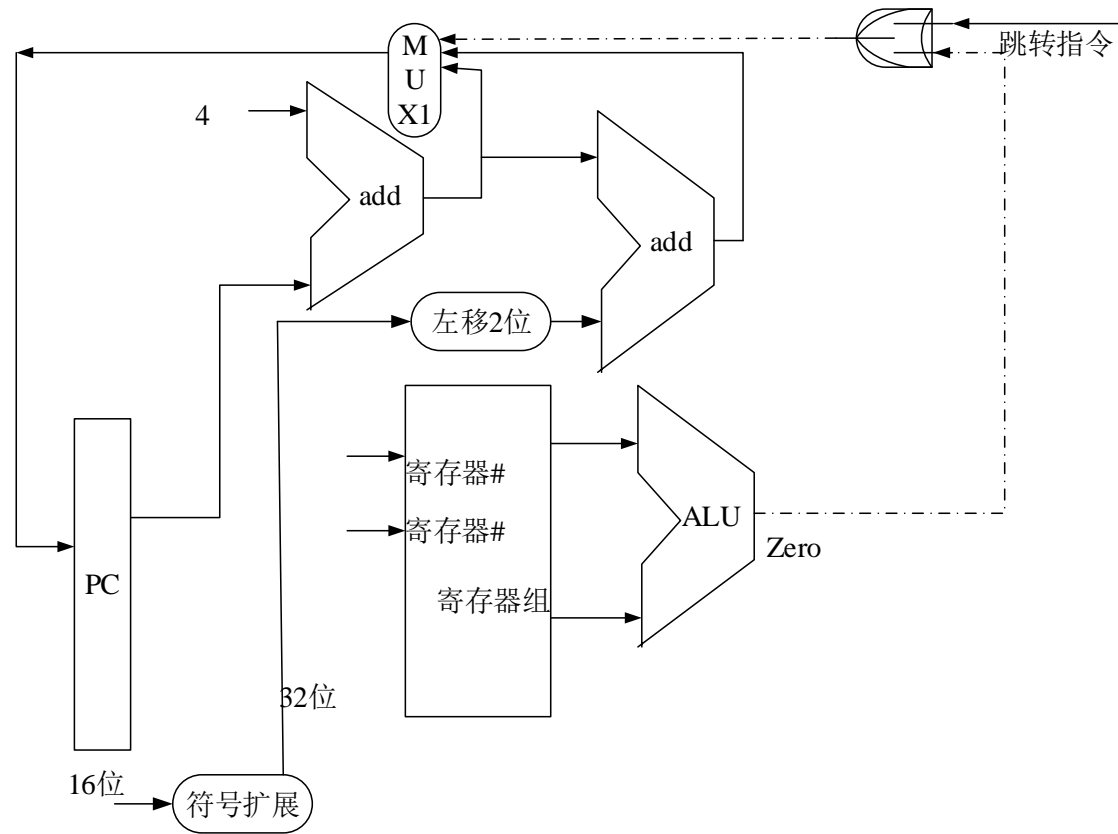


If Reg.File[Rs] is equal to Reg.File[Rt] then branch to label.

指令获取部件

条件跳转控制

- PC = PC + (Imm16符号数扩展<<2)

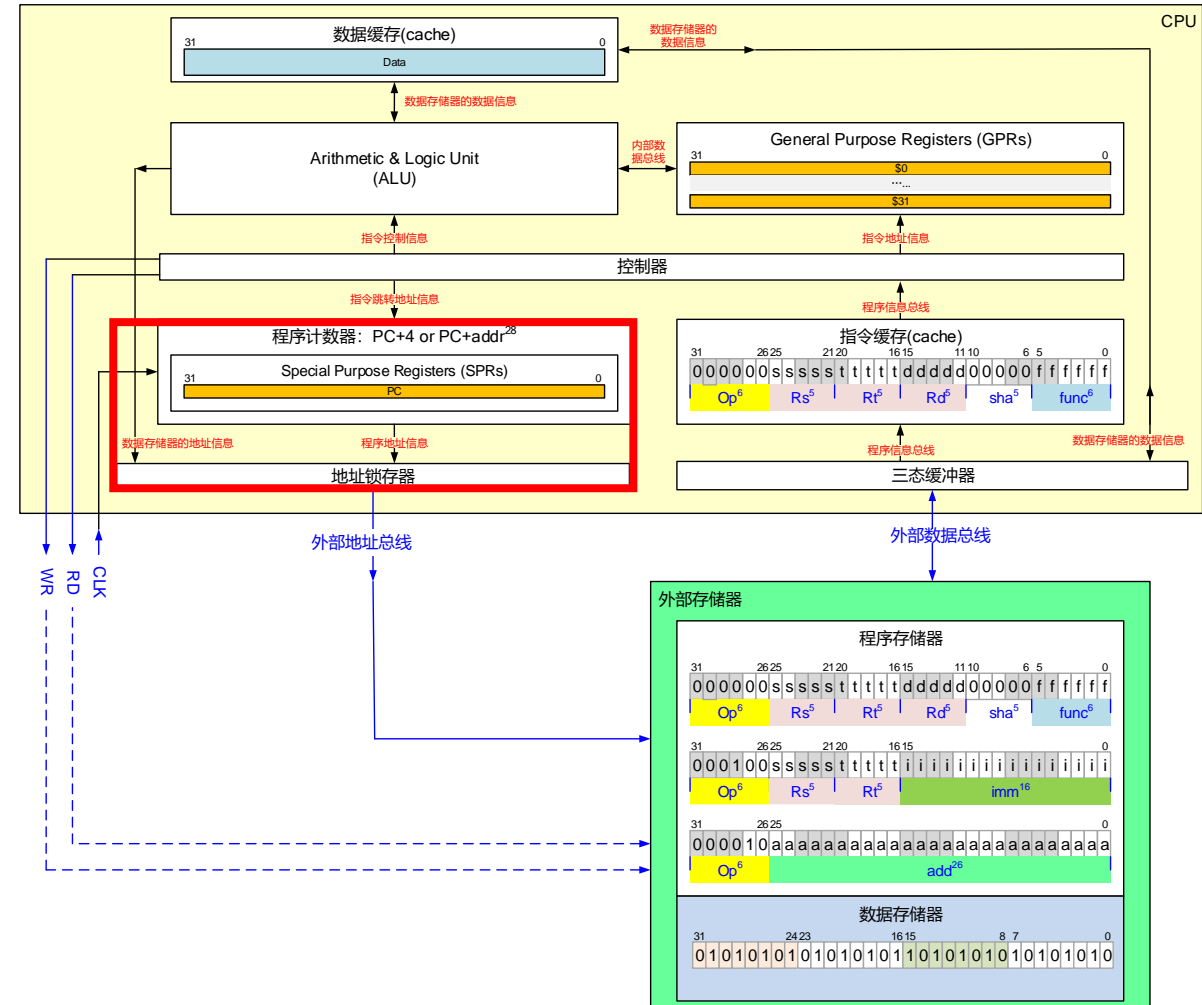
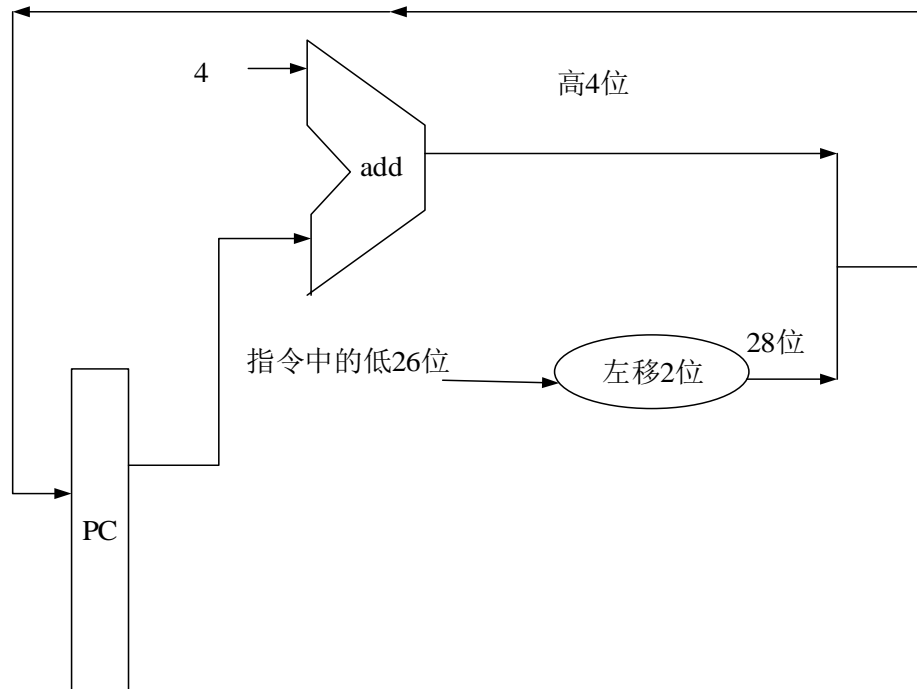
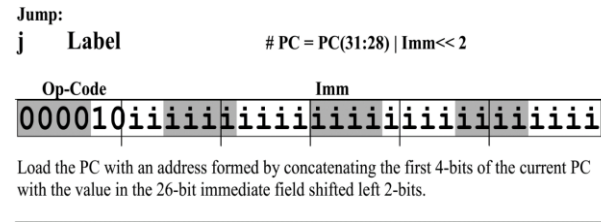


3.3 数据通路实现原理

指令获取部件

无条件伪直接寻址

$$PC = PC[31:28] \mid Imm26 \ll 2$$



3.3 数据通路实现原理

指令获取部件 (综合)

- 顺序执行时
 - PC自动加4

Add:
add Rd, Rs, Rt # $RF[Rd] = RF[Rs] + RF[Rt]$

Op-Code	Rs	Rt	Rd	Function Code
000000	ssssst	ttttt	ddddd	00000100000

Add contents of Reg.File[Rs] to Reg.File[Rt] and store result in Reg.File[Rd].
If overflow occurs in the two's complement number system, an exception is generated.

- 条件跳转控制
 - $PC = PC + (Imm16 \text{ 符号数扩展} \ll 2)$

Branch if Equal:
Beq Rs, Rt, Label # If $(RF[Rs] == RF[Rt])$ then $PC = PC + Imm \ll 2$

Op-Code	Rs	Rt	Imm
000100	ssssst	ttttt	iiiiiii

If Reg.File[Rs] is equal to Reg.File[Rt] then branch to label.

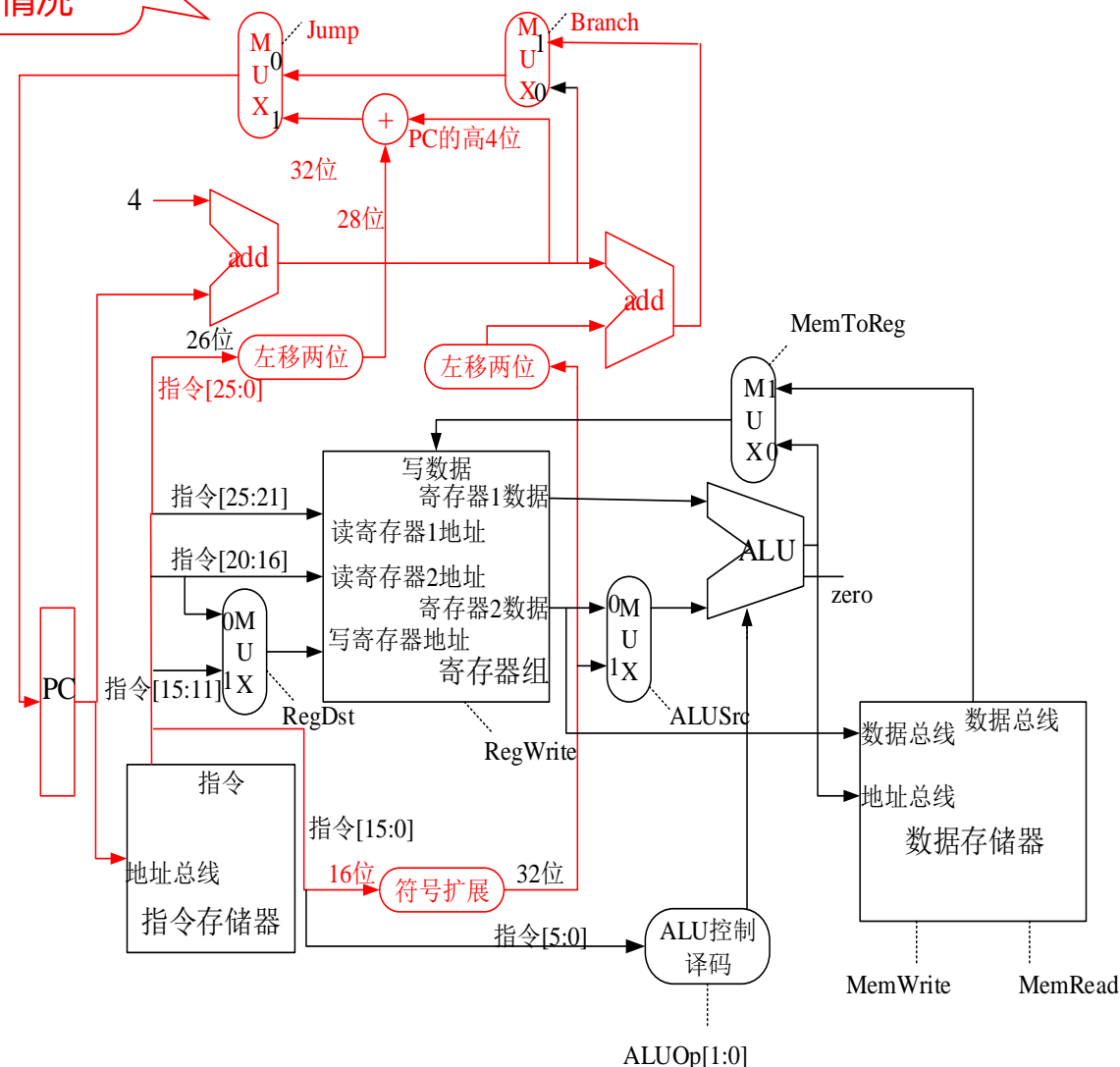
- 无条件伪直接寻址
 - $PC = PC[31:28] \mid Imm26 \ll 2$

Jump:
j Label # $PC = PC(31:28) \mid Imm \ll 2$

Op-Code	Imm
000010	iiiiiii

Load the PC with an address formed by concatenating the first 4-bits of the current PC with the value in the 26-bit immediate field shifted left 2-bits.

通过两个复用器来区分不同情况



3.3 数据通路实现原理

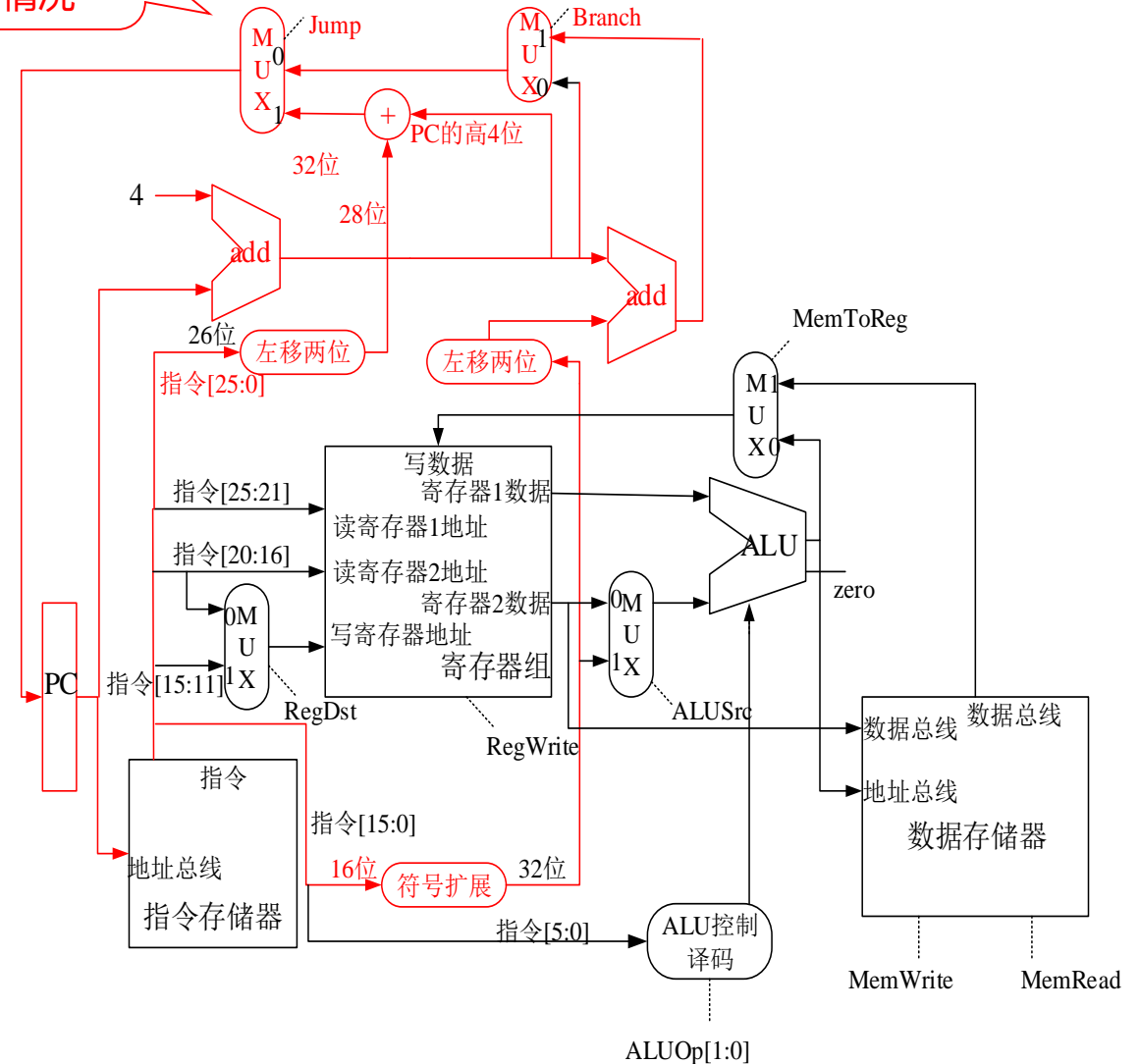
► 指令获取部件（综合）

- 顺序执行时
 - PC自动加4
 - Branch = 0、Jump = 0
- 条件跳转控制
 - $PC = PC + (\text{Imm16符号数扩展} \ll 2)$
 - Branch = 1、Jump = 0
- 无条件伪直接寻址
 - $PC = PC[31:28] \mid \text{Imm26} \ll 2$
 - Branch = x、Jump = 1

