计算机组成原理与系统结构

第七章 控制器

http://jpkc.hdu.edu.cn/computer/zcyl/dzkjdx/

杭州電子科技大学
HANGZHOU DIANZI UNIVERSITY

第七章　控制器

7.1 控制器的组成及指令的执行

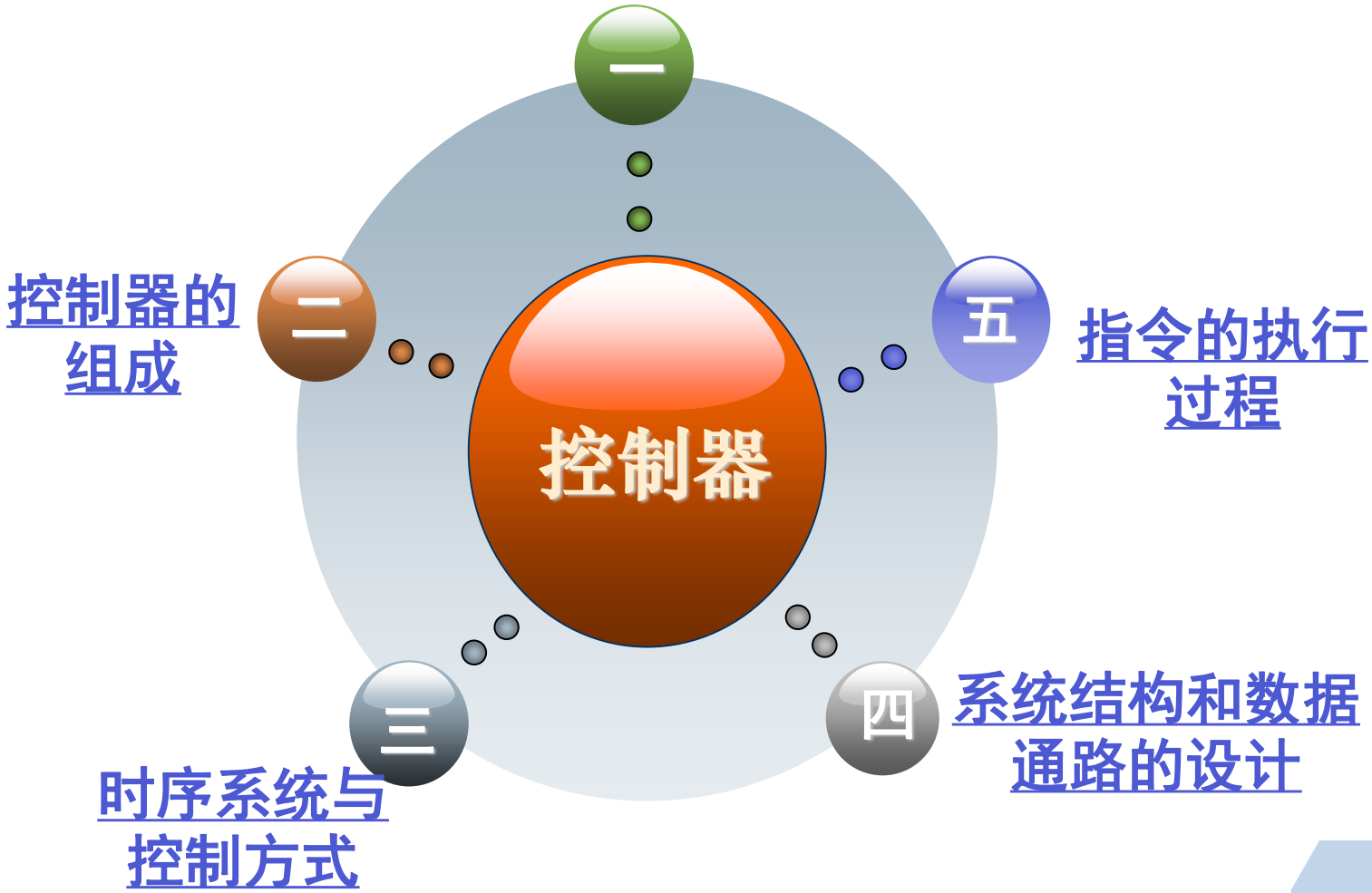
7.2 硬布线控制器

7.3 微程序控制器

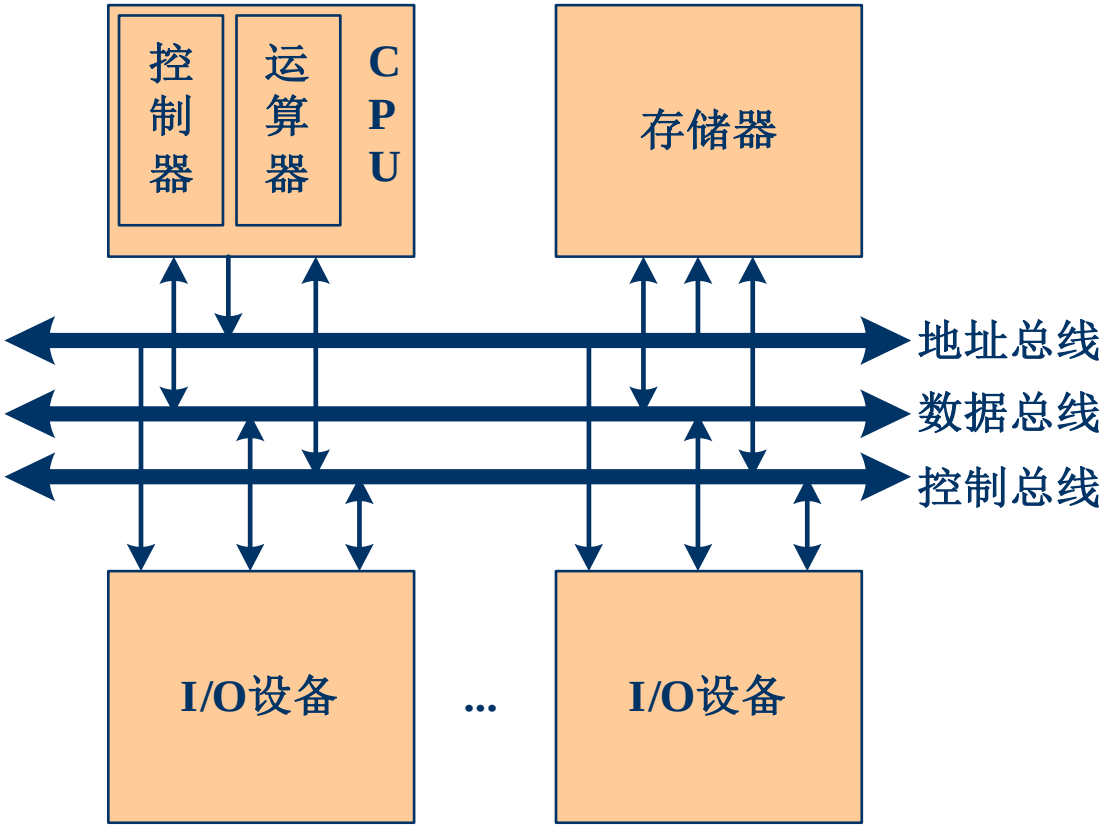
本章小结

BACK

7．1控制器的组成及指令的执行



一、基本的计算机组成和功能



- AB：CPU或总线主设备→存储器或IO设备（单向）
- DB：各部件之间（双向）
- CB：包含许多不同的控制信号线和状态信号线（单/双向））

CPU的基本功能

① **指令控制：**确保计算机指令按程序的顺序执行。

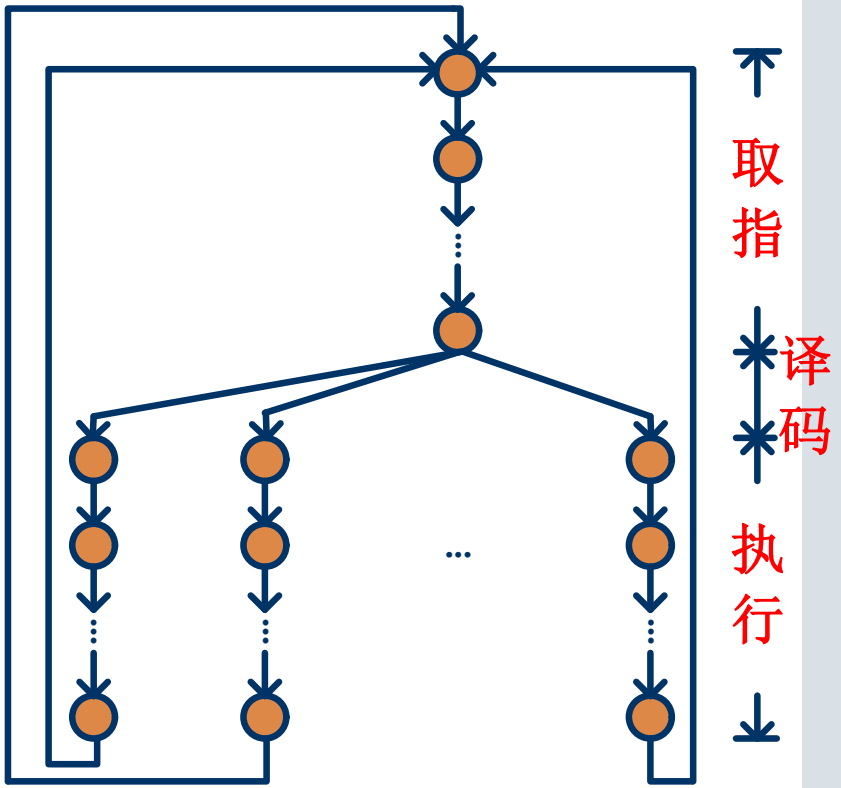
② **操作控制：**一条指令的功能通常有若干个操作信号（微操作）组合起来实现，CPU控制这些微操作的产生、组合、传送和管理。

③ **时间控制：**使各种微操作和指令的执行严格按照时间序列进行。

④ **数据加工：**由运算器对数据进行算术运算和逻辑运算。

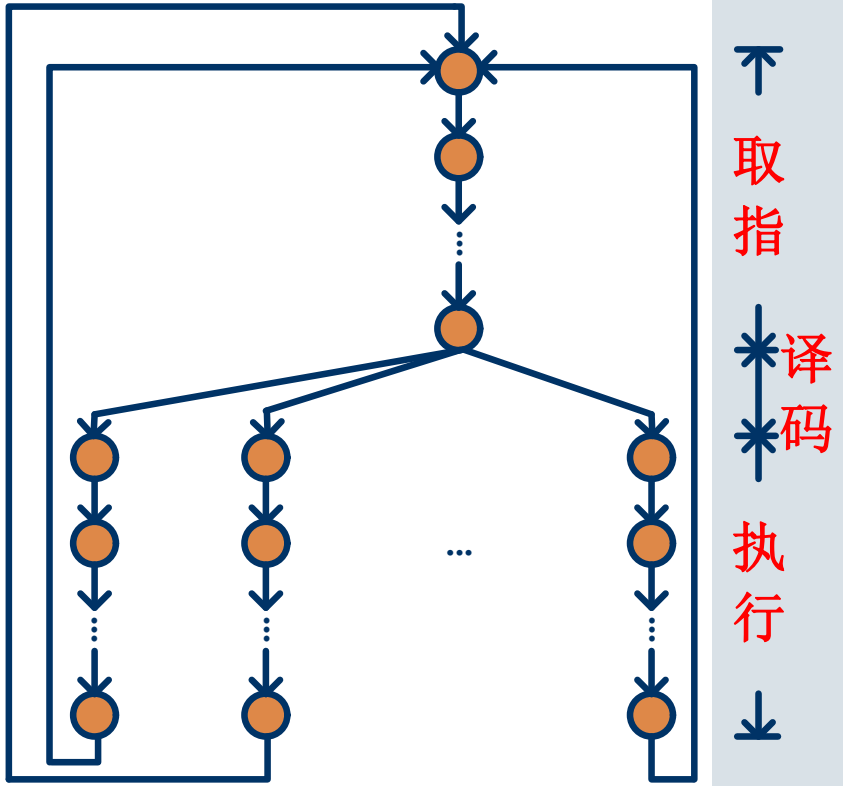
一、基本的计算机组成和功能

- CPU：实际上就是一个复杂的有限状态机。
- 设计CPU的过程：就是确定它所具有的所有状态，及其对应的微操作
- 确定的依据：指令系统



一、基本的计算机组成和功能

- 指令的执行可分为：
  - **取指令周期**：从存储器中取出一条指令，并对该指令的操作码译码；
  - **执行周期**：执行该指令。
- **译码可能并不对应任何状态**，它只是取指令结束后到各条指令的执行周期之间的一个多路选择。



二、控制器的组成

- **控制器的功能**：从存储器中**取指令**、对**指令译码**、产生控制信号并控制计算机系统各部件有序地执行，从而**实现这条指令的功能**。
- 具体：

  ①取指令

  ②分析指令

  ③执行指令

  ④中断处理和响应特殊请求

二、控制器的组成

❖控制器的组成：

1.专用寄存器：

①程序计数器（PC）：存放指令地址

- 顺序执行时，由PC+1产生下一条指令的地址
- 遇到转移指令时，转移地址→PC作为下一条指令的地址。

②指令寄存器（IR）：存放机器指令码

③地址寄存器（AR）：用于存放CPU访问存储器或者IO设备的地址码。

④数据寄存器（DR）：用于存放CPU访问存储器或者IO设备的数据。

二、控制器的组成

2. **指令译码器：** 对指令的操作码进行译码，以识别该指令所要求的操作。

3. **操作控制信号形成部件：** 根据指令的操作码以及时序信号，产生取出指令和执行这条指令所需的各种操作控制信号，以便正确地建立数据通路，完成取出指令和执行指令的控制。分为：

   - **硬布线控制器：** 主要由组合逻辑电路构成
   - **微程序控制器：** 主要由存储逻辑电路构成

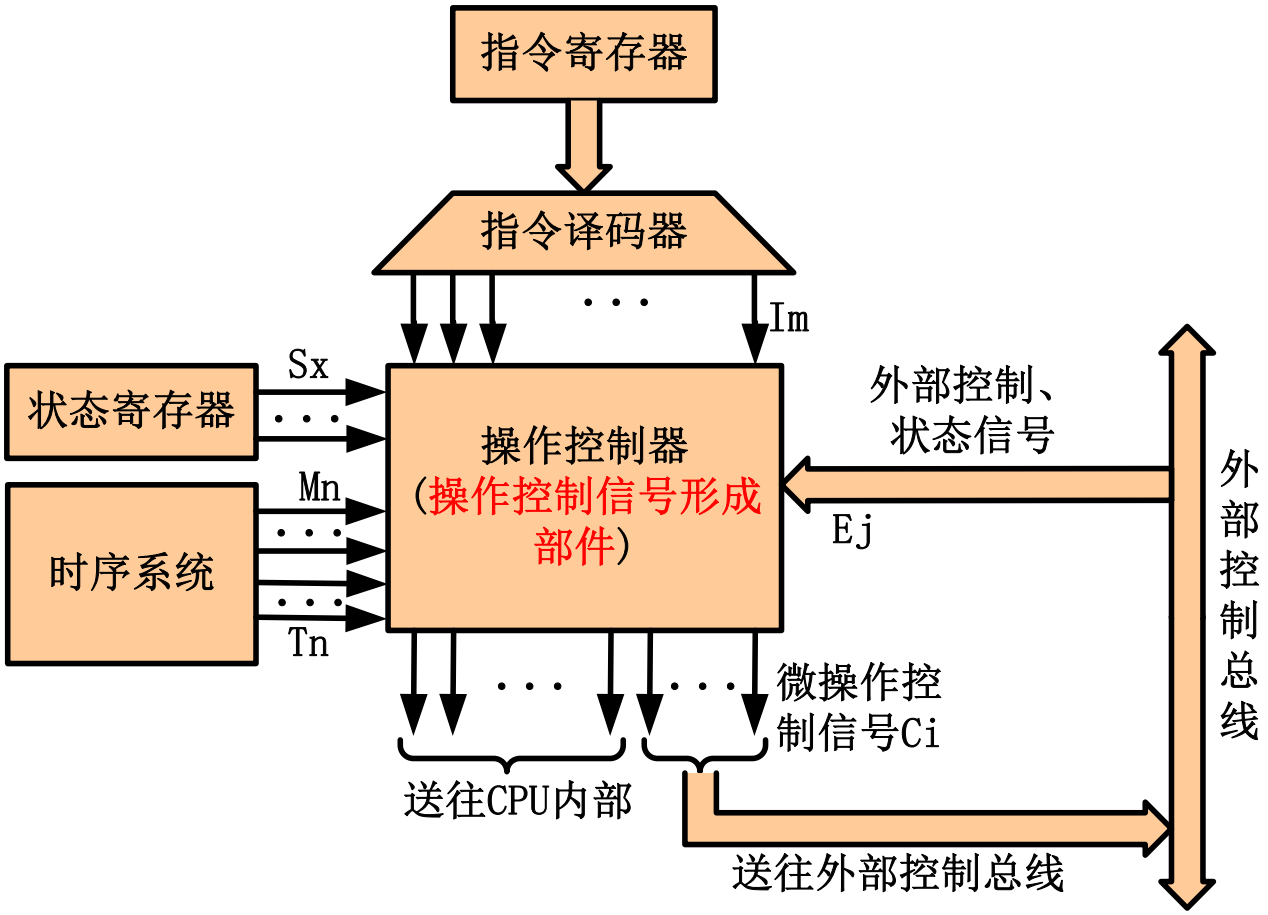
二、控制器的组成

4. **时序信号产生器：**负责提供时钟信号和机器周期信号，以规定每个操作的时间。包括：

- **脉冲源：**产生一定频率的脉冲信号，为整个计算机提供基准时钟信号。
- **启停线路：**负责控制时钟脉冲的送出与封锁，从而实现计算机的启动与停止。
- **时序信号产生及其控制部件：**以脉冲源为基准，产生不同的计算机所对应的多级时序信号，用以控制计算机的每一步微操作。

