计算机组成原理

Computer Organization

2022 . 秋

西安交通大学 计算机科学与技术学院

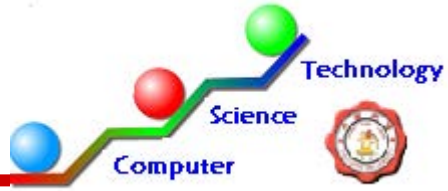
计算机组成原理课程组

http://corg.xjtu.edu.cn

计算机组成原理

第六章 中央处理器

目录



第六章 中央处理器

CPU的功能

CPU是计算机中执行程序的部件，具有五大功能：

- **顺序控制**

  对程序中指令的执行顺序加以控制

- **操作控制**

  按照指令功能，向相应部件发出操作控制信号

- **时间控制**

  对各种操作执行的时间进行定时

- **数据加工**

  对数据进行各种运算处理

- **中断控制**

  对程序中断进行开、关，检测、响应等

CPU的基本组成

CPU由运算器和控制器组成，有五个基本功能部件：

- **算术逻辑单元ALU**

  执行算术/逻辑运算

- **寄存器组**

  存放初始数据和运算结果数据

- **内部总线**

  连接CPU内各部件的数据通路

- **中断系统**

  接收中断请求并响应中断

- **控制单元CU**

  发出全机运行所需要的全部控制信号

CPU的基本组成（续）

CPU分为两个互相关联的部分：数据通路（Datapath）和控制单元（Control Unit，CU）。

- **数据通路**

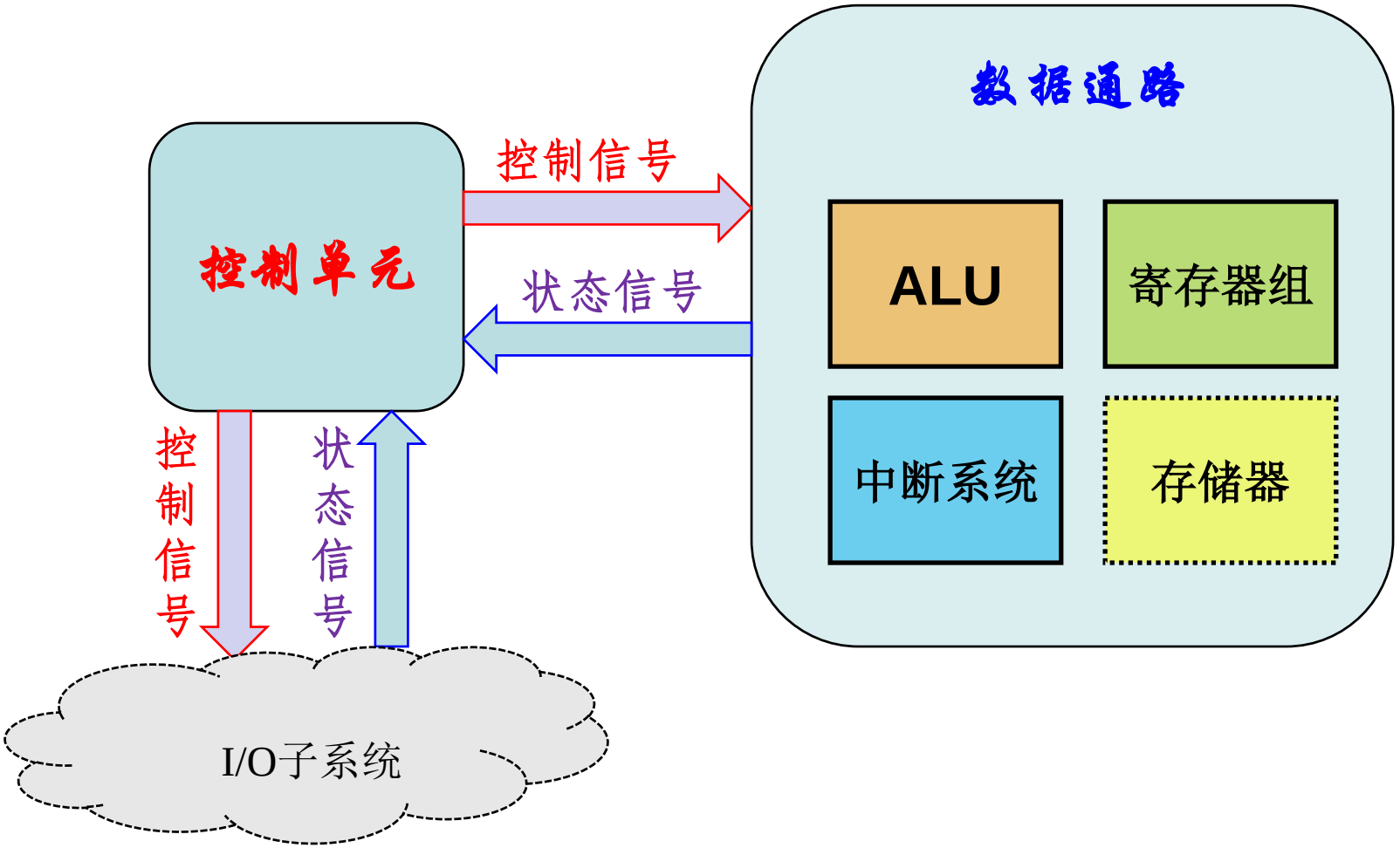
  CPU中全部执行部件的组合，包含了寄存器、ALU和中断处理子系统等。数据通路是指令执行的基础。

- **控制单元**

  CU是计算机的指挥系统，决定在什么时间根据什么条件应该做什么事情，即产生计算机运行所需要的全部控制命令。

主存储器虽然不位于CPU中，但是由于CPU在执行程序过程中要不断地访问主存以获取指令和数据，所以描述CPU数据通路时往往也要把主存描述出来。

CPU的基本组成（续）



微操作和微命令

计算机中的操作是具有层次性的，即一个较大的操作可被分解为若干个较小的操作，如此分层直到不可分解为止。

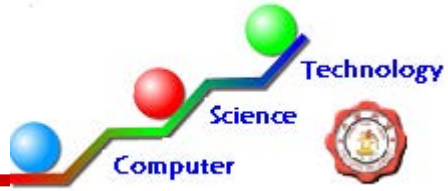
- **微操作**

  指计算机中最基本的不可再分解的操作。例如，逻辑线路中的开/关设置、寄存器数据打入和ALU基本运算等。

- **微命令**

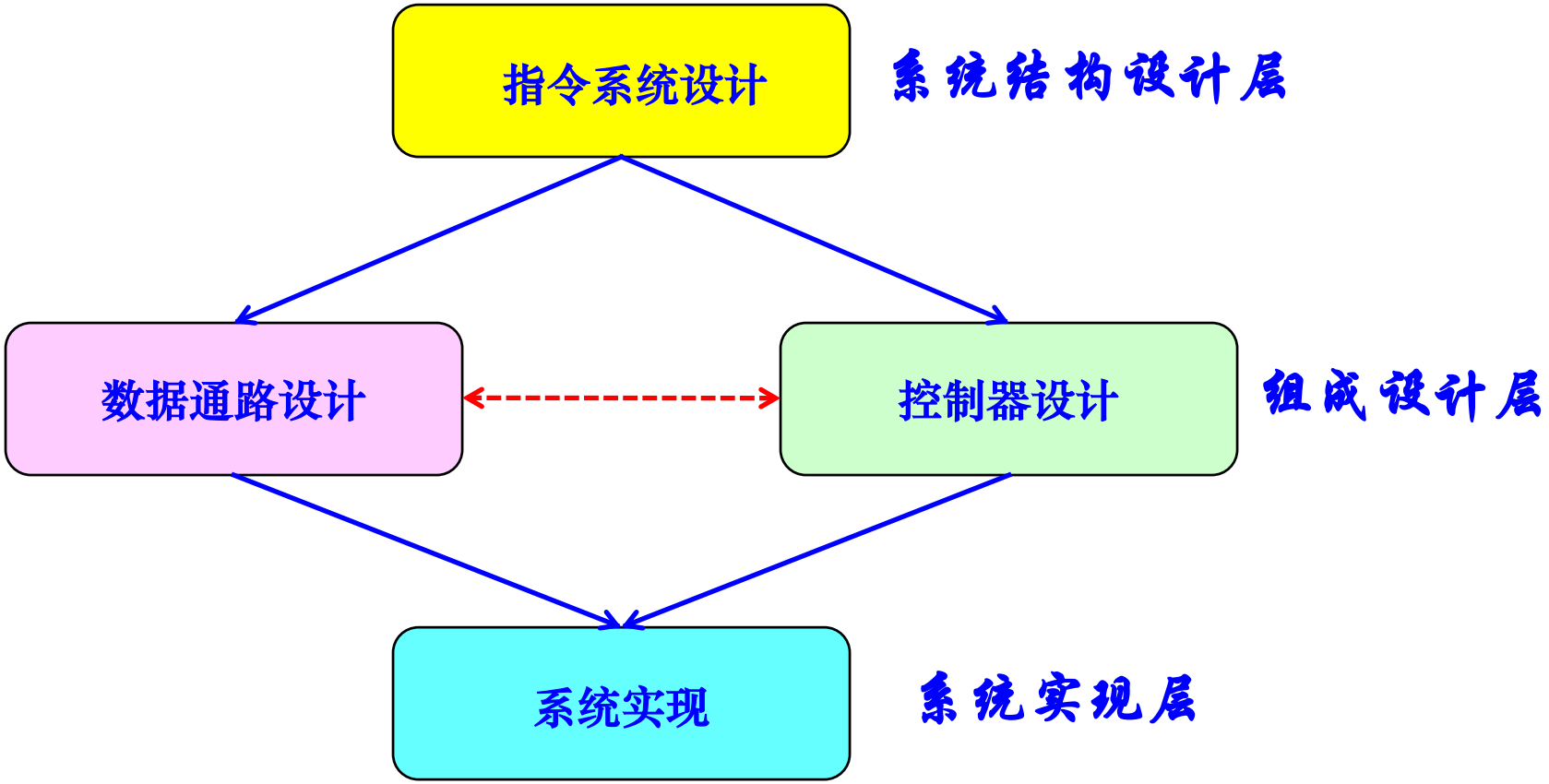
  执行微操作所需要控制信号称为微命令。在微命令的作用下，某个微操作在某个确定的时刻、由某个确定的部件完成。例如，寄存器写入、存储器读/写（-WE）等。

目录

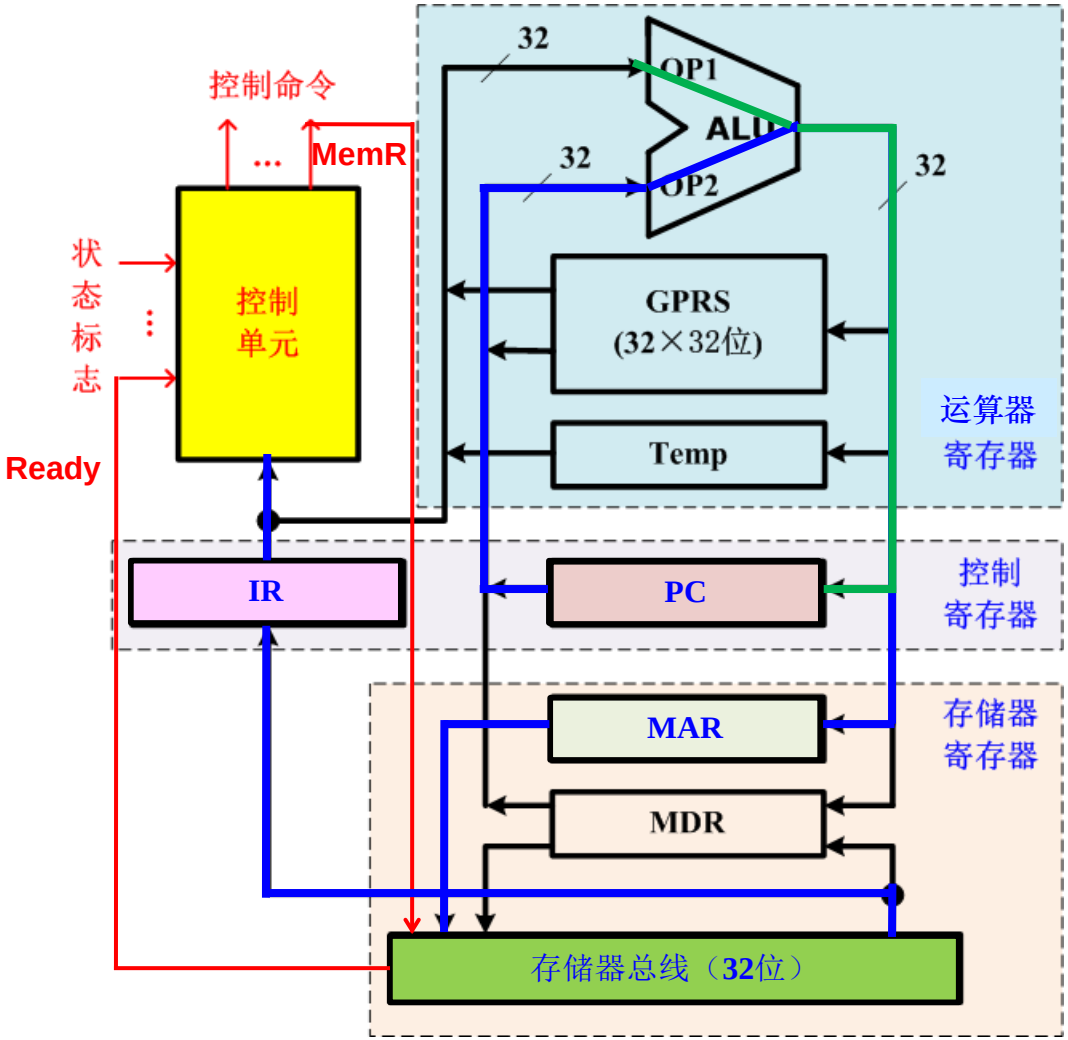


第六章 中央处理器

CPU设计过程



寄存器传输语言RTL



模型机数据通路

取指令操作序列

寄存器传输语言RTL（续）

❍主存储器与CPU异步

MAR ← (PC)