Branch、Jump怎么来？

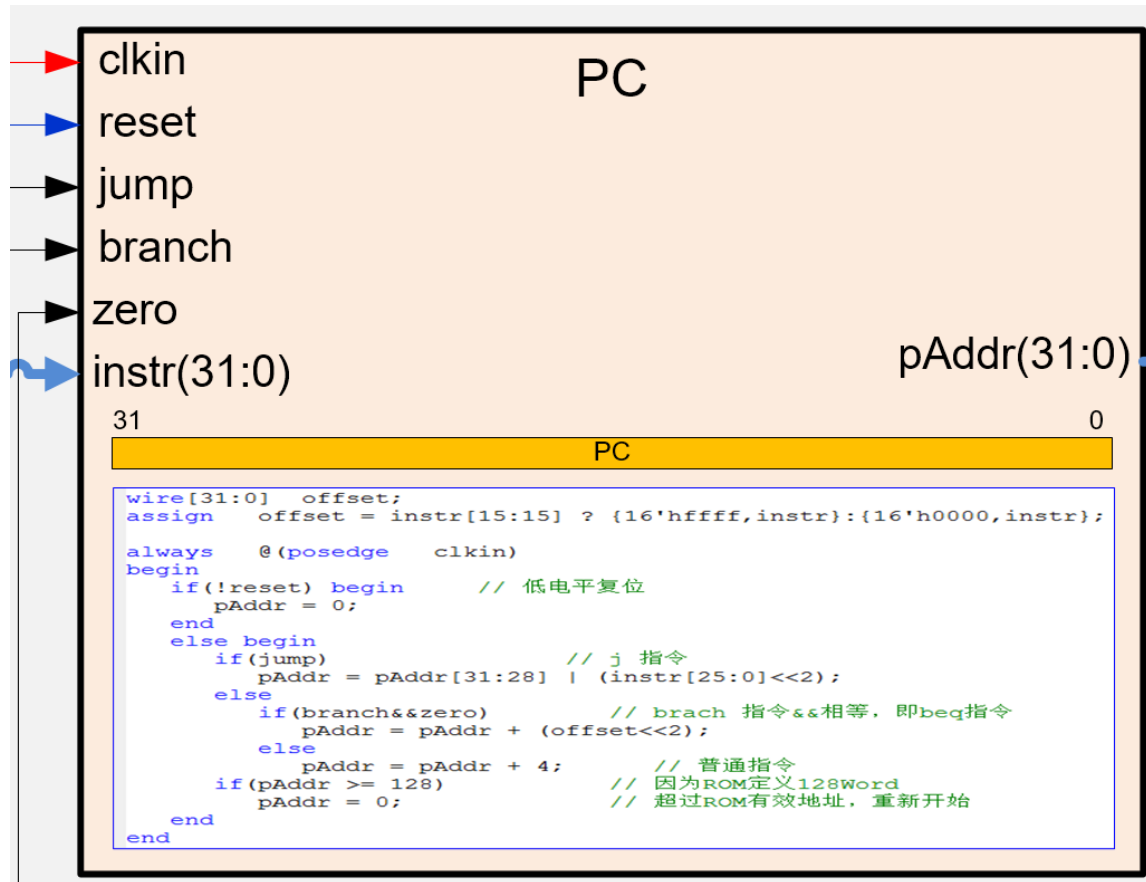
——由控制器，根据指令机器码（不同功能）产生

通过两个复用器来区分不同情况

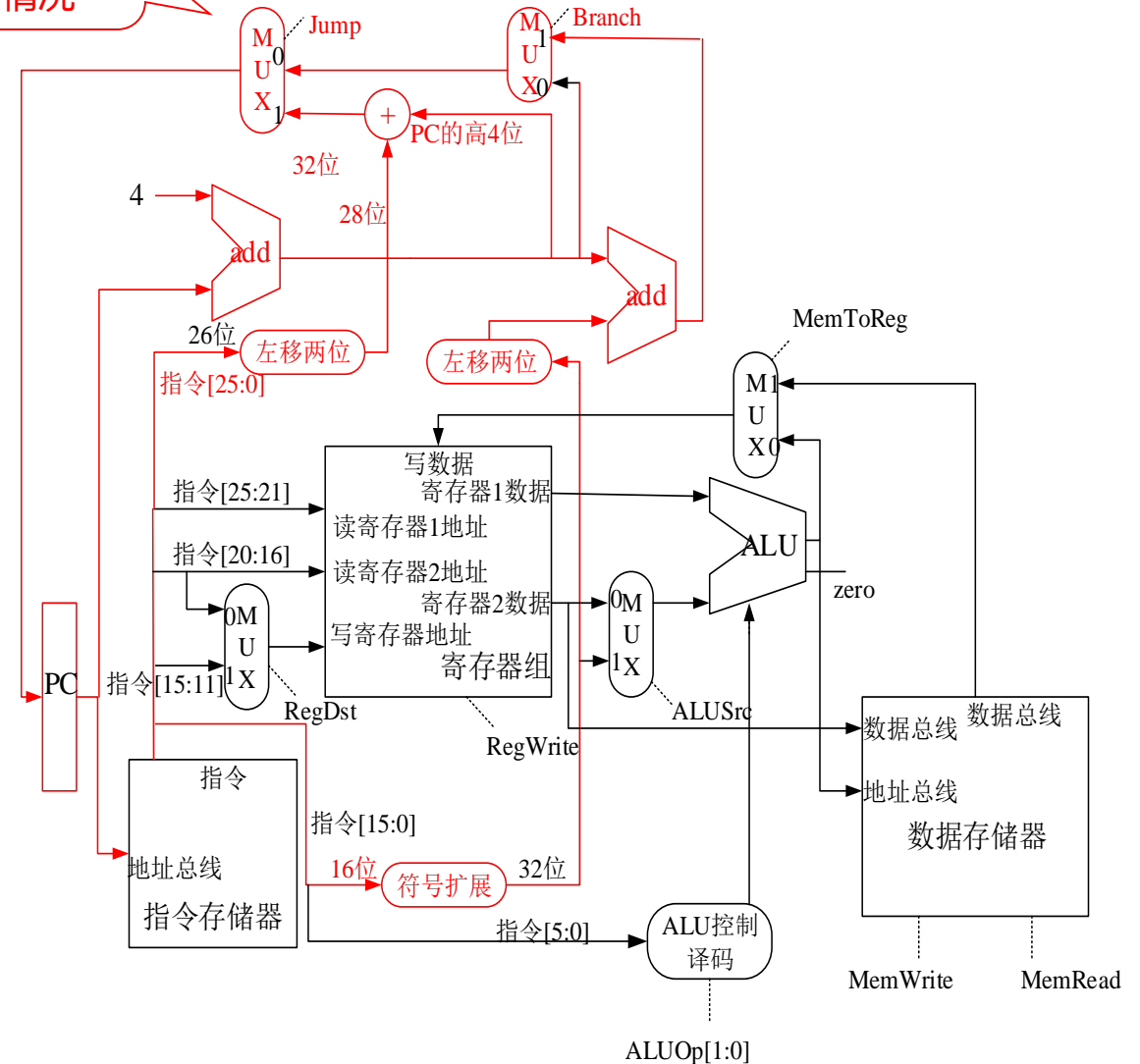


3.3 数据通路实现原理

指令获取部件的实现



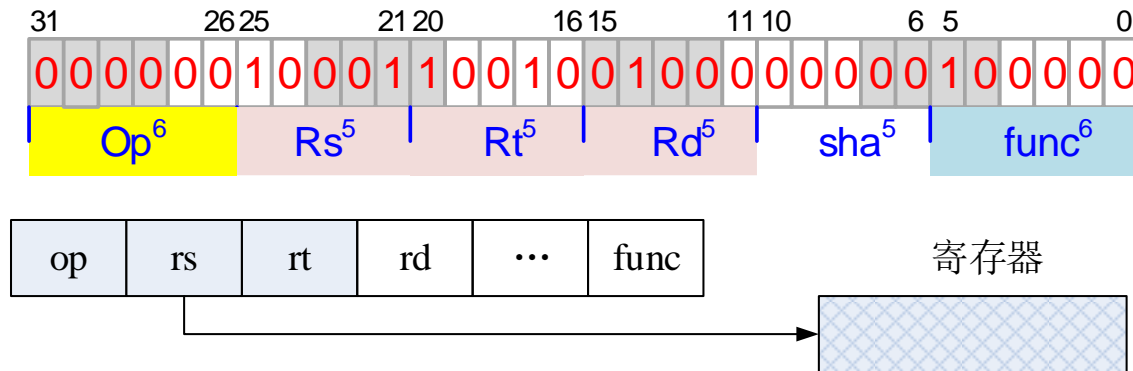
通过两个复用器来区分不同情况



3.3 数据通路实现原理

► R型指令实现部件

- 如：add \$t0, \$s1, \$s2



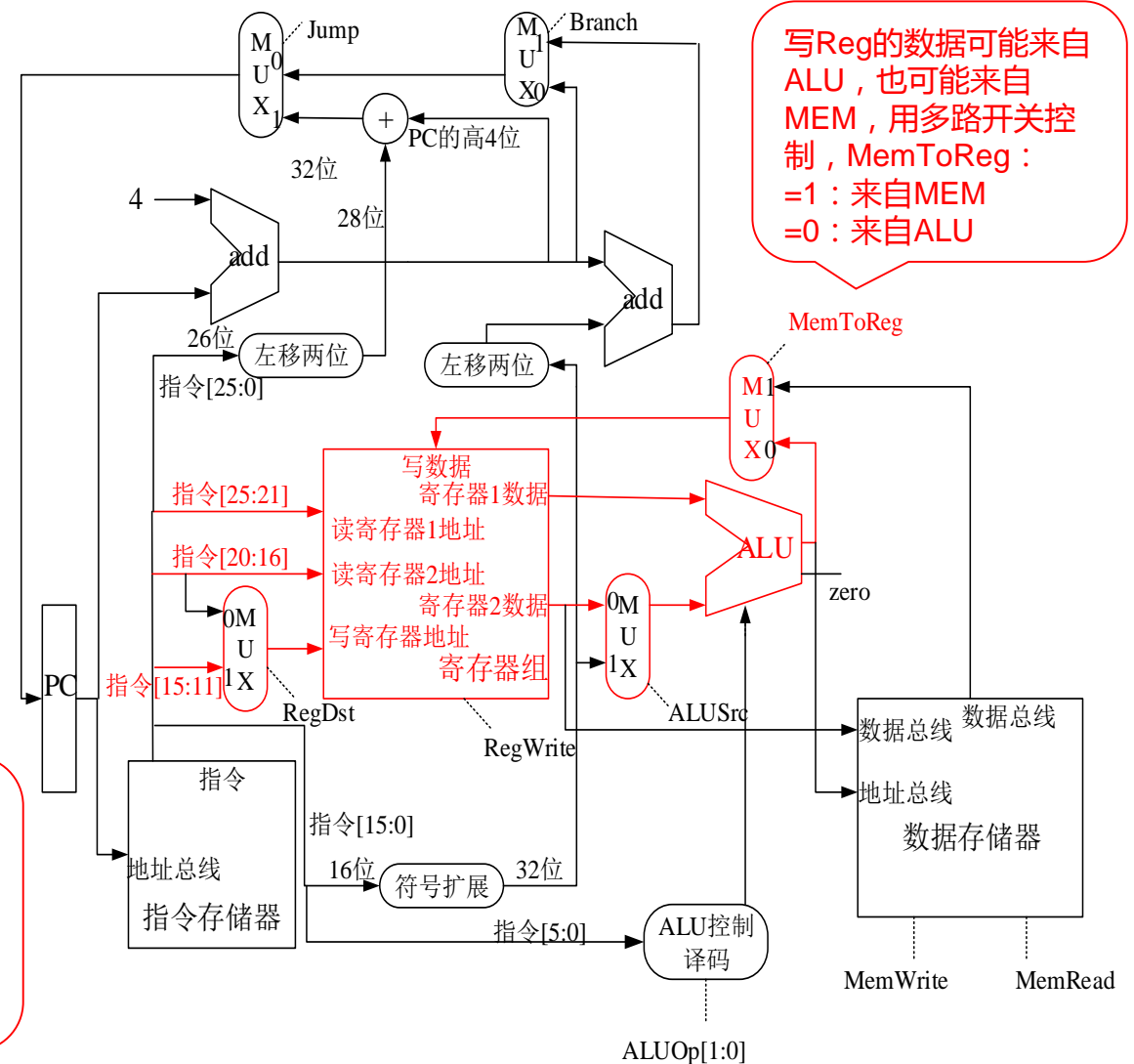
▪ 三个寄存器地址域

- 读Reg1的编号：Instr[25:21]
- 读Reg2的编号：Instr[20:16]
- 写Reg的编号：Instr[15:11]

▪ 三组数据总线：

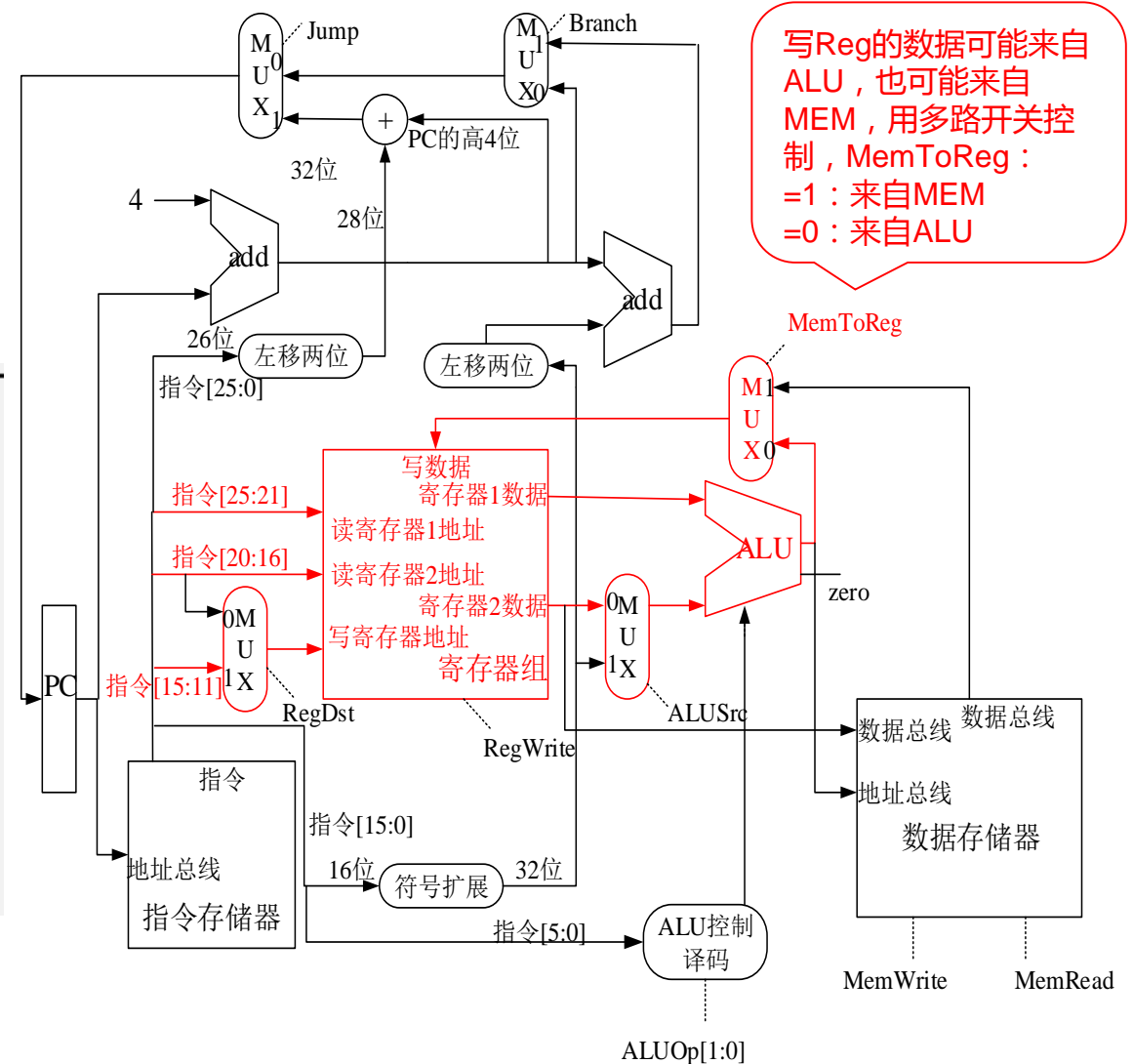
- 寄存器1数据：Rs的数据
- 寄存器2数据：Rt的数据
- ALU的结果：运算结果

R型指令中，写寄存器编号为Instr[15:11]，而I型指令中，写寄存器编号为Instr[20:16]，用RegDst区分：
=1：R型
=0：I型



► R型指令实现部件

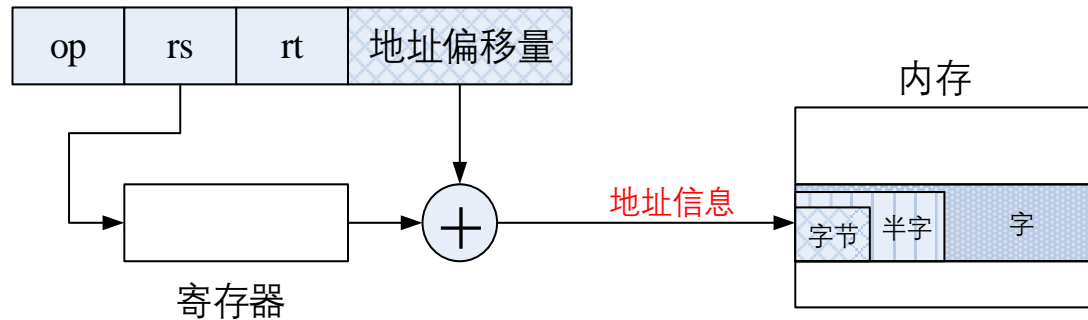
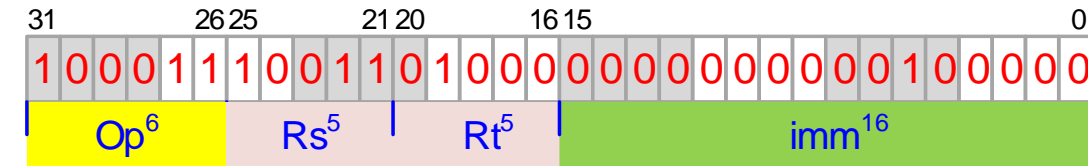
-
- 31 26 25 21 20 16 15 11 10 6 5 0
- 0 0 0 0 0 0 1 0 0 0 1 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0
- Op⁶ Rs⁵ Rt⁵ Rd⁵ sha⁵ func⁶



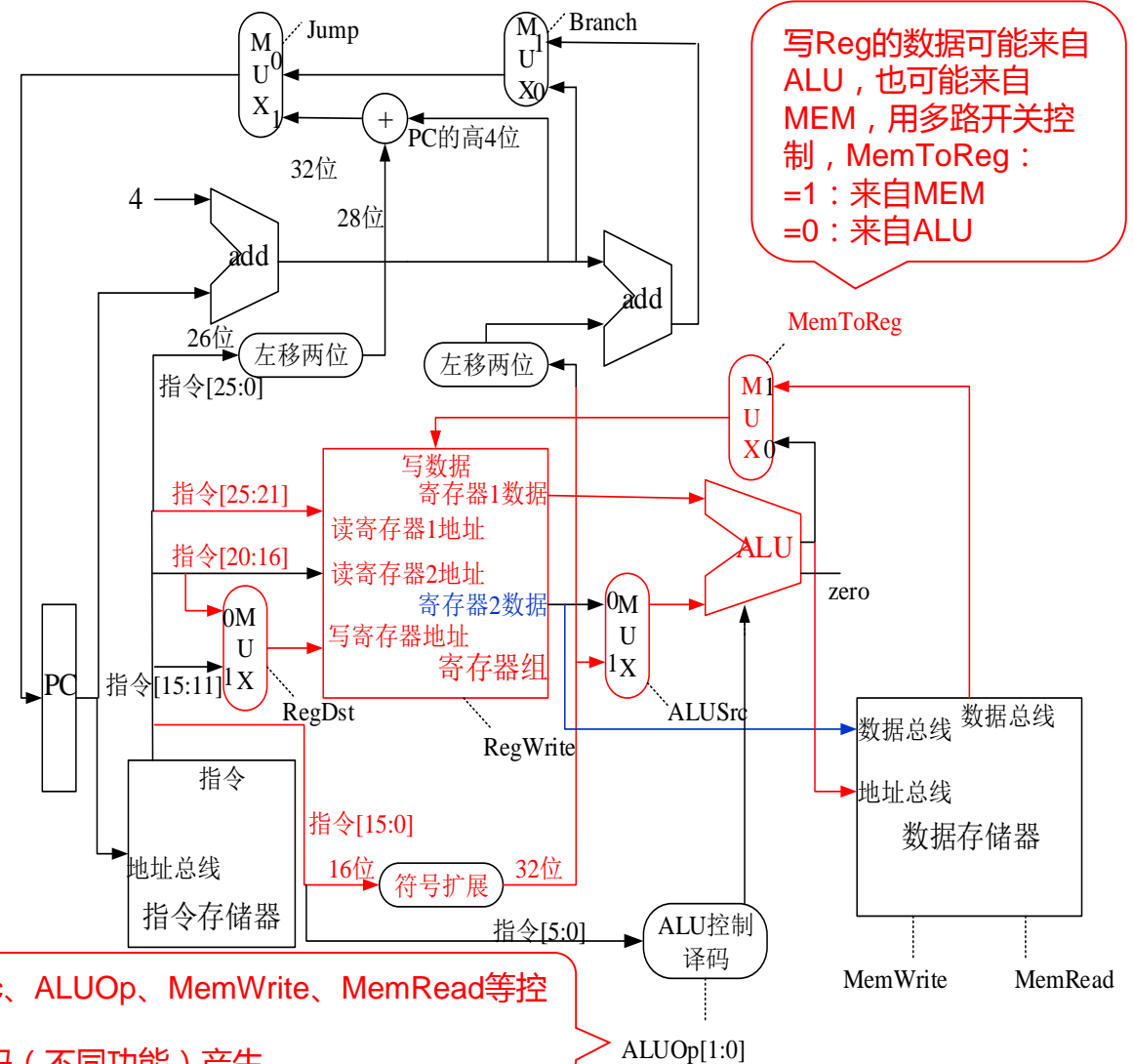
3.3 数据通路实现原理

► 存储器操作实现部件

- 如：lw \$t0, 32(\$s3)



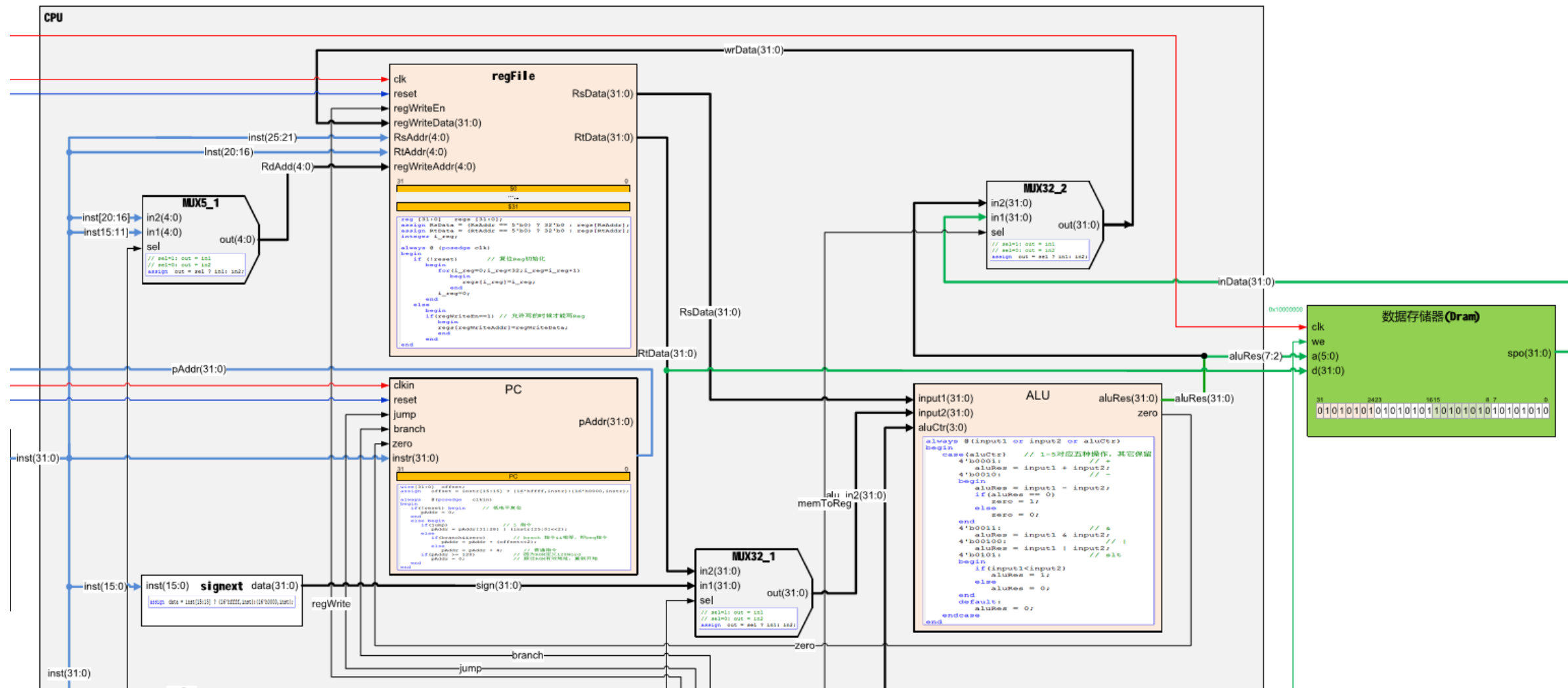
- 基址寻址来访问存储器
 - Instr[25:21]编号寄存器内容 + 符号偏移
- 读存储器时，数据来自MEM
 - MemToReg = 1、ALUSrc = 1、RegDst = 0、RegWrite = 1
- 写存储器时，数据来自Reg
 - RegWrite = 0