二、控制器的组成

控制器的结构框图



三、时序系统与控制方式

1

2

3

计算机中的时序信号

时序系统

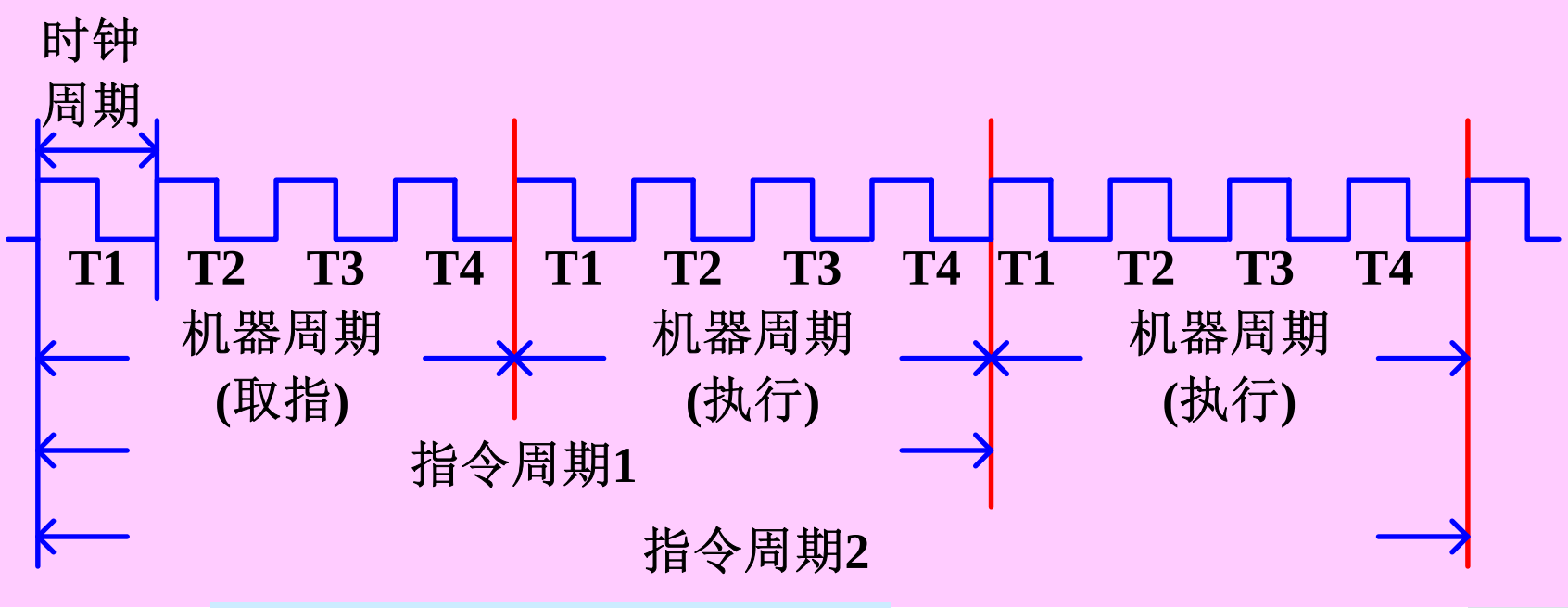
控制方式

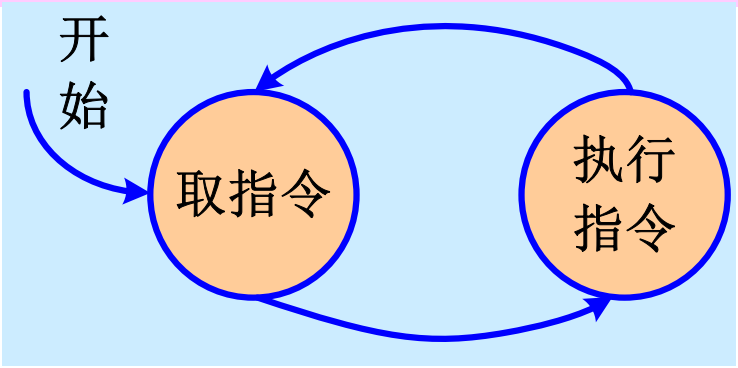
1、计算机中的时序信号

- **指令周期**：是指计算机从内存取出一条指令并完成该指令的执行所需要的时间。
  - 不同指令的指令周期是不相同的。
  - 一个指令周期可能由若干个机器周期组成。
- **机器周期**：又称为**CPU周期**，用于完成1次内存的读或写操作，或者1次ALU的运算，或者1次总线传送
  - 一般规定为CPU与内存交换1次信息（读或写内存）所需要的时间。
  - 一个机器周期的功能需要多个时钟周期完成。
- **时钟周期**：又称为**节拍**，是指CPU执行一个微操作命令（即控制信号）的最小时间单位，也即T周期。

1、计算机中的时序信号

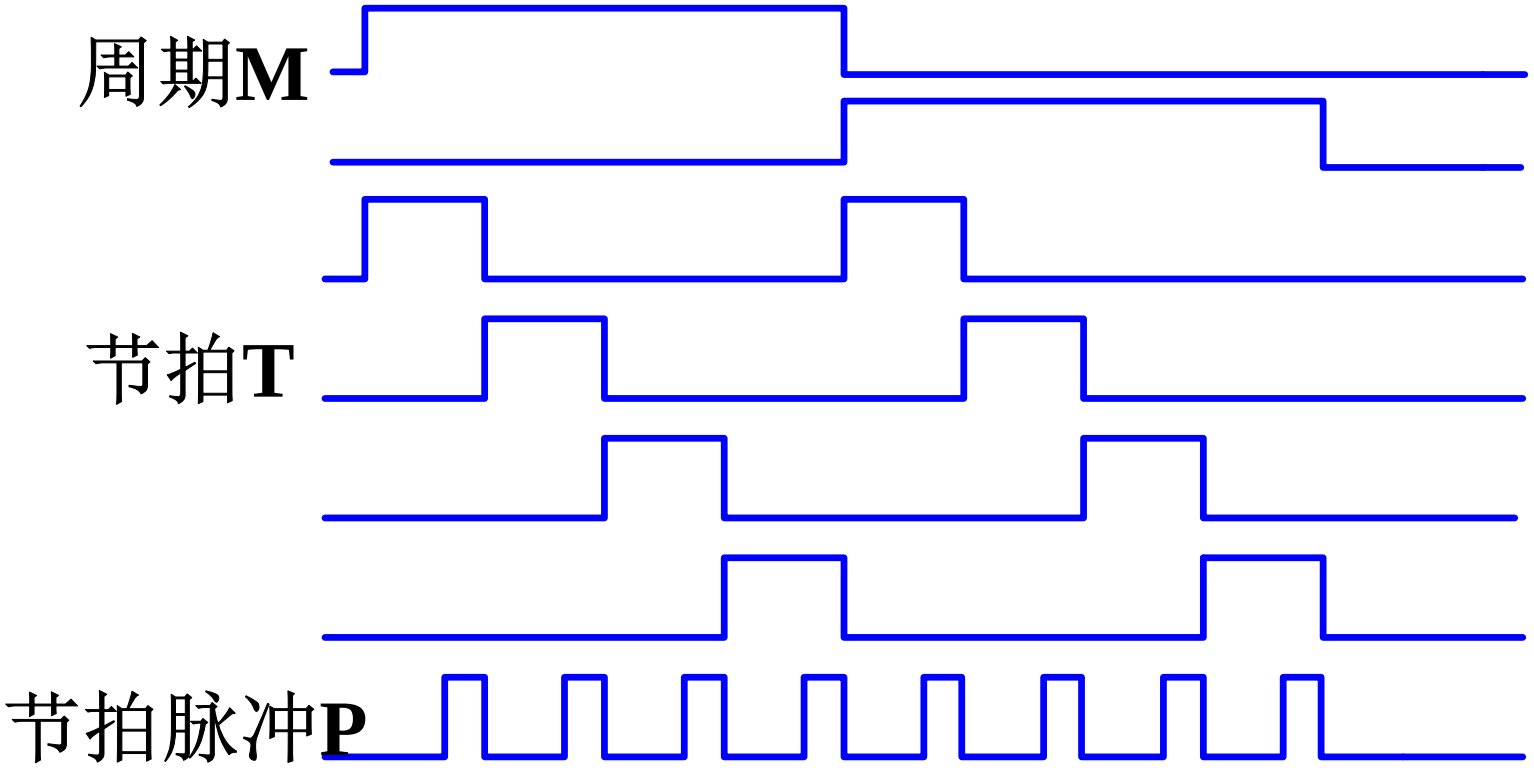
- 指令周期、机器周期、时钟周期的关系（多周期CPU）





1、计算机中的时序信号

- 三级时序信号

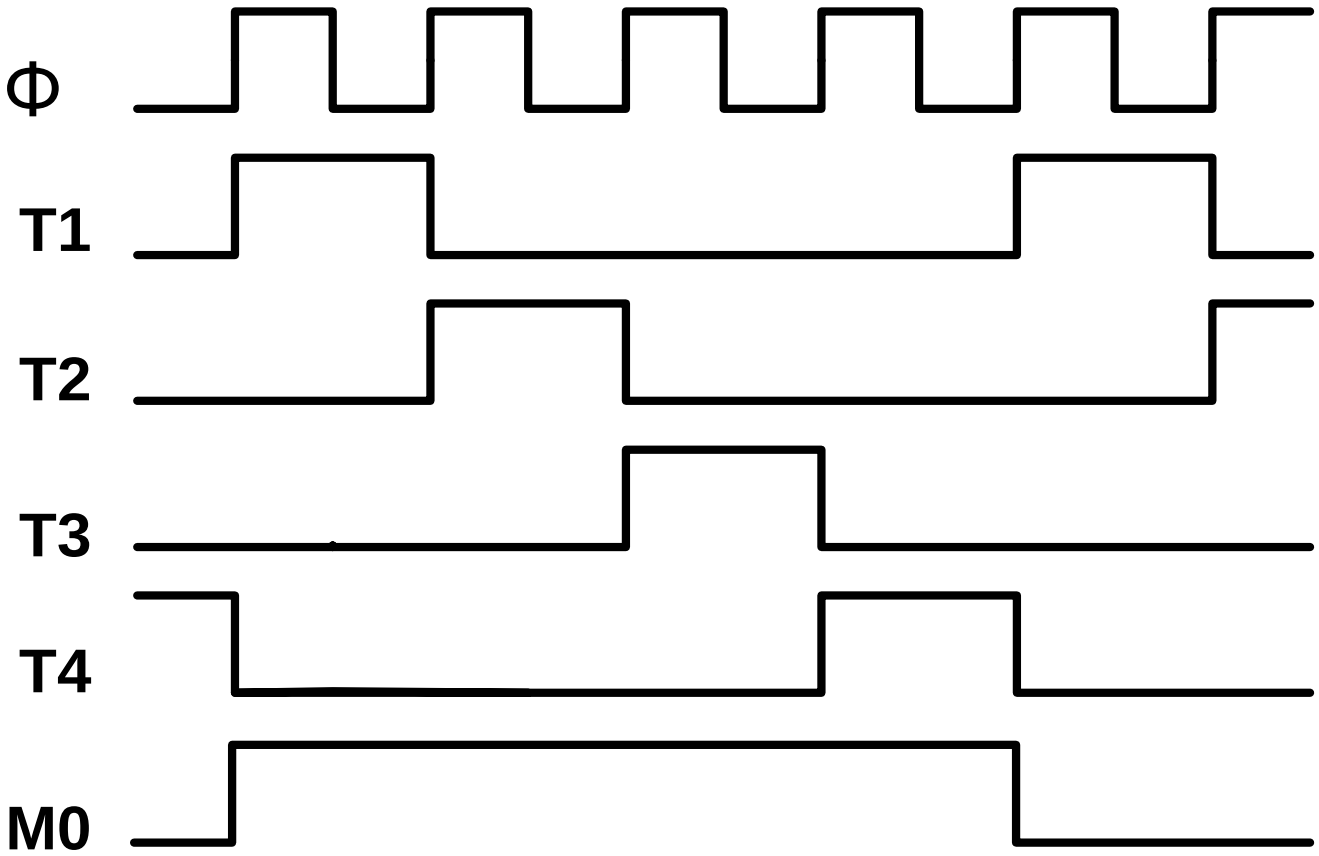


2、时序系统

①时序脉冲发生器

- 根据时钟产生一定频率的各种同步信号（M、T、P）作为整个机器工作的时序信号；
- 机器周期M：通常用访问一次主存取指或取数据的时间来作为机器周期的基本时间。
- 节拍T：按照数据通路与控制方式来决定机器周期中的节拍数。
- 工作脉冲P：每个节拍1～2个。
- 控制器的时钟输入实际上是节拍脉冲序列，其频率即为机器的主频。

2、时序系统



2、时序系统

②启停控制电路

- 保证在适当的时刻**准确可靠地开启或封锁计算机工作时钟**，以控制微操作命令序列的产生或停止，从而启动或停止计算机的运行。

3、控制方式

- 讨论的问题：一条指令有几个机器周期？每个机器周期又有几个节拍？
- 定义：对于不同的微操作控制信号序列**如何定时及同步**，并将信号序列衔接起来，从而保证计算机各部件有节奏地依次执行规定的各种操作。

3、控制方式

① 同步控制方式：统一节拍法

- 又称为固定时序控制方式或无应答控制方式。
- 以微操作序列最长的指令为标准，确定控制微操作运行的时钟周期数（节拍数）。控制器产生统一的、顺序固定的、周而复始的机器周期信号和节拍。
- 优点：电路简单，
- 缺点：运行速度慢。

3、控制方式

② 异步控制方式：分散节拍法

- 异步控制方式又称**可变时序控制方式或应答控制方式**。每条指令需要多少节拍，就产生多少节拍；当指令执行完毕，发出回答信号；控制器收到回答信号时，才开始下条指令的执行。采用**不定长机器周期**。
- **优点**：指令的运行效率高；
- **缺点**：控制器的电路比较复杂。
- 异步控制方式在计算机中得到广泛的应用。例如CPU对内存的读写；I/O设备与内存的数据交换等都采用异步控制方式，以保证高速度的执行。

3、控制方式

③ 联合控制方式：延长节拍法或者时钟周期插入

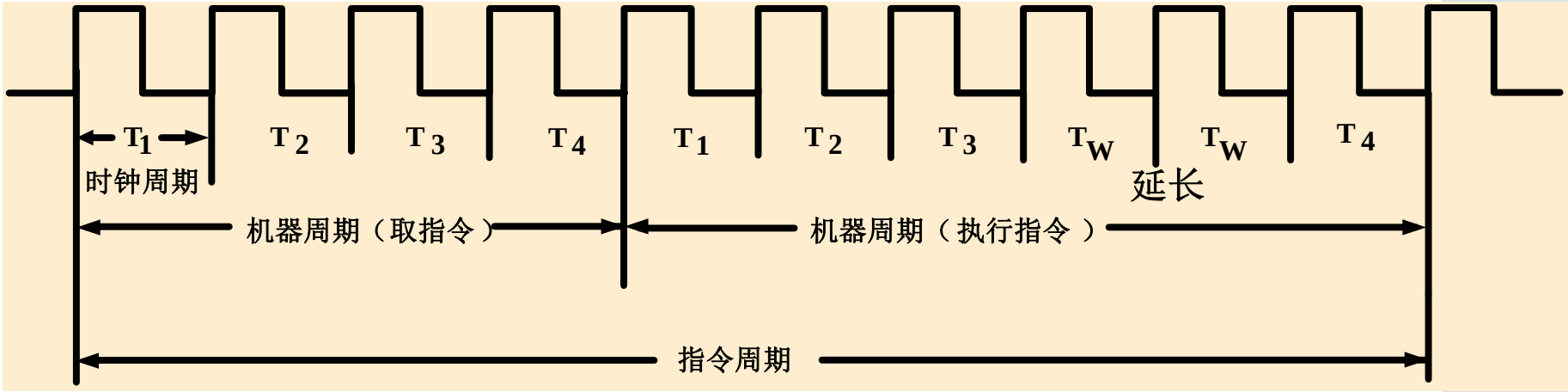
- 把同步控制方式和异步控制方式结合使用的一种方式。
- 大部分指令安排在统一的机器周期内完成，即同步控制；而将较少数特殊指令，或微操作序列过长或过短，或微操作时间难以确定的，采用异步控制来完成。

3、控制方式

③ 联合控制方式：延长节拍法或者时钟周期插入

- 延长节拍法：大多数机器周期采用相同的基本节拍数，若某个机器周期无法完成该周期的全部微操作，则可延长节拍。
- 时钟周期插入：大多数机器周期采用相同的时钟周期数，对于复杂操作的机器周期，则插入等待时钟周期。（例如8086CPU）

3、控制方式



8086CPU延长机器周期的时序图

3、控制方式

③ 联合控制方式：延长节拍法或者时钟周期插入

- 现代计算机系统大多采用联合控制方式，其一般设计思想是，在功能部件内部采用同步控制方式，而在功能部件之间采用异步控制方式。
- 优点：能保证一定的运行速度
- 缺点：控制电路设计相对比较复杂。

四、系统结构和数据通路的设计

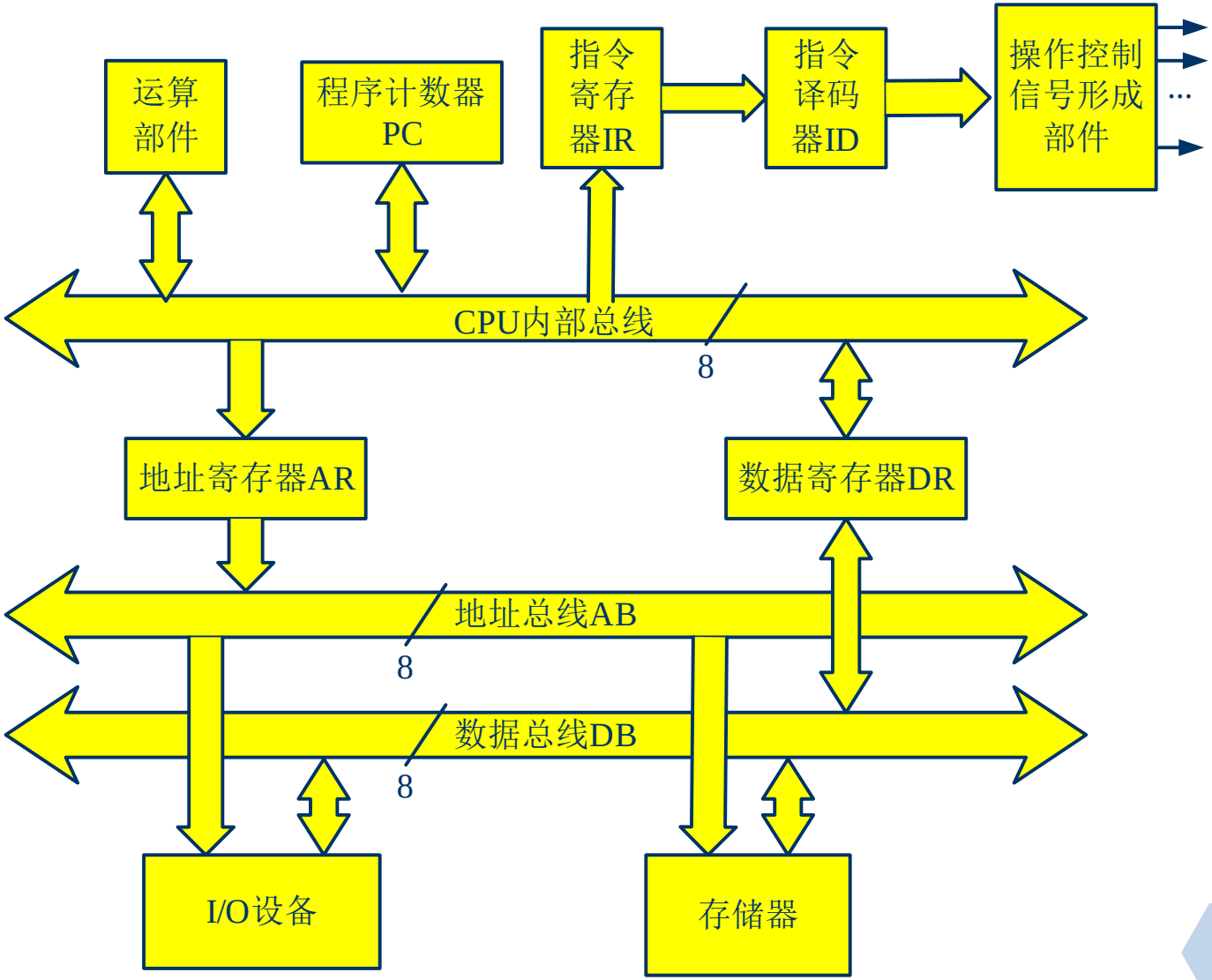
- 计算机的系统结构主要取决于指令系统、机器字长等因素。
- 各部件间的数据通路设计，有两种方案：
- 第一种是在所有需要传送数据的部件之间创建一条直接通路，这种方案对于很小的计算机系统来说是可行的，但是如果所要设计的CPU的复杂度增加的话，用这种方案来设计数据通路将变得越来越不现实。（单周期CPU的简单方案）
- 第二种方案是在CPU内部创建一条总线，并且在各个部件之间使用总线来传递数据。（多周期CPU）

四、系统结构和数据通路的设计

- 确定系统结构和建立数据通路的依据：指令系统中各种指令所需。
  - ①确定实现某种类型指令功能所需的部件；
  - ②在部件之间建立数据通路；
  - ③确定对应的控制信号。