MemR ← 1

PC ← (PC) + 1

Wait until Ready=1

IR ← M[MAR]

MemR ← 0

控制命令
或
状态反馈

寄存器传输语言RTL（续）

❍主存储器与CPU异步

MAR ← (PC)

PC ← (PC) + 1

IR ← M[MAR] ; MemR=1, *Until* Ready=1

寄存器传输语言RTL（续）

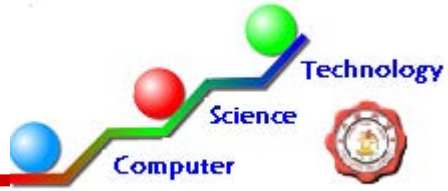
- 主存储器与CPU同步

MAR ← (PC)

PC ← (PC) + 1

IR ← M[MAR] ； MemR=1

目录



第六章 中央处理器

数据通路操作分析

通过分析指令周期操作过程，确定数据通路基本组成。

❍取指令

该阶段的任务是将一条指令从主存取出送到CPU。

✧CPU中设置程序计数器PC，将PC的内容作为地址读取主存单元的内容；

✧将取出的指令存入CPU中的指令寄存器IR；

✧修正PC的内容使其指向下一条指令。

❍指令译码

该阶段的任务是分析指令的操作码以及寻址特征，确定指令功能、操作数类型以及寻址方式等。

✧设置一个或多个译码器完成指令译码。

数据通路操作分析（续）

❍读取数据

该阶段的任务是将指令要处理的源操作数读到CPU。

- ✧RR型指令，源操作数在通用寄存器中；
- ✧RS型指令，源操作数在通用寄存器或存储器中；
- ✧设置存储器地址寄存器MAR存放存储器地址；
- ✧设置存储器数据寄存器MDR存放存储器数据；
- ✧有效地址计算时可能需要加法器。

数据通路操作分析（续）

❍执行操作

该阶段的任务是完成指令的具体功能。算术逻辑运算单元ALU的基本功能包括：

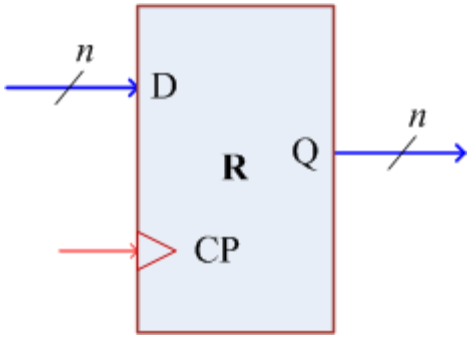
- ✧定点数四则运算；
- ✧浮点数四则运算；
- ✧逻辑运算；
- ✧移位运算。

❍存放结果

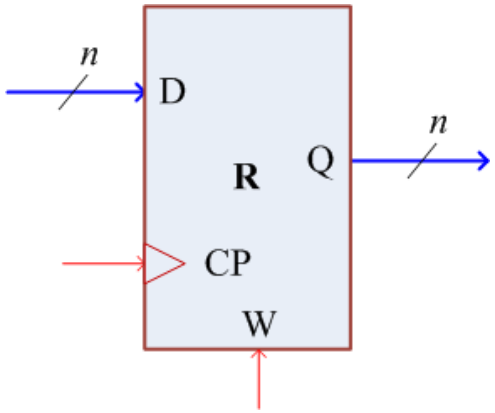
该阶段的任务是保存处理后的数据（目的操作数）。

- ✧RR型，目的操作数存放在通用寄存器；
- ✧RS型，目的操作数存放在通用寄存器或存储器。

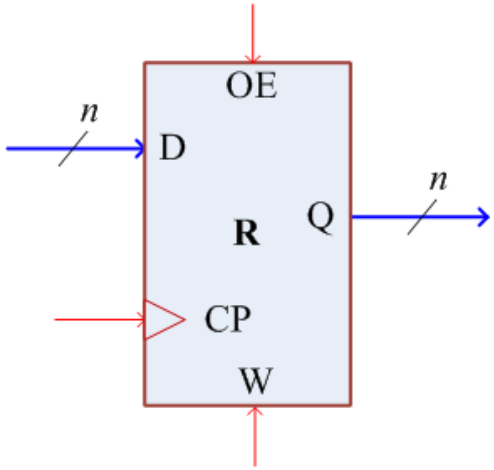
❍寄存器



(a) n位寄存器



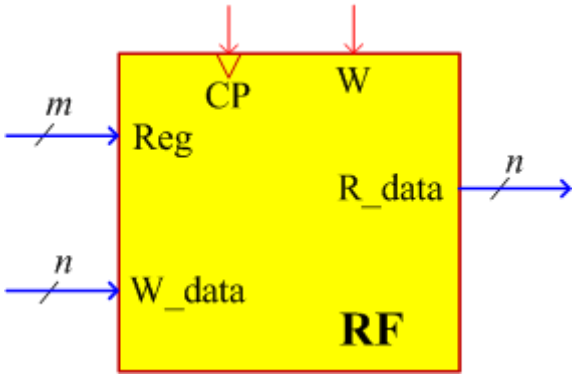
(b) 带写入控制n位寄存器



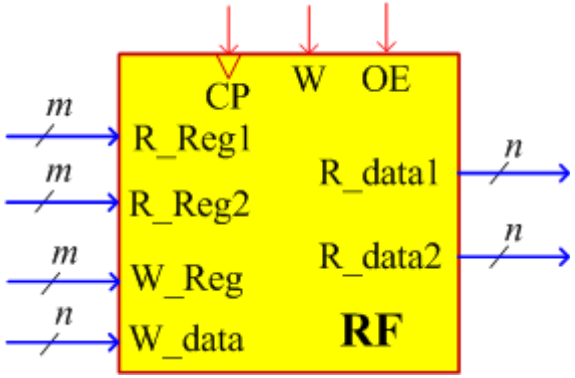
(c) 带写入控制和输出使能n位寄存器

数据通路基本部件（续）

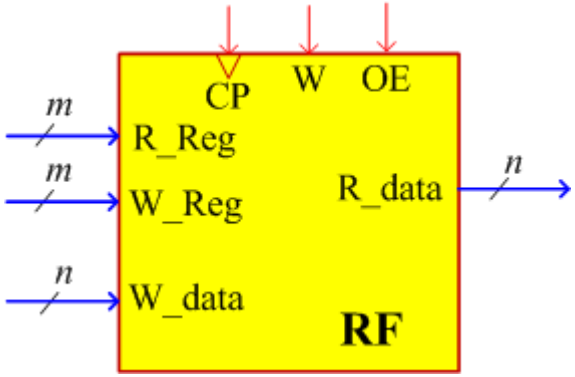
❍通用寄存器组（Register File，RF）



(a) RF



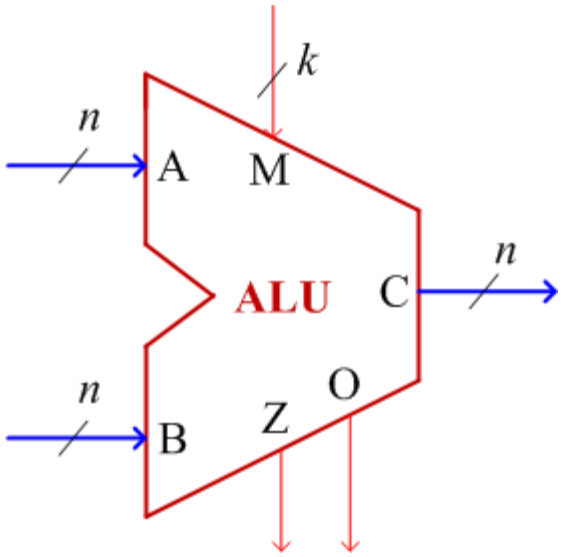
(c) 2R+1W RF



(b) 具有输出使能的1R+1W RF

数据通路基本部件（续）

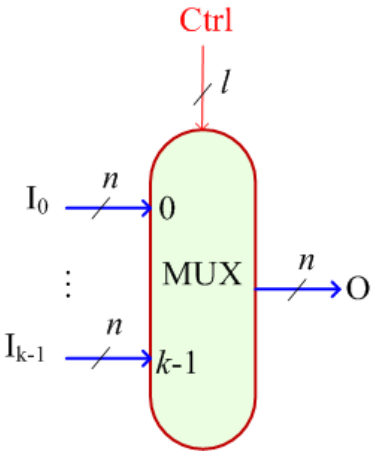
❍ALU



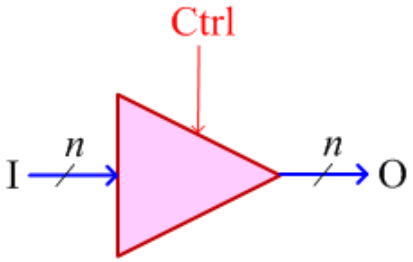
M	操作功能
000	AND
001	OR
010	ADD
011	SUB
100	NOR
101	NOT

数据通路基本部件（续）

- 多路选择器（MUX）



- 三态控制器

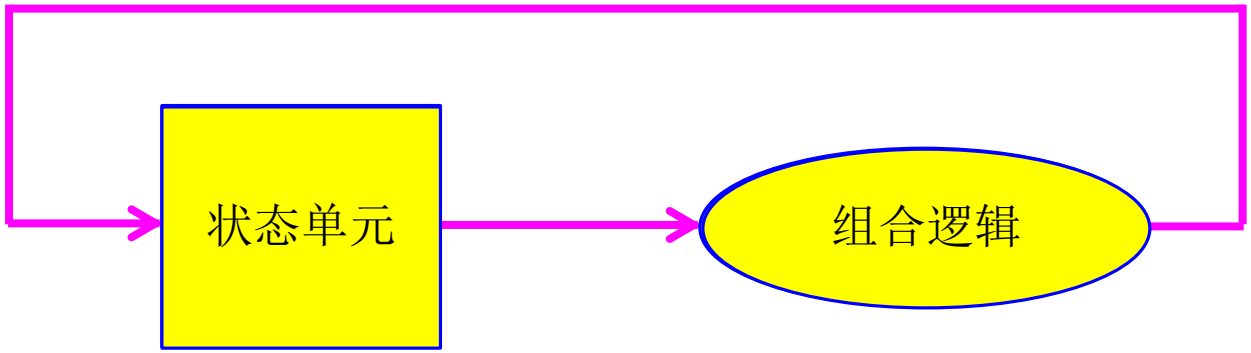


CPU结构分类

根据指令周期和时钟周期的关系，CPU结构分为三类：

- 单周期CPU

  指令周期仅包含一个时钟周期，即指令周期=时钟周期。

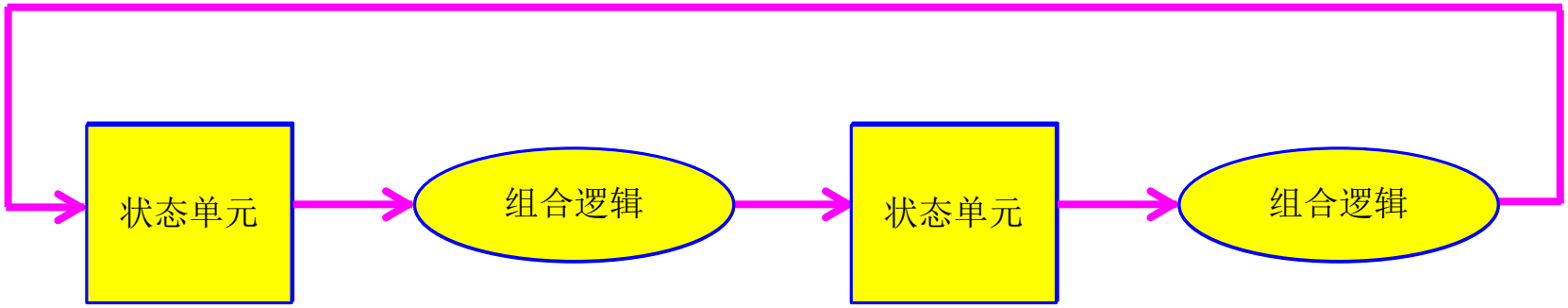


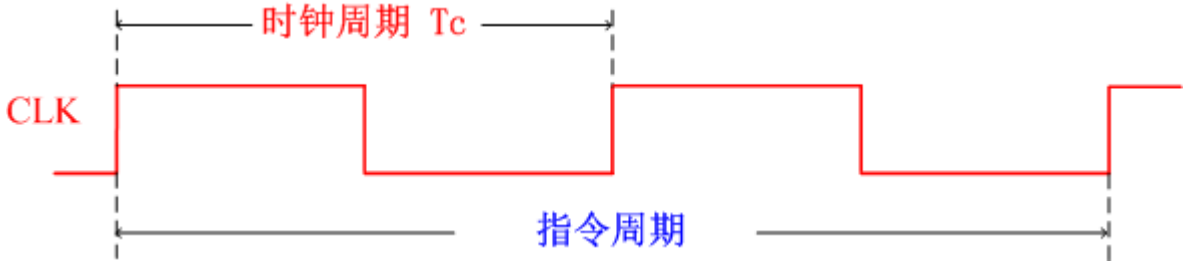
CPU结构分类（续）

❍多周期CPU

一个指令周期包含多个较短的时钟周期，即

指令周期 = *k* ×时钟周期。





CPU结构分类（续）

❍流水CPU

- ✧数据通路类似于单周期，时序控制上类似于多周期。
- ✧利用单周期数据通路中的部件冗余技术，但指令周期由多个时钟周期组成，每个时钟周期操作结束后缓存其结果。
- ✧多条指令在数据通路上重叠执行。
- ✧相对于多周期CPU来说，指令周期长度不变，但程序运行时间缩短。