MemtoReg、RegDst、ALUSrc、ALUOp、MemWrite、MemRead等控制位怎么来？
——由控制器，根据指令机器码（不同功能）产生

3.3 数据通路实现原理

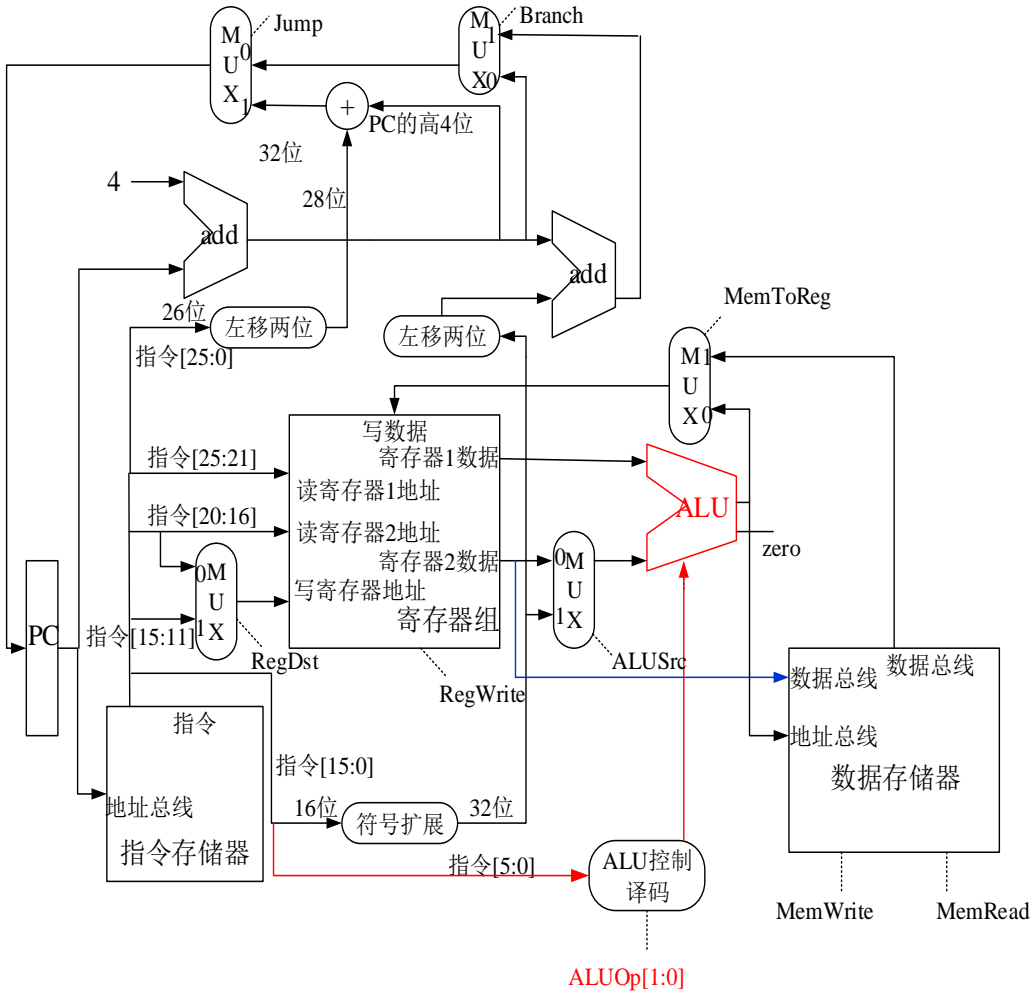
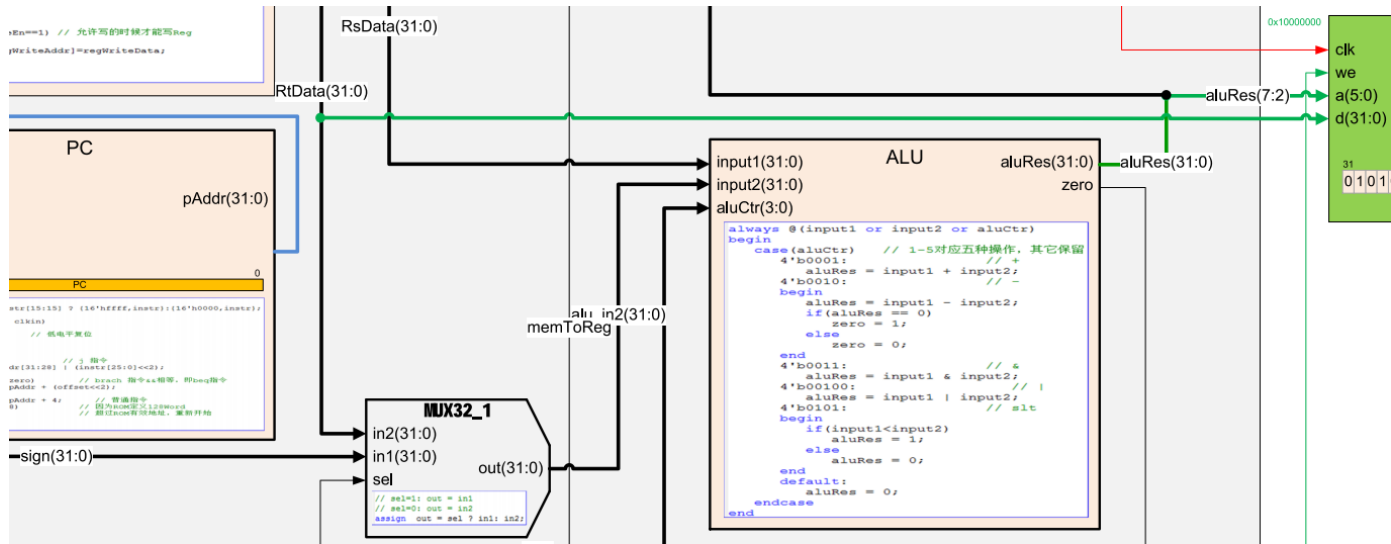
▶ 存储器操作实现部件



3.3 数据通路实现原理

- **ALU部件**
- ALU单元利用4根输入控制线的译码决定执行何种操作

输入控制信号编码	操作类型
0011	与
0100	或
0001	加
0010	减
0101	小于设置

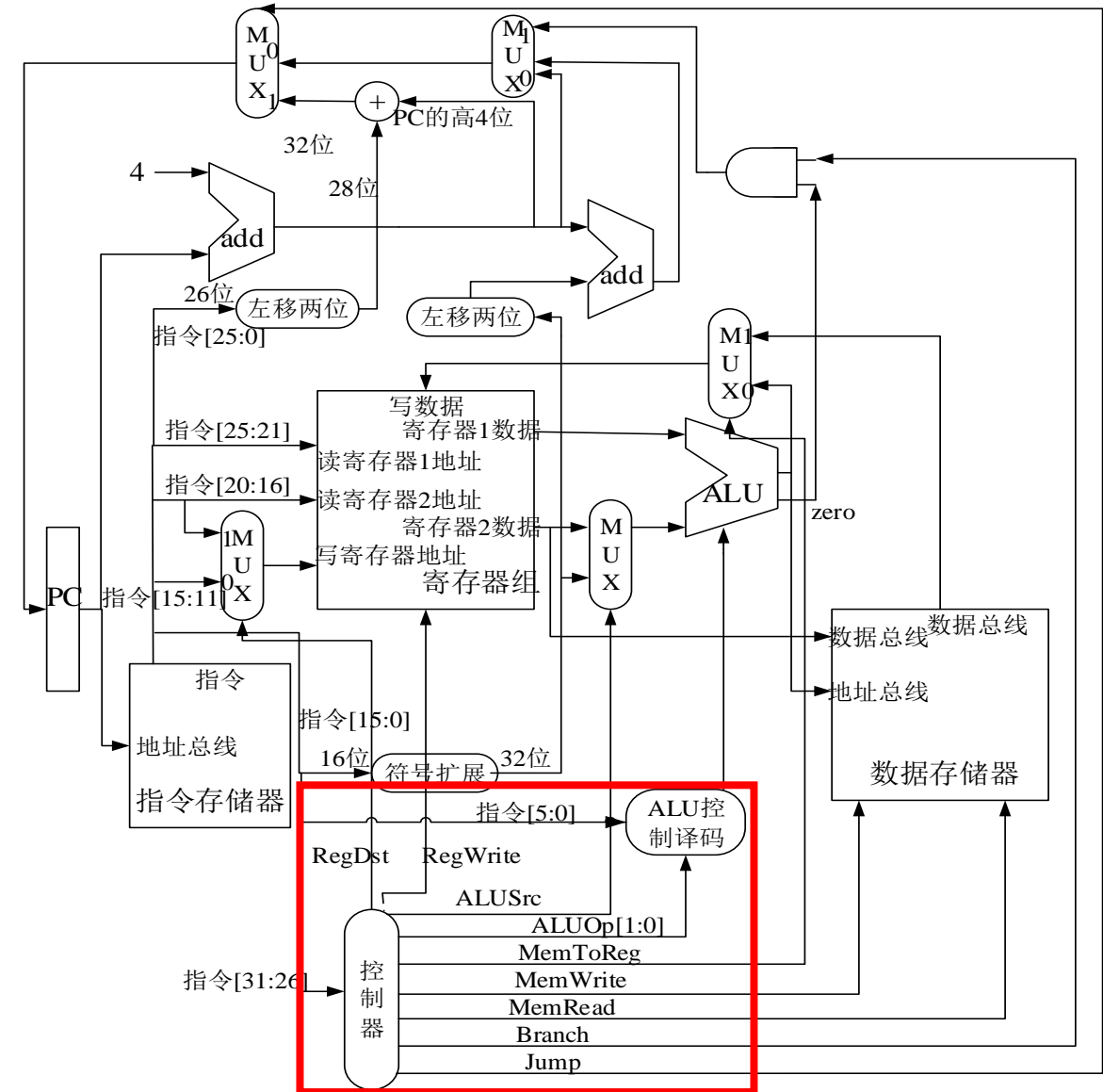


3.4 控制器实现原理

▶ 控制器：需产生下列控制信号

- ALU控制aluCtr[3:0]
 - 微处理器需要支持add、sub、and、or、slt运算指令，需要利用ALU单元实现运算
 - 数据存储指令sw、lw需要通过ALU单元计算存储器地址（加）
 - 条件跳转指令beq需要ALU来比较两个寄存器是否相等（减）
 - 包含的操作为加、减、与、或、小于设置等5种不同的操作
- Branch、Jump
- MemRead、MemWrite、MemToReg
- ALUSrc、RegDst、RegWrite

add、sub、and、or、slt，lw、sw、beq，j 指令的机器码不同（体现在Op⁶、func⁶），可以据此来区分不同指令，从而产生上述不同控制信号

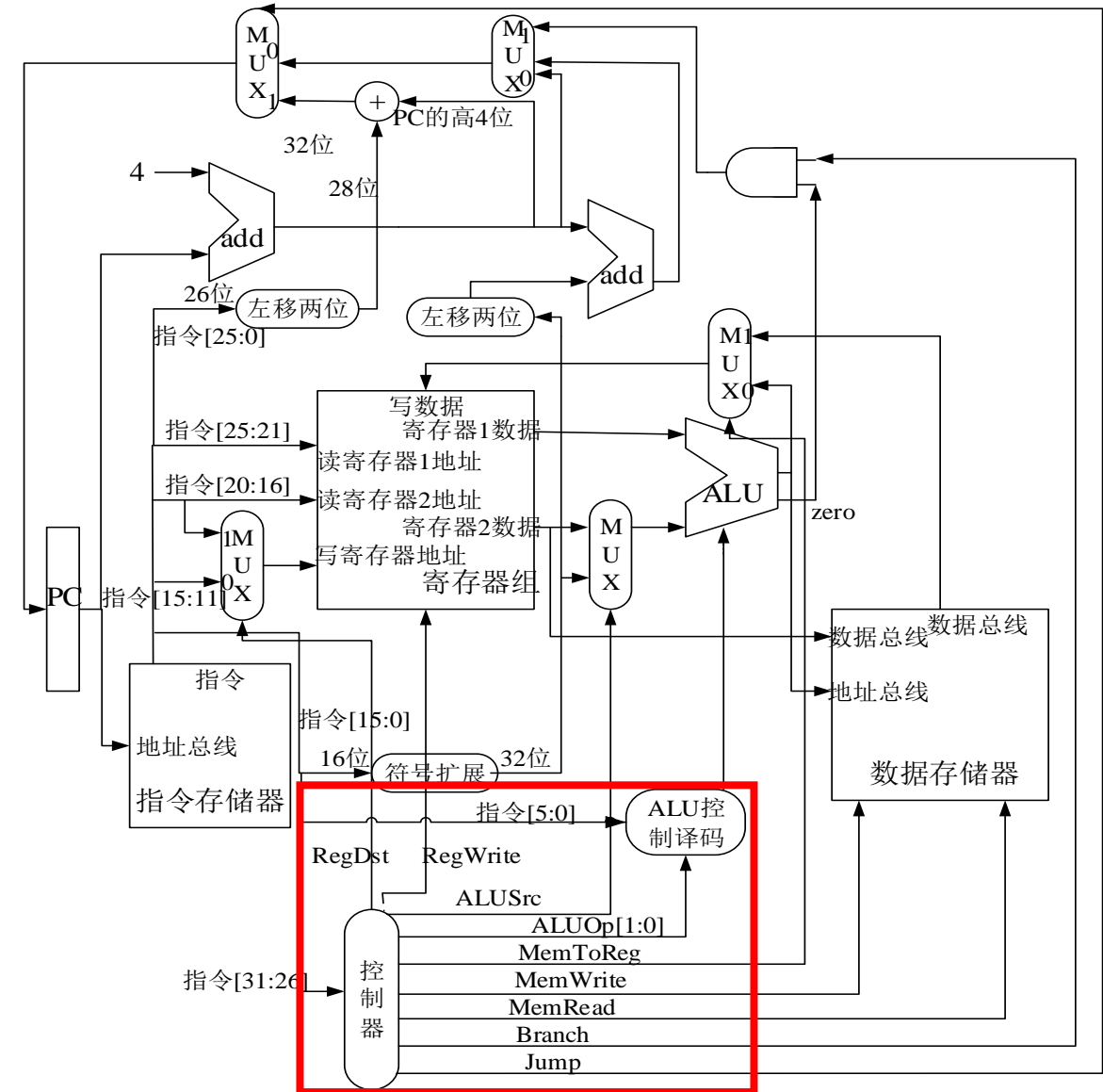


3.4 控制器实现原理

► 控制信号定义

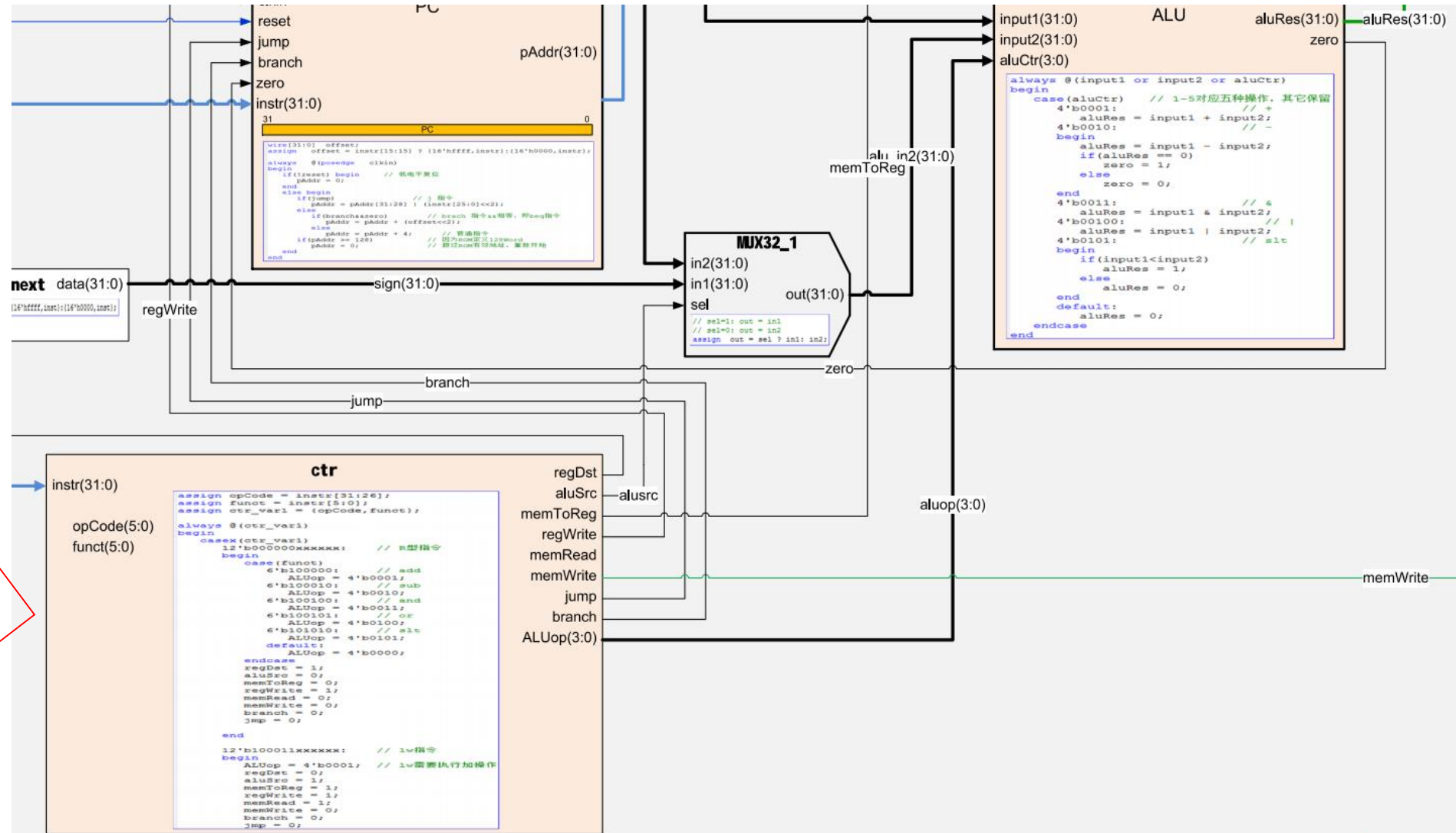
ALU的控制信号 aluCtr[3:0]	ALU的 运算
0010	加
0110	减
0000	与
0001	或
0111	小于设置

控制信号 名称	置1	清0
RegDst	表示写寄存器的地址来自指令[15:11]	来自指令[20:16]
Jump	表示PC的值来自伪直接寻址	来自另一个复用器
Branch	表示下一级复用器的输入来PC相对寻址加法器	来自PC+4
MemToReg	表示写寄存器数据来自存储器数据总线	来自ALU结果
ALUSrc	表示ALU的第二个数据源来自指令[15:0]	来自读寄存器2
RegWrite	将写寄存器数据存入写寄存器地址中	无操作
MemWrite	将写数据总线上的数据写入内存地址单元	无操作
MemRead	将内存单元的内容输出到读数据总线上	无操作



3.4 控制器实现原理

▶ 控制器实现部件

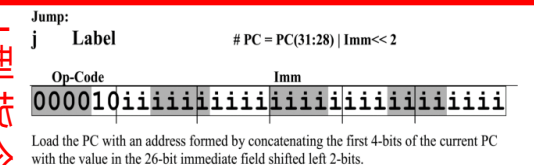
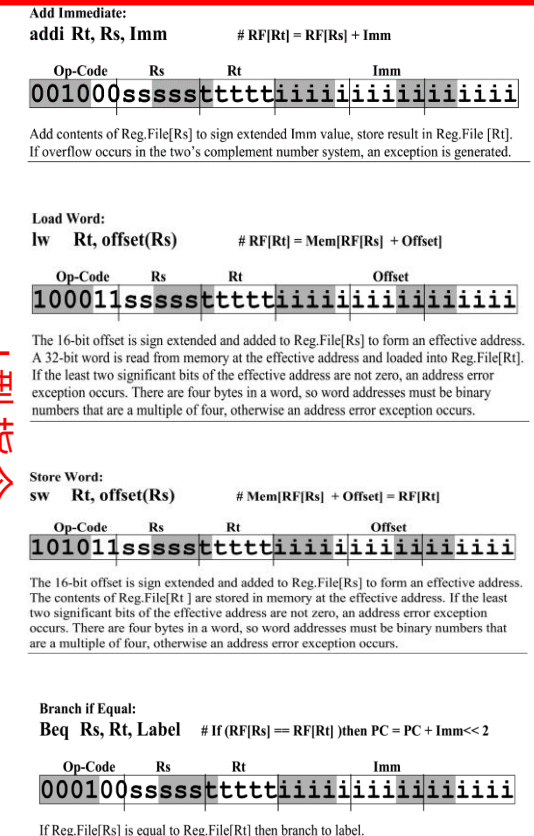
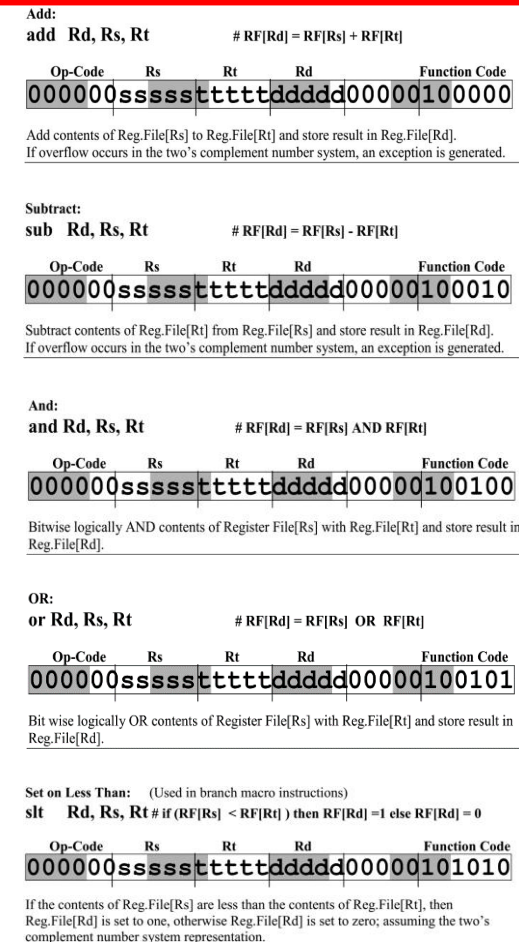


add、sub、and、or、slt、lw、sw、beq、j 指令的机器码不同（体现在Op⁶、func⁶），可以据此来区分不同指令，从而产生上述不同控制信号

3.4 控制器实现原理

► 指令类型及机器码

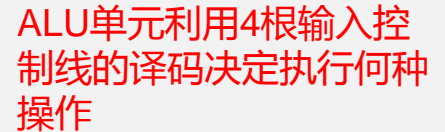
- 支持add, sub, and, or, slt, **andi**, lw, sw, beq, j 等指令
 - 加、减、与、或、slt的op-code均为**000000**, 通过function code来区分(2级译码):
 - Func = 100000 : add
 - Func = 100010 : sub
 - Func = 100100 : and
 - Func = 100101 : or
 - Func = 101010 : slt
 - lw: op-code = **100011**
 - sw: op-code = 101011
 - beq: op-code = 000100
 - j: op-code = 000010