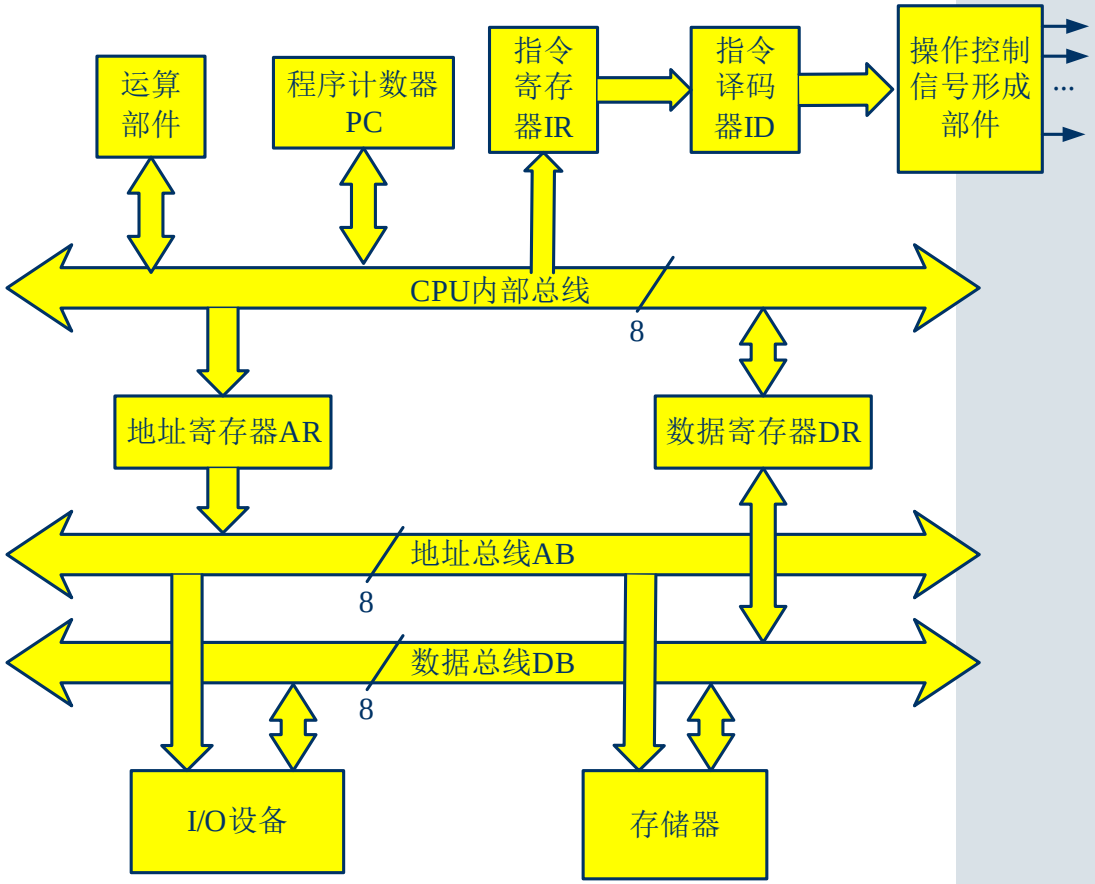
- 4.1 简单计算机系统
- 4.2 MIPS单周期CPU

选择第二种方案，勾画出简单计算机系统的结构



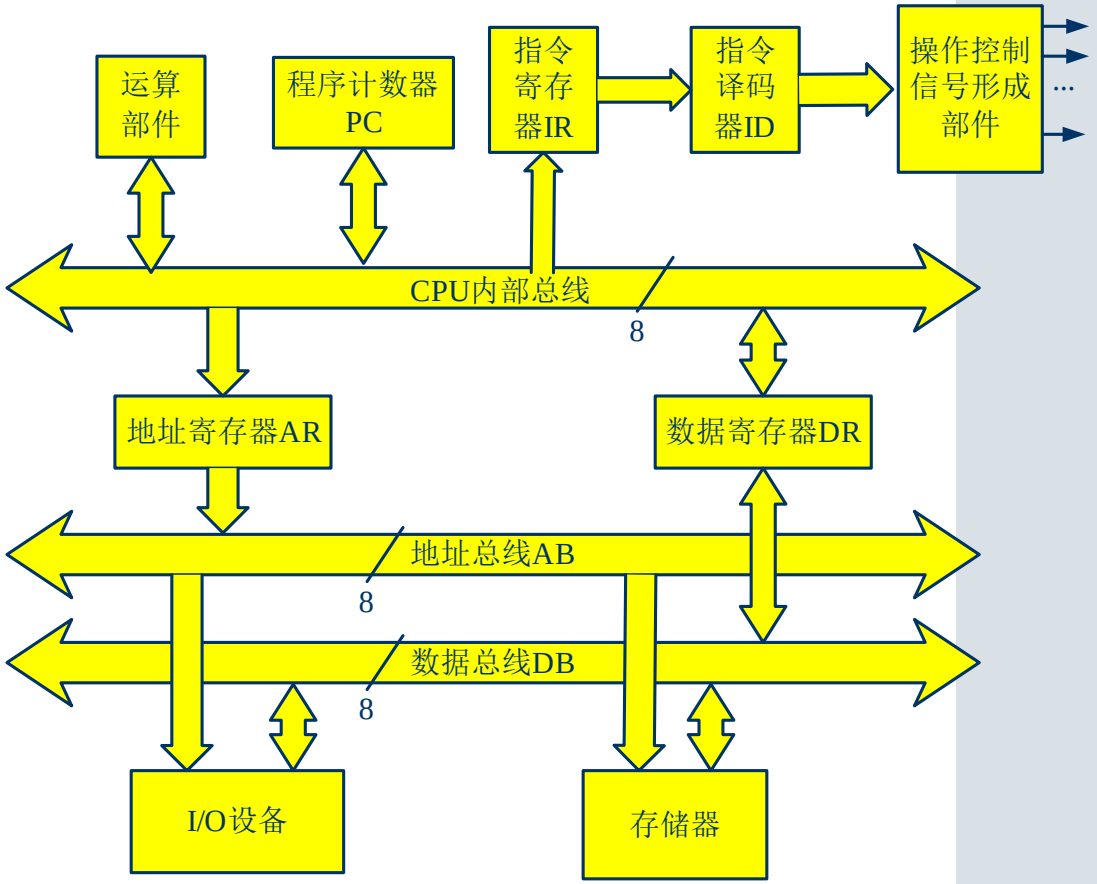
得到访存的数据通路如下：

- 存储器读操作：
  - 送地址到CPU片内总线，并打入地址寄存器AR；控制器发送存储器读信号，启动存储器读操作，并将读出的数据从数据总线上接收至数据寄存器DR。



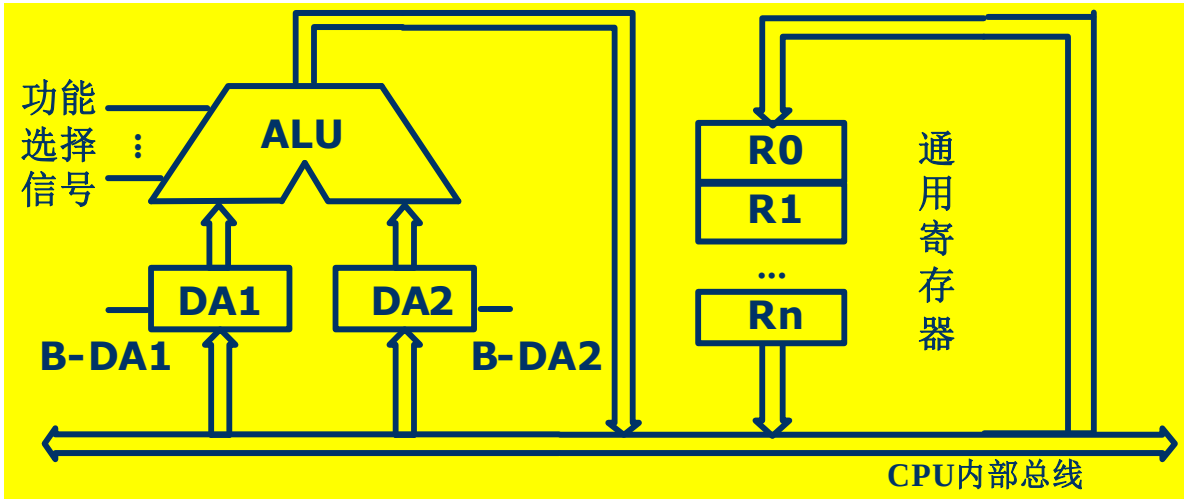
得到访存的数据通路如下：

- **存储器写操作：**
  - 送地址到CPU片内总线，并打入地址寄存器AR；
  - 送数据到DR，DR将数据送到数据总线，控制器发送存储器写信号，启动存储器写操作。



系统结构和数据通路的设计

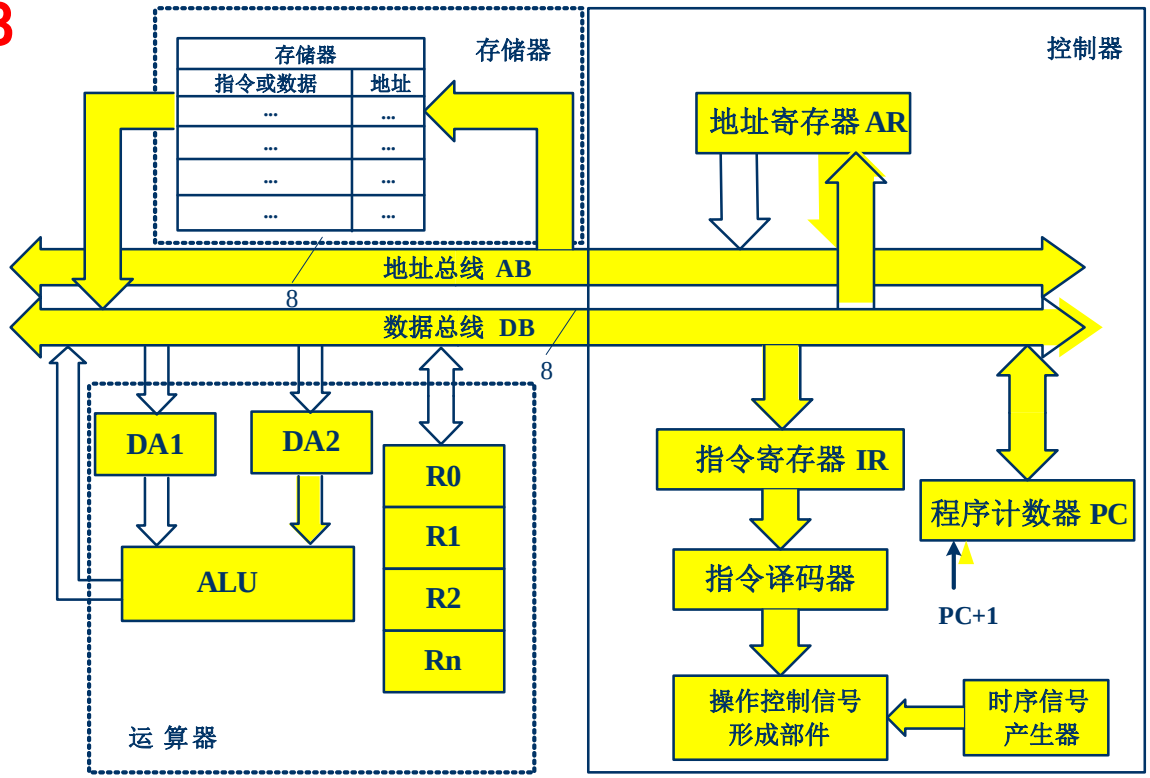
细化图中的运算部件，发现还需要不止一个寄存器以便暂时保存数据，这些寄存器称为通用寄存器，通常CPU会把它们中的一个命名为累加器（AC），它与其他的寄存器有些许的不同，但是对这个系统来说，并不一定要有个专门的AC，结合第四章所学的定点运算器内部单总线结构和通路，画出系统运算部件的内部结构图，如图所示。



运算部件内部结构图

系统结构和数据通路的设计

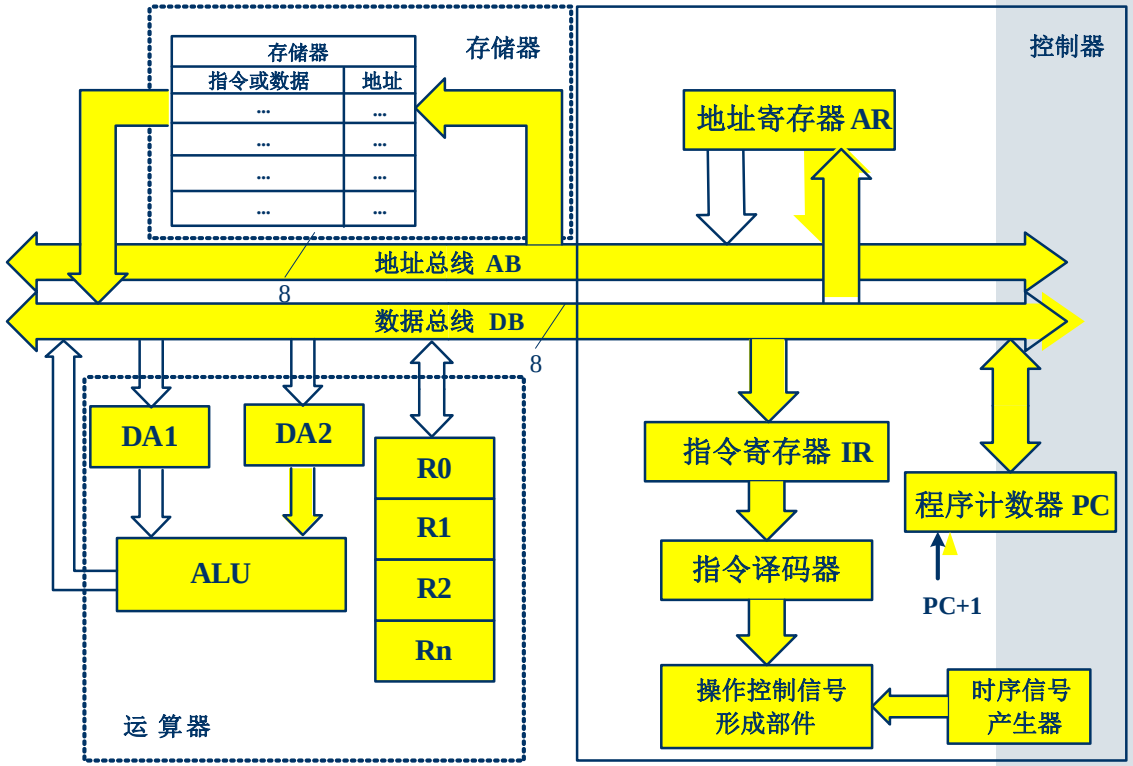
系统结构简化为：与外部地址总线连接的是AR，与外部数据总线连接的是CPU内部总线；因此，**省略了DR**。于是把片外的数据总线和CPU片内总线合并成一条总线，称之为**数据总线DB**



简化后的简单计算机系统的结构图

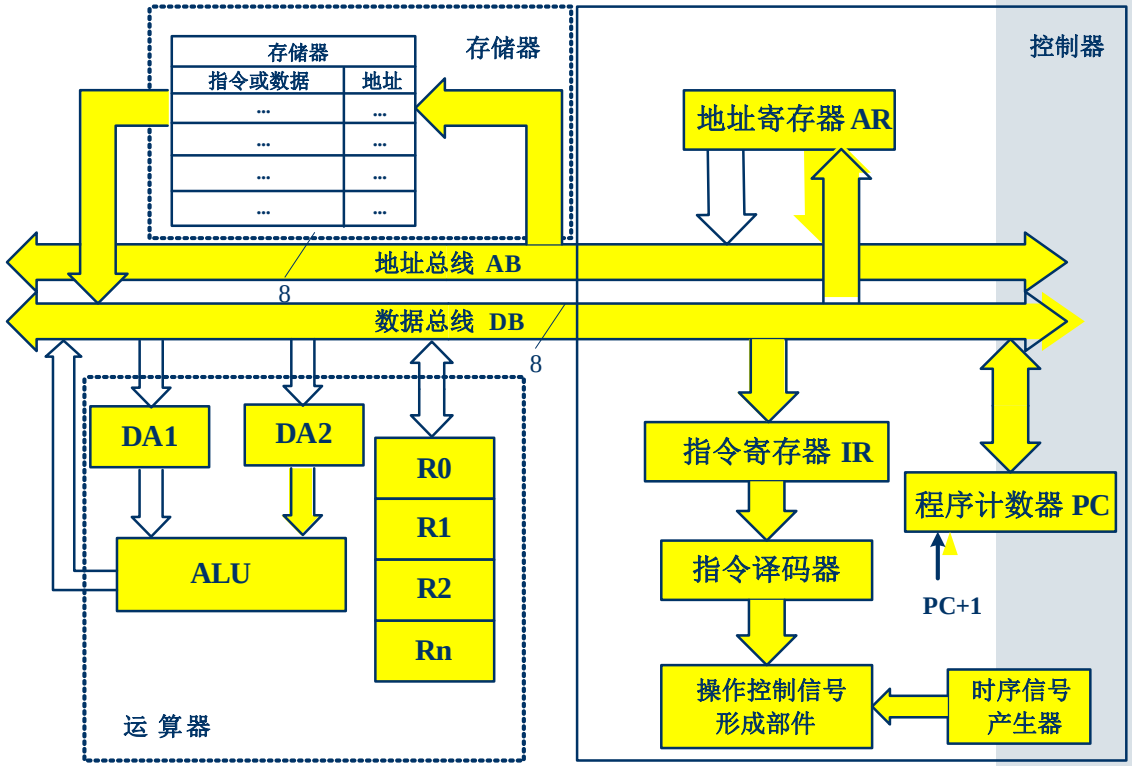
简化结构图上访存的数据通路：

- 存储器读操作：
  - 送地址到CPU片内总线，并打入地址寄存器AR；
  - 控制器发送存储器读信号M-R#=0，启动存储器读操作，并将读出的数据从数据总线上接收至目的寄存器。
- 例如取指令操作