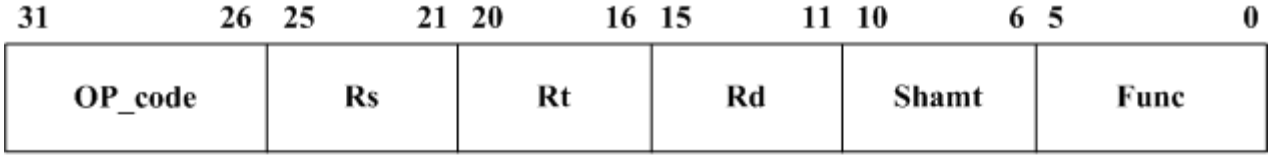
目标指令

以实现 MIPS 32 指令集中的 10 条指令为例，讲解数据通路的构建方法，以及基于不同数据通路的指令执行过程。

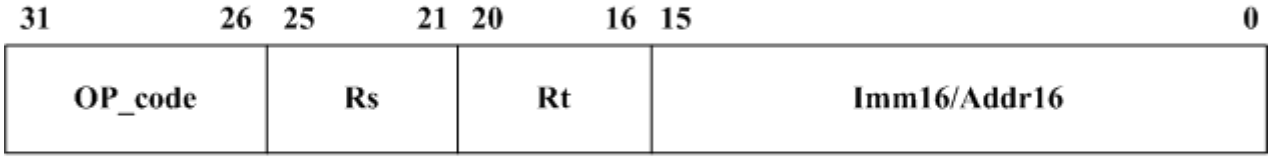
指令类型	指令	指令格式类型	指令功能描述	
			取指令阶段	执行指令阶段
存储器访问指令	lw Rt, Imm16(Rs)	I-型	（1）M[PC] 或者 IR←M[PC] （2）PC←(PC)+ 4	（3）addr←(Rs)+ SigExt(Imm16) （4）Rt←M[addr]
	sw Rt, Imm16(Rs)			（3）addr←(Rs)+ SigExt(Imm16) （4）M[addr]←(Rt)
算术/逻辑运算指令	add Rd, Rs, Rt	R-型		（3）Rd←(Rs)+(Rt)，判溢出
	sub Rd, Rs, Rt			（3）Rd←(Rs)-(Rt)，判溢出
	addu Rd, Rs, Rt			（3）Rd←(Rs)+(Rt)
	and Rd, Rs, Rt			（3）Rd←(Rs)∧(Rt)
	or Rd, Rs, Rt			（3）Rd←(Rs)∨(Rt)
	nor Rd, Rs, Rt			（3）Rd←(Rs)⊕(Rt)
程序转移指令	beq Rs, Rt, Addr16	I-型		（3）if ((Rs) == (Rt) PC← (PC)+ SigExt(Addr16)×4
	j Addr26	J-型		（3）PC←PC[31~28] ‖ (Addr26)×4

目标指令（续）

MIPS 32 格式如下：



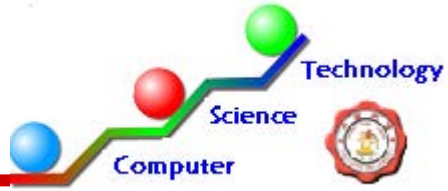
(a) R-型指令



(b) I-型指令

(c) J-型指令

目录



第六章 中央处理器

中断源分类

中断是现代计算机能有效、合理地发挥效能和提高效率的一个重要功能。实现这种功能需要软硬配合来完成，通常，把实现中断的软硬件称为中断系统，或中断技术。

- 中断源分类
  - 人为设置中断
  - 程序性异常
  - I/O设备
  - 硬件故障

中断源分类（续）

- 中断系统：由两部分组成
  - 中断软件
    - 中断初始化程序
    - 中断处理程序，或称中断服务程序
  - 中断硬件
    - 集中在CPU
    - 专门的中断控制器

本章仅以程序性异常中的未定义指令和算术溢出为例介绍。

中断响应

- 响应中断的条件
  - CPU允许中断：EINT=1
  - 有中断请求（未屏蔽）：$INTR_i$=1
- 响应中断的时间
  - 指令周期结束后
  - 原因：此时CPU现行程序的现场最简单、最稳定。
- 识别中断源
  - 外部中断源：由中断控制器完成
  - 内部中断源：通过不同的条件判断逻辑实现。

中断响应（续）

- 中断判优
  - 软件法：由CPU执行一段程序查询实现
  - 硬件法：若在CPU内部，采用并行判优逻辑
- 中断响应
  - CPU自动完成三个操作
    - 关中断
    - 保护程序断点
    - 获得中断服务程序入口地址
      - 软件查寻法
      - 硬件向量法

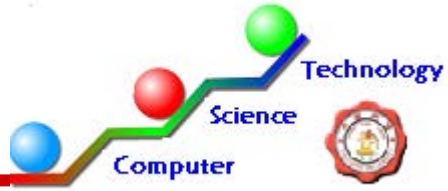
中断响应（续）

- MIPS 中的异常
  - 设置异常程序计数器（EPC），保存出错指令的地址。
  - 设置原因寄存器（Cause），记录中断的原因。

中断类型	寄存器值
硬件中断	0x00000000
系统调用	0x00000020
断点/除数为0	0x00000024
未定义指令	0x00000028
算术溢出	0x00000030

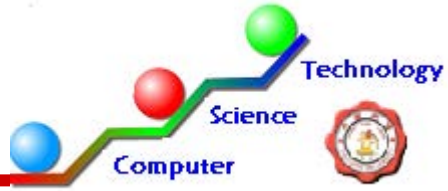
  - 异常发生后，程序转移到操作系统（地址：0x8000 0180）。根据Cause寄存器的值识别中断源，然后提供相应服务。服务结束后，从EPC获取返回地址。

目录



第六章 中央处理器

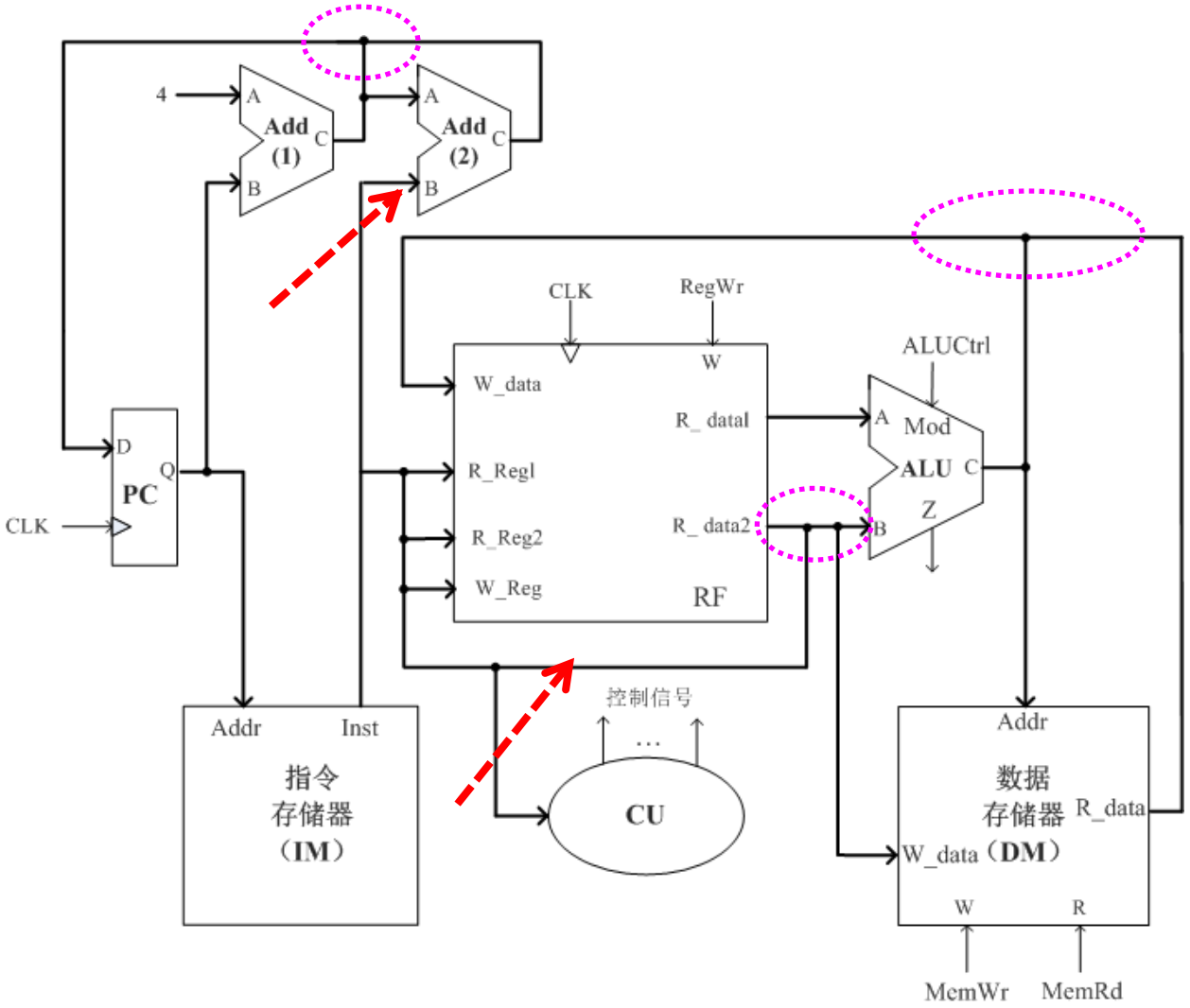
单周期CPU数据通路



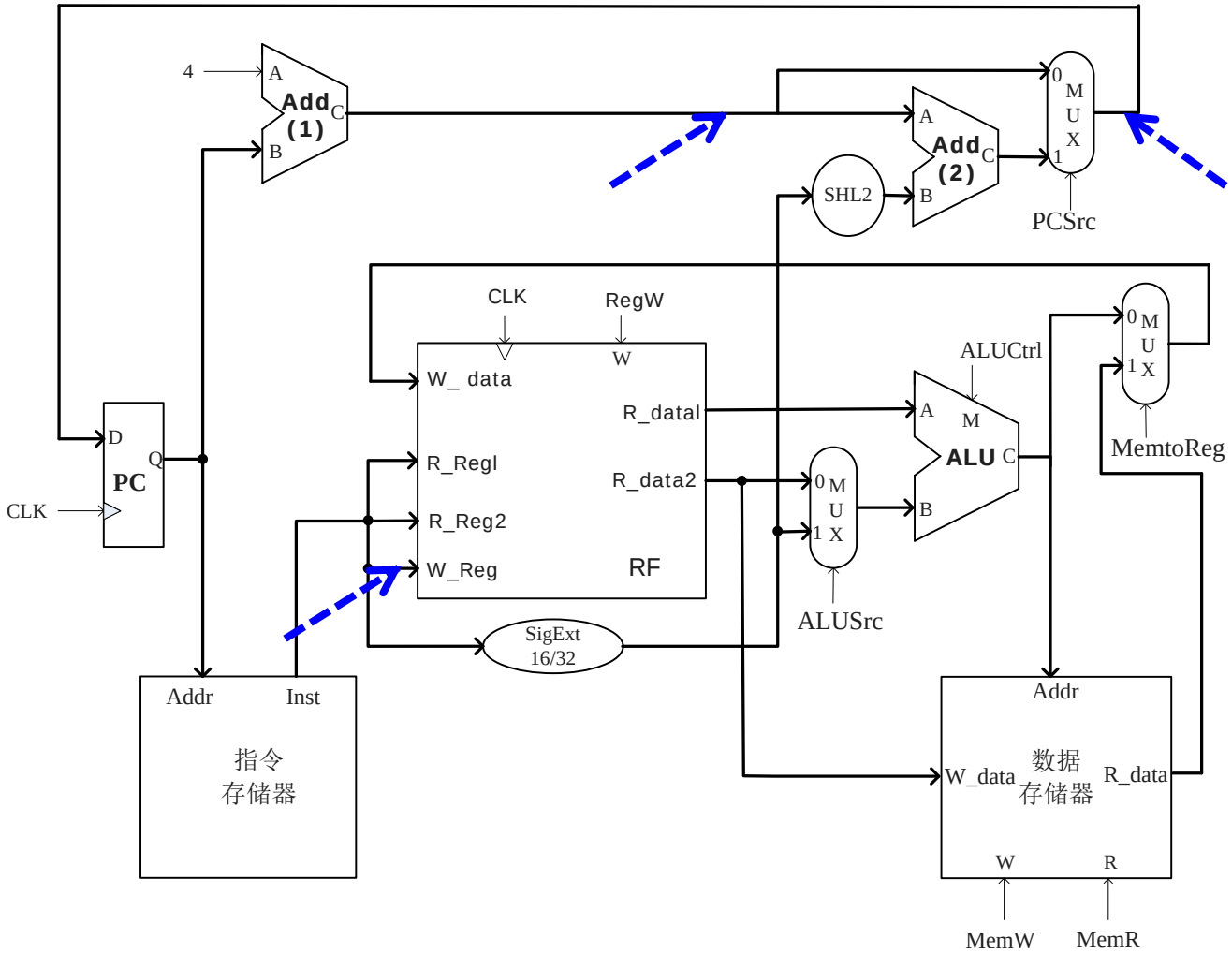
单周期数据通路的主要特点：部件冗余。

- 单周期数据通路的基本组成
  - 指令存储器IM，存放指令并执行读操作
  - 数据存储器DM，存放数据，并执行读或写操作
  - 程序计数器PC，存放即将执行指令的地址
  - 寄存器组（或称寄存器堆RF），临时存放数据或地址
  - 算术逻辑单元ALU，执行数据或地址运算
  - 加法器，对PC值进行修正(+4)