► 控制信号的产生

- 【提醒1】操作码**
aluCtr[3:0]的取值和课本不一致，只要ctr模块和ALU模块中的信号一致就行！！

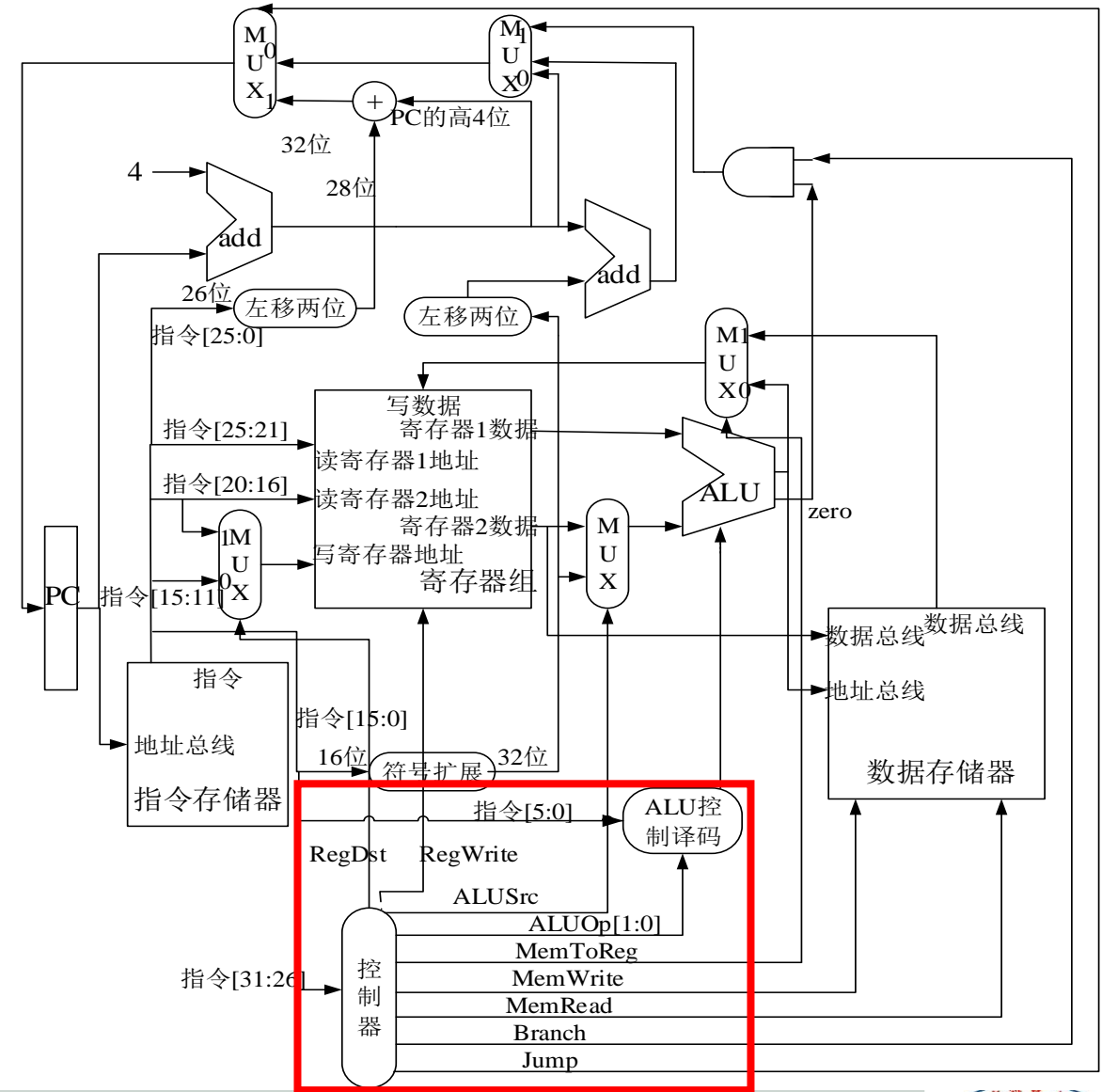
输入控制信号编码	操作类型
0001	加
0010	减
0011	与
0100	或
0101	小于设置



ctr模块产生的控制信号
ALUOp和ALU模块中的
aluCtr信号所对应的操作
要一致！！
—— 编程者决定

► 控制信号的产生

- 课本上的ALUOp只有2位，还要进一步和Instr[5:0]组合译码，才能产生控制ALU的4位控制信号aluCtr[3:0]；
本例中直接对Instr[31:26]、Instr[5:0]进行译码，产生控制ALU的4位控制信号

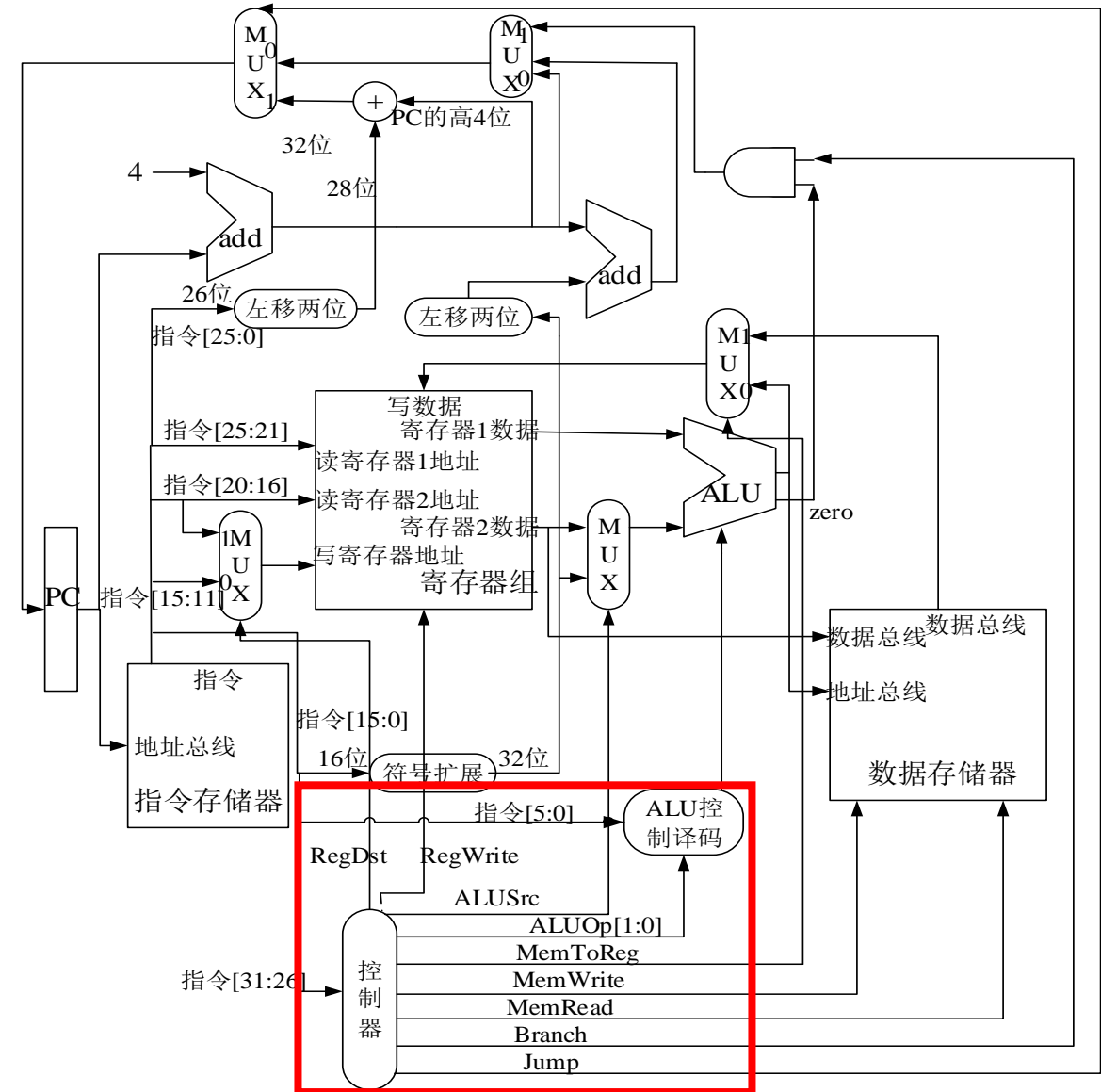


3.4 控制器实现原理

► 控制信号的产生

- aluCtr[3:0]的另一种产生方法【课本】
 - 先根据Instr[31:26]产生2位的中间控制信号ALUOp[1:0]
 - ALUOp[1:0]再和功能码Instr[5:0]一起产生四位ALU控制信号aluCtr[3:0]。

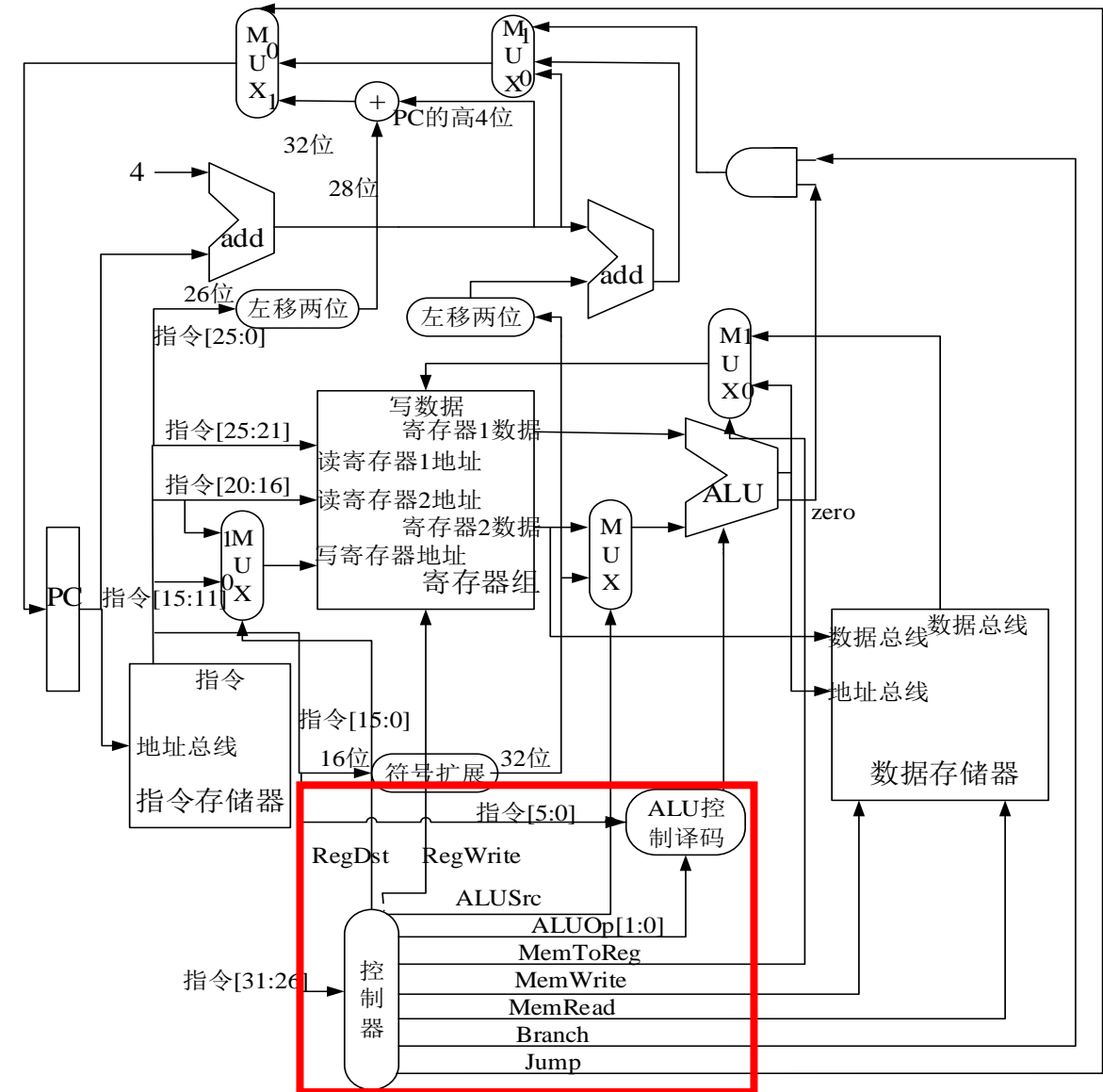
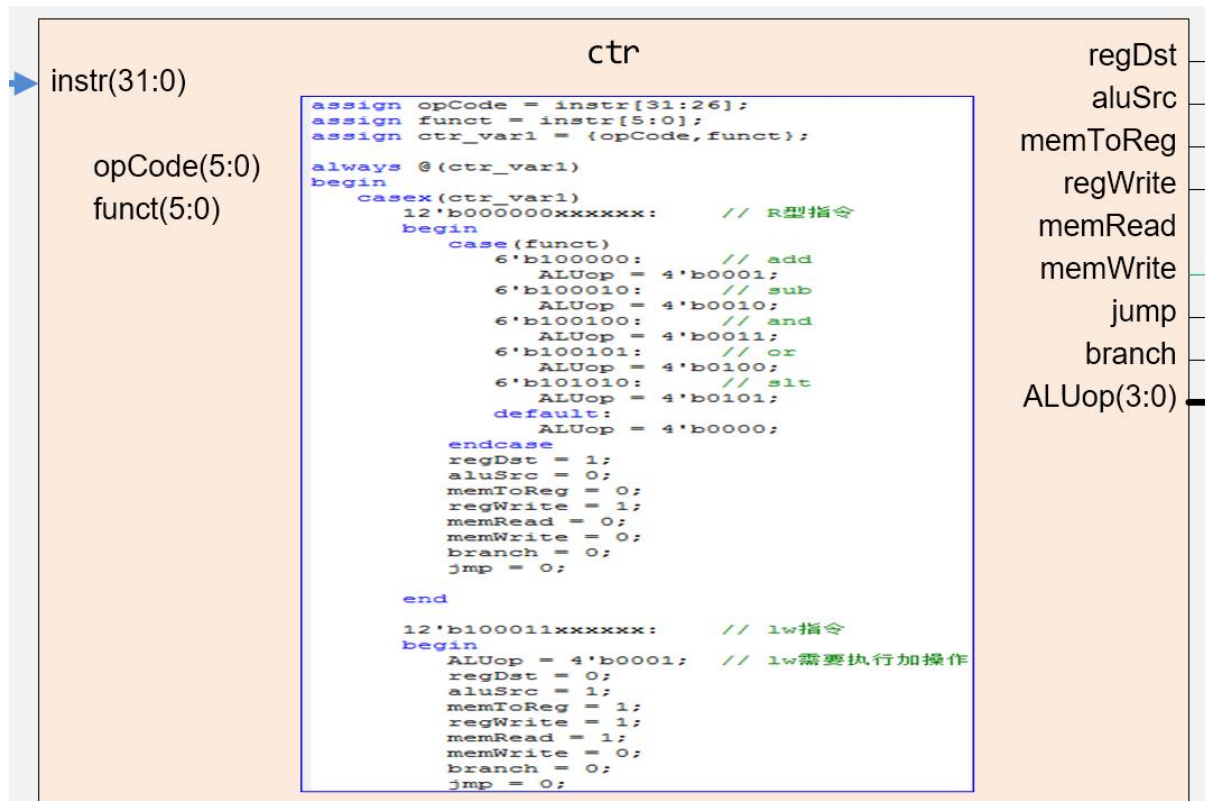
指令	2位操作码	指令功能	6位功能码	ALU的运算	ALU的控制信号
LW	00	取字	XXXXXX	加	0010
SW	00	存字	XXXXXX	加	0010
BEQ	01	相等跳转	XXXXXX	减	0110
R型指令	10	加	100000	加	0010
R型指令	10	减	100010	减	0110
R型指令	10	与	100100	与	0000
R型指令	10	或	100101	或	0001
R型指令	10	小于设置	101010	小于设置	0111



3.4 控制器实现原理

► 控制信号的产生

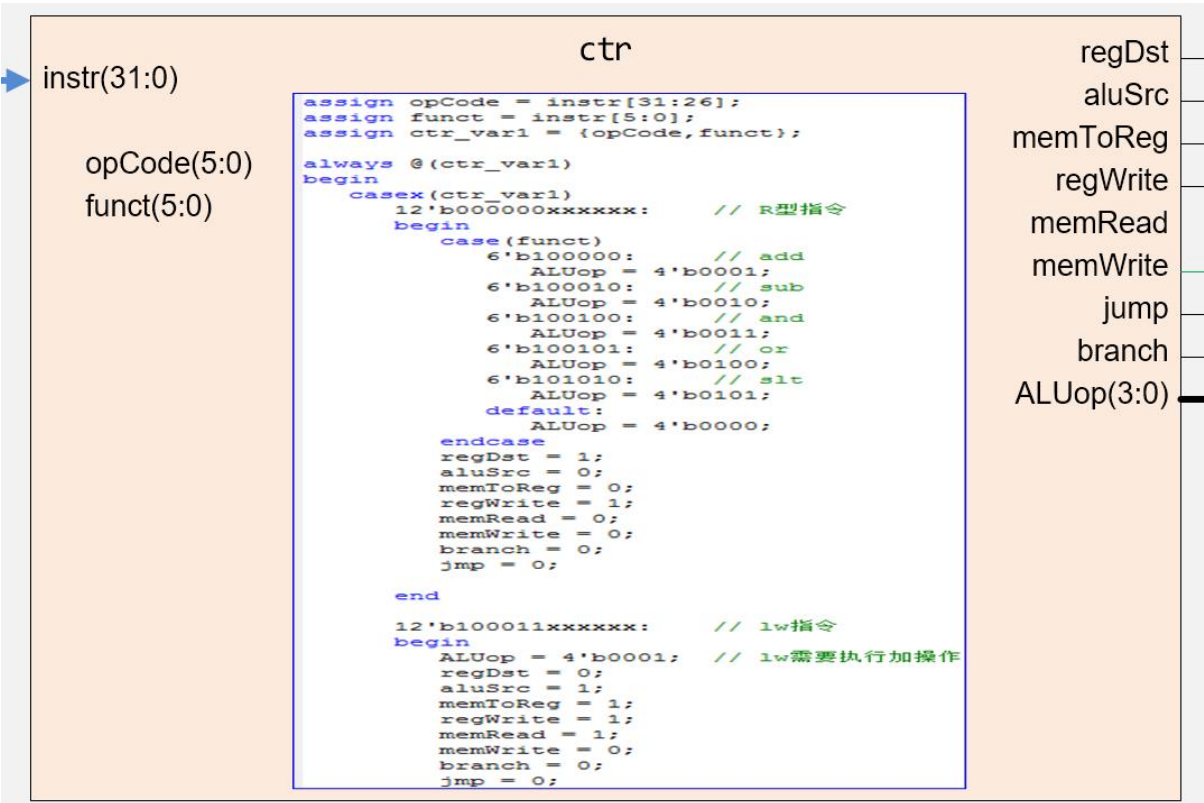
- Branch、Jump
- MemRead、MemWrite、MemToReg
- ALUSrc、RegDst、RegWrite



3.4 控制器实现原理

► 控制信号的产生

- Branch、Jump
- MemRead、MemWrite、MemToReg
- ALUSrc、RegDst、RegWrite

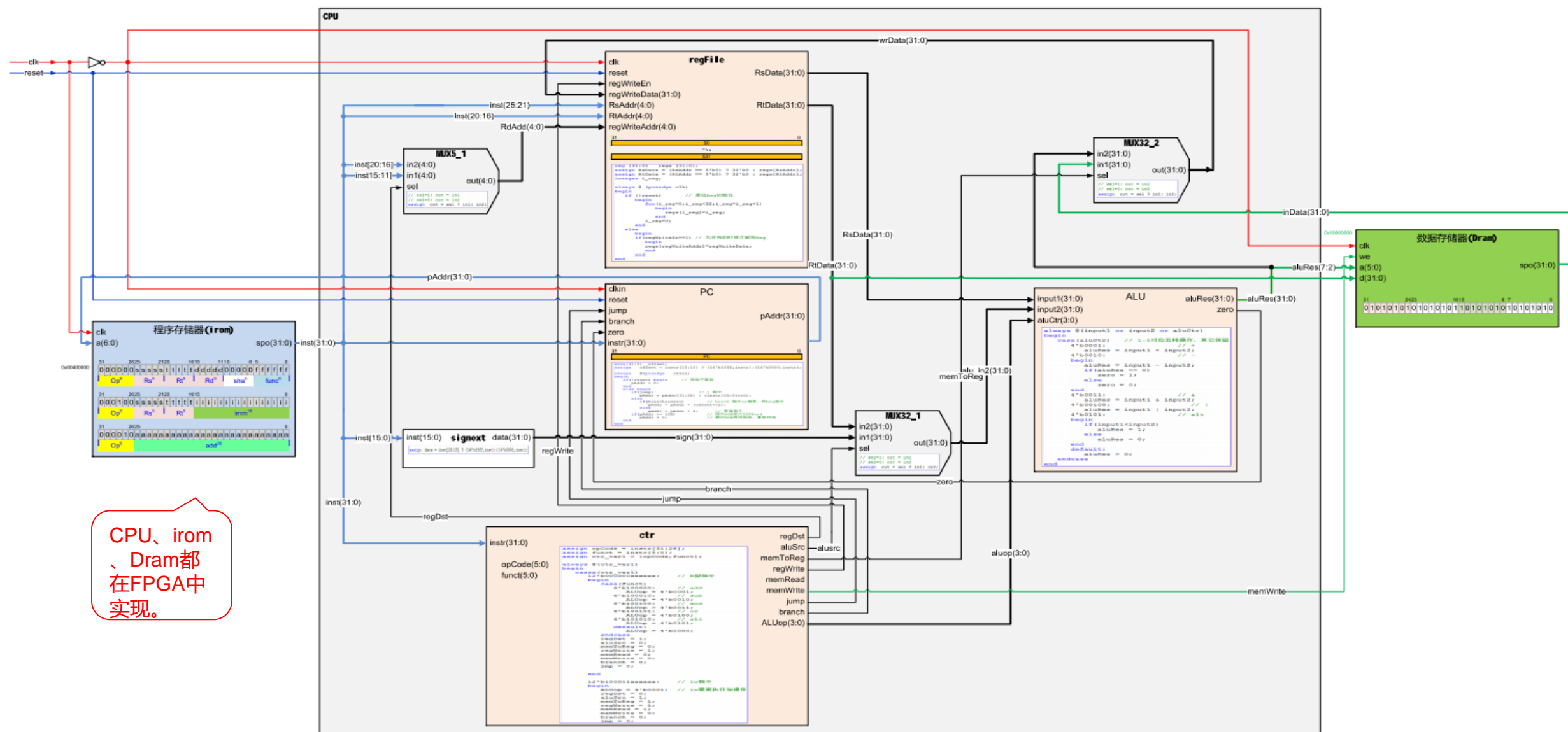


控制信号名称	置1	清0
RegDst	表示写寄存器的地址来自指令[15:11]	来自指令[20:16]
Jump	表示PC的值来自伪直接寻址	来自另一个复用器
Branch	表示下一级复用器的输入来PC相对寻址加法器	来自PC+4
MemToReg	表示写寄存器数据来自存储器数据总线	来自ALU结果
ALUSrc	表示ALU的第二个数据源来自指令[15:0]	来自读寄存器2
RegWrite	将写寄存器数据存入写寄存器地址中	无操作
MemWrite	将写数据总线上的数据写入内存地址单元	无操作
MemRead	将内存单元的内容输出到读数据总线上	无操作