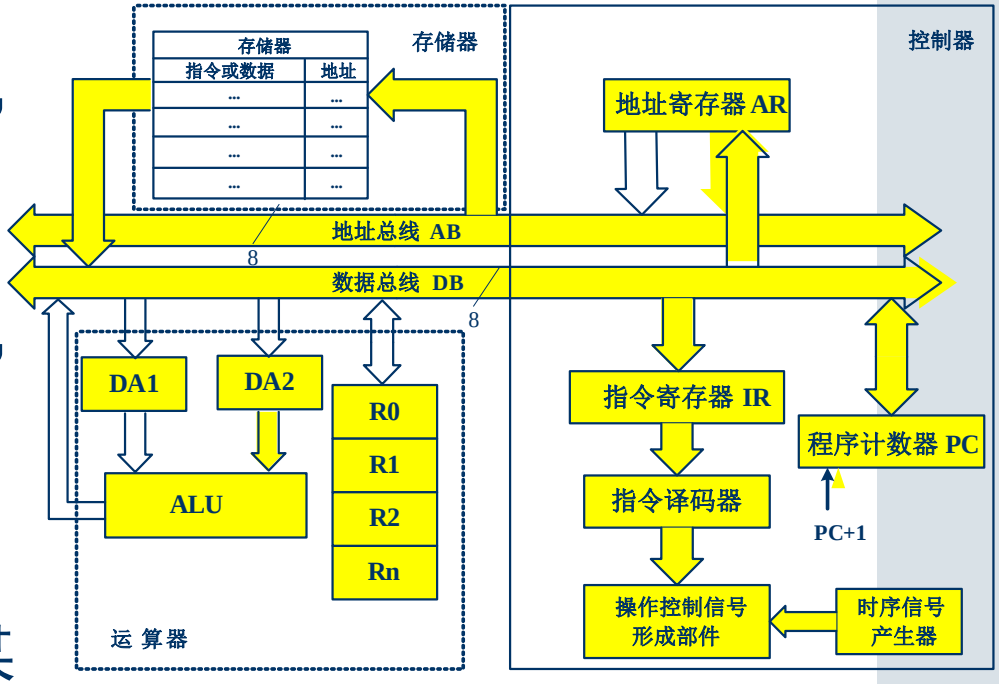
简化结构图上访存的数据通路：

- 存储器写操作：
  - 送地址到CPU片内总线，并打入地址寄存器AR；
  - 将数据送到数据总线，控制器发送存储器写信号M-W# =0，启动存储器写操作。



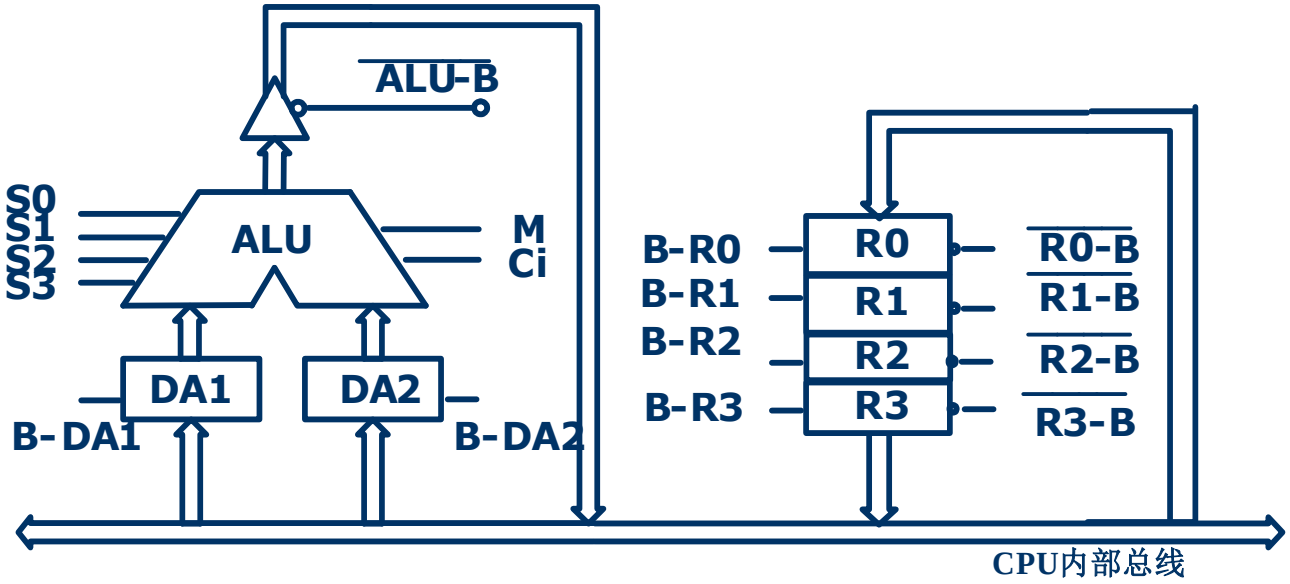
ALU的数据通路：

- 运算器的运算操作：
  - 送第一个数据到总线，并打入ALU暂存器DA1（或DA2）；
  - 送第二个数据到总线，且打入ALU暂存器DA2（或DA1）；
  - 发送运算器功能选择信号，控制ALU进行某种运算，并将结果通过数据总线DB送目的部件（例如某通用寄存器）。



简单计算机系统主机各部件的实现方案

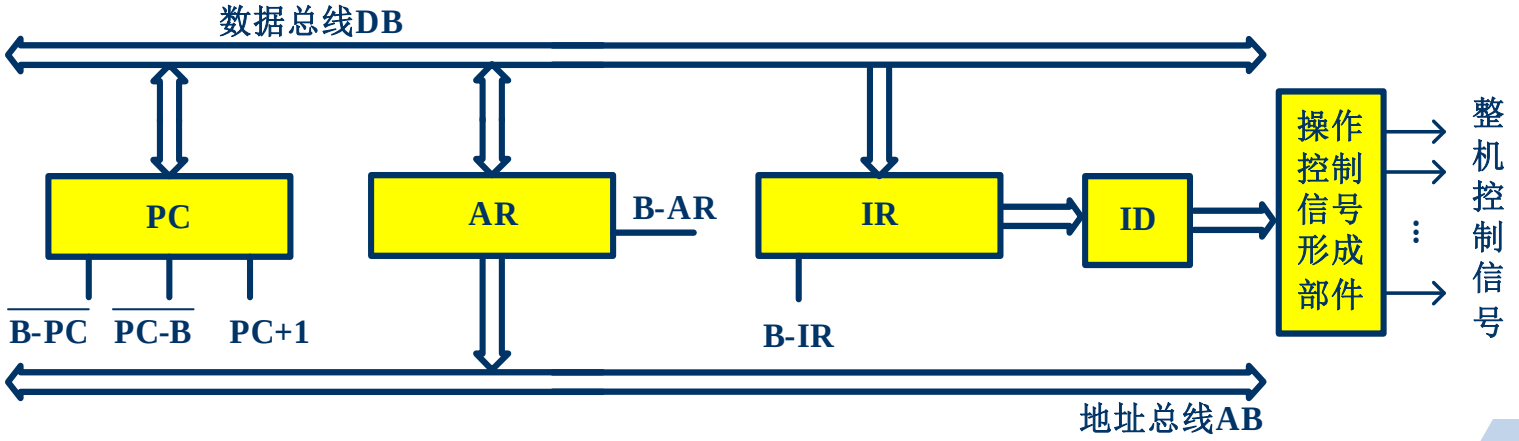
- 1. 运算器
  - 8位的算术逻辑运算器，2个暂存器为DA1和DA2，控制信号用B-DA1和B-DA2,在输出端用一个三态门控制数据是否送上总线，控制信号是ALU-B#，S0~S3、M、Ci是该ALU的运算选择信号。
  - R0-B#~R3-B#分别是读R0~R3的控制信号，B-R0~B-R3分别是写R0~R3的控制信号。



简单计算机系统主机各部件的实现方案

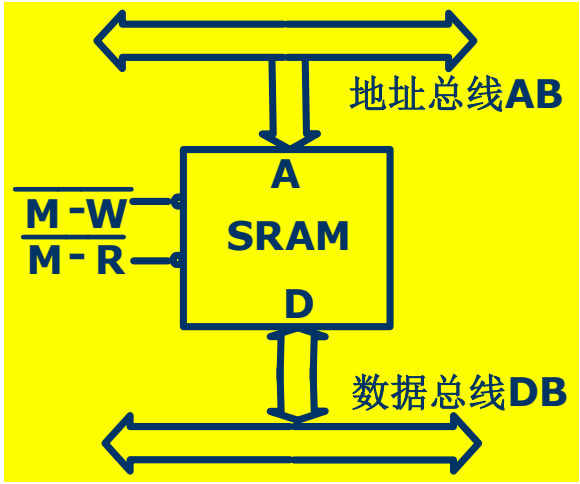
2. 控制器

- 程序计数器PC, PC-B#是将PC值送上数据总线的控制信号，PC+1是PC的自增1控制信号。B-PC#信号控制将数据总线的值送入PC。
- 地址寄存器AR的输入控制信号是B-AR，指令寄存器IR的输入控制信号是B-IR。



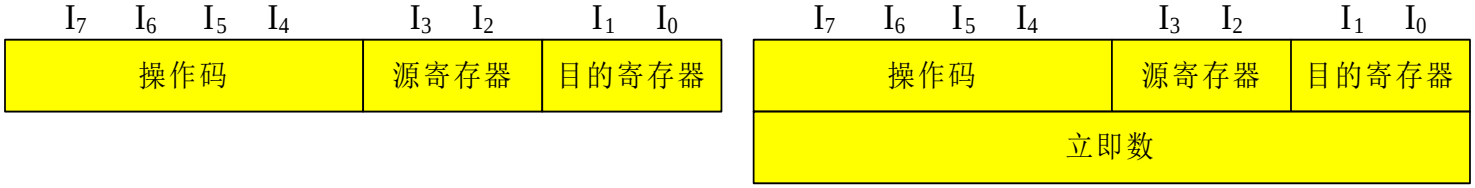
简单计算机系统主机各部件的实现方案

- 3. 存储器
- 用一片SRAM芯片就可以满足存储需要
- SRAM芯片存储单元是字节。
- 用两个信号M-R#和M-W#来控制读和写这个存储器。



指令的执行过程

- 把指令具体化，对于加法指令（ADD R，#data）具体化为ADD R0，06H
- 对于无条件跳转指令（JMP　addr）具体化为JMP 04H
- 根据所设计的指令格式，它们都应该是双字节的指令
- 假如加法指令的操作码是0101，则加法指令ADD R0，06H对应的机器码是50H和06H；
- 假如无条件跳转指令的操作码是1000，则无条件跳转指令JMP 04H对应的机器码是80H和04H；



（a）单字节指令格式　　(b)双字节指令格式

机器指令格式

指令的执行过程

- 假设它们已经在存储器中了，且位于地址04H~07H的单元内，下表给出了这两条指令的内容和在存储器中的位置。

存放在存储器中的二条指令内容

指令地址	指令机器码	助记符
0000 0100	0101 0000	ADD R_0, 06H
0000 0101	0000 0110	立即数
0000 0110	1000 0000	JMP 04H
0000 0111	0000 0100	转移地址

1、取指令

① 控制器先将第一条指令的地址置入PC

② PC将当前指令的地址送到地址寄存器AR，同时程序计数器PC的内容递增以指向下一条指令的地址；

③ AR的输出通过地址总线送到存储器的地址端，指明指令所在的地址单元，控制器发出读控制信号，控制从存储器中读出这条指令；

④ 该指令通过数据总线送到指令寄存器IR。

⑤ 指令取到指令寄存器IR后，指令译码器对其译码；

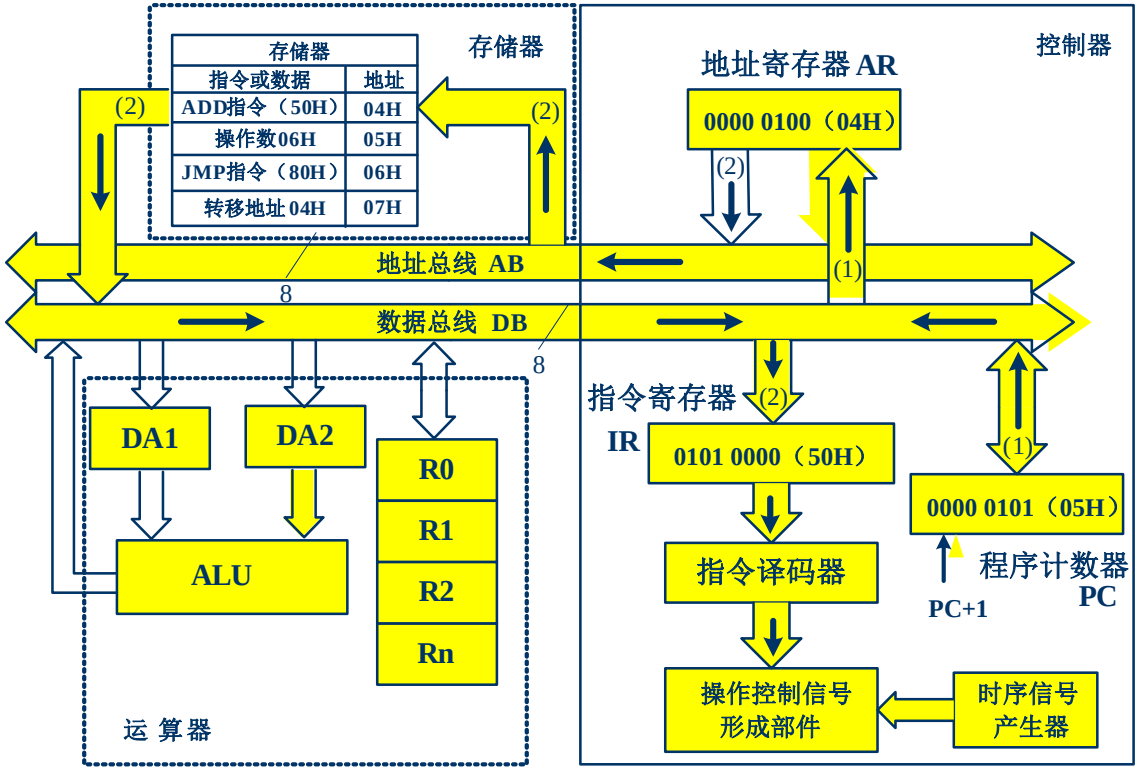
⑥ 指令译码器将译码结果传递给操作控制信号形成部件，至此，取指令的过程完成。

执行指令

- 操作控制信号形成部件根据指令译码信息和时序周期信号，发出该指令所需的所有部件的有一定时序关系的控制信号序列，完成指令的执行。执行指令与指令的内容有很大的关系。

取加法指令

- PC置为04H，并送到AR，AR的地址通过AB送到存储器的地址端，PC+1，指向05H，以准备取立即数；
- 控制器读，将该地址单元的内容50H读出，通过DB送到IR，IR中的指令送到指令译码器ID进行译码，将结果信息送到操作控制信号形成部件；

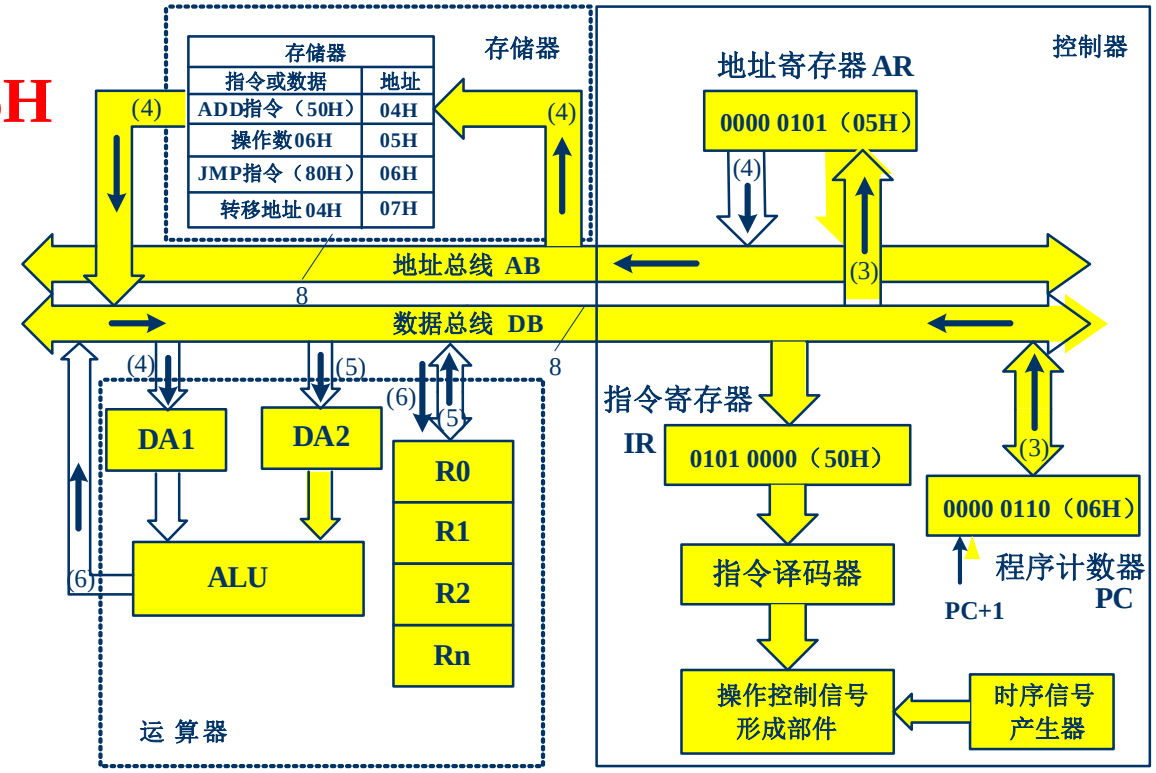


ADD R0, 06H

执行ADD指令

- 将PC的内容05H送到AR，同时PC+1；
- 从存储器05H单元中读出操作数，送到暂存器DA1；
- 根据IR中的低4位，由寄存器地址译码后，寻址源操作数寄存器为R0，从R0中取出另一操作数，送DA2；
- 在ALU中进行加法运算，并将结果送到目的寄存器R0中存放

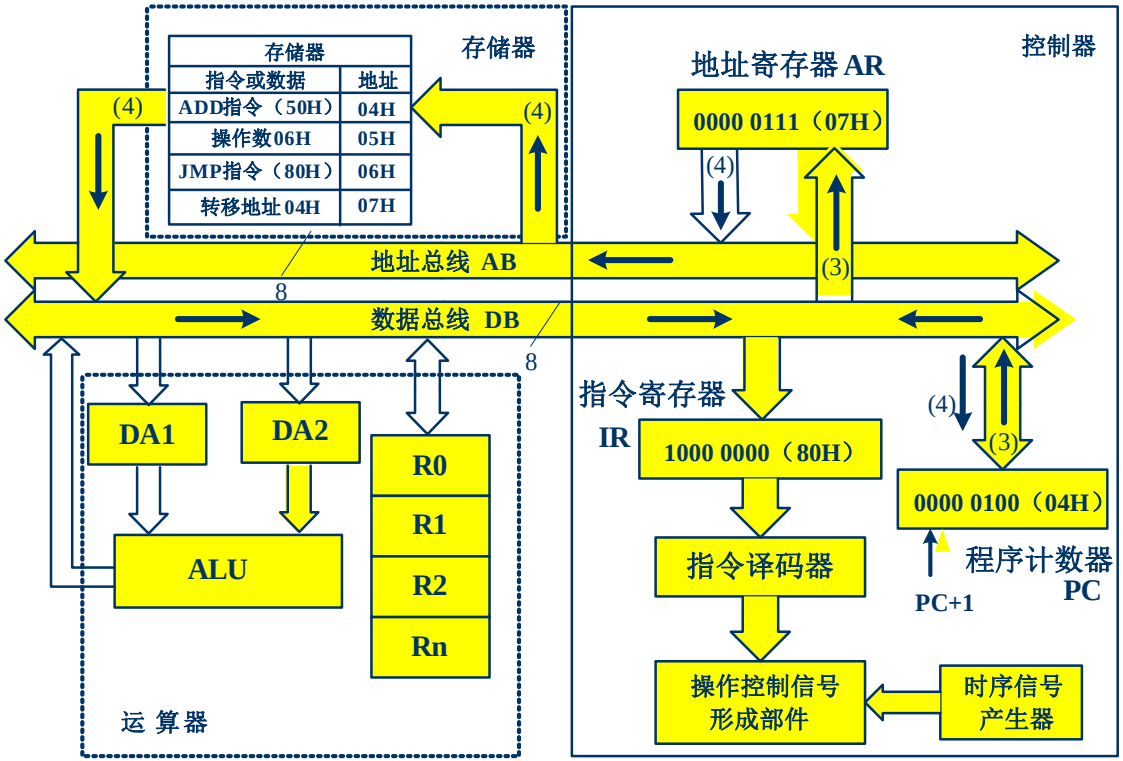
ADD R0, 06H



执行JMP指令的过程

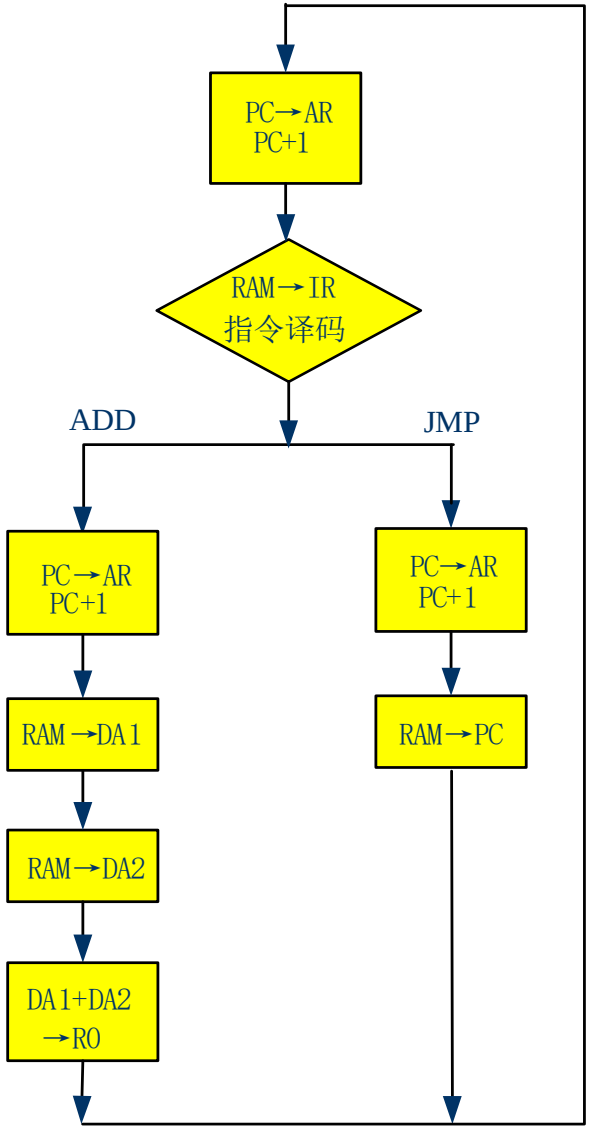
- 转移指令是将程序转移到04H地址的指令去执行。
- 执行指令阶段将PC的内容07H送到AR，同时PC+1；
- 控制器发读信号，从存储器07H单元中读出转移地址，并通过数据总线送到PC，即实现转移操作。