单周期CPU数据通路（续）



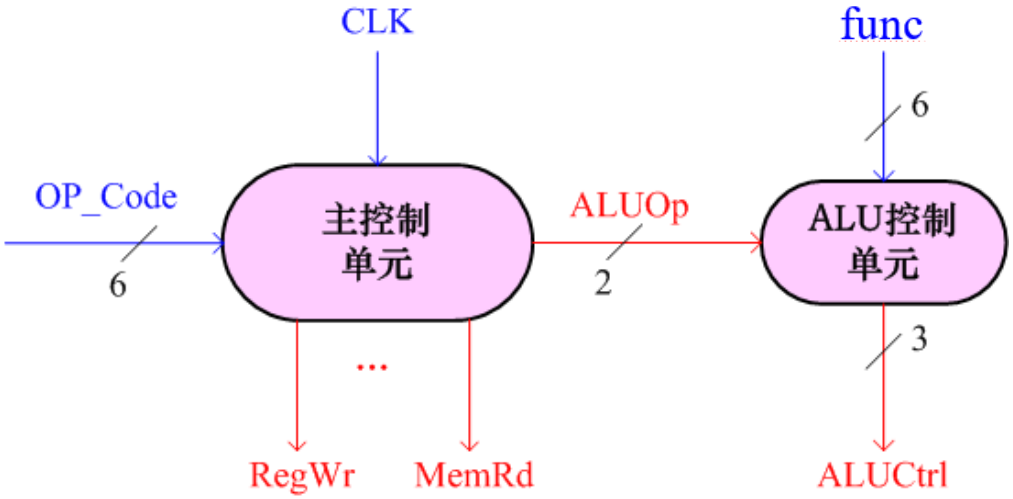
单周期CPU数据通路（续）



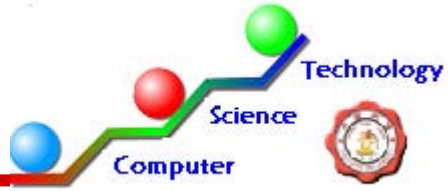
单周期CPU数据通路（续）

为了适应 MIPS指令格式，控制单元采用分散式设计方案。

- 控制单元组成
  - 主控制单元MCU：发出各部件所需的控制信号，包括ALU的操作类型控制信号。
  - ALU控制单元ALUCU：根据ALU的操作类型控制信号发出ALU的具体操作控制信号。



单周期CPU数据通路（续）



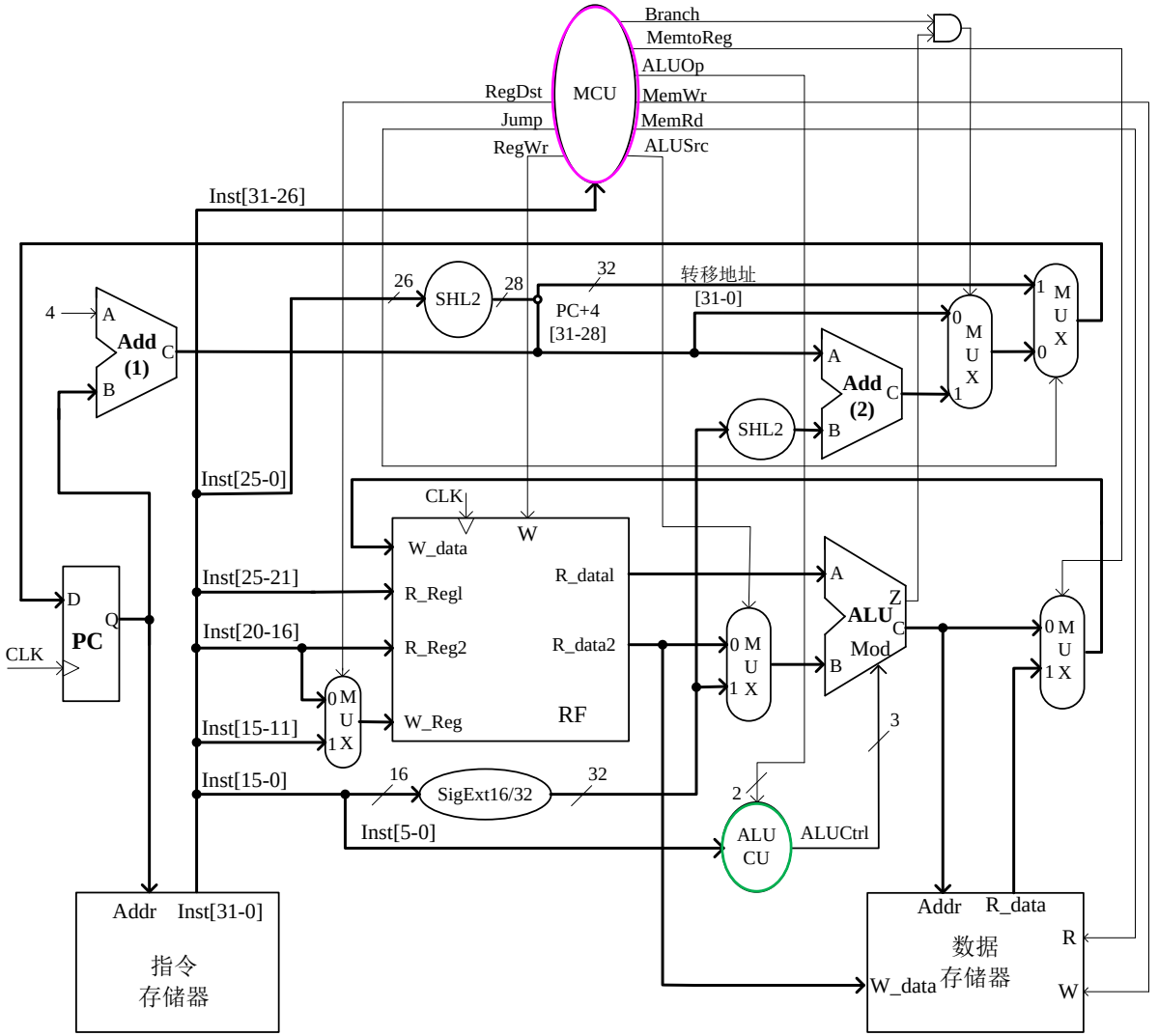
❒ 主控制单元真值表

指令	操作码	RegDst	RegWr	ALUSrc	MemRd	MemWr	MemtoReg	Branch	Jump	ALUOp
R-类型	000000	1	1	0	0	0	0	0	0	10
lw	100011	0	1	1	1	0	1	0	0	00
sw	101011	X	0	1	0	1	X	0	0	00
beq	000100	X	0	0	0	0	X	1	0	01
j	000010	X	0	X	0	0	X	0	1	XX

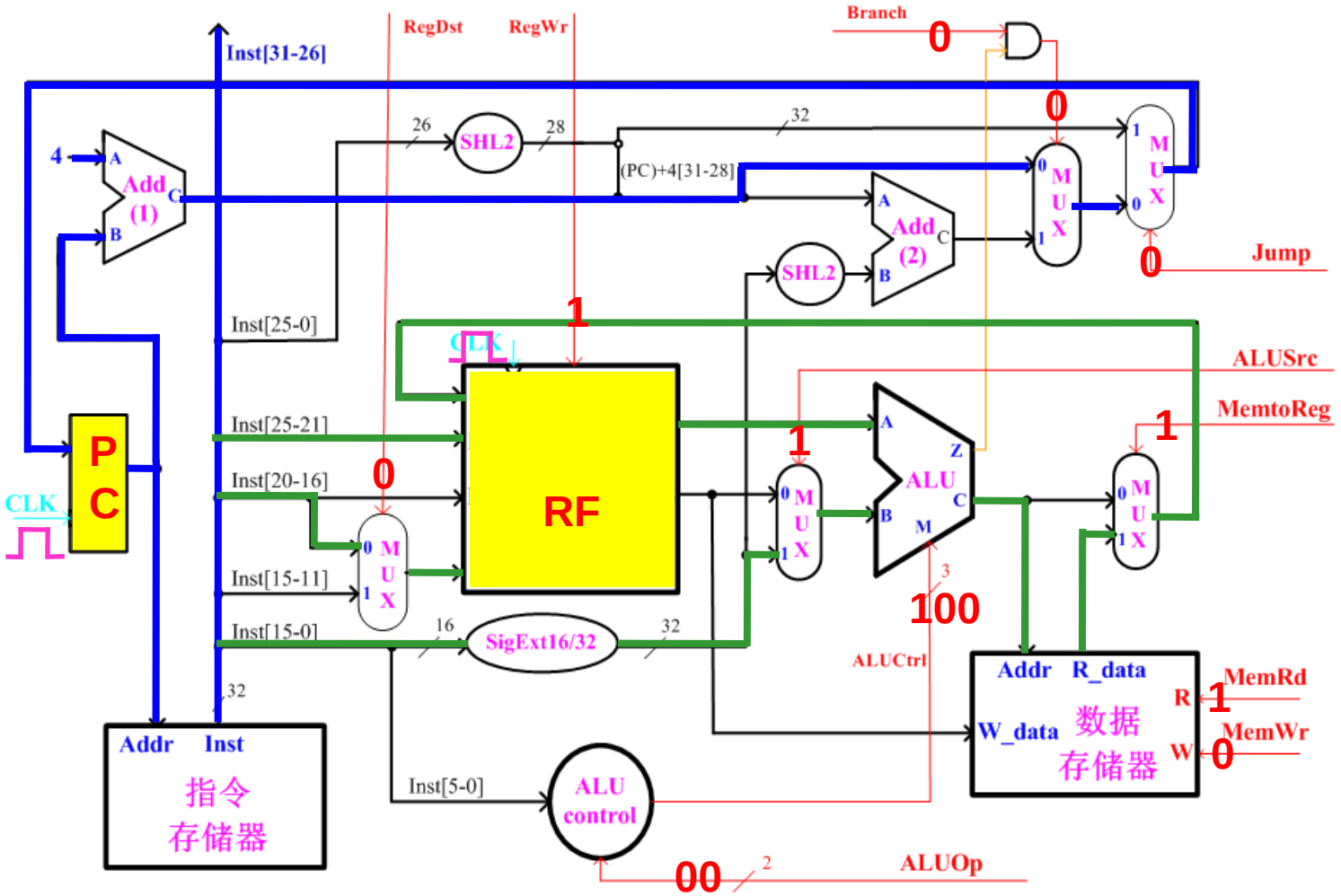
❒ ALU控制单元真值表

ALUOp	funct	ALUCtrl
00	X	100 (add)
01	X	110 (sub)
1X	100000	100 (add)
1X	100001	101 (addu)
1X	100010	110 (sub)
1X	100100	000 (and)
1X	100101	001 (or)
1X	101010	011 (nor)