输入输出	信号名称	R型	lw	sw	beq	j
输入	op5	0	1	1	0	0
	op4	0	0	0	0	0
	op3	0	0	1	0	0
	op2	0	0	0	1	0
	op1	0	1	1	0	1
	op0	0	1	1	0	0
	ALUop1	1	0	0	0	X
输出	ALUop0	0	0	0	1	X
	RegDst	1	0	X	X	X
	ALUSrc	0	1	1	0	X
	RegWrite	1	1	0	0	0
	MemWrite	0	0	1	0	0
	MemRead	0	1	0	0	0
	Branch	0	0	0	1	0
	Jump	0	0	0	0	1
	MemToReg	0	1	X	X	X



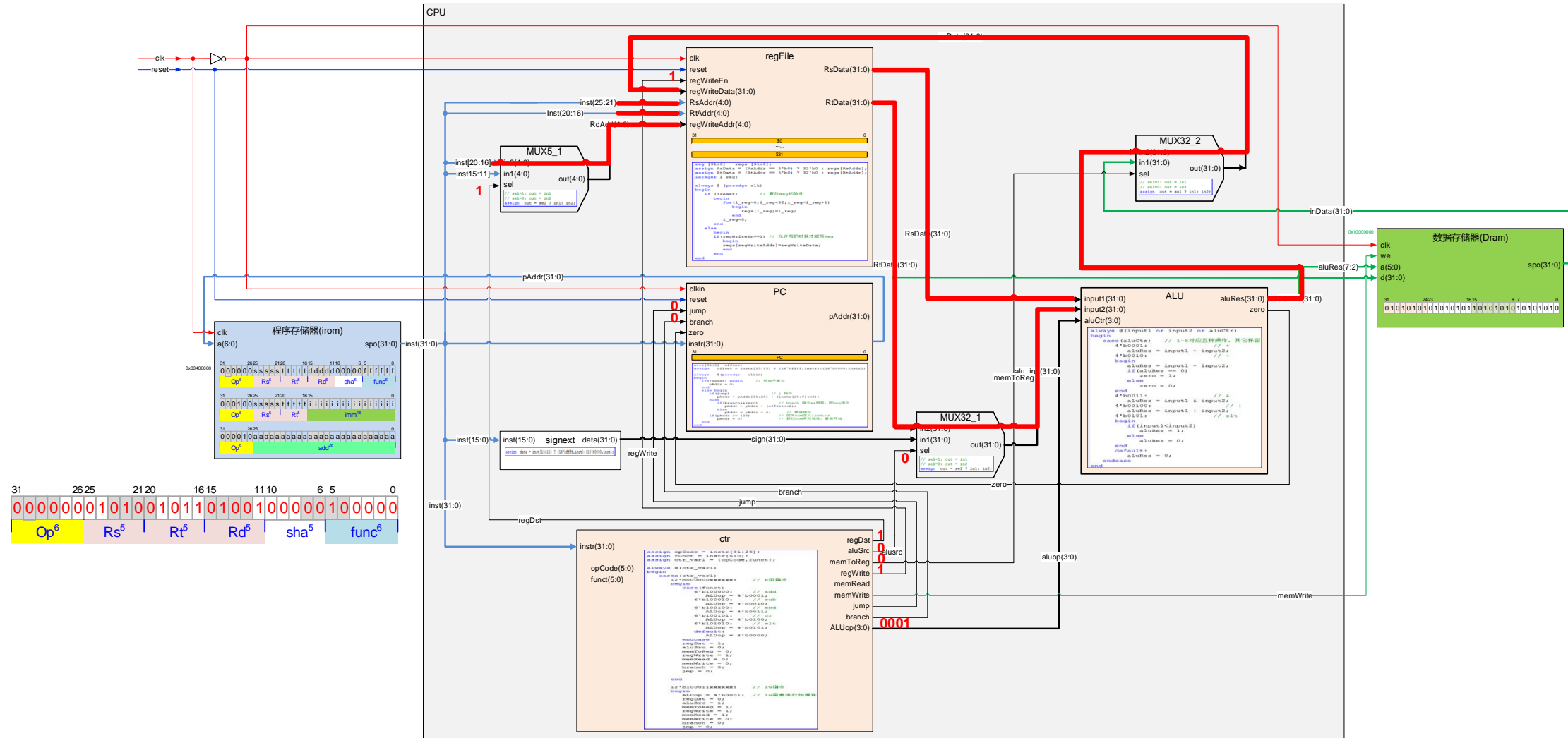
MIPS CPU的实现框图



CPU、irom、Dram都在FPGA中实现。

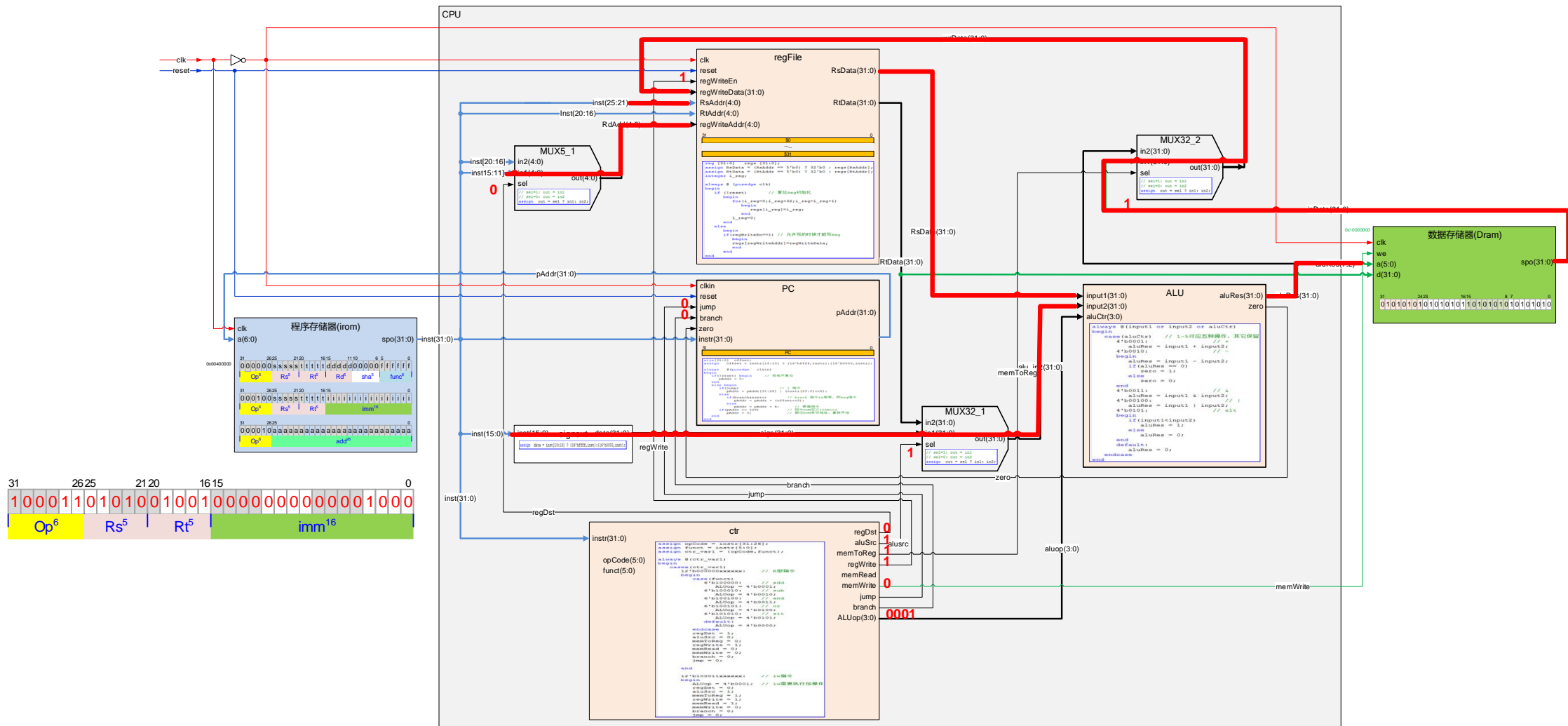
3.4 控制器实现原理

► 不同指令执行过程描述 (add \$t1, \$t2, \$t3 指令)



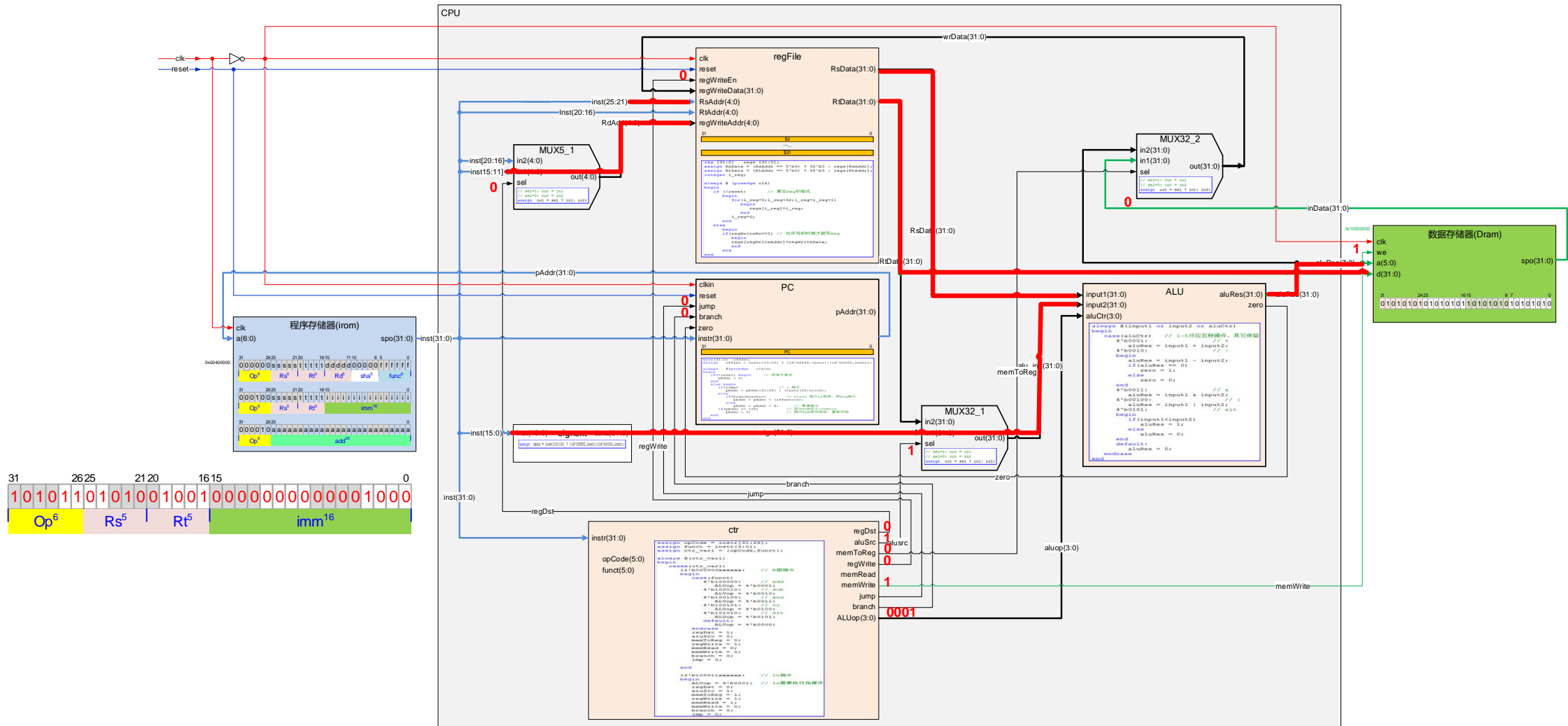
3.4 控制器实现原理

► 不同指令执行过程描述 (lw \$t1,8(\$t2)指令)



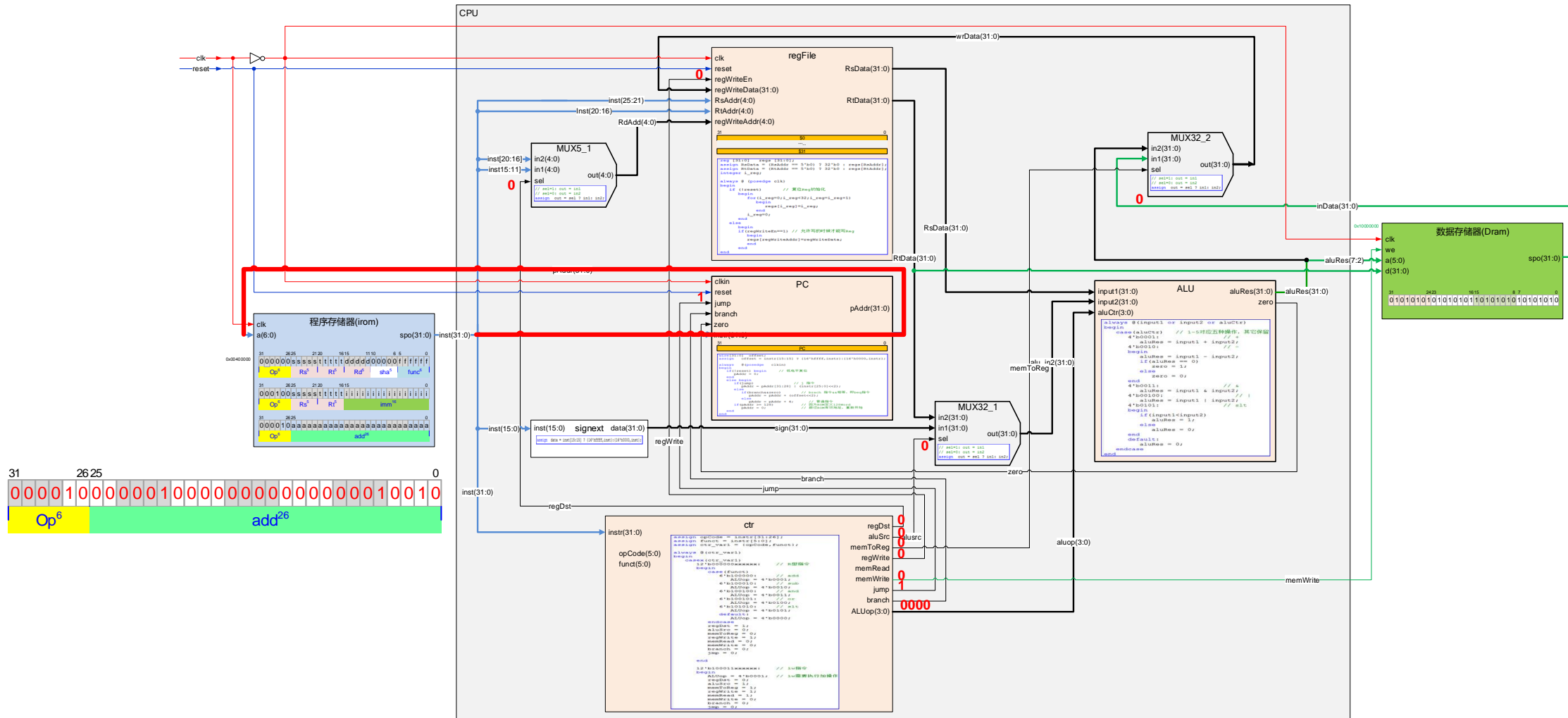
3.4 控制器实现原理

► 不同指令执行过程描述 (sw \$t1,8(\$t2)指令)



3.4 控制器实现原理

不同指令执行过程描述 (j xxxx指令)



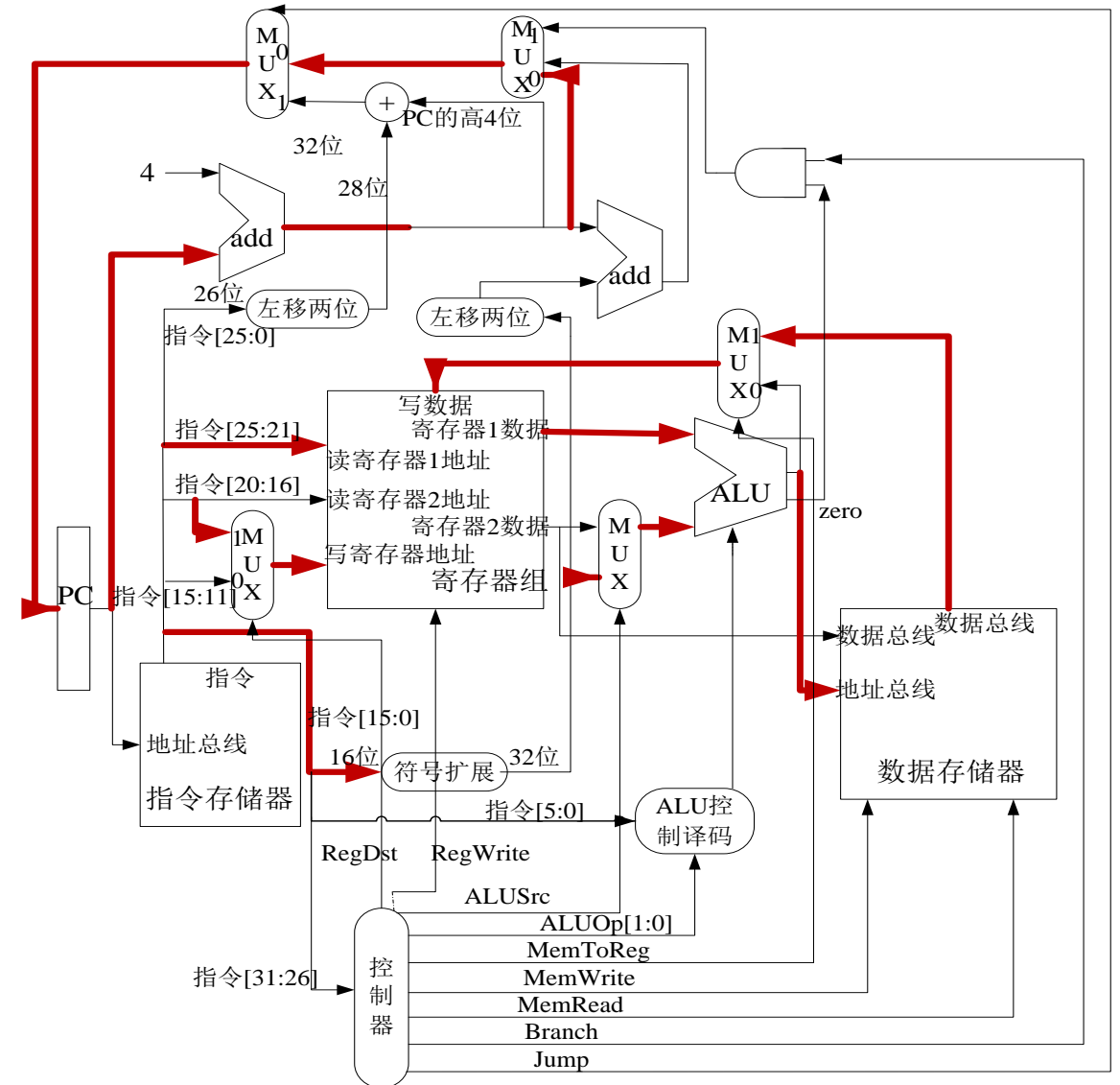
► 不同指令执行过程描述

-
- Diagram illustrating the input vector for the SHA-256 compression function. The input is a 31-bit vector, partitioned into six fields: Op^6 (6 bits), Rs^5 (5 bits), Rt^5 (5 bits), Rd^5 (5 bits), sha^5 (5 bits), and $func^6$ (6 bits). The bits are represented by red circles, with some circles containing '1' and others empty.