JMP 04H



简单CPU的状态图

- 每个方框和菱形框代表了CPU的一个状态
- 简单CPU其实就是一个仅具有8个状态的有限状态机。
- 当执行了某个状态中的微操作时，CPU就从一个状态转移到了另一个状态。



4.2、MIPS单周期CPU系统结构和数据通路的设计

- 假设实现MIPS的核心指令子集：
  - 算术逻辑运算指令（R型）：add、sub、and、or、slt
  - 访存指令（I型）： lw和sw
  - 转移类指令：分支指令beq（I型），跳转指令j（J型）

MIPS核心指令子集：各指令功能

指令类型	指令	格式	功能
运算类	add/sub/and/or/slt rd,rs,rt	R型	（rs）op （rt）→rd
访存类	lw rt, offset(rs)	I型	mem[(rs)+offset]→rt
	sw rt, offset(rs)	I型	rt → mem[(rs)+offset]
转移类	beq rs, rt, offset	I型	If (rs=rt) then PC+4+offset*4→PC
	J target	J型	{PC[31:28],target,2'b00}→PC

- 各类指令执行过程分析：
  - 取指令：根据PC从内存取出指令，PC+4→PC
  - 执行指令：
    - 首先根据指令字的字段，读取1~2个寄存器内容；
    - 再根据指令具体功能，做不同的操作：
      - 算术逻辑类指令：ALU运算，写寄存器
      - 访存指令：计算存储器地址（ALU运算），读存储器或者写存储器
      - 转移类指令：比较数据（ALU运算），PC+偏移量→PC

各类指令执行过程分析

指令类型	指令执行过程				
运算类（R型）	取指令	读寄存器rs、rt	ALU执行op运算	写寄存器rd	
取数lw（I型）	取指令	读寄存器rs	ALU加运算EA=(rs)+offset	读存储器	写寄存器rt
存数sw（I型）	取指令	读寄存器rs、rt	ALU加运算EA=(rs)+offset	写存储器	
分支beq（I型）	取指令	读寄存器rs、rt	ALU减运算（比较）	*写PC（有条件）	
跳转（J型）	取指令	写PC			

单周期CPU设计和多周期CPU设计

- 单周期CPU：每条指令均在一个时钟周期内完成。
  - 所有指令的执行时间等长：1个时钟周期，CPI=1
  - 时钟周期需要选择所有指令中的执行路径（时间）最长的那条指令为准。
    - 应该是lw指令的执行时间
- 多周期CPU：将指令的执行过程分解成一系列步骤，每个步骤占用一个时钟周期，即：
  - 每条指令的执行可占用多个周期；
  - 允许不同指令占用的周期数不同；
  - 一个功能单元可以在某个指令执行过程中共享。

单周期CPU设计和多周期CPU设计

- 现代计算机并不采用单周期CPU设计方式，原因：效率太低。
- CPU性能评价：最重要的是：CPU执行时间

  CPU执行时间=程序的指令数×每条指令的时钟周期数（CPI）×时钟周期长度
  - 单周期CPU：CPI=1，但是时钟周期很长（最长指令）
  - 多周期CPU：虽然CPI≥1，但是时钟周期很短，且某些硬件部件可共享
- 多周期CPU的性能和硬件成本优于单周期CPU。
- 单周期CPU适用于简单指令系统。
- 多周期CPU适用于流水线。

实现MIPS核心指令子集的单周期CPU设计

- 选择单周期CPU实现MIPS的核心指令子集：
  - 所有指令都在一个时钟周期内完成；
  - 因此，部件不能共享；
  - 需要部件：
    - 指令存储器及取指令部件；
    - 寄存器堆部件及其读写电路；
    - ALU部件；
    - 数据存储器及其读写电路；
    - PC的自增与更新部件；

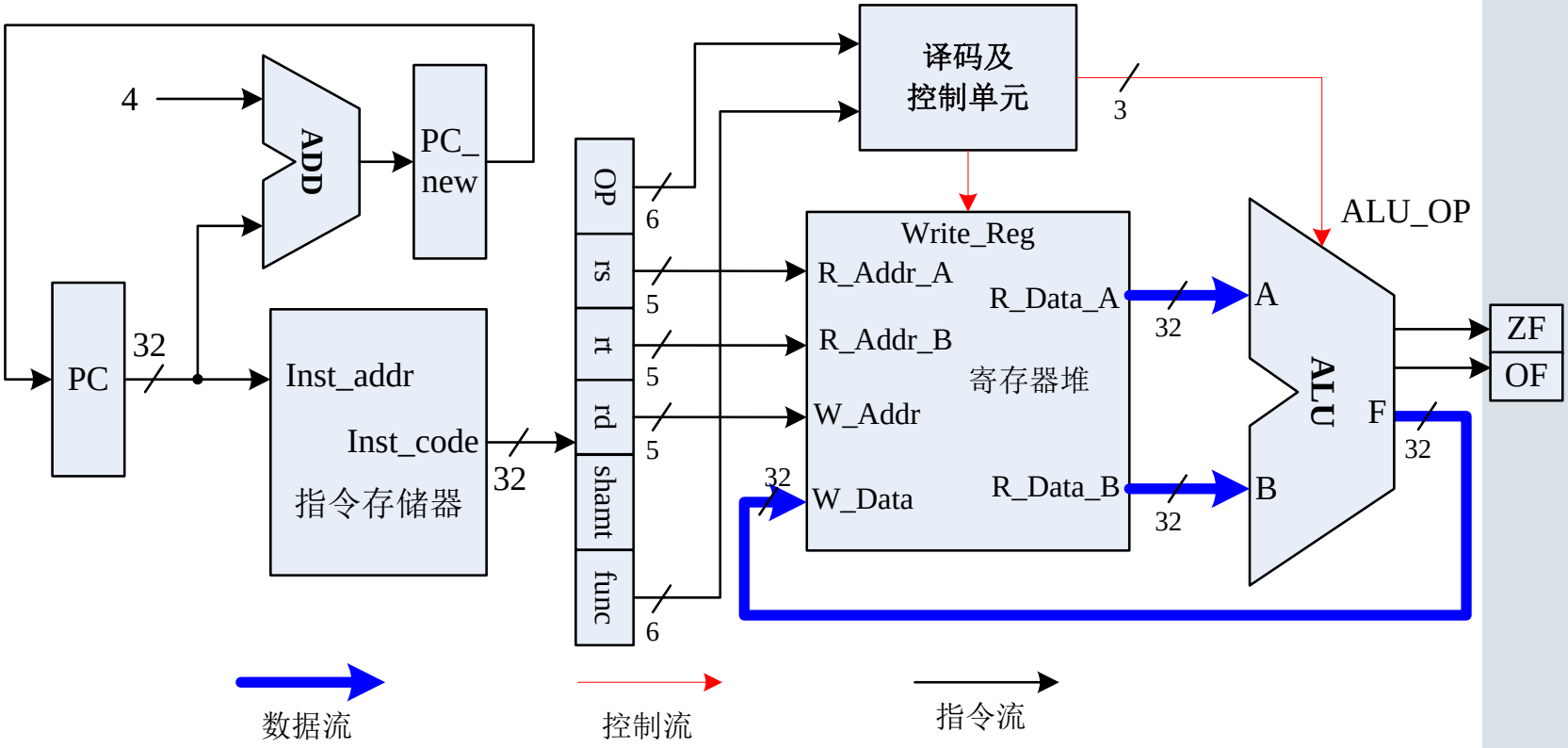
实现MIPS核心指令子集的单周期CPU设计

- 1、R型指令（运算类指令）
- 2、I型指令（访存指令）
- 3、转移指令（I型分支指令、J型跳转指令）

- 均从3个方面考虑：
  - 部件的设置
  - 数据通路的建立
  - 控制信号

1、R型指令（运算类指令）

- 数据通路建立



1、R型指令（运算类指令）-部件

- **指令存储器：**
  - 取指令：由PC提供地址，在时钟周期的上跳沿完成读操作；（也可以不受时钟控制，但是PC值在本周期内不能改变）
- **PC的控制：**
  - 自增：通过加法器，+4运算，可以不受时钟控制
  - 更新：必须在时钟周期的下[illegible]本周期内PC值不变）；
- **指令译码：**
  - 不设置指令寄存器IR：why？
  - 读出的32位指令，分别将各字段送各数据通路部件：
    - rs、rt、rd送寄存器堆；
    - OP（6'b000000）和func送译码控制单元译码。

因为在本周期内读出的指令不会改变

1、R型指令（运算类指令） -部件

- **寄存器堆：**
  - 2个读端口：rs、rt指定读端口的地址；
  - 一个写端口：rd指定写端口的地址；
  - 2个读数据输出端口：直接送ALU的A和B端口；
  - 1个写数据输入端口：从ALU的运算结果输出端引入；
  - 写控制信号： Write_Reg
- **ALU：**
  - 运算功能：8种；
  - 输入A和B：从寄存器堆的2个读端口接入(rs)、(rt)
  - 运算结果输出F：接寄存器堆的写数据端口

1、R型指令（运算类指令）

- R型指令的控制信号：
  - ALU_OP：由指令译码产生；选择ALU的运算功能；
  - Write_Reg：在时钟周期的下降沿有效；控制将ALU的运算结果写入rd指定的寄存器

指令	func	ALU_OP	操作
add rd,rs,rt	100000	100	算术加
sub rd,rs,rt	100010	101	算术减
and rd,rs,rt	100100	000	逻辑位与
or rd,rs,rt	100101	001	逻辑位或
sltu rd,rs,rt	101011	110	小于置位

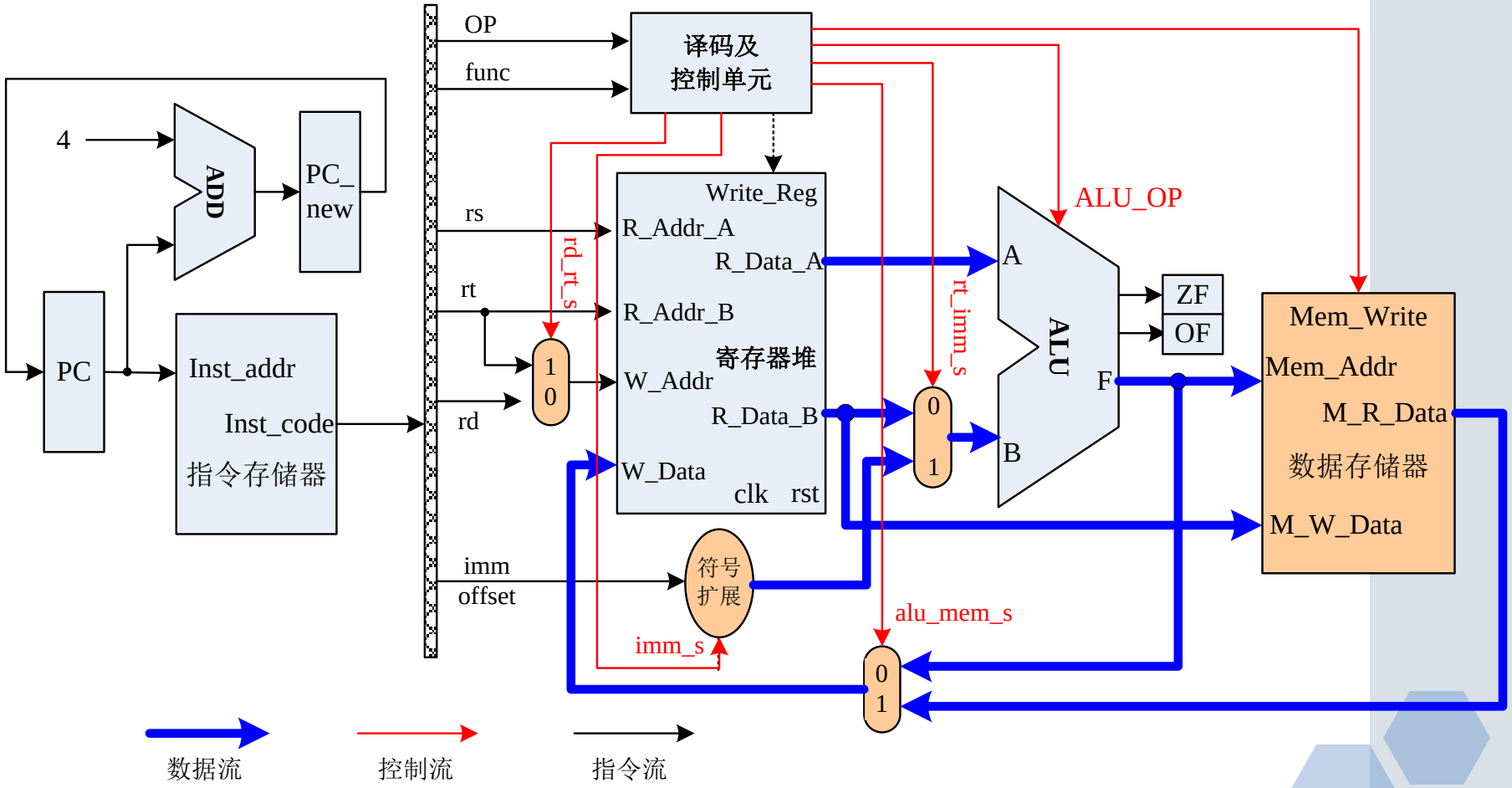
2、I型指令（访存指令）

指令类型	指令执行过程				
取数lw（I型）	mem[(rs)+offset]→rt				
	取指令	读寄存器 rs	ALU加运算 EA=(rs)+offset	读存储器	写寄存器 rt
存数sw（I型）	rt → mem[(rs)+offset]				
	取指令	读寄存器 rs、rt	ALU加运算 EA=(rs)+offset	写存储器	

2、I型指令（访存指令） -部件

- 相比R型指令的组成部件，还需要添加：
  - 数据存储器；
    - •哈佛结构；（为何不能和指令存储器共享同一个存储器？）
    - •读出的数据：到哪去？寄存器堆（rt）
    - •写入的数据：从哪来？寄存器堆（rt）
  - 寄存器堆：
    - •可以向rt对应的寄存器写入；
    - •设置多路选择器；
  - EA的计算：加法运算；
    - •Offset的符号扩展部件；
    - •共享ALU还是另单独设计一个地址加法器？共享ALU
    - •问题：ALU的B口数据多了一个来源；
    - •解决：使用多路选择器

2、I型指令（访存指令） -数据通路



2、I型指令（访存指令） -部件

- **数据存储器：**
  - 地址：由ALU计算得到的EA提供，接ALU的F端；
  - 写入的数据：由寄存器堆rt对应的读端口数据提供，接寄存器B口读出数据；
  - 读出的数据：送寄存器堆的写端口，写rt指定的寄存器
- **寄存器堆的多路选择器：**
  - 需要设置2个多路选择器：
    - 寄存器写端口地址：指令的rd或者rt字段，二选一；
    - 寄存器写端口数据：ALU运算结果F或者存储器读出数据，二选一
- **立即数字段的符号扩展部件：**
  - 由16位的立即数或者偏移量，扩展成32位，高16位填充符号位或者0；
  - 32位符号扩展数据送ALU的B端口进行计算；

2、I型指令（访存指令）-部件

- ALU：
  - B端口设置多路选择器：寄存器堆的读B端口数据或者符号扩展输出数据，二选一
- 指令译码：
  - 读出的32位指令，分别将各字段送各数据通路部件：
    - rs、rt、rd送寄存器堆；
    - Imme/offset送符号扩展部件；
    - OP和func送译码控制单元译码。

2、I型指令（访存指令） -控制信号

❖I型访存指令的控制信号：

- 数据存储器的写信号Mem_Write；
- 三个多路选择器的选择信号；
- 符号扩展的控制信号；控制有符号扩展和无符号扩展

信号	R型指令	lw rt, offset(rs)	sw rt, offset(rs)
rd_rt_s	0	1	——
imm_s	——	1	1
rt_imm_s	0	1	1
alu_mem_s	0	1	——
ALU_OP	***	100	100
Write_Reg	1	1	0
Mem_Write	0	0	1

3、转移指令

指令类型	指令执行过程			
分支（I型） beq rs,rt,offset	If (rs=rt) then PC+4+offset*4→PC			
	取指令	读寄存器 rs、rt	ALU减运算 （比较）	*写PC（ZF=1）
跳转（J型） J target	{PC[31:28],target,2’b00}→PC			
	取指令	写PC		

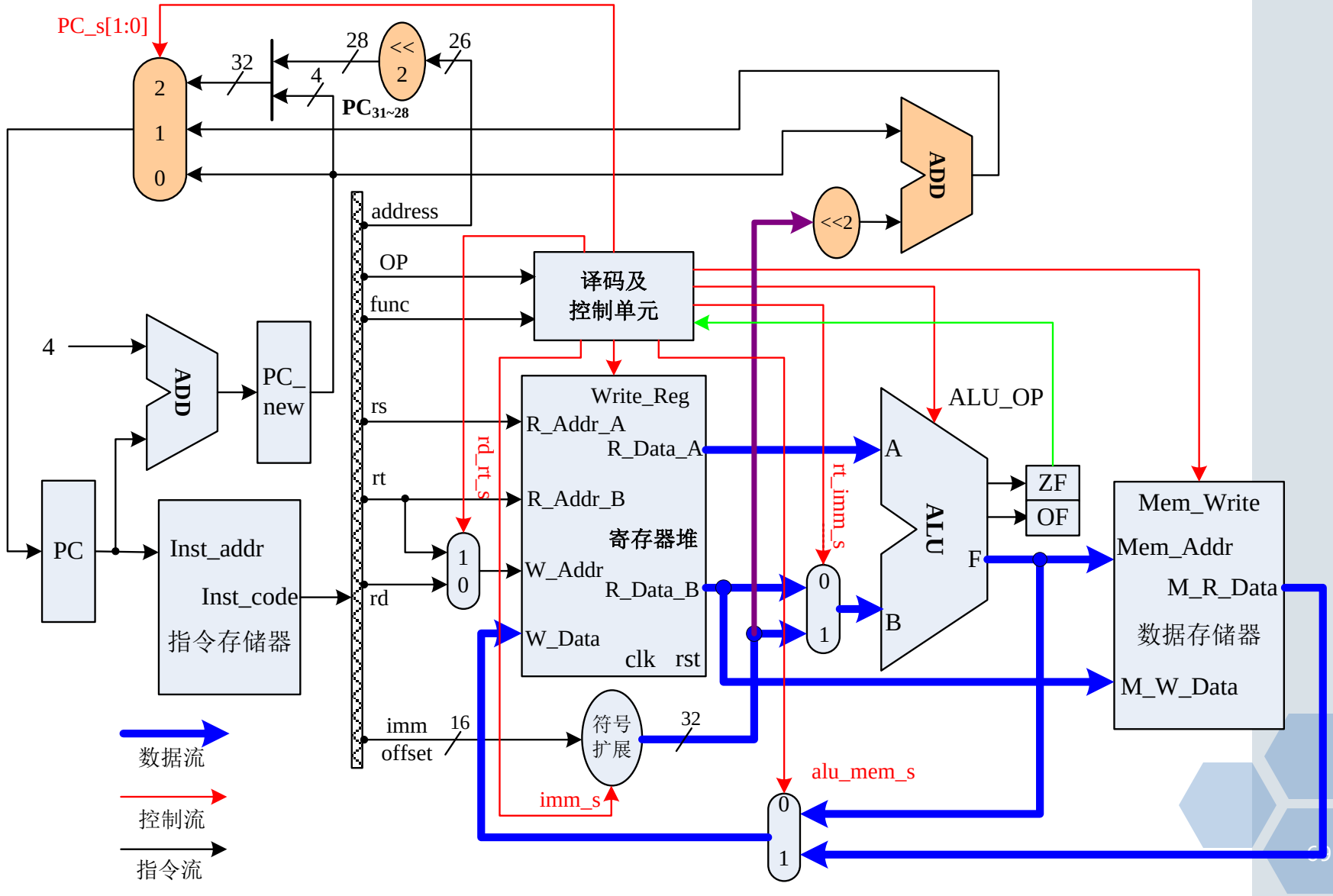
3、转移指令 –分析

- 对于beq指令：
  - rs和rt的比较：使用ALU实现，置位ZF标志；
  - 相对转移的地址计算：需要一个新的加法器实现；
  - Offset*4的实现：将符号扩展后的32位offset左移2位，需要一个移位器
  - PC值的修改：除了自增4之外，多了一个选择——计算得到的相对转移地址（ZF=1时，转移地址→PC）
- 对于J指令：
  - 页面寻址方式：不需要单独部件，信号拼接即可得到跳转地址EA
  - PC值的修改：又多了一个选择——拼接得到的跳转地址