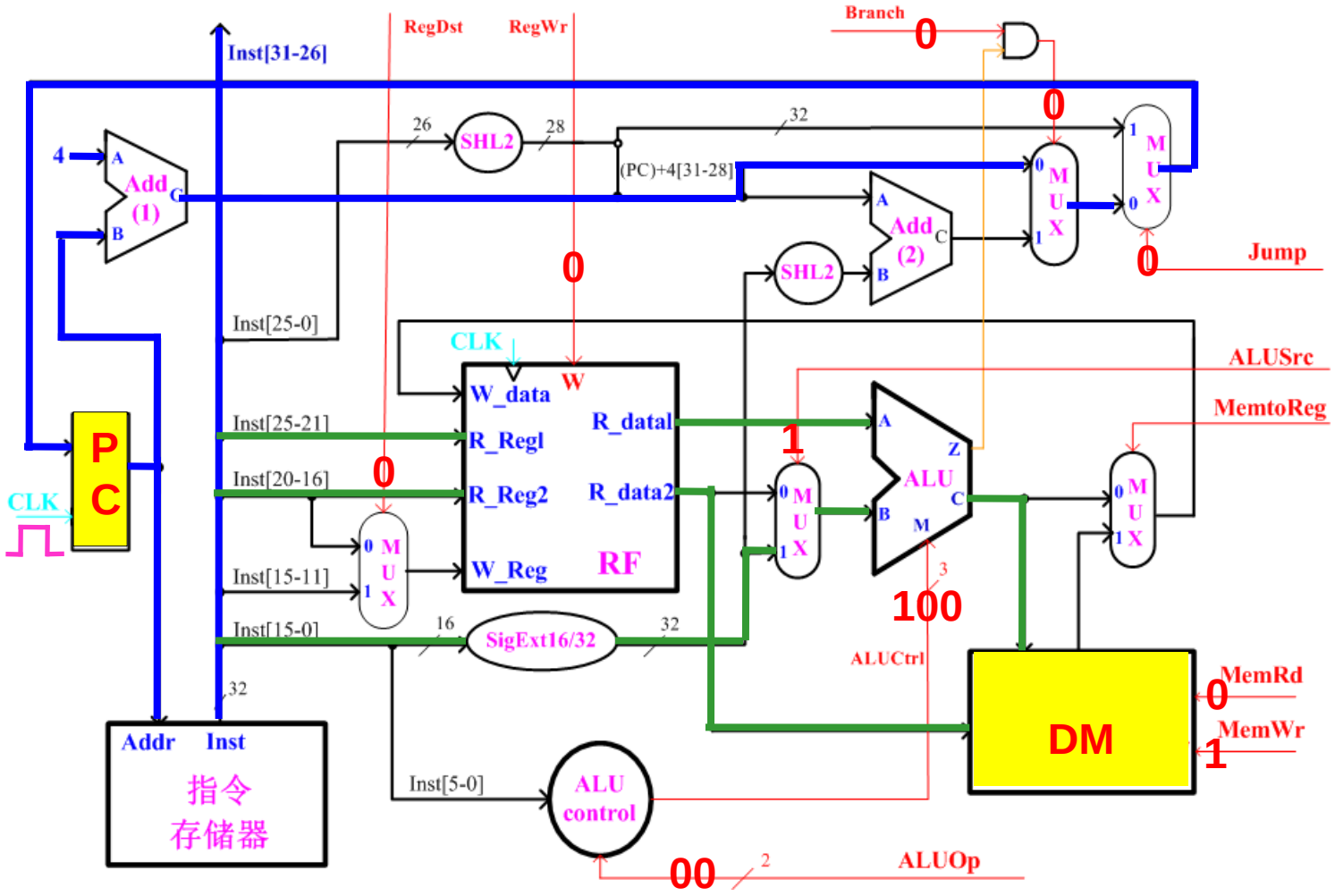
单周期CPU数据通路（续）



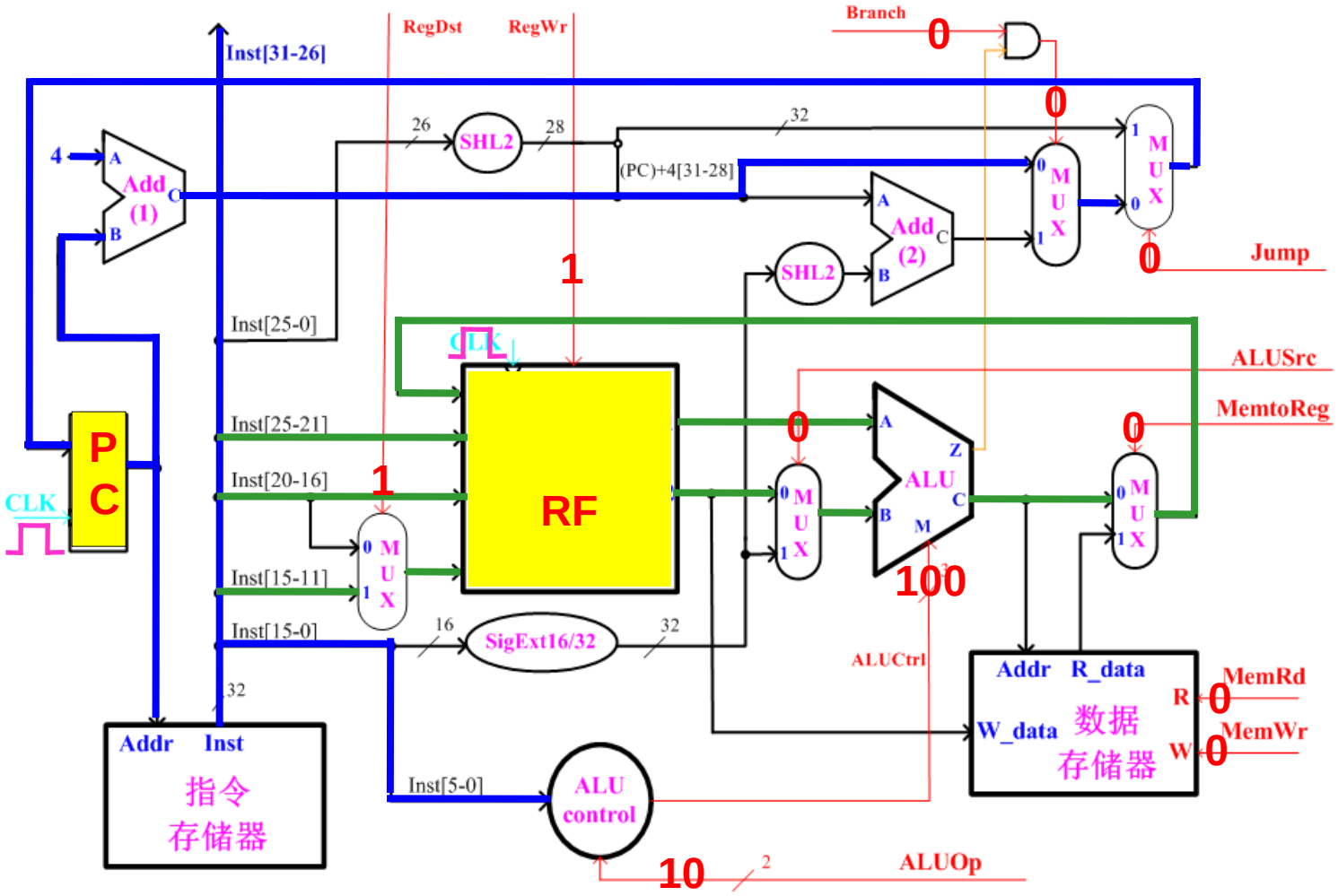
取数指令周期数据流图



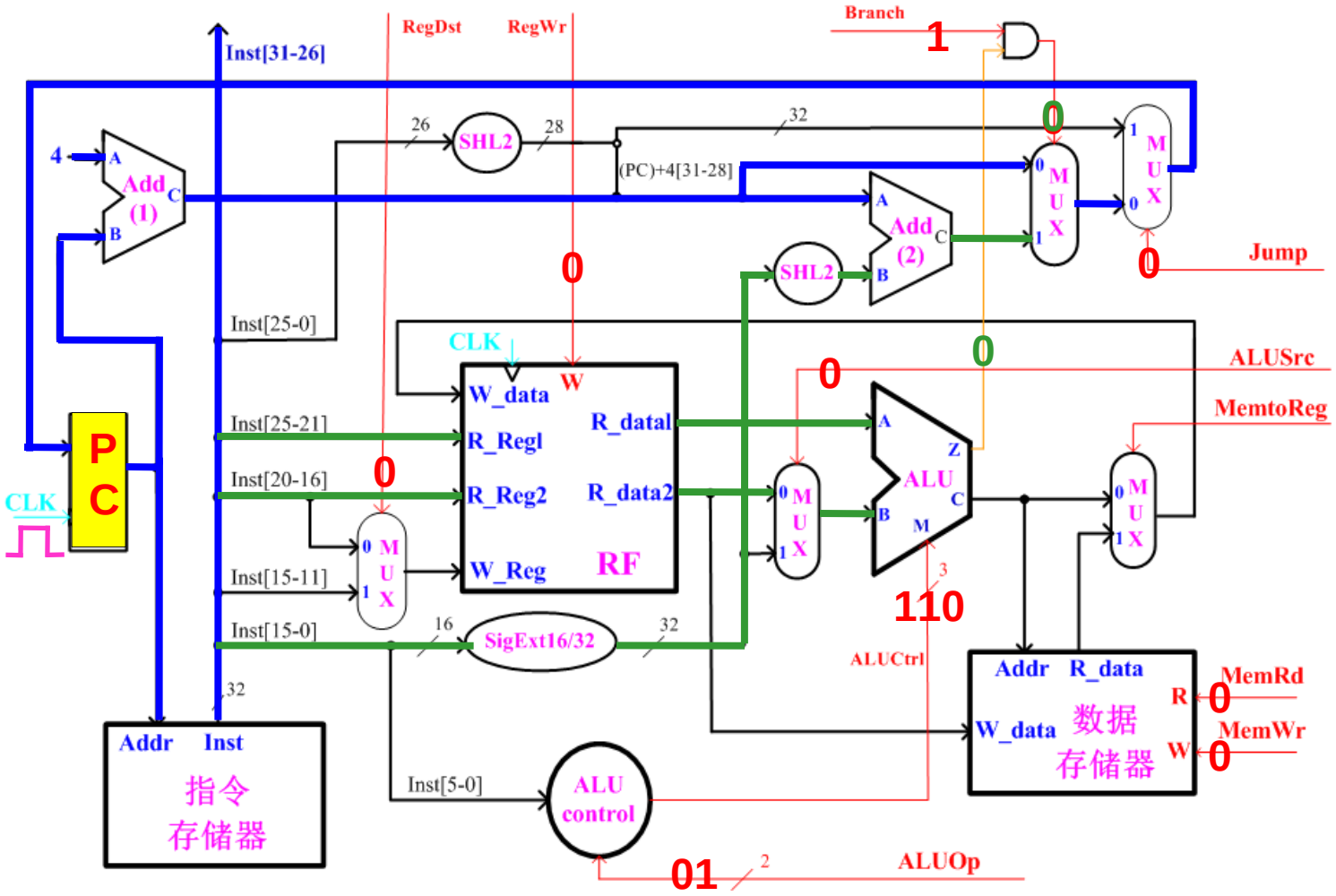
存数指令周期数据流图



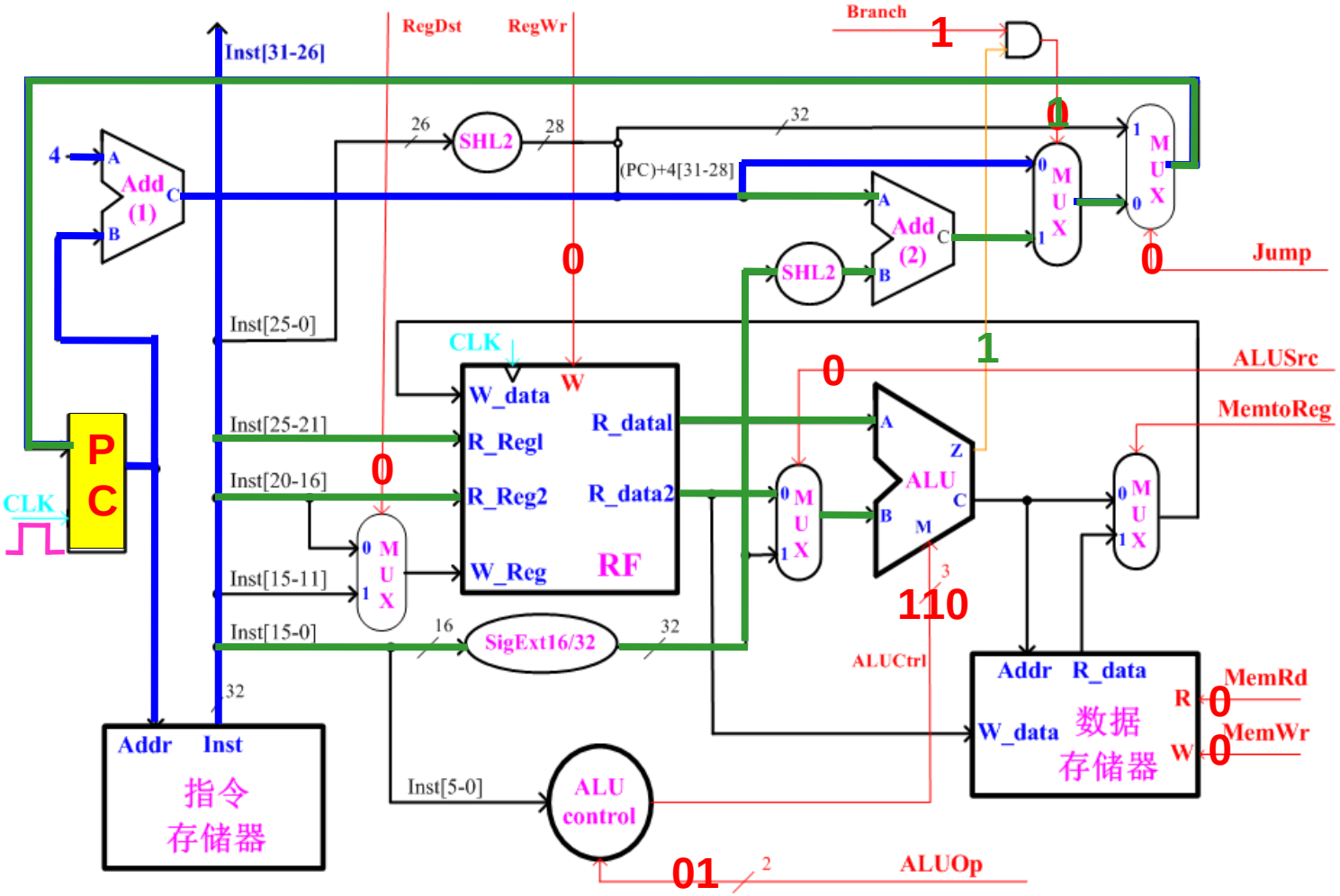
加法指令周期数据流图



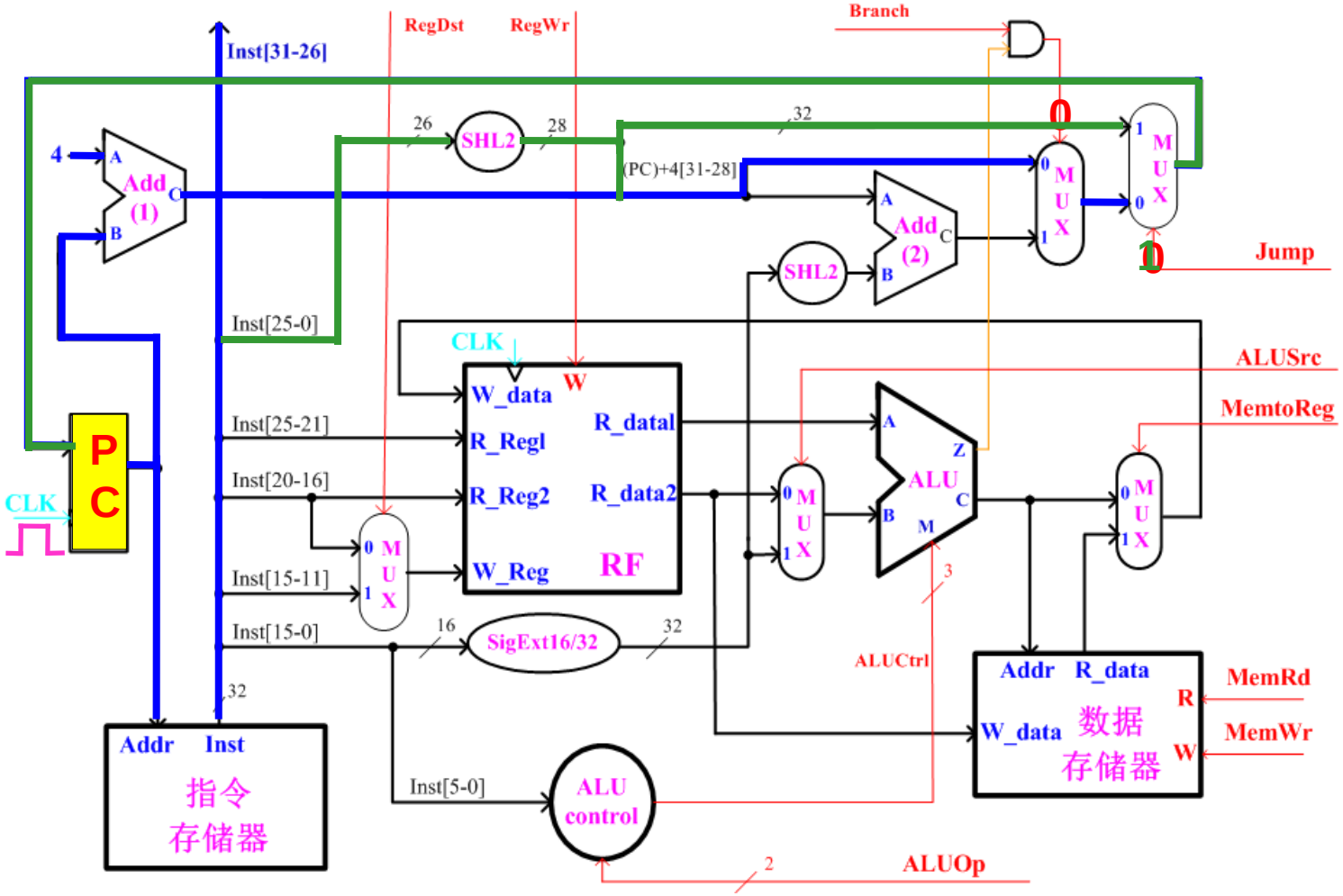
分支指令周期数据流图：不转移



分支指令周期数据流图：转移



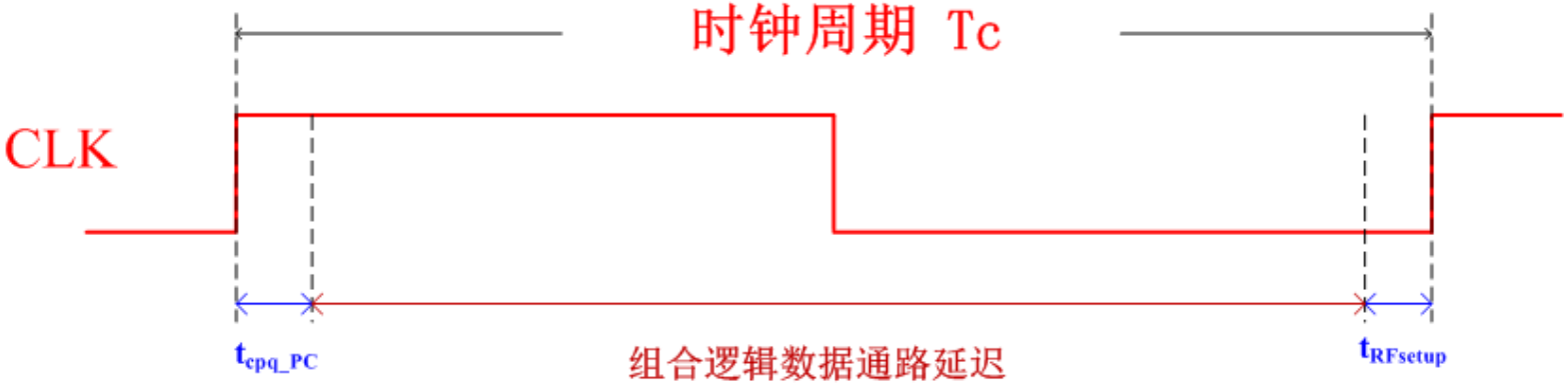
跳转指令周期数据流图



单周期CPU数据通路（续）

❒ 指令周期分析

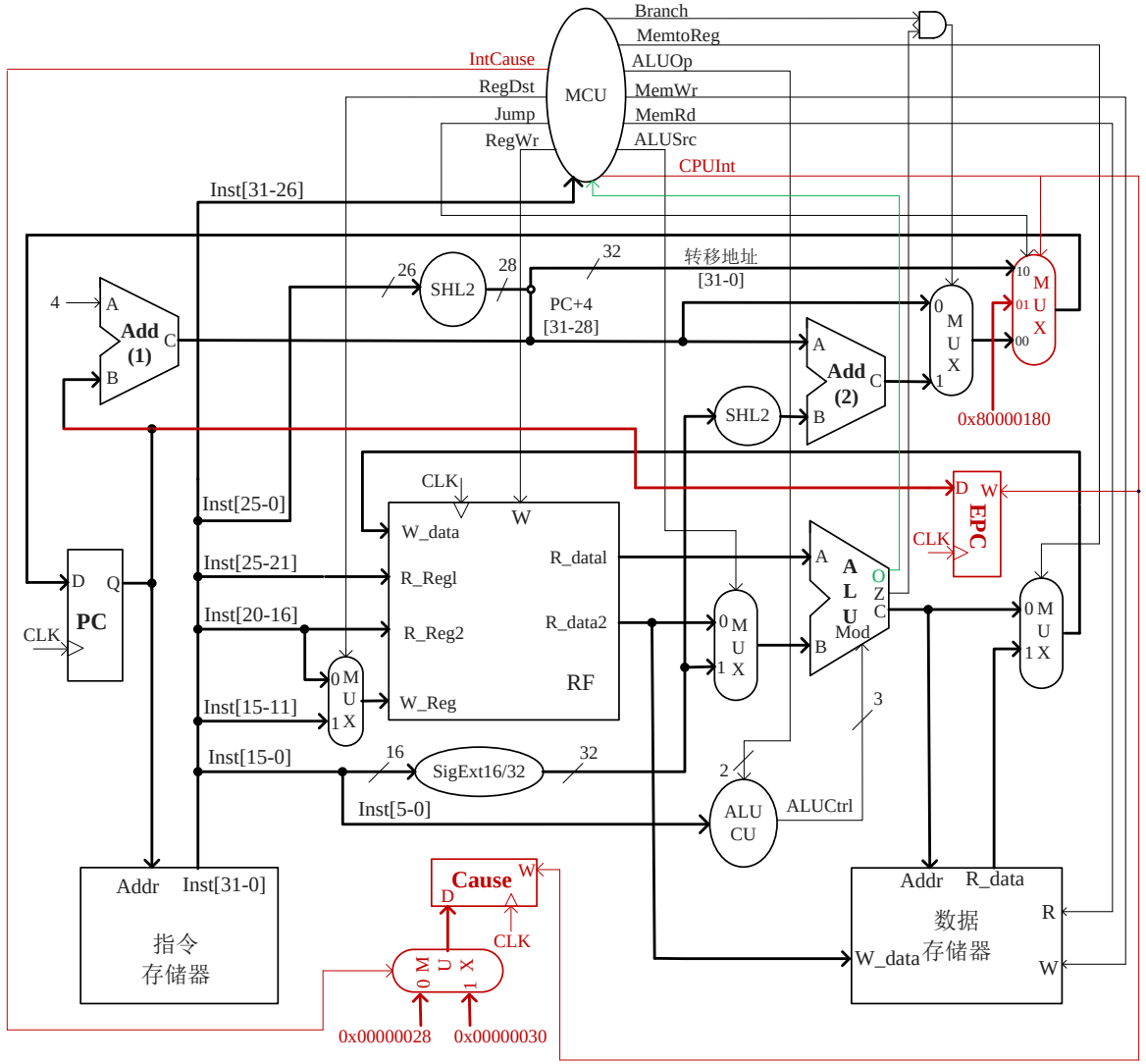
- ❍单周期CPU采用**定长指令周期**，其指令周期长度由执行时间**最长的指令执行时间**来决定。
- ❍**取数指令（load）**执行过程中在数据通路中经历的路径最长，所以其执行时间最长。



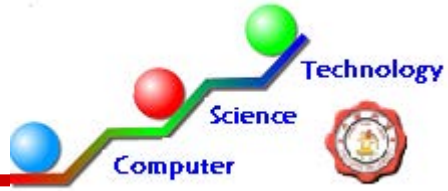
$$T_c = t_{cpq\_PC} + 2t_{mem} + t_{RFread} + t_{ALU} + t_{mux} + t_{RFsetup}$$