► 不同指令执行过程描述

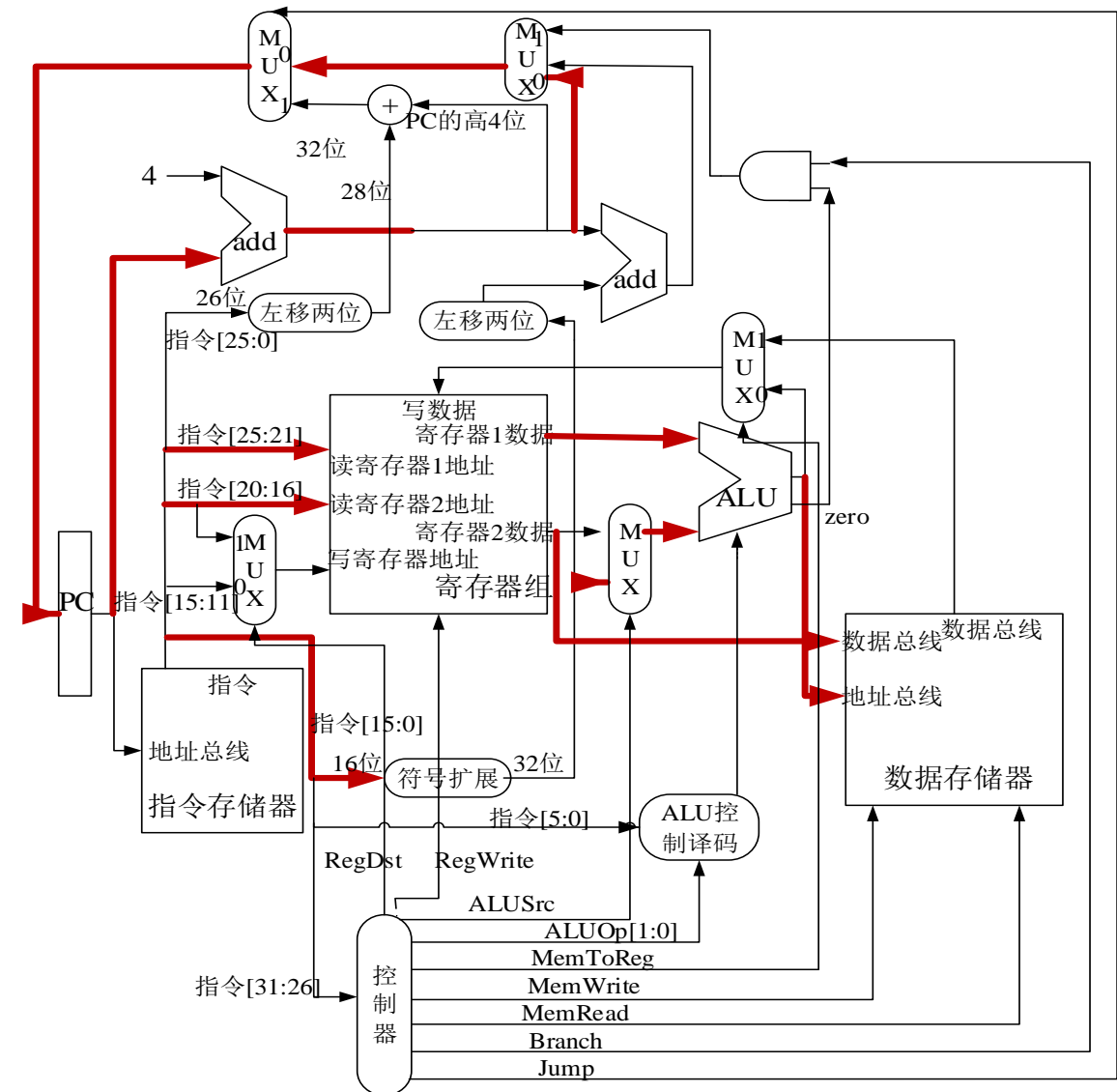
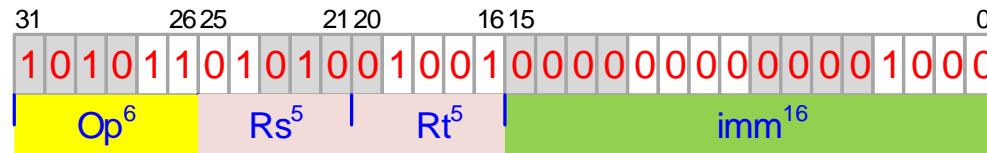
- 同：lw \$9, 8(\$10)
- 机器码：0x8d490008



3.4 控制器实现原理

► 不同指令执行过程描述

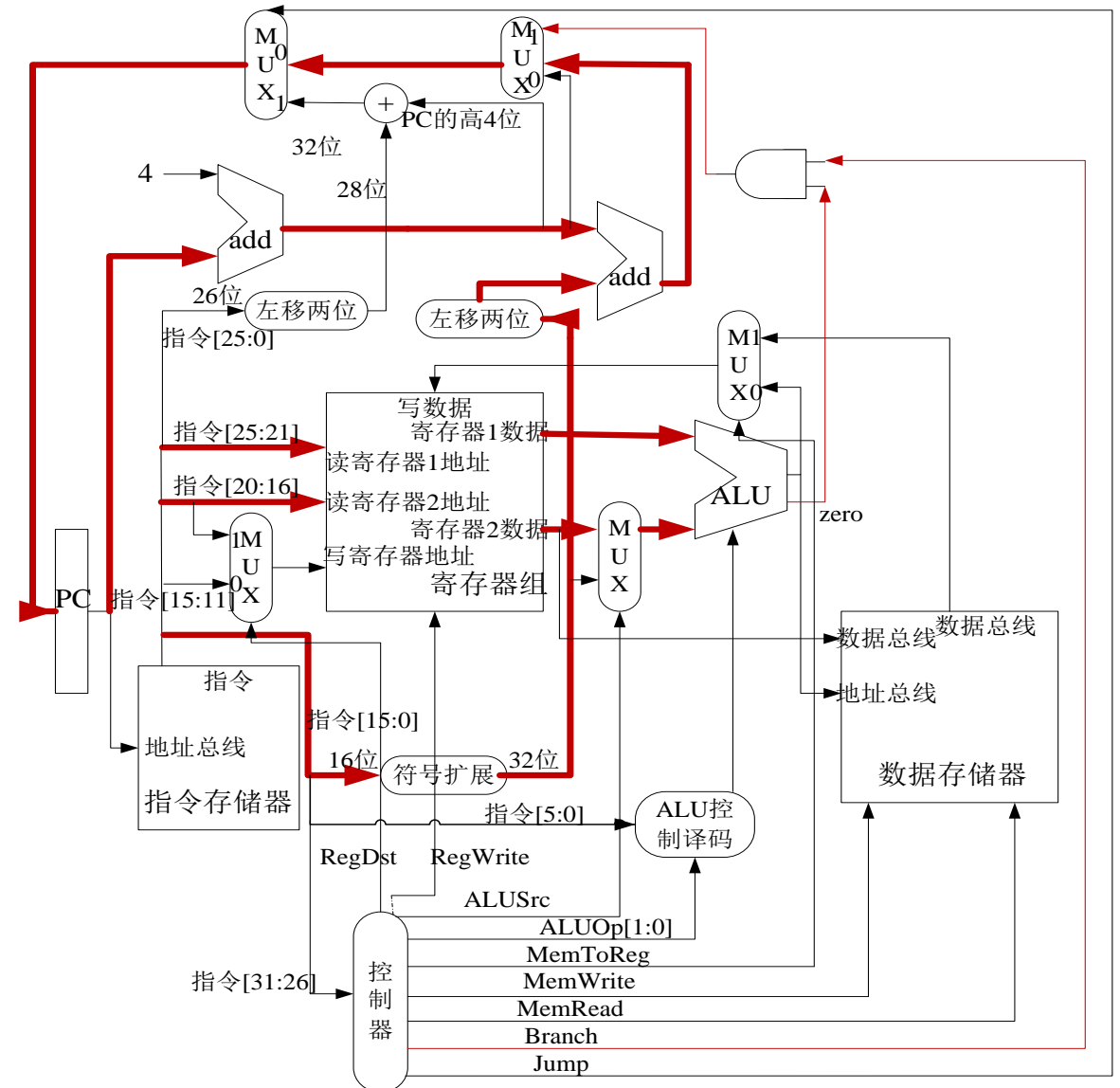
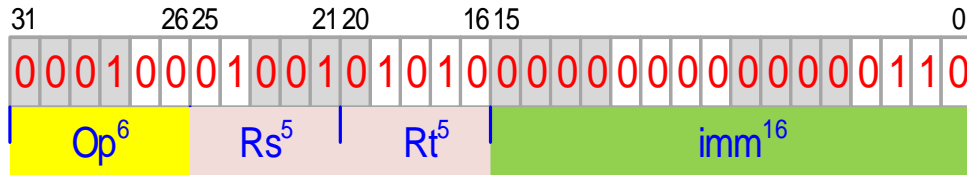
- I型指令 `sw $t1, 8($t2)`
 - 同： `sw $9, 8($10)`
 - 机器码： `0xad490008`



3.4 控制器实现原理

► 不同指令执行过程描述

- I型指令 beq \$t1, \$t2, L0
 - 同： beq \$9, \$10, L0
 - 机器码：0x112a0006

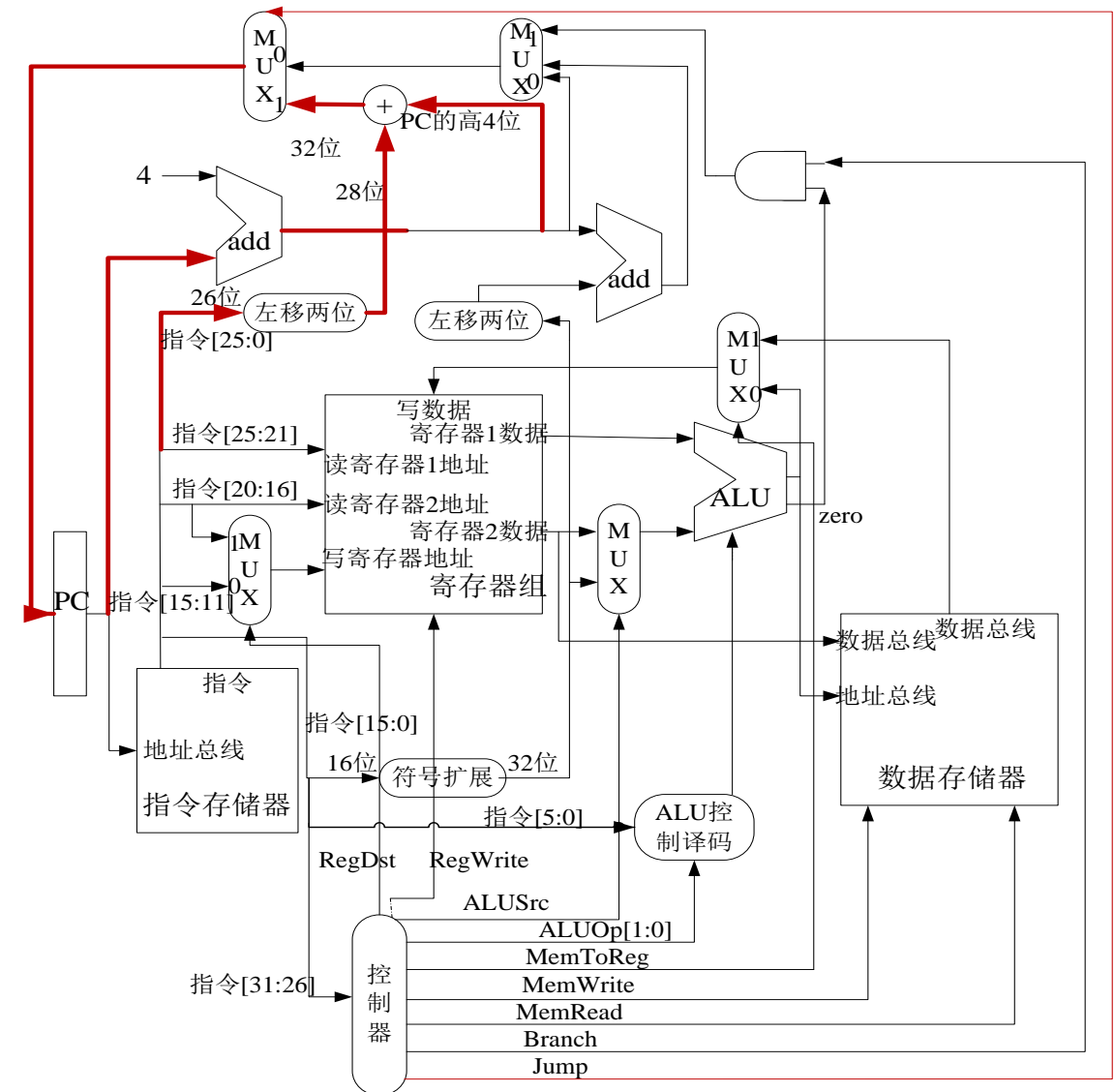
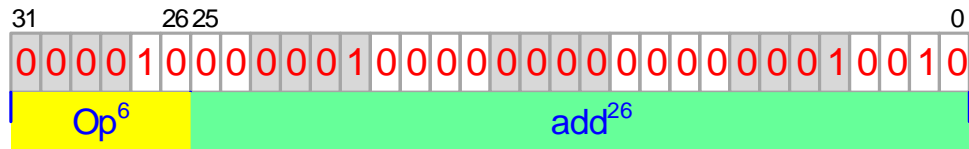


3.4 控制器实现原理

► 不同指令执行过程描述

• J型指令 j L0

- 机器码：0x08100012



▶ 第三章作业（一）

- 第二版P125：1、2、3

▶ 要求

- 微助教提交
- 下次课之前提交

► 内容

- 微处理器的基本构成
- 简单MIPS微处理器各部件原理及设计
- 现代微处理器的流水线技术原理
- 现代微处理器的超标量技术原理
- 微处理器异常处理机制和外部接口
- MicroBlaze微处理器简介

► 目标

- 理解处理器的基本操作、基本构成部件
- 能用Verilog语言设计简单MIPS微处理器
- 理解解现代微处理器设计的新技术
- 了解微处理器异常处理机制和外部接口
- 了解MicroBlaze微处理器特点

3.5 现代微处理器新技术

- ▶ 流水线技术
- ▶ 超标量技术

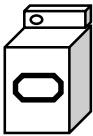



▶ 流水线技术

- 3级流水线例子

- 3级

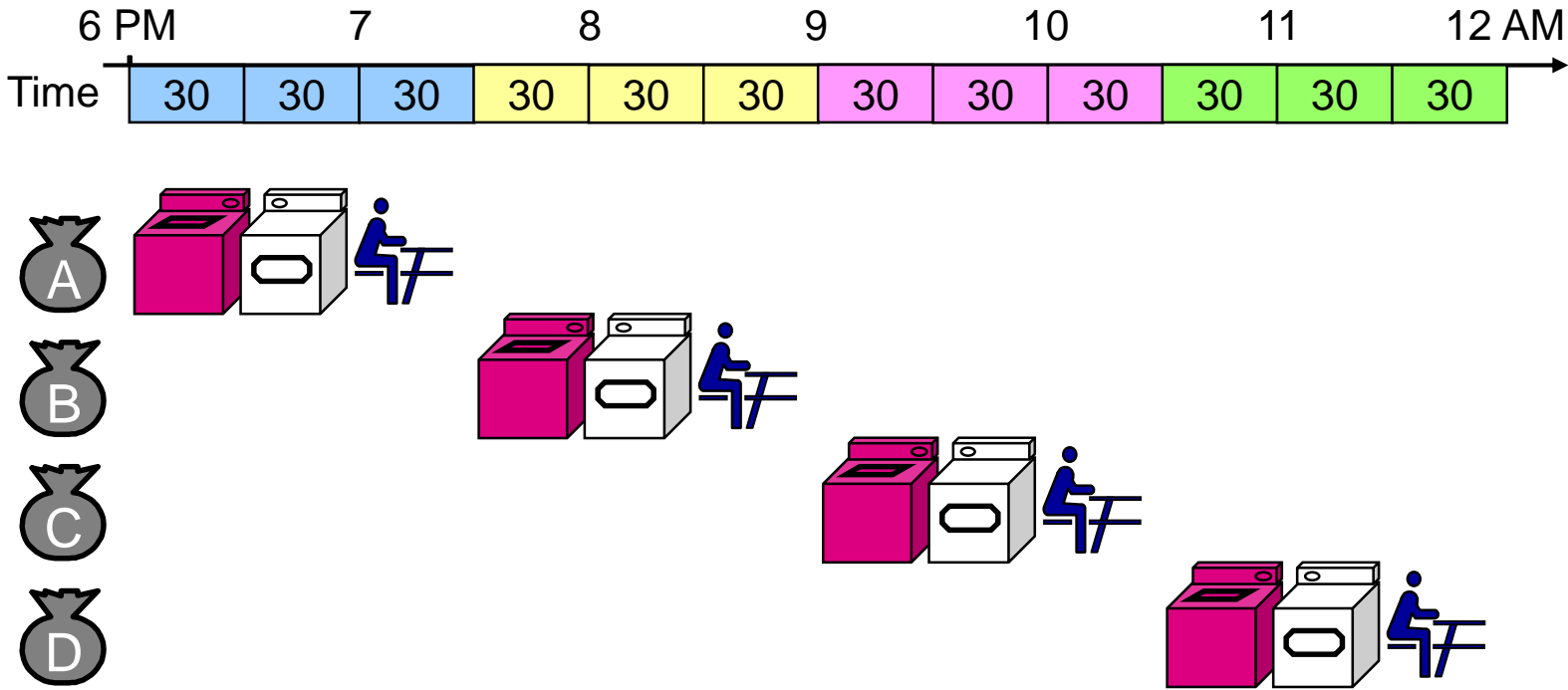
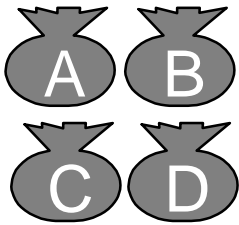
- 洗衣：

- 烘干：

- 整理：

- 每级流水耗时30min

- 共有4套衣服



顺序执行，4套衣服处理完，需要6小时



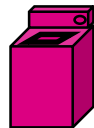
3.5 现代微处理器新技术

► 流水线技术

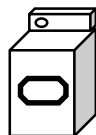
- 3级流水线例子

- 3级

- 洗衣：



- 烘干：

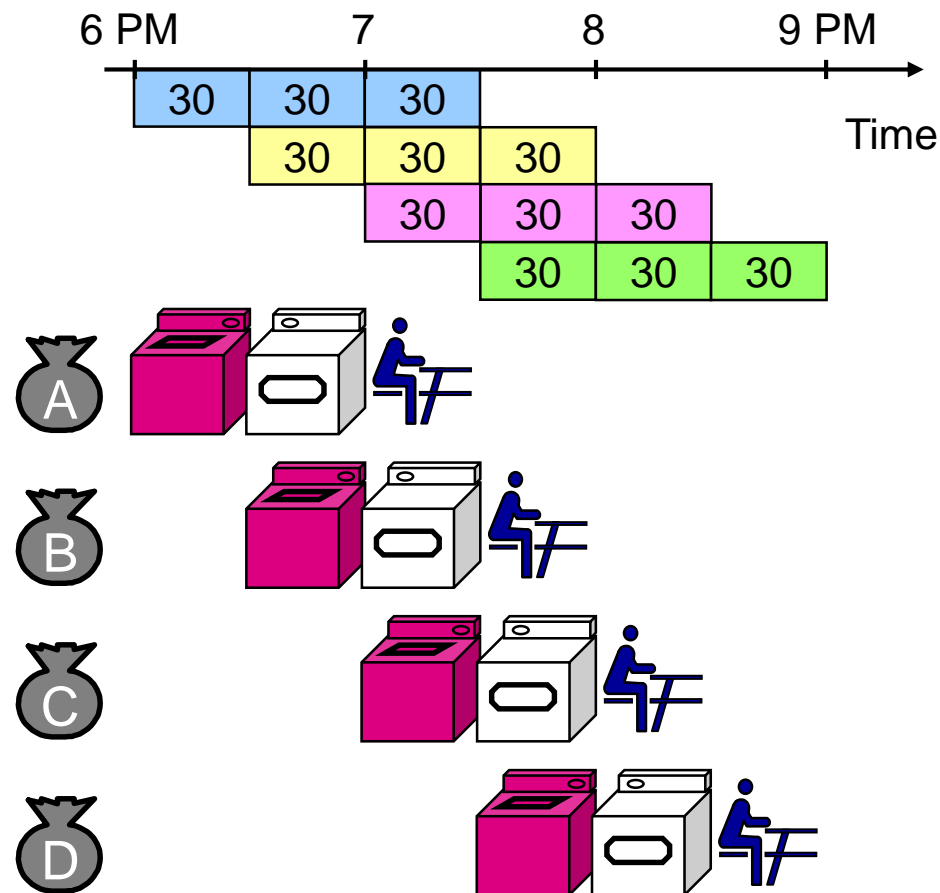
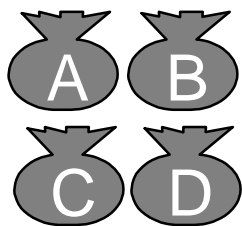


- 整理：



- 每级流水耗时30min

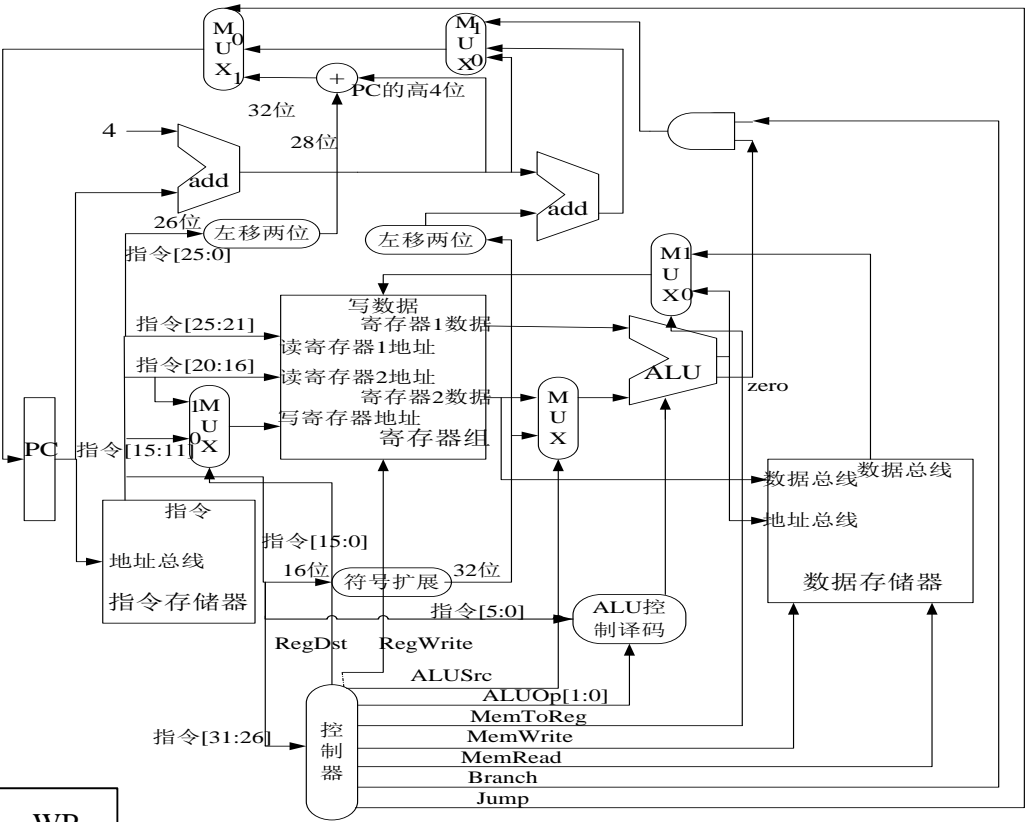
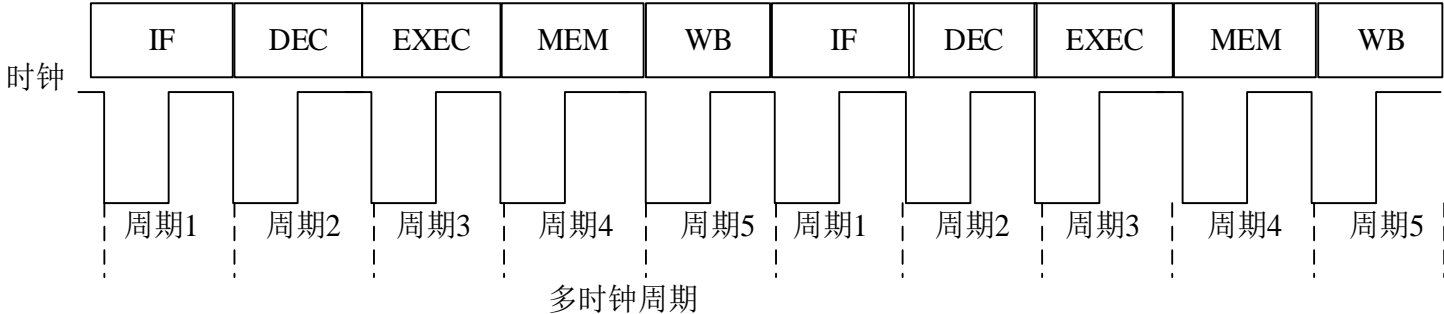
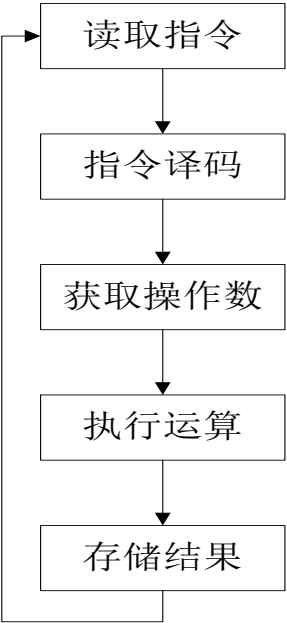
- 共有4套衣服