3、转移指令 –部件

- **新增部件：**
  - 相对转移地址加法器；
  - 偏移量移位器；
  - PC更新的多路选择器：三选一
    - 自增：即PC+4（顺序指令）
    - 相对转移地址：PC+4+offset*4，即相对转移地址加法器的输出（分支指令）
    - 页面寻址的转移地址：拼接得到转移地址（跳转指令）

3、转移指令 –数据通路


PC_s[1:0]
2
1
0
32
4
28
<<
2
26
PC31~28
ADD
<<2
address
OP
func
译码及
控制单元
4
ADD
PC_
new
rs
rt
rd
rd_rt_s
Write_Reg
R_Addr_A
R_Data_A
R_Addr_B
寄存器堆
W_Addr
R_Data_B
W_Data
clk
rst
ALU_OP
rt_imm_s
A
ALU
B
F
ZF
OF
Mem_Write
Mem_Addr
M_R_Data
数据存储器
M_W_Data
PC
Inst_addr
Inst_code
指令存储器
imm
offset
16
符号
扩展
32
imm_s
alu_mem_s
数据流
控制流
指令流

3、转移指令 –部件

- （相对转移）地址加法器：
  - 功能：加法
  - 输入数据：
    - PC的新值（PC+4）
    - 16位offset经过符号扩展后的32位偏移量再左移2位的值；
  - 输出数据：加法结果，送PC的多路选择器
- PC的多路选择器：
  - 三选一：由2位信号PC_s[1:0]选择；
    - 2’b00：PC的新值PC_New（=PC+4）；
    - 2’b01：地址加法器的输出（= PC+4+offset*4）；
    - 2’b10：页面寻址的拼接地址{PC[31:28],address,2’b00}

3、转移指令 –控制信号

❖转移类指令的控制信号：

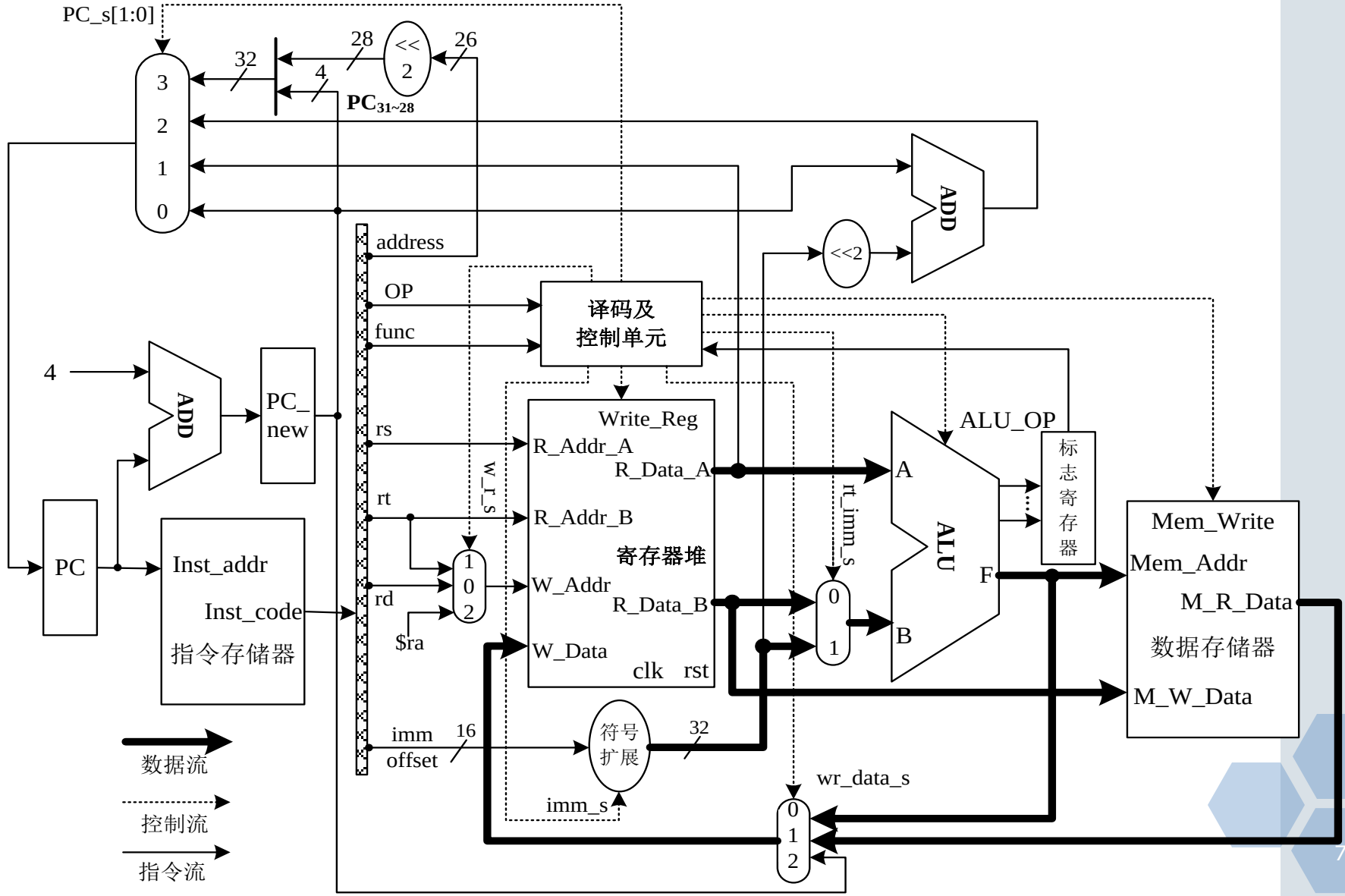
- PC_s[1:0]：选择PC的更新值

信号	R型指令	lw	sw	beq	J
rd_rt_s	0	1	——	——	——
imm_s	——	1	1	1	——
rt_imm_s	0	1	1	0	——
alu_mem_s	0	1	——	——	——
ALU_OP	***	100	100	101	——
Write_Reg	1	1	0	0	0
Mem_Write	0	0	1	0	0
PC_s[1:0]	00	00	00	ZF=1:01 ZF=0:00	10

3、转移指令 （jr rs; jal label）

指令类型	指令执行过程			
jr rs（R型）	rs→PC			
	取指令	读寄存器 rs	写PC	
jal label（J型）	(PC+4)→$31,{PC[31:28],target,2'b00}→PC			
	取指令	写寄存器$31, 写PC		

3、转移指令 （jr rs; jal label）



五、指令的执行过程

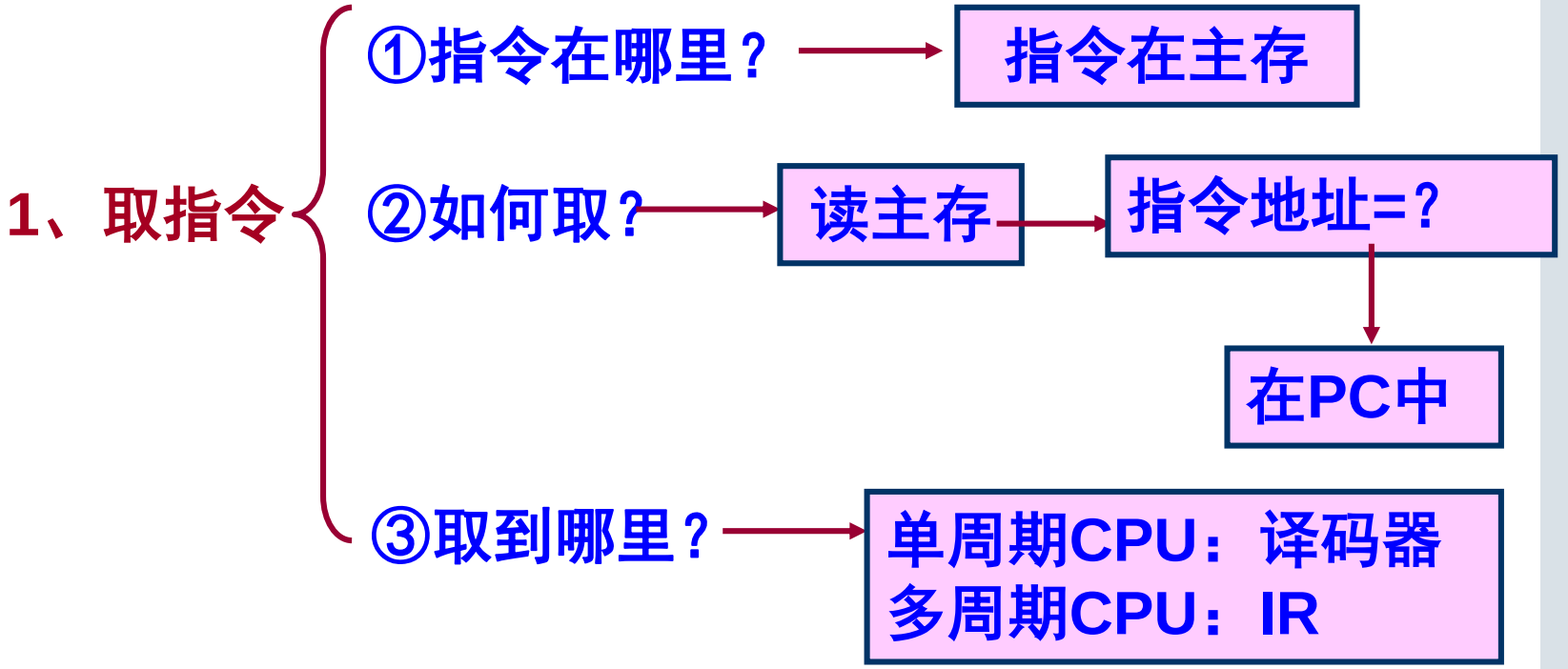
(一) 指令执行过程概述

(二) 典型指令的执行过程

(三) 计算机的工作过程

（一）指令执行过程概述

- 一条指令的执行过程包括取指令、分析指令、执行指令三个步骤：



- 总结：取指令就是以PC为地址读存储器，读出的指令送IR或者指令译码器，PC自增。

（一）指令执行过程概述

2、分析指令：

①指令的功能？ → 对OP字段译码

②操作数在哪里？ → 识别寻址方式

③指令含几个字？ → 确定是否要继续取指令第二字

- 总结：分析指令就是控制指令译码器ID工作，产生识别指令OP和寻址方式的控制信号。

（一）指令执行过程概述

3、执行指令

①取操作数 → 取指令剩余字（如果需要的话）；根据寻址方式计算操作数的EA；取操作数。

②执行操作或处理 → 根据指令功能执行：传送/计算/移位/转移等操作

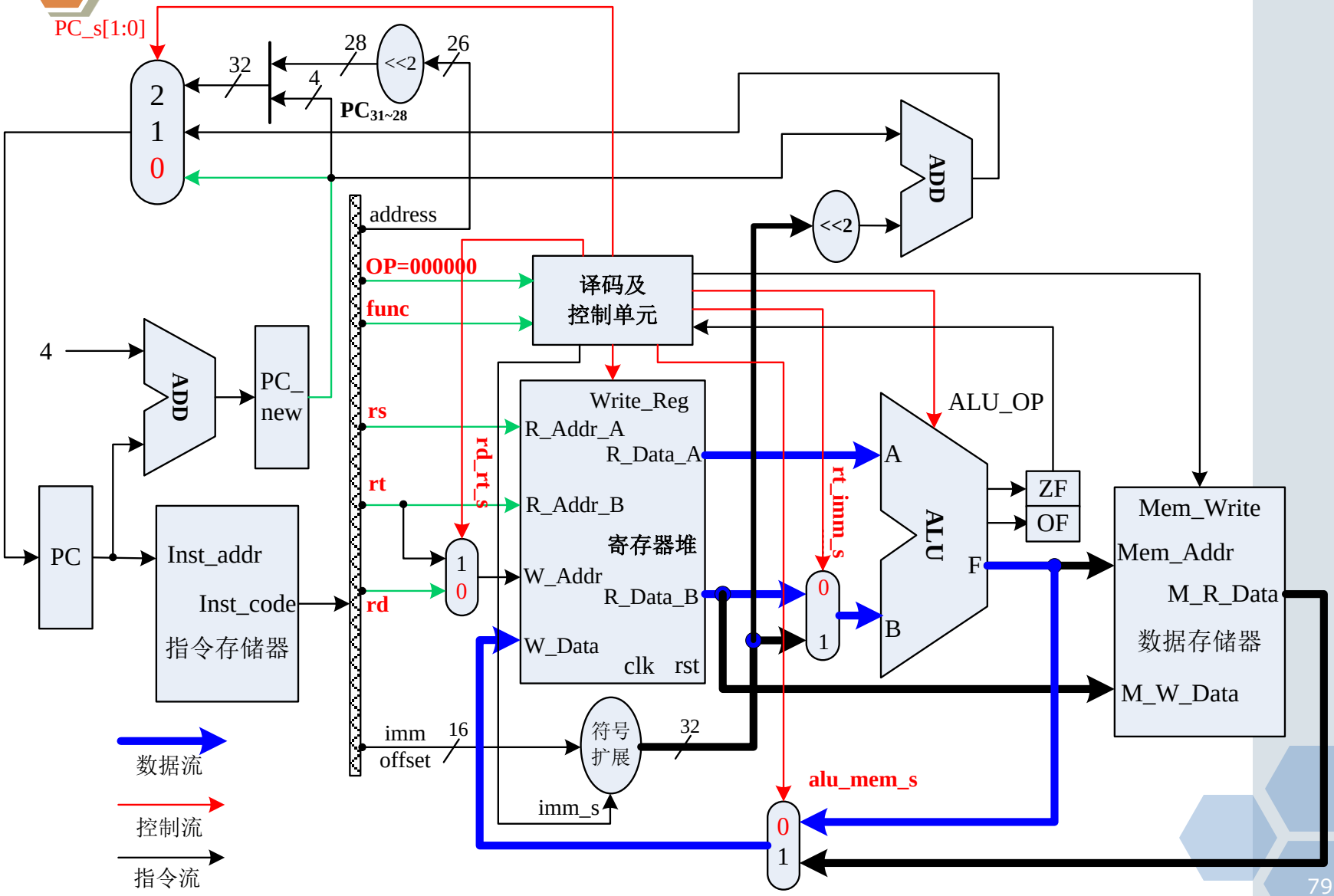
- 总结：执行指令的具体操作取决于指令的功能与寻址方式。

（二）典型指令的执行过程

- 1、R型指令
- 2、I型访存指令
- 3、I型分支指令
- 4、J型跳转指令

地址	机器码	汇编指令	功能
0040 0000H	000000 01000 01001 10000 00000 100000	L:add $s0,$t0,$t1	($t0)+($t1)→$s0
0040 0004H	100011 10000 10001 0000 0000 1100 1000	lw $s1,200($s0)	mem[($s0)+200]→$s1
0040 0008H	101011 01011 10010 0000 0001 0010 1100	sw $s2,300($t3)	$s2→mem[($t3)+300]
0040 000CH	000100 10000 10001 1111 1111 1111 1100	beq $s0,$s1,L	If ($s0=$s1) PC+4+(-4)*4→PC
0040 0010H	000010 0000 0100 00 00 0000 0000 0000 00	J L	{PC[31:28],address,2’b00} →PC

1、R型指令 –数据通路


PC_s[1:0]
2
1
0
32
4
28
<<2
26
PC31~28
address
OP=000000
func
译码及
控制单元
ADD
<<2
4
ADD
PC_
new
rs
rt
rd
rd_rt_s
Write_Reg
R_Addr_A
R_Data_A
R_Addr_B
寄存器堆
W_Addr
R_Data_B
W_Data
clk
rst
ALU_OP
rt_imm_s
A
ALU
B
F
ZF
OF
Mem_Write
Mem_Addr
M_R_Data
数据存储器
M_W_Data
PC
Inst_addr
Inst_code
指令存储器
imm
offset
16
符号
扩展
32
imm_s
alu_mem_s
数据流
控制流
指令流

1、R型指令 –执行过程

（1）根据PC从指令存储器取出指令，PC自增

- 得到I_mem[PC]，（PC）+4

（2）译码及控制单元工作，置各控制信号状态；读寄存器值

- 译码：若OP=000000，则置rd_rt_s=0，rt_imm_s=0，alu_mem_s=0，Write_reg=1,再根据func置ALU_OP的编码。
- 读寄存器rs、rt的值送ALU；

（3）ALU执行规定的运算操作

（4）ALU运算结果写入rd指向的寄存器。

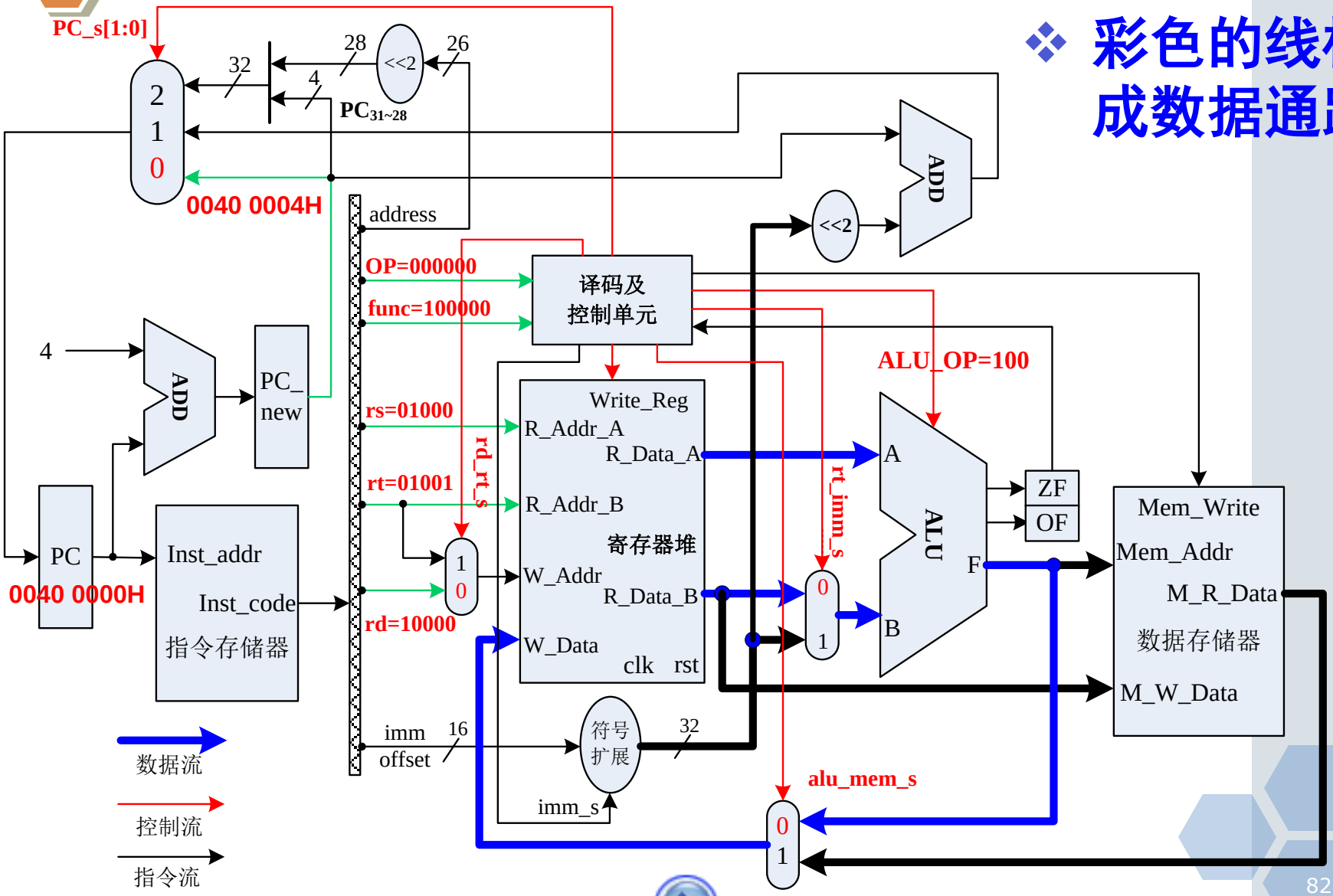
1、R型指令 –举例：add $s0,$t0,$t1

助记符	指令格式（位）					
	31..26	25..21	20..16	15..11	10..6	5..0
R型指令	op	rs	rt	rd	shamt	func
add rd,rs,rt	000000	rs	rt	rd	00000	100000
add $s0,$t0,$t1	000000	01000	01001	10000	00000	100000