支持内部中断的数据通路

支持内部中断的单周期数据通路



目录



第六章 中央处理器

多周期CPU数据通路

- 多周期数据通路基本设计思想
  - 把每条指令的执行过程划分成多个时间间隔大致相等的阶段
  - 把每个阶段的微操作序列安排在一个时钟周期内完成
  - 一个时钟周期内最多完成一次访存或者一次寄存器读写或者一次ALU操作
  - 前一个时钟周期的执行结果由下一个时钟信号的上升沿打入到相应的状态单元
  - 时钟周期的宽度以最复杂操作阶段所用时间为基准，通常，取一次存储器读写的时间，即时钟周期等于主存储器的读写周期。

多周期CPU数据通路（续）

- 多周期数据通路分类
  - 分散互连结构
  - 单总线结构
  - 双总线结构
  - 三总线结构

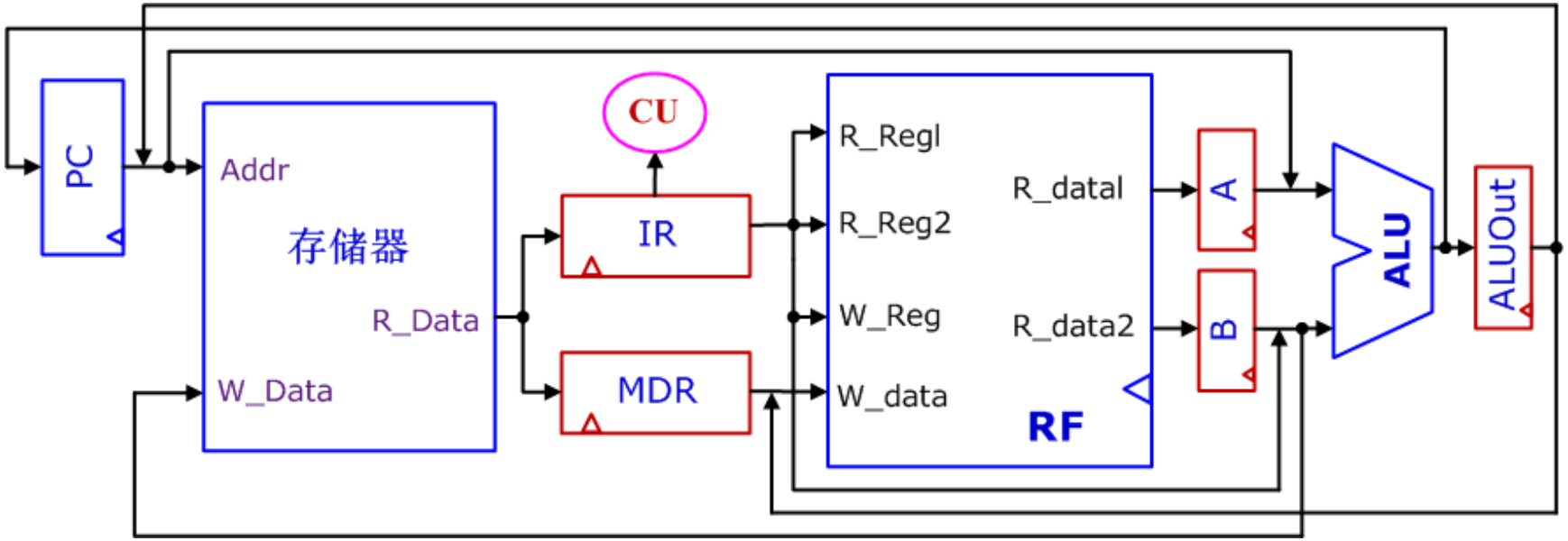
分散互连结构

在单周期数据通路基础上，通过合并冗余处理单元，并增加处理单元之间的临时寄存器，便可构造出分散互连的多周期数据通路。

- 将指令存储器和数据存储器合并成一个存储器，将加法器和 ALU合并成一个 ALU。
- 需要增加的临时寄存器包括：
  - 指令寄存器（IR）用于暂存从存储器读出的指令；
  - 存储器数据寄存器（MDR）用于暂存从存储器读出的数据；
  - 临时寄存器 A和 B用于暂存从寄存器堆中读出的操作数；
  - 寄存器 ALUOut用于暂存 ALU输出的数据。