并行执行，4套衣服处理完，需要3小时

3.5 现代微处理器新技术

► 流水线技术

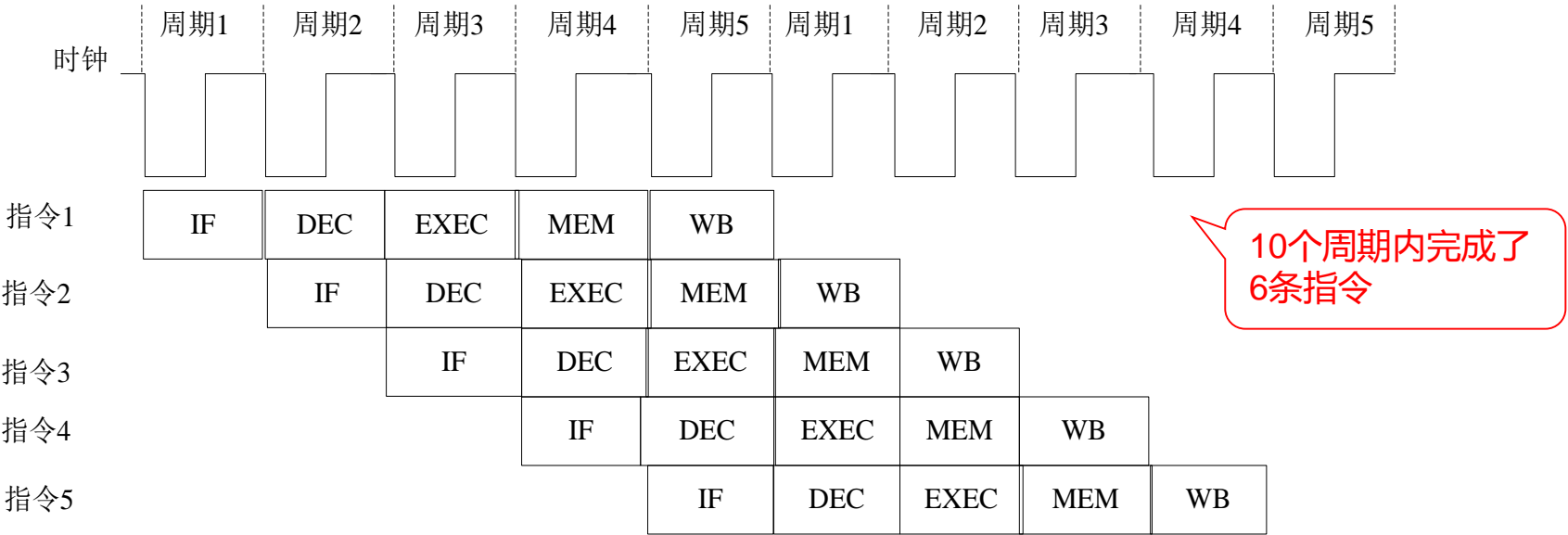


10个周期内仅完成了2条指令



▶ 流水线技术

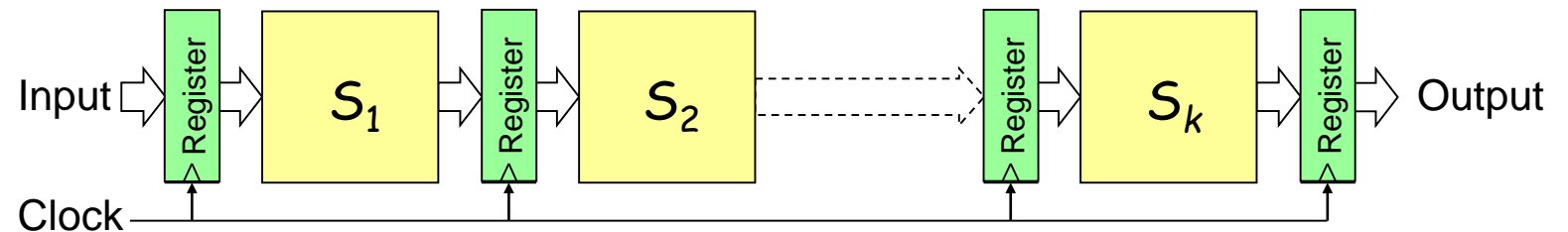
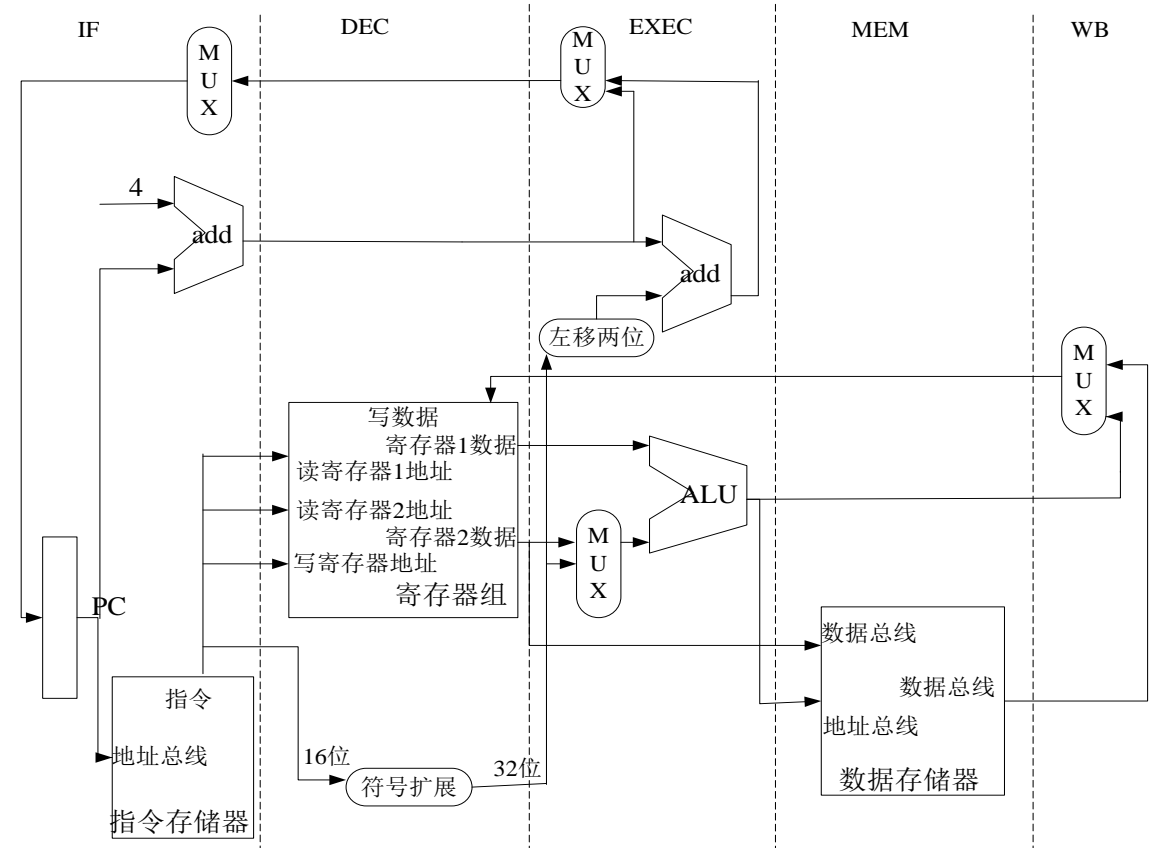
- 功能分解，空间上顺序依次进行，时间上重叠并行。
 - 5个部分设计为相对独立部件
 - 流水线具有多少个不同的执行部件，则称这条流水线具有多少级
 - 在同一条指令中顺序执行
 - 在不同指令中并行执行



3.5 现代微处理器新技术

► 流水线技术

- 数据通路的5级流水线划分
 - 每两个流水线部件之间必须增加器件来实现流水线不同部件之间的接口。
 - Uses **clocked registers** between stages
 - Upon arrival of a clock edge ...
 - All registers hold the results of previous stages simultaneously
 - The pipeline stages are **combinational logic circuits**
 - It is desirable to have **balanced stages**
 - Approximately equal delay in all stages
 - Clock period is determined by the **maximum stage delay**



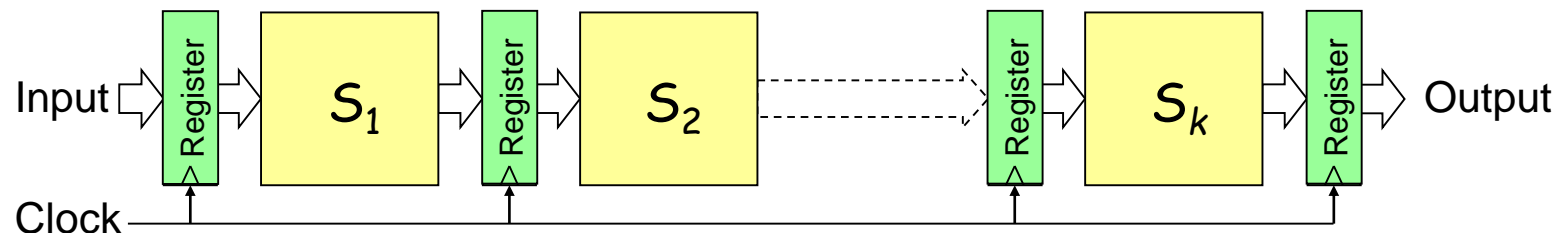
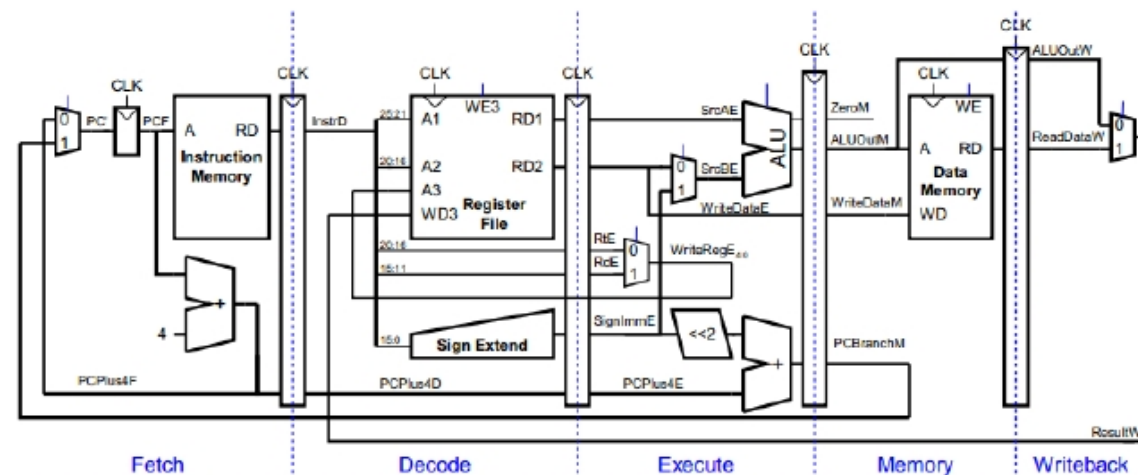
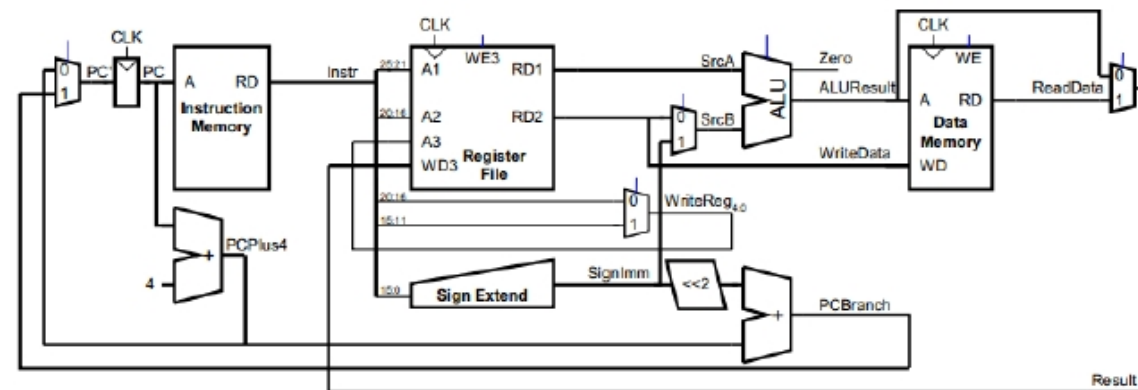
3.5 现代微处理器新技术

► 流水线技术

- 数据通路的5级流水线划分

- 每两个流水线部件之间必须增加器件来实现流水线不同部件之间的接口。

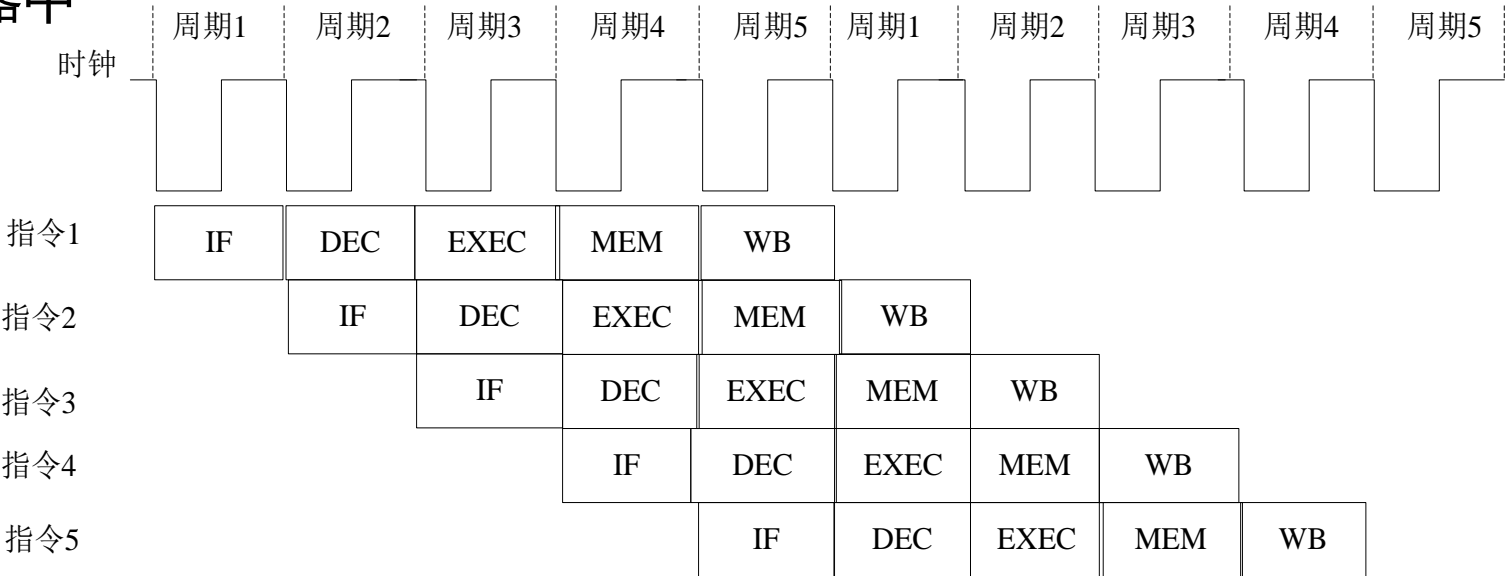
- Uses **clocked registers** between stages
- Upon arrival of a clock edge ...
- All registers hold the results of previous stages simultaneously
- The pipeline stages are **combinational logic circuits**
- It is desirable to have **balanced stages**
- Approximately equal delay in all stages
- Clock period is determined by the **maximum stage delay**



► 流水线技术

- 不足

- 流水线处理器中增加少量的器件就可以较大幅度提高处理器执行效率，但是对于下列问题，不能从根本上解决执行效率问题
 - 结构相关：由于硬件资源不足而导致流水线不通畅
 - 数据相关：由于指令的源操作数依赖其它指令的计算结果而导致读数据不正确
 - 转移相关：由于流水线操作而导致在转移发生之前，若干条转移指令的后继指令已被取到流水线处理器中



3.5 现代微处理器新技术

► 超标量技术

- 微处理器集成多个ALU、多个译码器和多条流水线，以并行处理的方式来提高性能
- 流水线处理器在每个时钟周期最多发出一条指令，而超标量处理器可以在每个时钟周期发出1~8条指令。同时发出的指令必须是不相关的并且不能出现资源冲突。
 - 假设处理器有一个整数部件和一个浮点部件，处理器至多能发出两条指令，一条是整数类型的指令，包括整数算术逻辑运算，存储器访问操作和转移指令；另一条必须是浮点类型的指令
- 指令调度策略是需要考虑的问题
 - 通常在处理器中会设计一个指令缓冲区，用来存放等待调度的指令。



3.6 微处理器异常处理原理

► 异常处理机制

• 异常事件

异常种类	来源	MIPS处理器命名
I/O设备	外部	中断
用户程序唤醒操作系统	内部	异常
计算结果溢出	内部	异常
未定义的指令（非法指令）	内部	异常
硬件出错	两者	异常或中断

• 异常处理

- 计算机在正常运行过程中，由于种种原因，使CPU暂时停止当前程序的执行，转而去处理临时发生的异常事件，处理完毕后，再返回去继续执行暂停的程序。
- 微处理器要能够实现异常处理需要完成以下几方面的功能：
 - 记录异常发生的原因
 - 记录程序断点处的指令在存储器中的地址
 - 记录不同种类的异常处理程序在内存中的地址
 - 建立异常种类与异常处理程序地址之间的对应关系。

才能在遇到某个特定的异常事件时去执行相应的异常处理程序。

3.6 微处理器异常处理原理

► 异常事件识别机制

- 状态位法 (MIPS) :
 - 在微处理器中利用一个寄存器对每种异常事件确定一个标志位，当有异常事件发生时，寄存器中对应的位置1，一个32位的寄存器可以表示32种不同类型的异常事件
- 向量法(Intel) :
 - 对不同类型的异常事件进行编码，这个编码叫中断类型码或异常类型码。

► 断点保存和返回

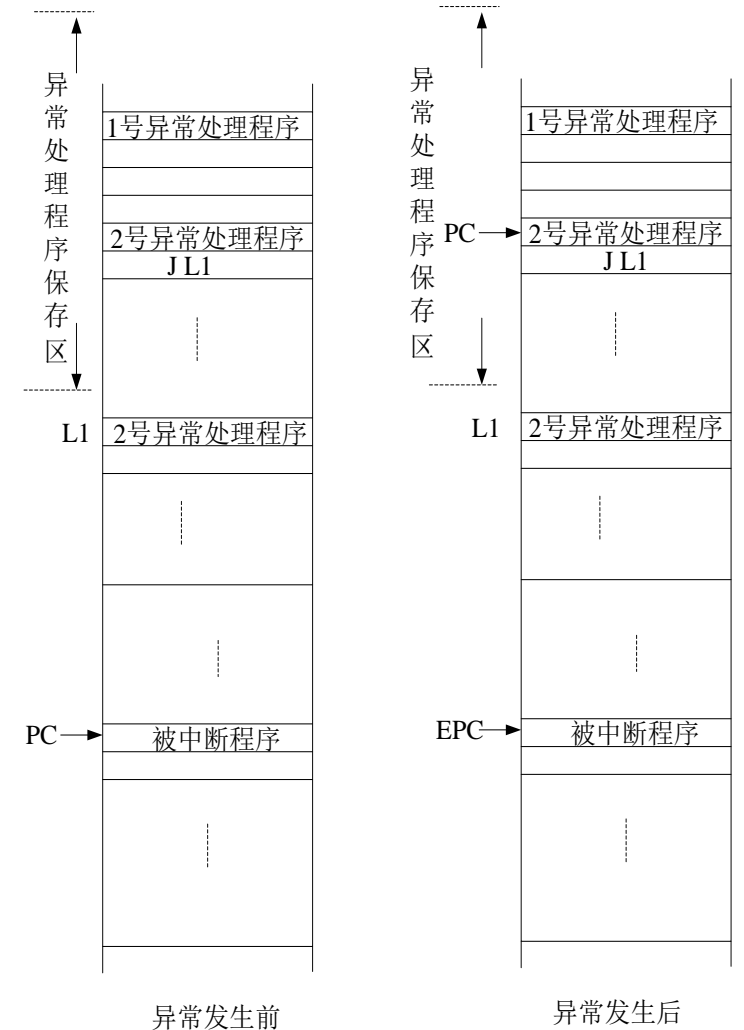
- 寄存器法 (嵌入式)
 - 在微处理器中设计一个寄存器EPC，当微处理器出现异常时，就将PC的值保存到EPC中。异常处理完之后，再把EPC的值赋给PC，这样就可以实现中断的返回
- 栈 (PC)
 - 微处理器直接将PC的值压入栈中，异常处理完之后，再从栈顶把值弹出来赋给PC