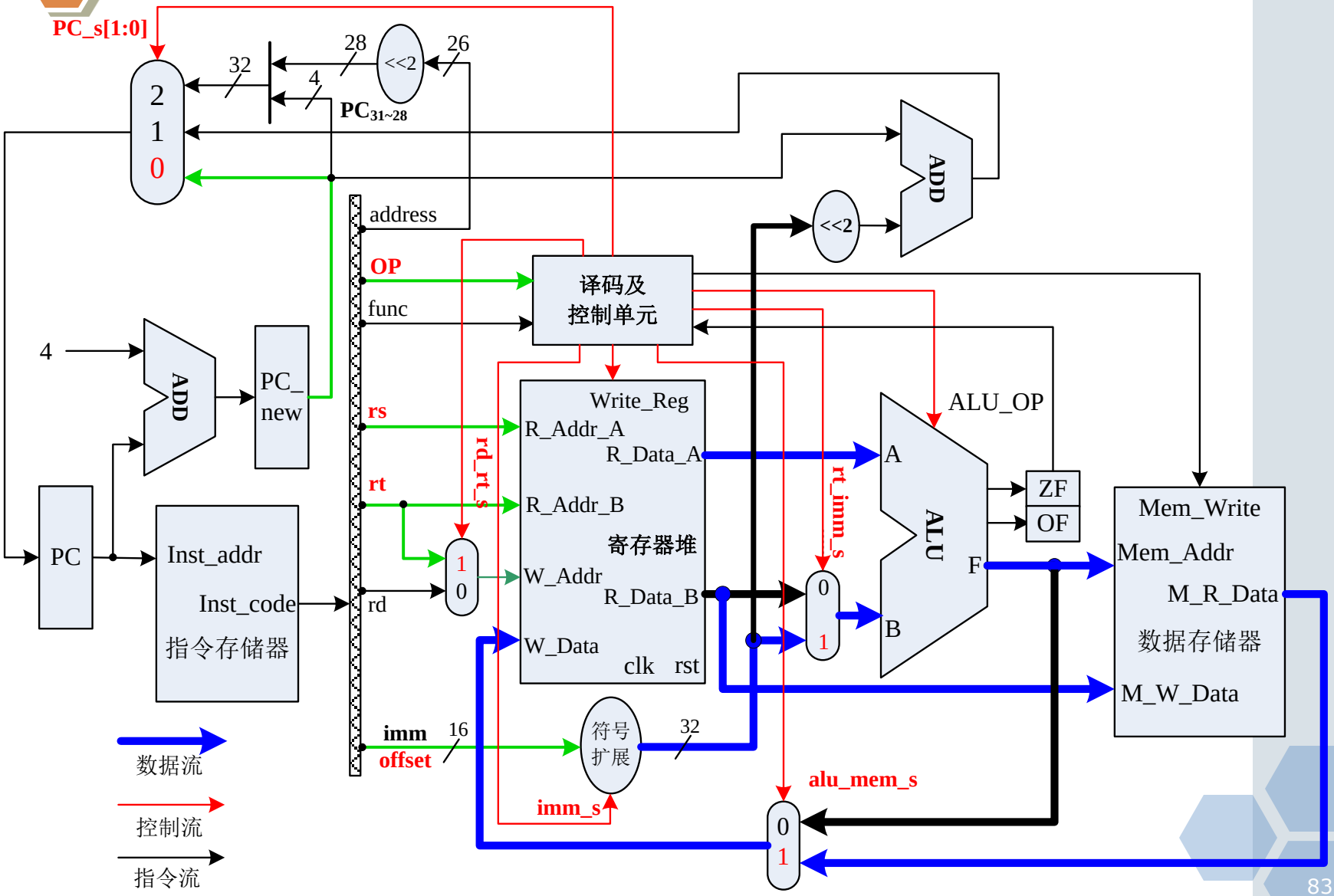
- func=100000：置ALU_OP=100，做算术加操作

1、R型指令 –举例：add $s0,$t0,$t1

❖ 彩色的线构成数据通路



2、I型访存指令 –数据通路



2、I型访存指令 –执行过程

（1）根据PC从指令存储器取出指令，PC自增

- 得到I_mem[PC]，（PC）+4

（2）译码及控制单元工作，置各控制信号状态；读寄存器值

❖取数指令：lw(OP=100011)

- 译码：置rd_rt_s=1,imm_s=1, rt_imm_s=1, alu_mem_s=1, Write_reg=1, ALU_OP=100, Mem_write=0;
- 读寄存器：rs的值送ALU;

❖存数指令：sw(OP=101011)

- 译码：置imm_s=1,rt_imm_s=1, Write_reg=0, ALU_OP=100, Mem_write=1;
- 读寄存器： rs的值送ALU，rt的值送数据存储器

（3）ALU执行算术加运算操作，计算EA；

❖ 取数指令lw：

（4）数据存储器执行读操作

（5）写入rt指向的寄存器

❖ 存数指令sw：

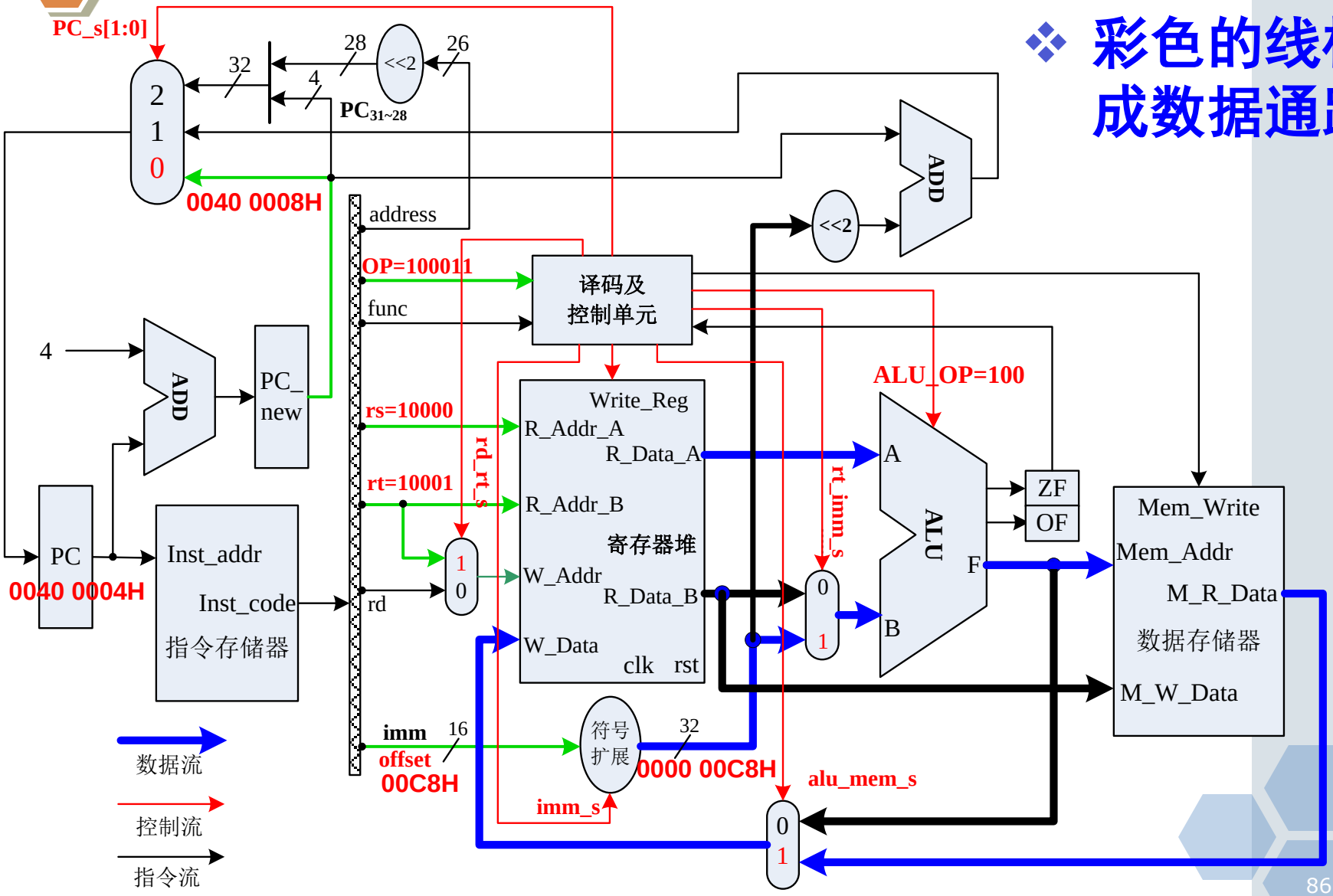
（4）数据存储器执行写操作

2、I型访存指令 –举例：lw、sw

助记符	指令格式（位）			
	31..26	25..21	20..16	15.. 0
I型指令	op	rs	rt	offset/imme
lw rt, offset(rs)	100011	rs	rt	offset
lw $s1,200($s0)	100011	10000	10001	0000 0000 1100 1000
sw rt, offset(rs)	101011	rs	rt	offset
sw $s2,300($t3)	101011	01011	10010	0000 0001 0010 1100

- EA=(rs)+offset：ALU_OP=100，做算术加操作

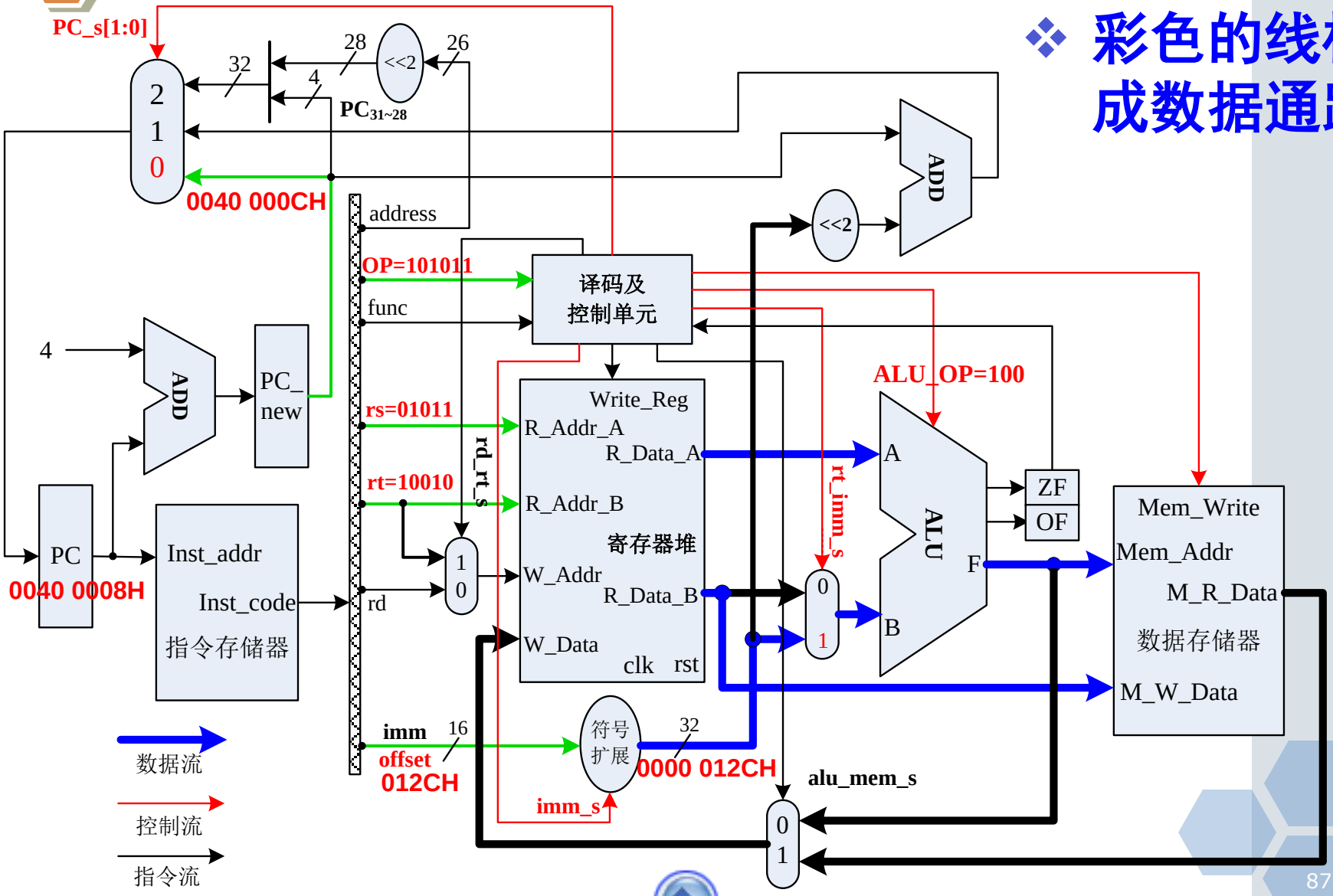
2、I型访存指令 –举例：lw $s1,200($s0)



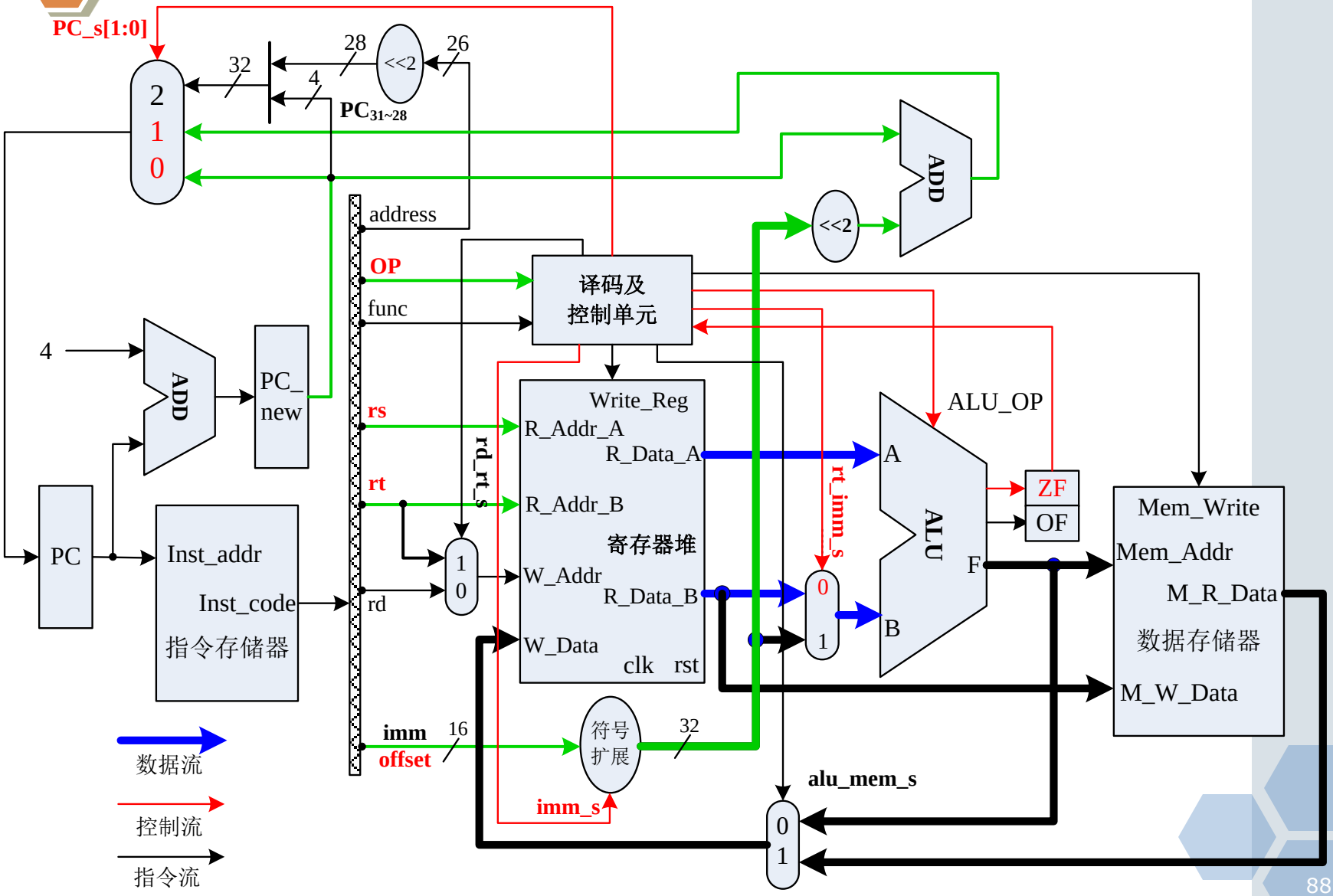
❖ 彩色的线构成数据通路

2、I型访存指令 –举例：sw $s2,300($t3)

❖ 彩色的线构成数据通路



3、I型分支指令 –数据通路(beq)



3、I型分支指令 –执行过程

（1）根据PC从指令存储器取出指令，PC自增

- 得到I_mem[PC]，（PC）+4

（2）译码及控制单元工作，置各控制信号状态；读寄存器值

- 译码：置imm_s=1,rt_imm_s=0, ALU_OP=101, Write_reg=0,Mem_write=0;
- 读寄存器：rs、rt的值送ALU;

（3）ALU执行算术减操作，比较(rs)和(rt)，并修改ZF；地址加法器执行加法，计算转移目标地址

- ALU执行(rs)-(rt)操作，根据结果置ZF标志(=0，则置ZF=1，≠0，则置ZF=0)
- 地址加法器执行(PC+4)+(offset*4)操作，计算转移目标地址