分散互连结构（续）

❒ 数据通路基本组成

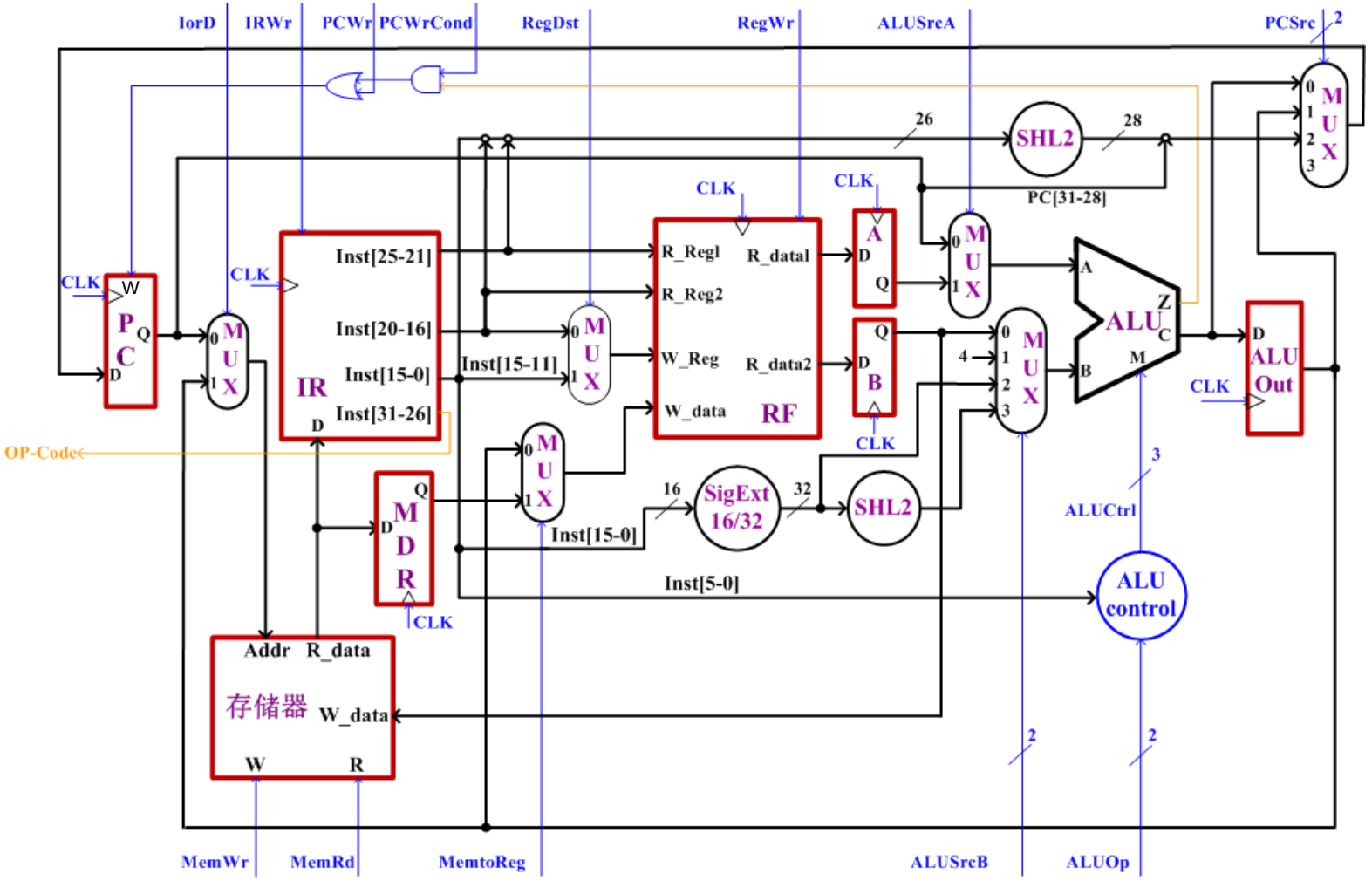


分散互连结构（续）

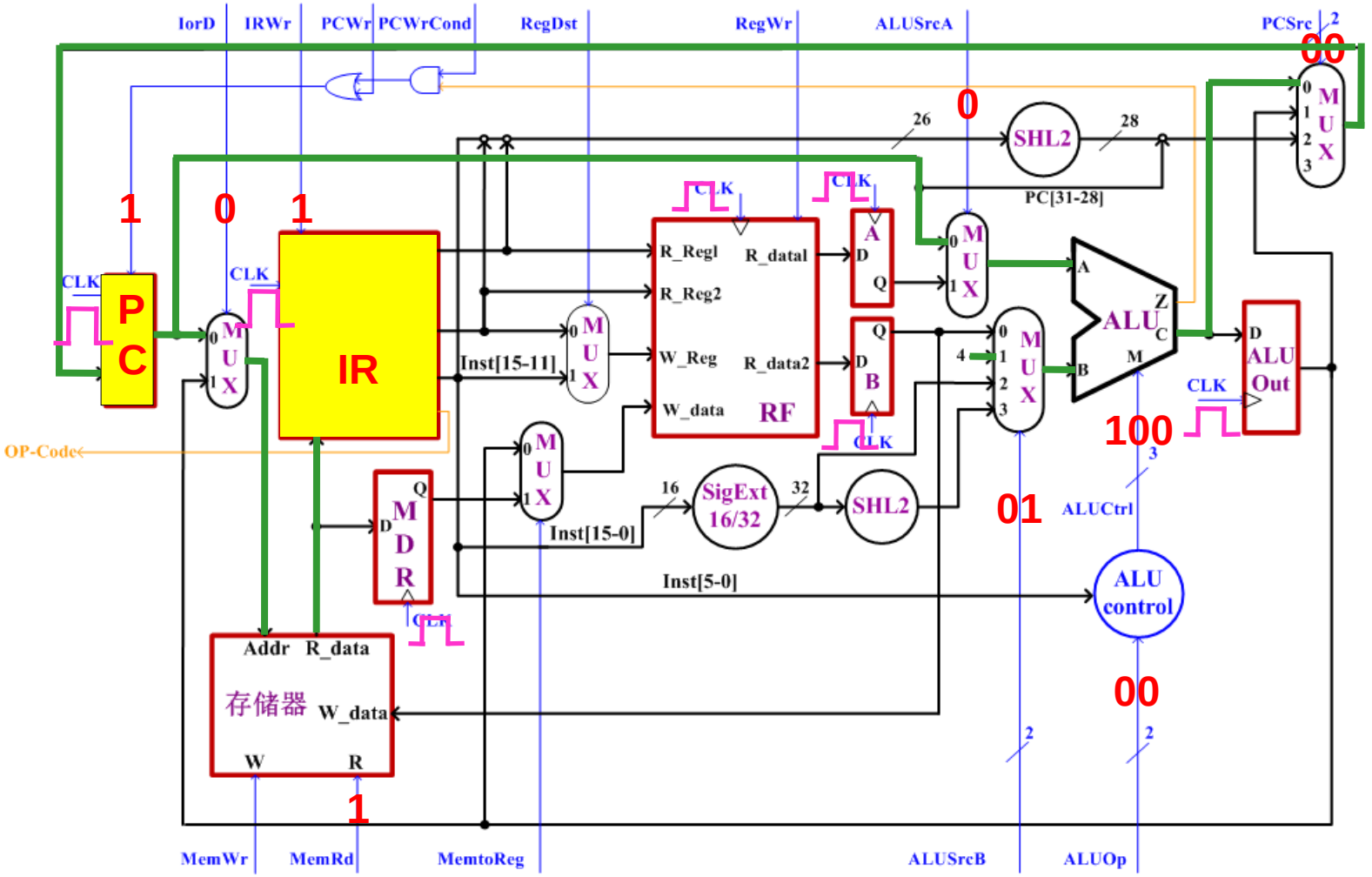
- 指令周期：分为5个时钟周期
  - T0：用PC值作为地址读取存储器，将指令存入指令寄存器IR；同时，通过ALU修正PC值。
  - T1：读寄存器堆，将数据存入临时寄存器A、B。
  - T2：ALU执行某种操作，并将结果存入 ALUOut。
  - T3：将 ALUOut 值写入寄存器堆或存储器，或者读取存储器单元内容放入MDR。
  - T4：将 MDR 值存入寄存器堆。