3.6 微处理器异常处理原理

► 异常处理程序进入方式

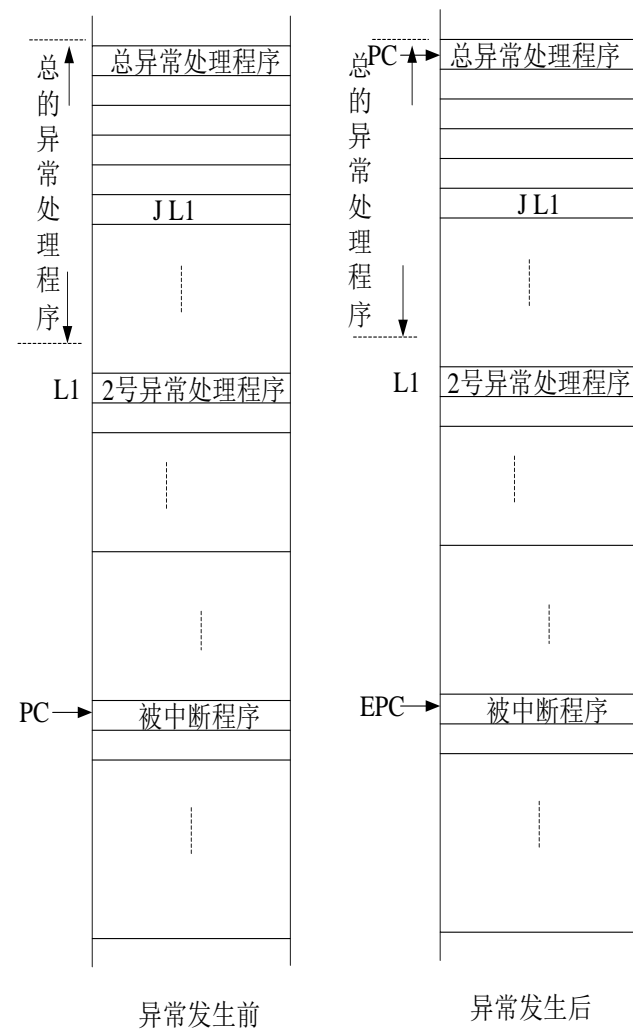
- 1) 专门的内存区域保存异常处理程序
 - 在这块内存区域中为**每个异常处理程序分配固定长度的空间**如32个字节或8条指令长度的空间，而且针对每个异常事件其异常处理程序的**存放地址是固定的**。
- 2) 仅提供一个异常处理程序存放地址
- 3) 分配一块专门的内存区域保存异常处理程序的入口地址



3.6 微处理器异常处理原理

► 异常处理程序进入方式

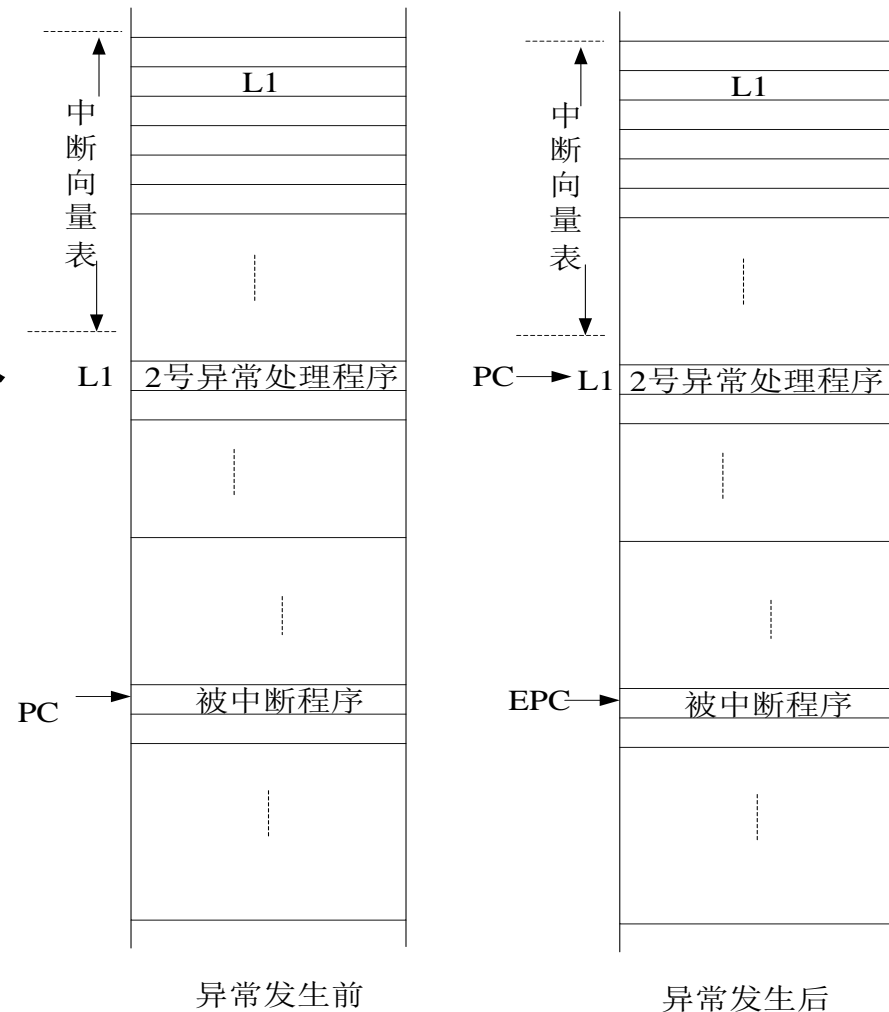
- 1) 专门的内存区域保存异常处理程序
- 2) 仅提供一个异常处理程序存放地址
 - 发生任何异常事件都首先转移到该地址执行总的异常处理，并在总异常处理程序中分析异常事件的原因，然后再根据异常的原因通过子程序调用的方式去执行相应的异常处理。
- 3) 分配一块专门的内存区域保存异常处理程序的入口地址



3.6 微处理器异常处理原理

► 异常处理程序进入方式

- 1) 专门的内存区域保存异常处理程序
- 2) 仅提供一个异常处理程序存放地址
- 3) 分配一块专门的内存区域保存异常处理程序的入口地址
 - 异常处理程序的入口地址叫中断向量
 - 保存异常处理程序的入口地址的内存区域叫做中断向量表
 - 异常处理程序可以存放在内存中的任意位置，只需要把该异常处理程序的入口地址保存到中断向量表中正确的地址中，当异常发生时，微处理器就可以通过中断向量表查找到中断服务程序的入口地址。



3.7 微处理器外部接口

► 微处理器必须具有总线接口

- 访问外部存储器

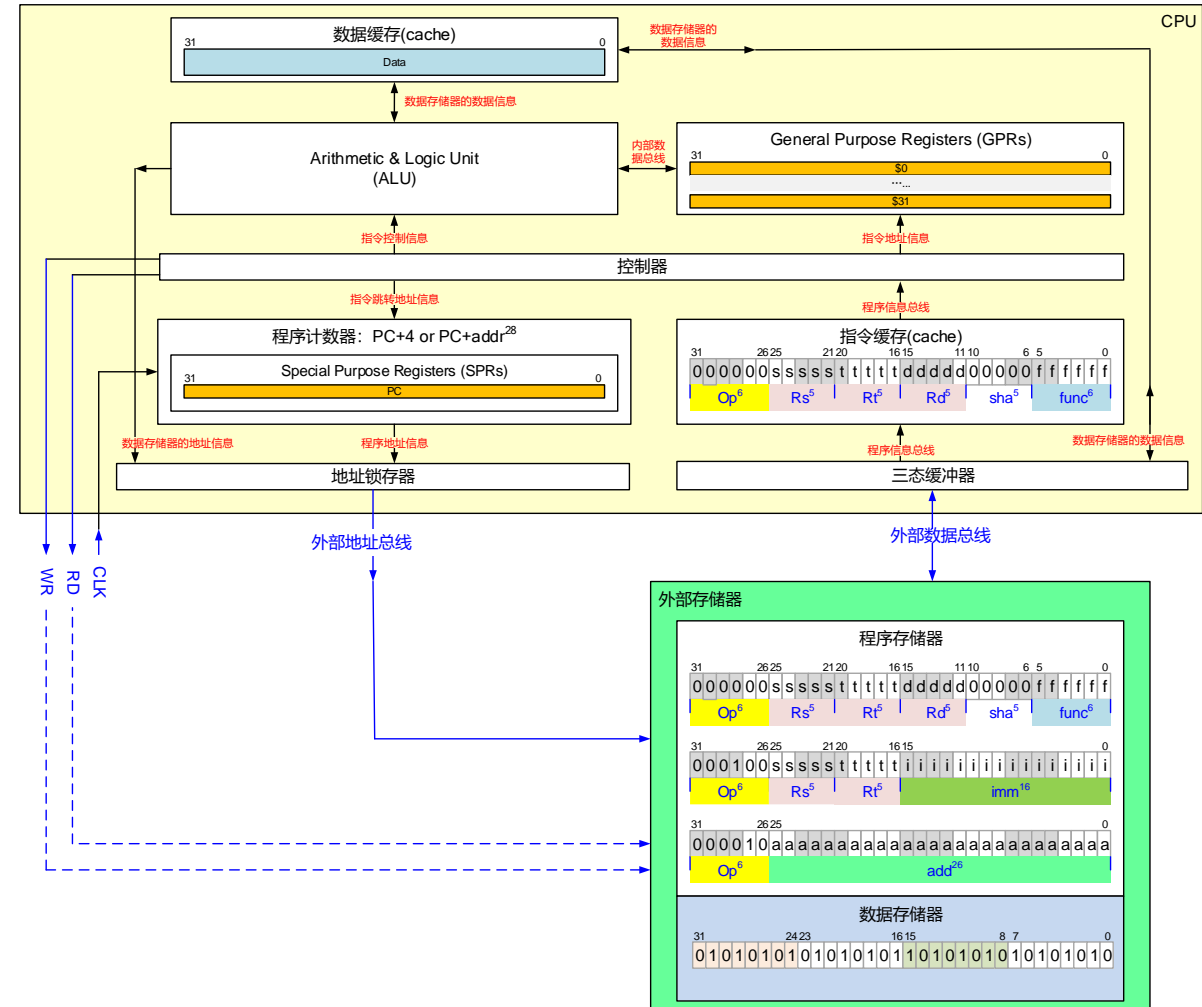
- 程序存储器
- 数据存储器

- 访问外部设备

- 键盘
- 鼠标
- 声卡
- 摄像头
- ...

- 三总线

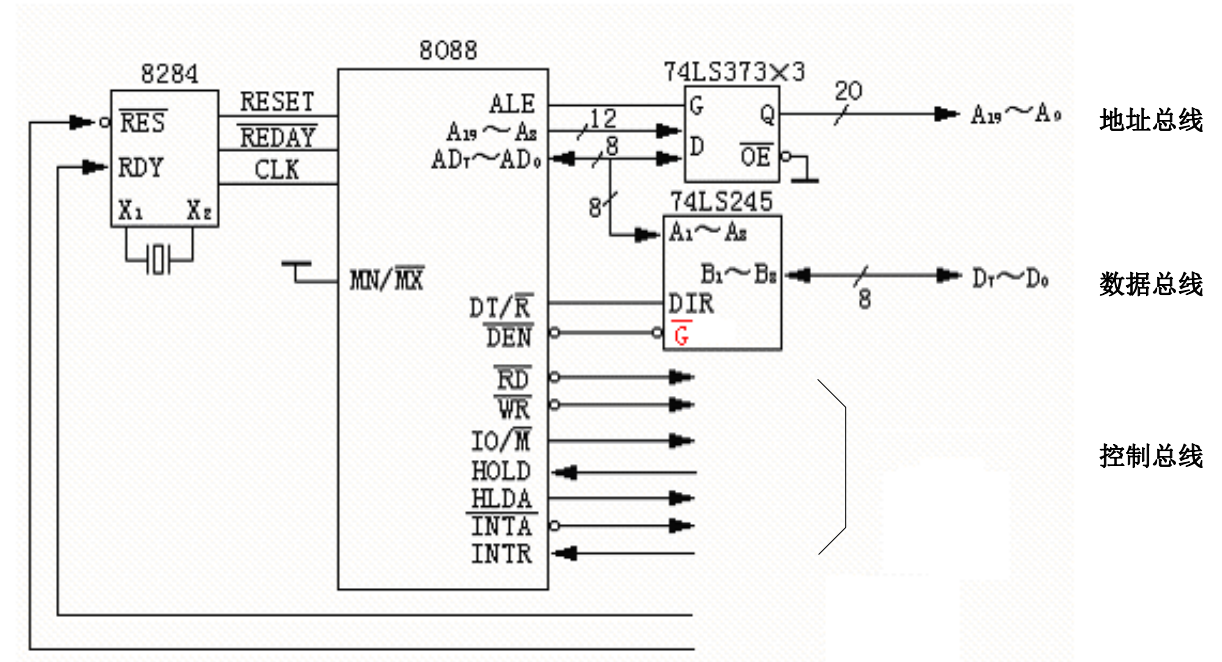
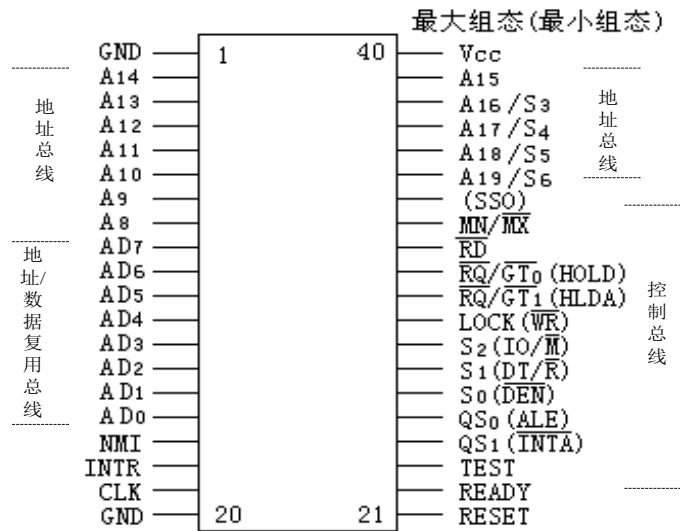
- 地址
- 数据
- 控制



3.7 微处理器外部接口

► 8088微处理器的外部接口

- 外部接口通过三总线体现

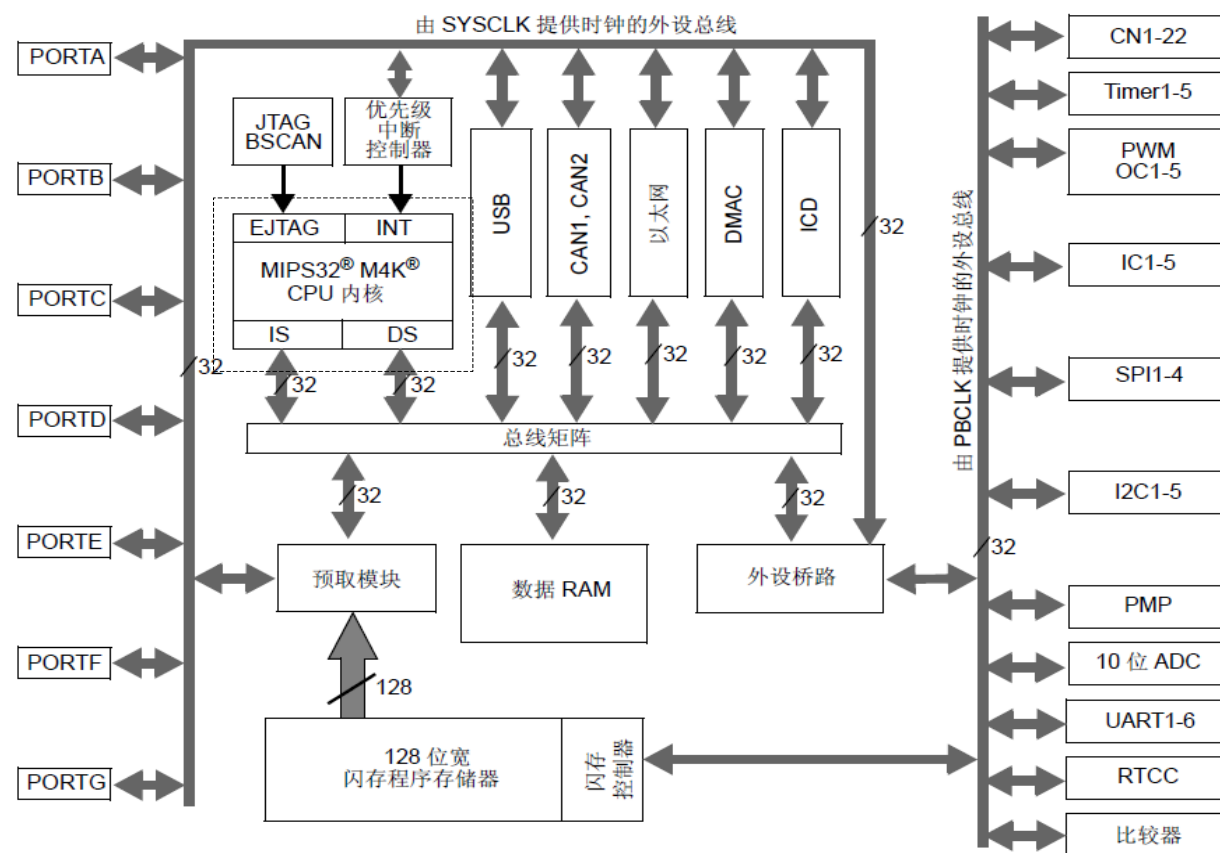


- 为便于用户接口设计，利用373/245等芯片，分离出来独立的三总线

3.7 微处理器外部接口

► 嵌入式芯片PIC32MX5XX/6XX/7XX系列框图

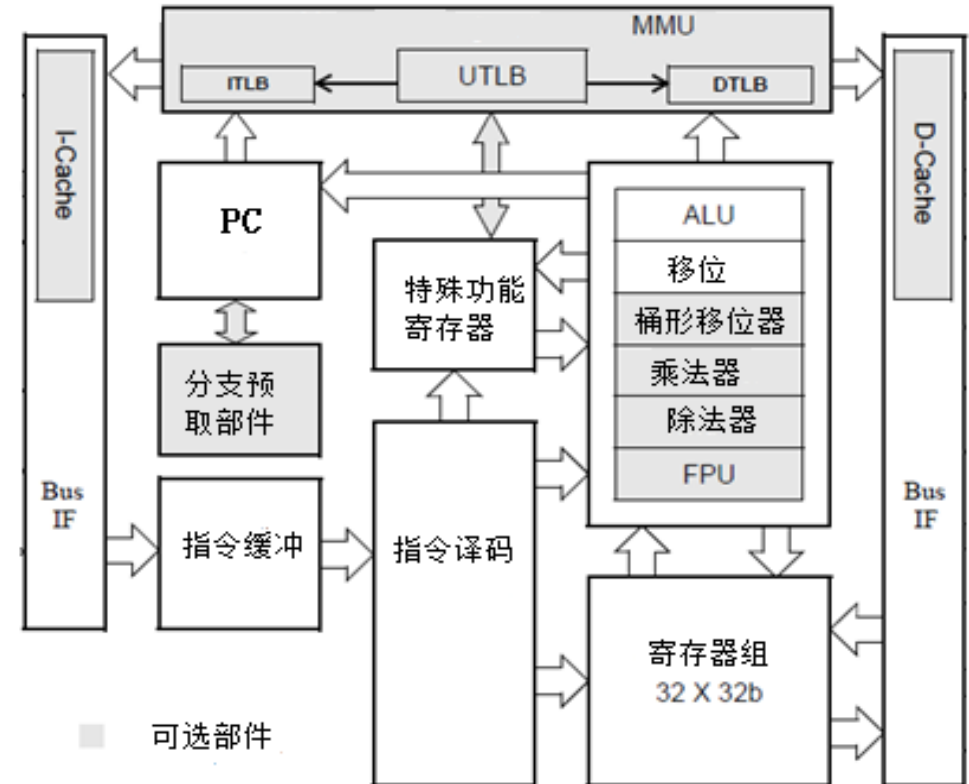
- 为降低用户设计复杂接口电路的难度，嵌入式CPU将大部分接口电路都集成到芯片内部，用户**直接使用接口模块**，**重点关注这些模块的使用**，而非接口的设计。



3.8 MicroBlaze微处理器简介

► MicroBlaze微处理器

- Xilinx公司基于FPGA开发的一个软核类MIPS指令集微处理器
- 包含简单MIPS微处理器的各个部件，支持32位的数据总线和32位的地址总线
- 可以配置为支持大字节序或小字节序，
 - 当采用PLB外部总线时为大字节序，
 - 采用AXI4外部总线时为小字节序。
- 支持三级或五级流水线，
- 具有可选的指令和数据cache，
- 支持虚拟内存管理。
- 支持通过PLB、LMB、AXI总线与外围接口或部件相连。
- 具有32个32位的通用寄存器，使用规则与MIPS微处理器的通用寄存器使用规则相同，命名为R0~R31。
 - R14，R15，R16，R17又用做异常返回地址寄存器，
 - 还具有18个32位的特殊功能寄存器，包括PC，MSR等

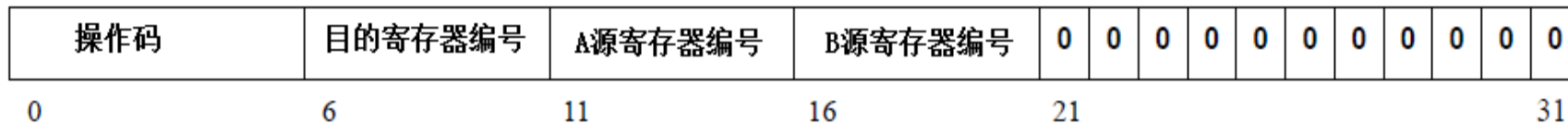


3.8 MicroBlaze微处理器简介

► MicroBlaze微处理器

- MicroBlaze指令架构

- **A型指令**：相当于MIPS中的R型指令



- **B型指令**：相当于MIPS中的I型指令



- 将MIPS中的J型指令合并到B型指令中。



► 作业（二）

- 第二版P126：5
- 要求
 - 微助教提交
 - 下次课之前提交

► 作业（三）

- 第二版P126：7
- 要求
 - 思考，自己完成
 - 完成数据通路和控制器各个独立模块的仿真，汇编程序到COE文件制作
 - 后续实验课验证

Thanks