（4）根据ZF标志，选择PC+4(ZF=0)或者转移目标地址(ZF=1)修改PC值。

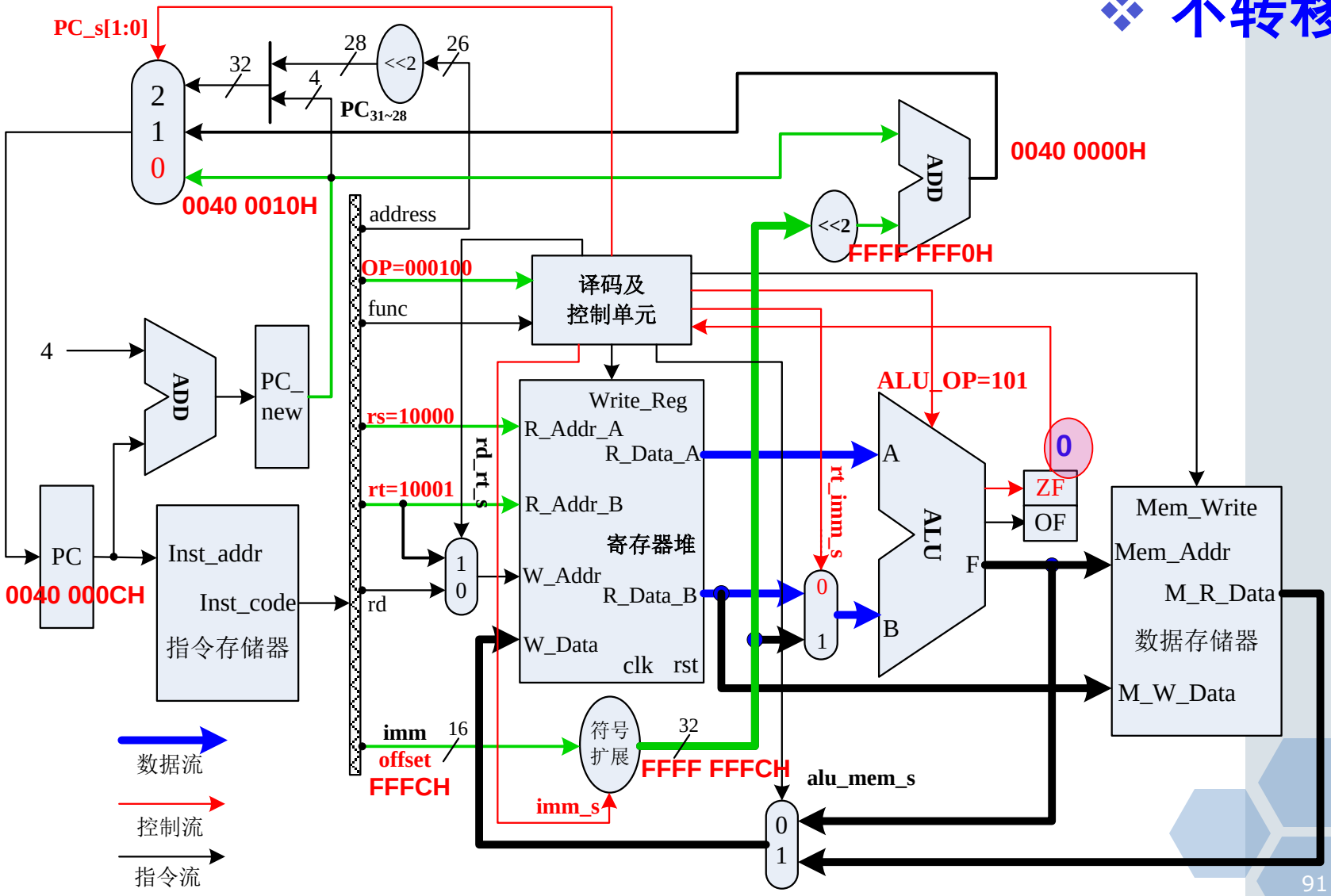
3、I型分支指令 –举例：beq $s0,$s1,offset

助记符	指令格式（位）			
	31..26	25..21	20..16	15.. 0
I型指令	op	rs	rt	offset/imme
beq rs,rt, offset	000100	rs	rt	offset
beq $s0, $s1,L	000100	10000	10001	1111 1111 1111 1100

- Offset的计算：当前指令的下一条指令与转移目标指令之间的偏移量（指令条数）；
- 譬如：L的地址是0040 0000H，而PC+4是0040 0010，偏移量是-10H，每条指令4B，即偏移指令条数是：-10H÷4=-4，所以偏移量=-4,用16位补码表示= 1111 1111 1111 1100

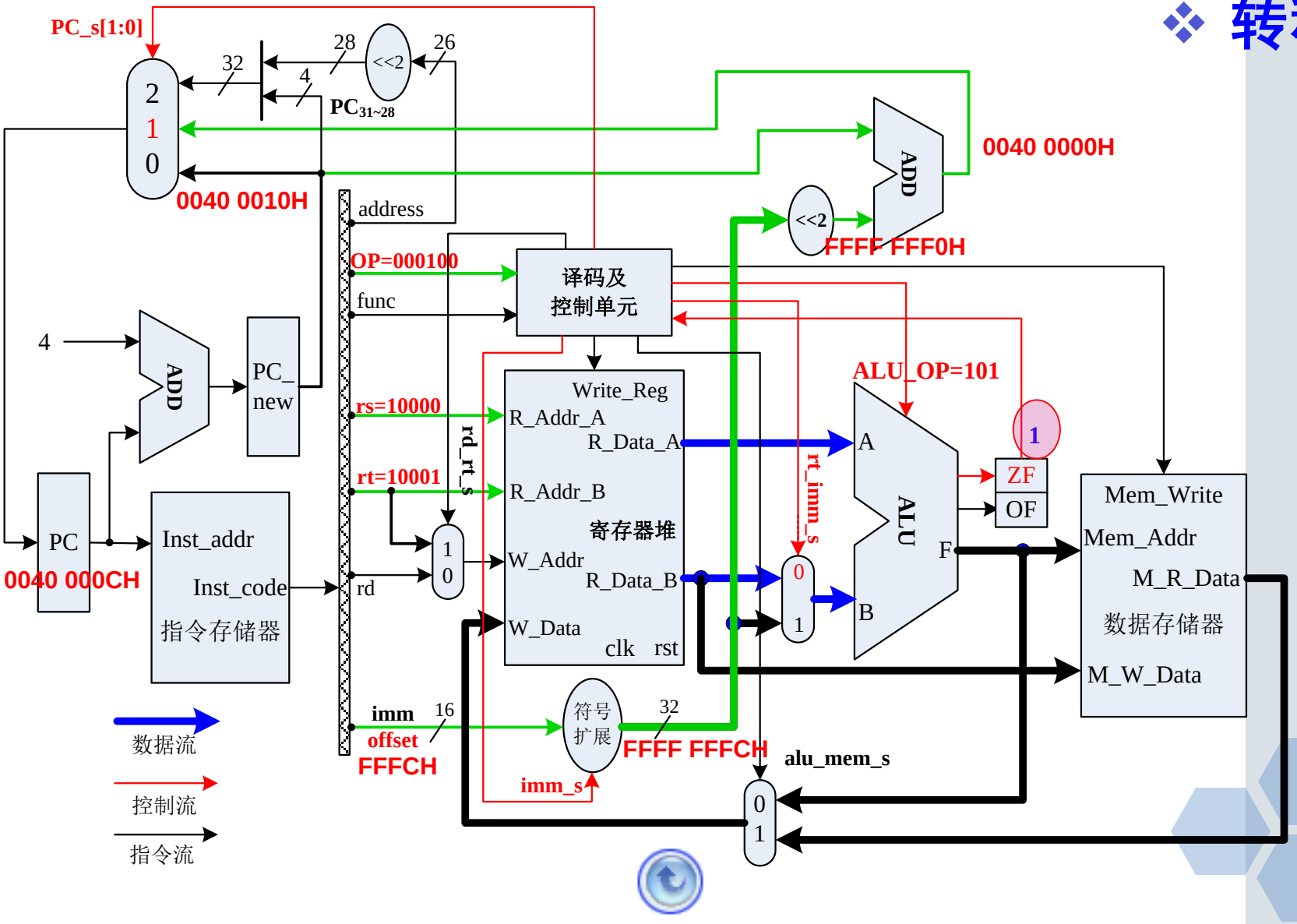
3、I型分支指令 –举例：beq $s0,$s1,offset

❖ 不转移

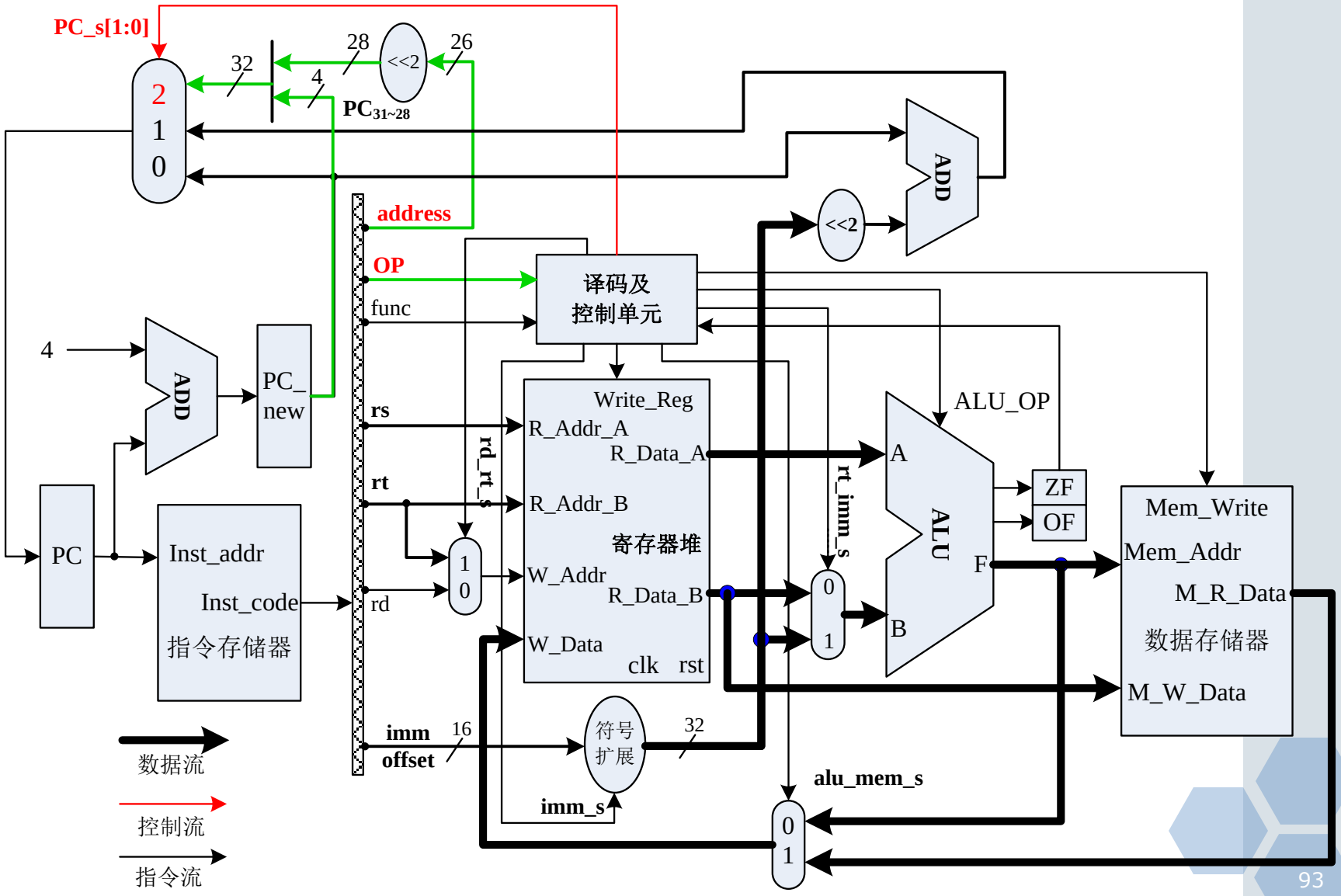


3、I型分支指令 –举例：beq $s0,$s1,offset

❖ 转移



4、J型跳转指令 –数据通路（J L）



4、J型跳转指令 –执行过程

（1）根据PC从指令存储器取出指令，PC自增

- 得到I_mem[PC]，（PC）+4

（2）译码及控制单元工作，置各控制信号状态；转移地址置PC

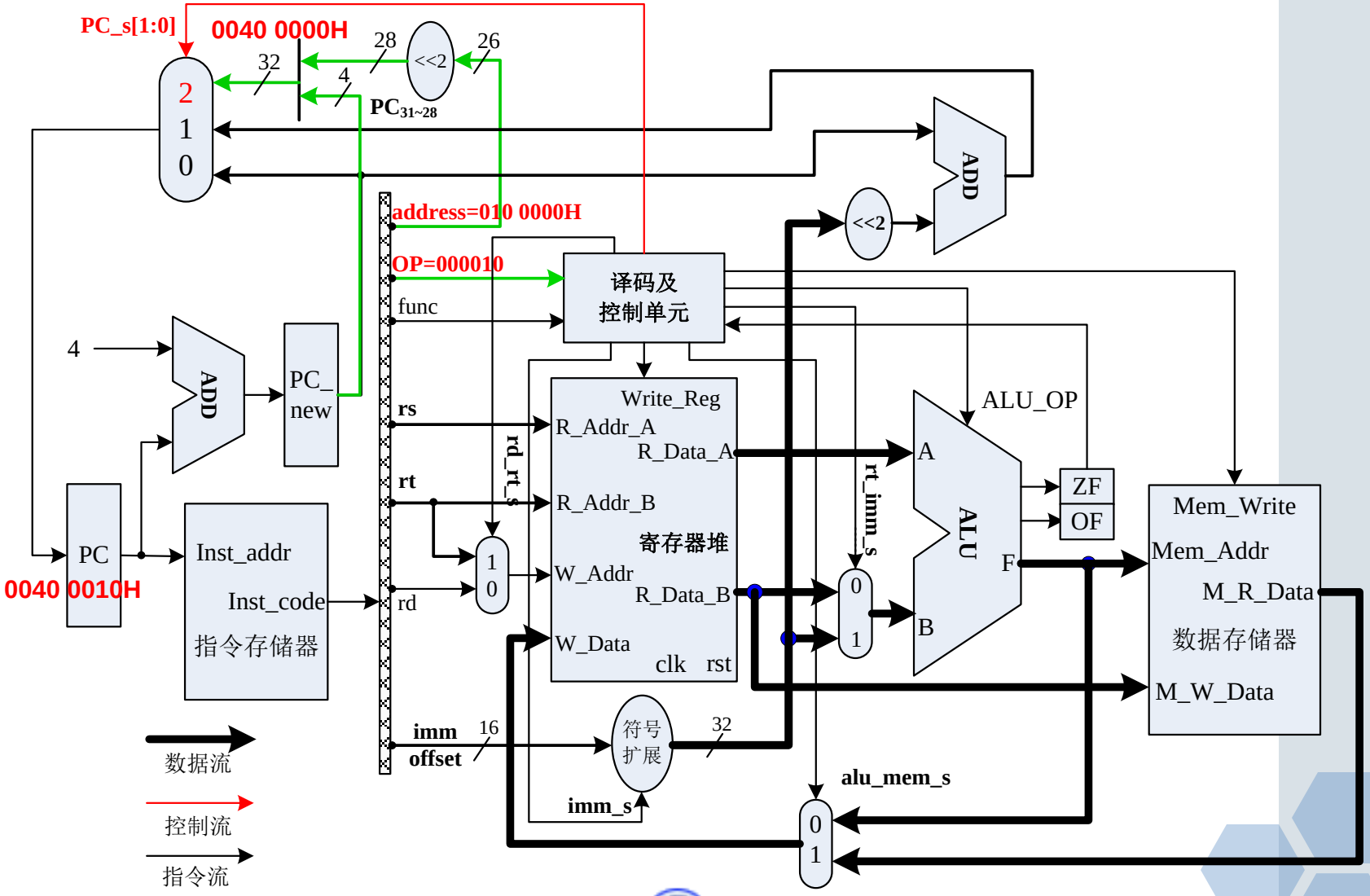
- 译码：置PC_s=10, Write_reg=0,Mem_write=0;
- 转移： {PC[31:28],address,2’b00} →PC

4、J型跳转指令 –举例：J L

助记符	指令格式（位）	
	31..26	25..0
J型指令	op	address
J Label	000010	address
J L	000010	0000 0100 0000 0000 0000 0000 00

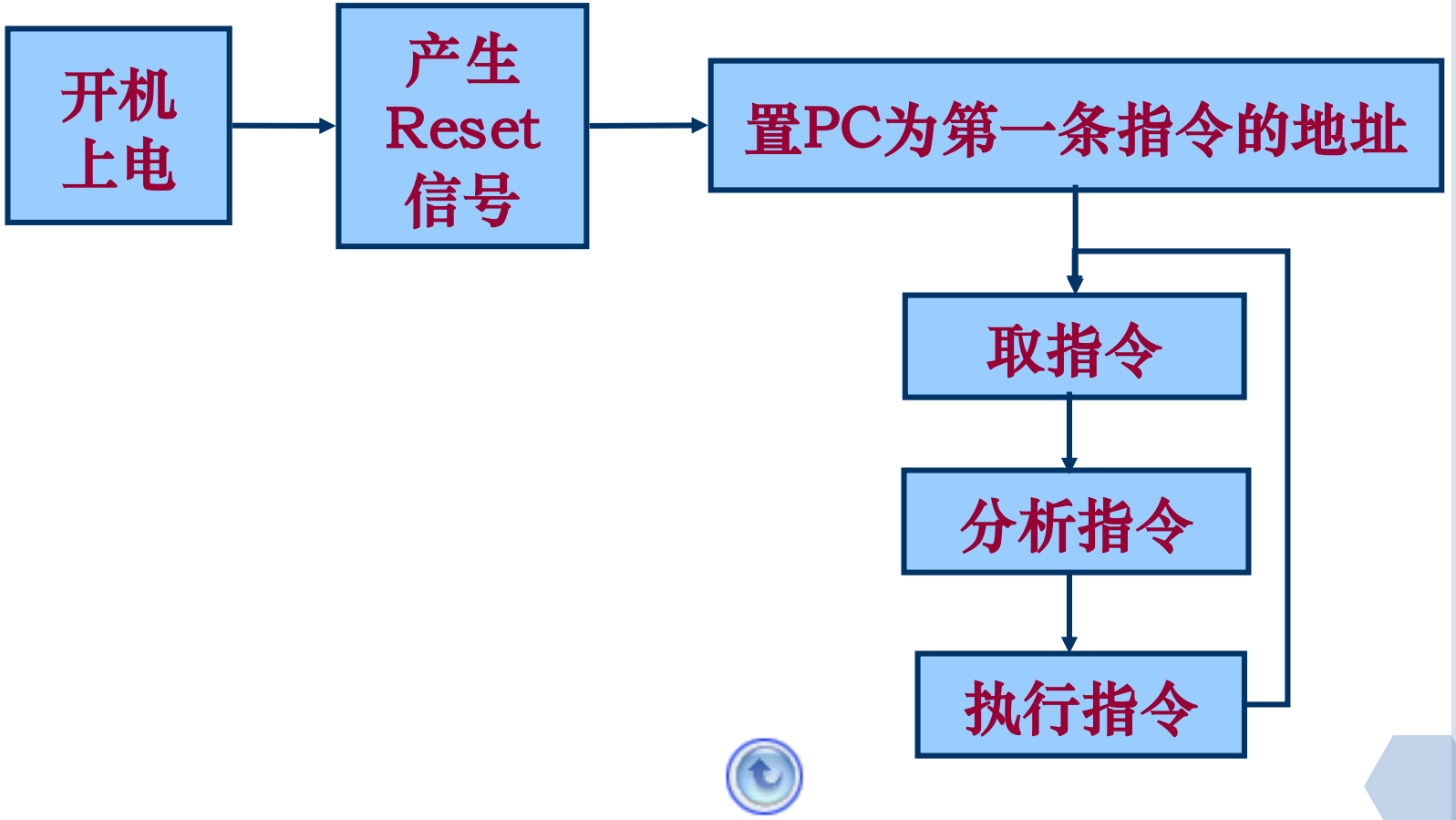
- address的计算：目标指令地址的第27位到第2位，即[27..2]；
- 譬如：L的地址是0040 0000H，address= 0000 0100 0000 0000 0000 0000 00 。
- 注意：J指令的转移目标地址必须与J指令本身在同一页面内（就是指令地址的高4位相等）。

4、J型跳转指令 –举例：J L



（三）计算机的工作过程

- 计算机的工作过程即是循环往复的取指令、分析指令、执行指令的过程。



The End!