分散互连结构（续）

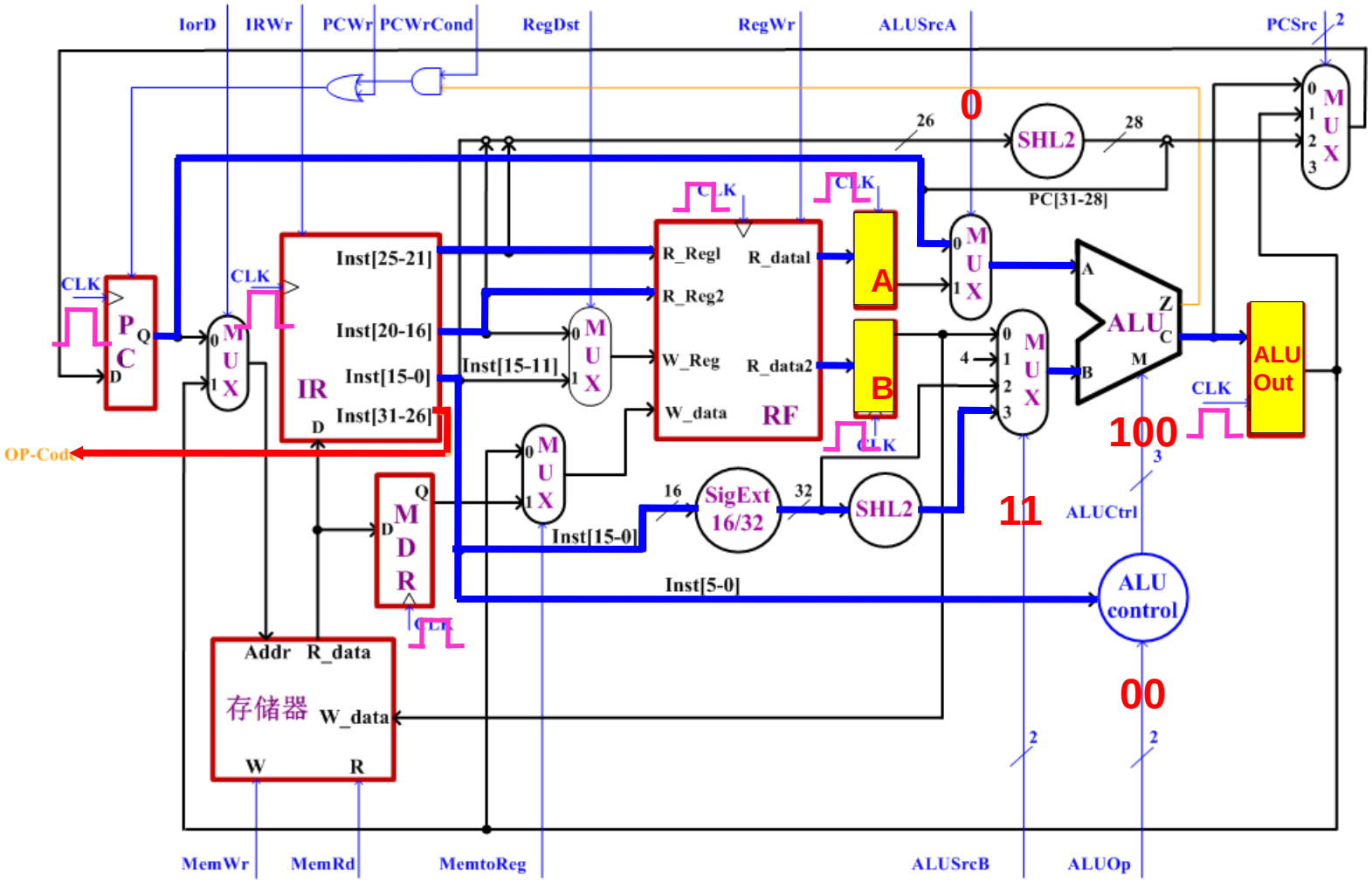
❒ 完整数据通路



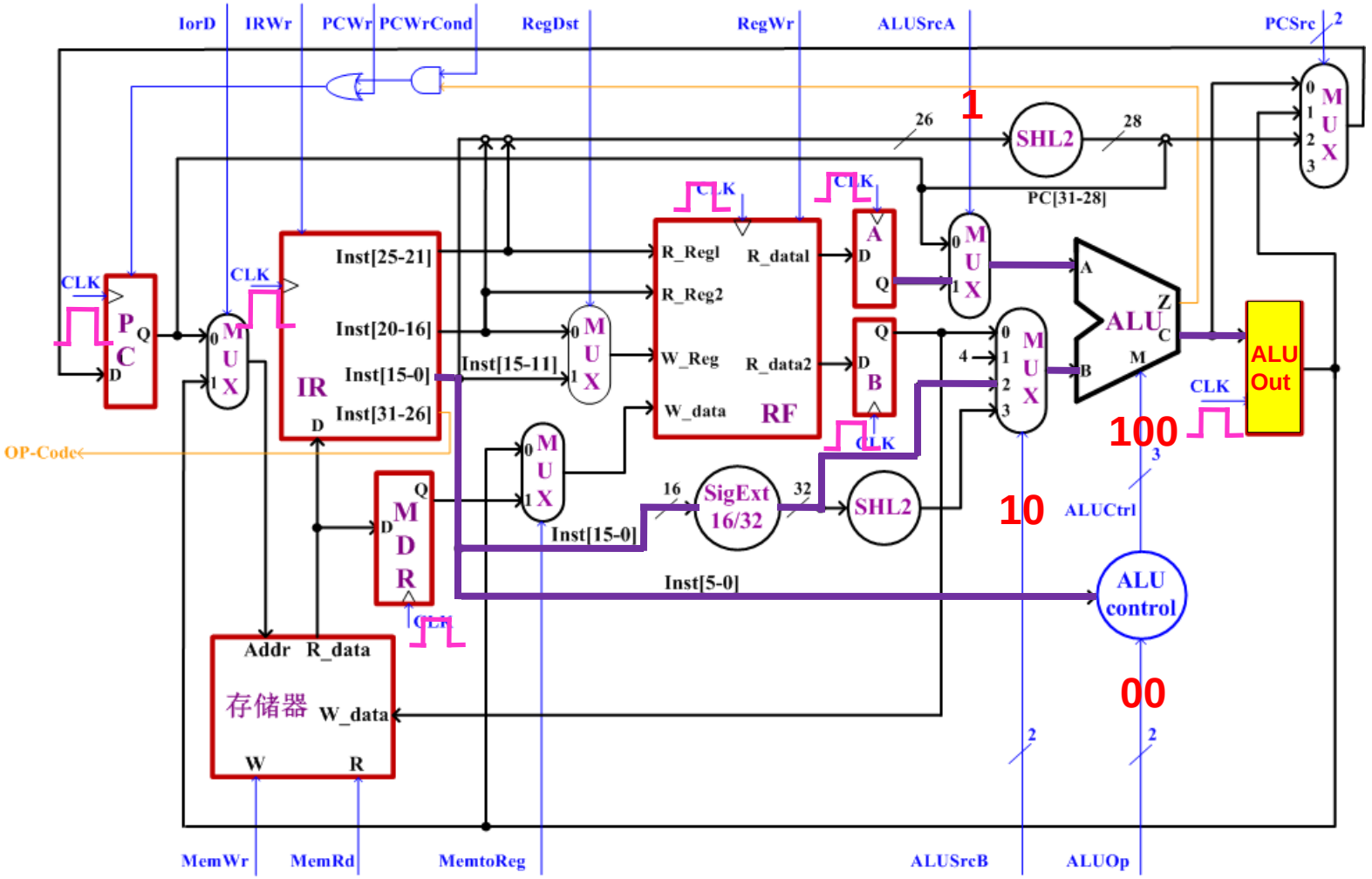
指令周期数据流图：第一个时钟周期



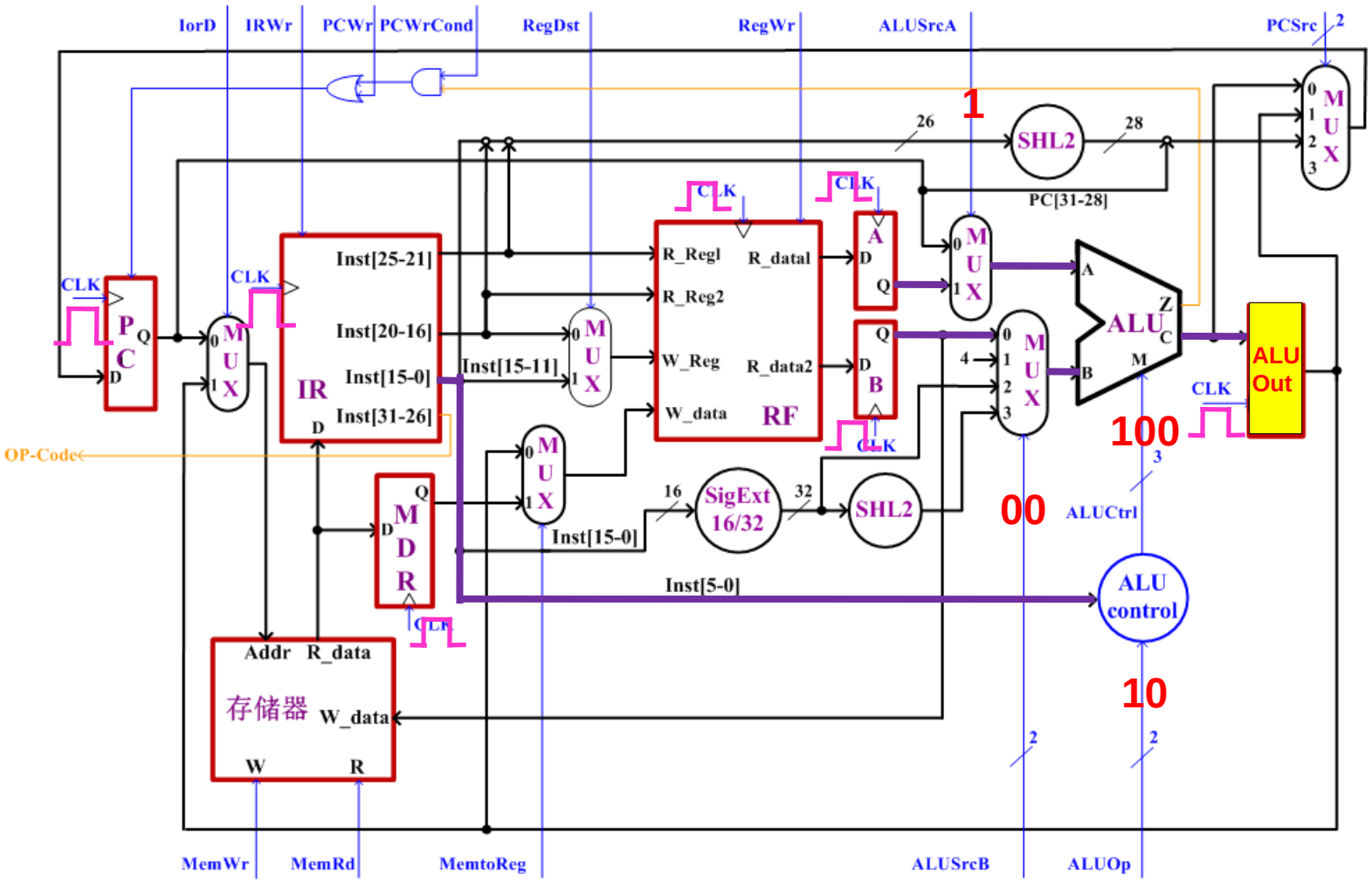
指令周期数据流图：第二个时钟周期



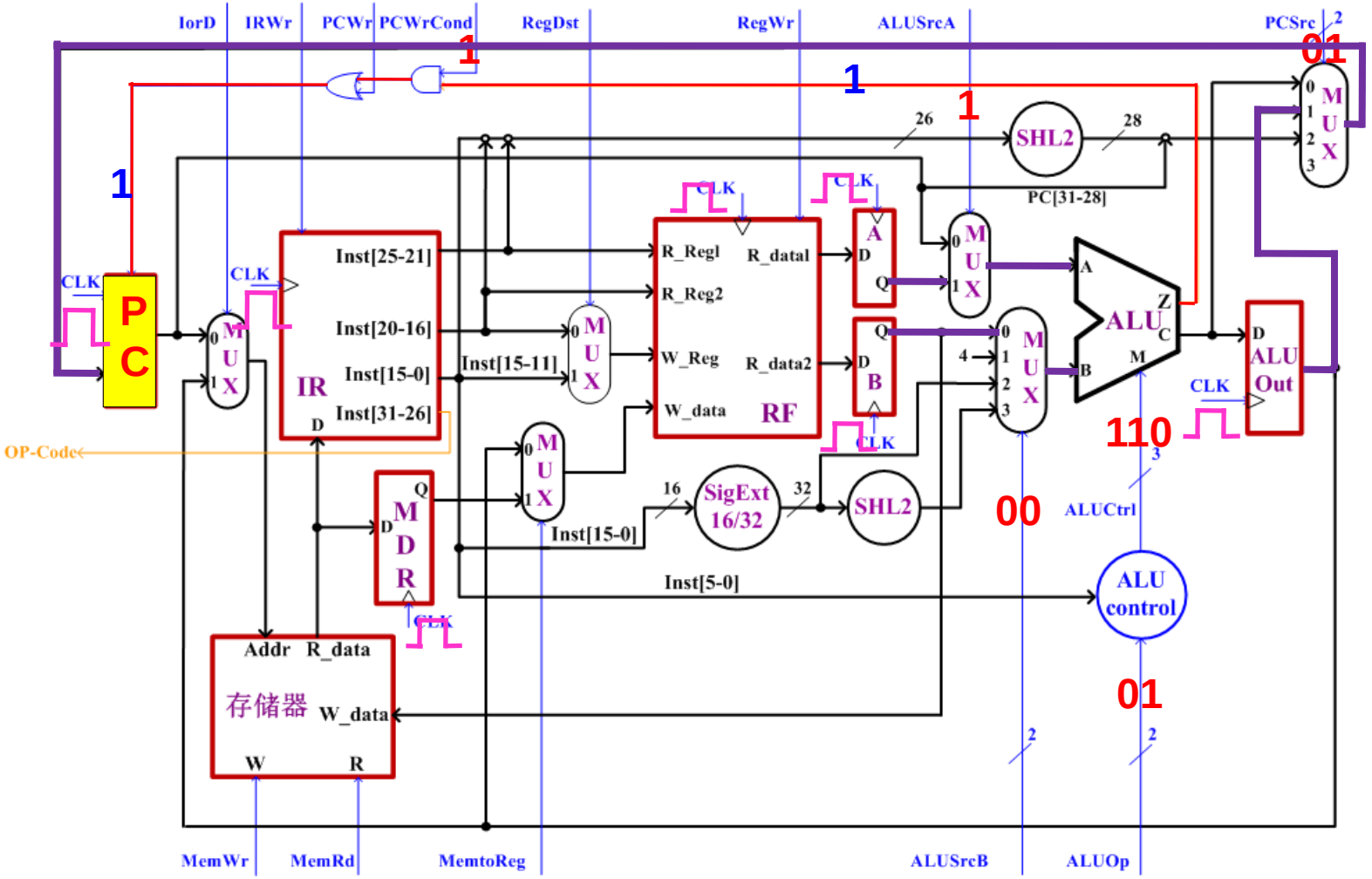
load/store指令第三个时钟周期



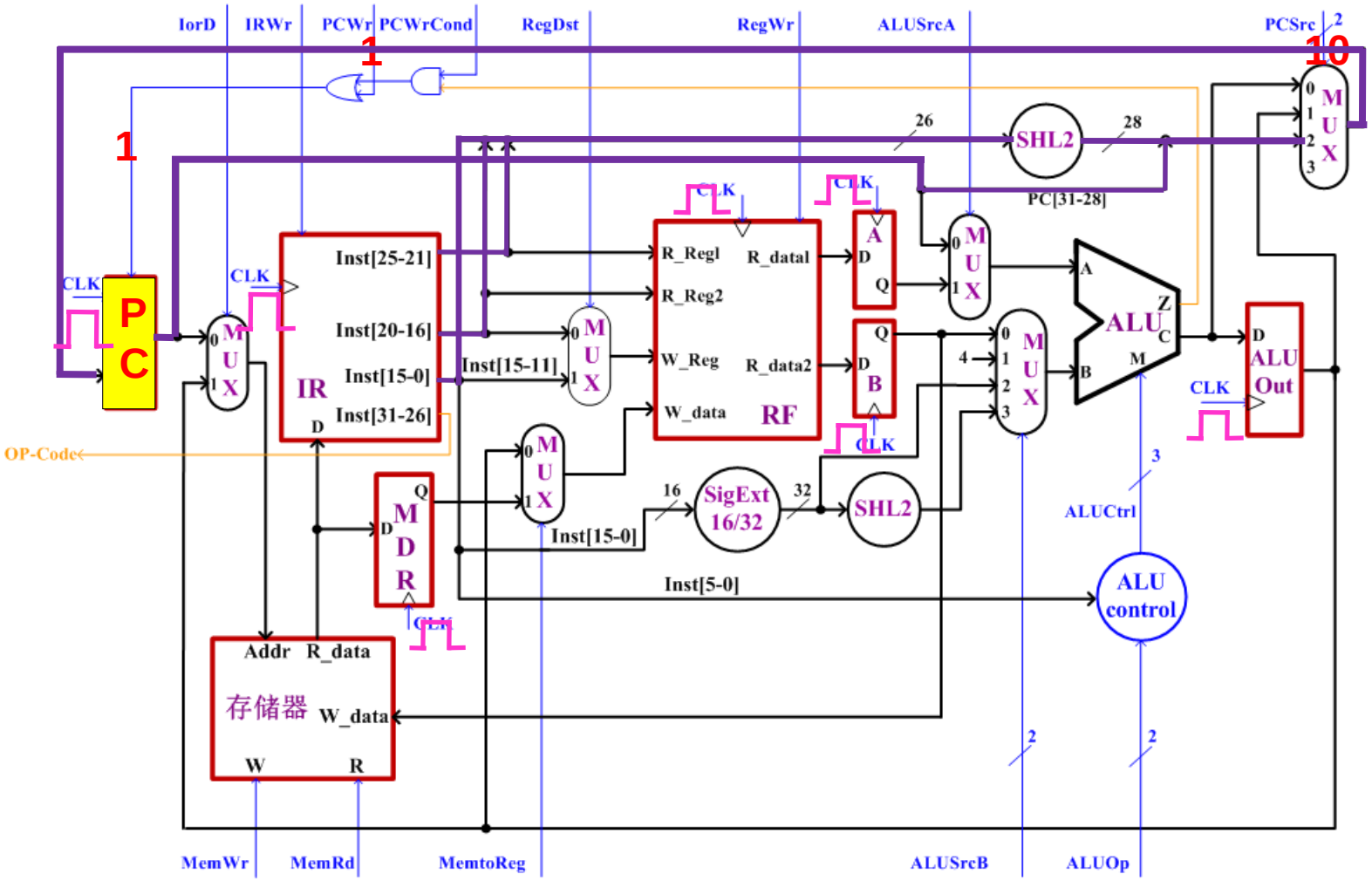
add指令第三个时钟周期



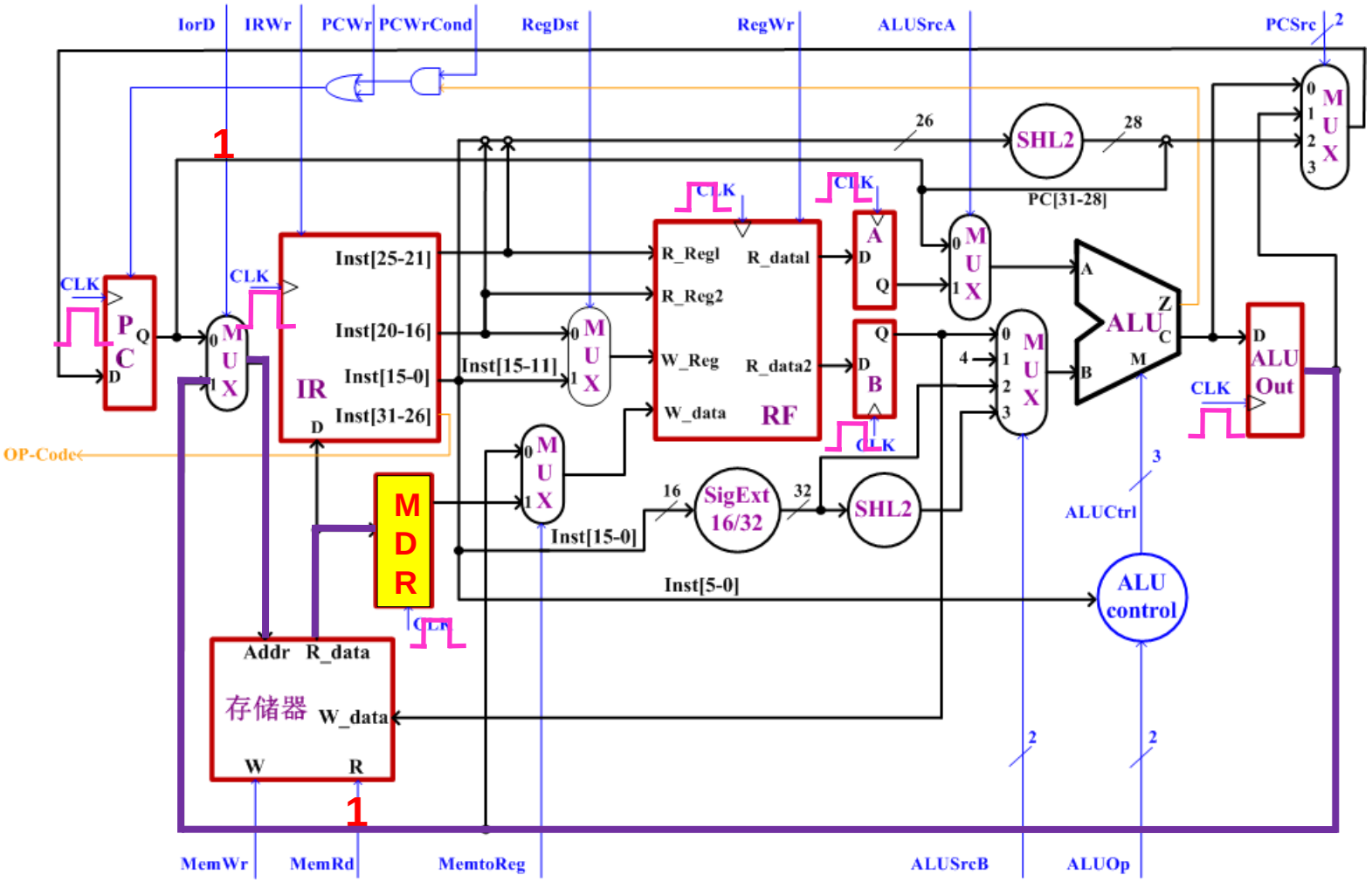
beq指令第三个时钟周期



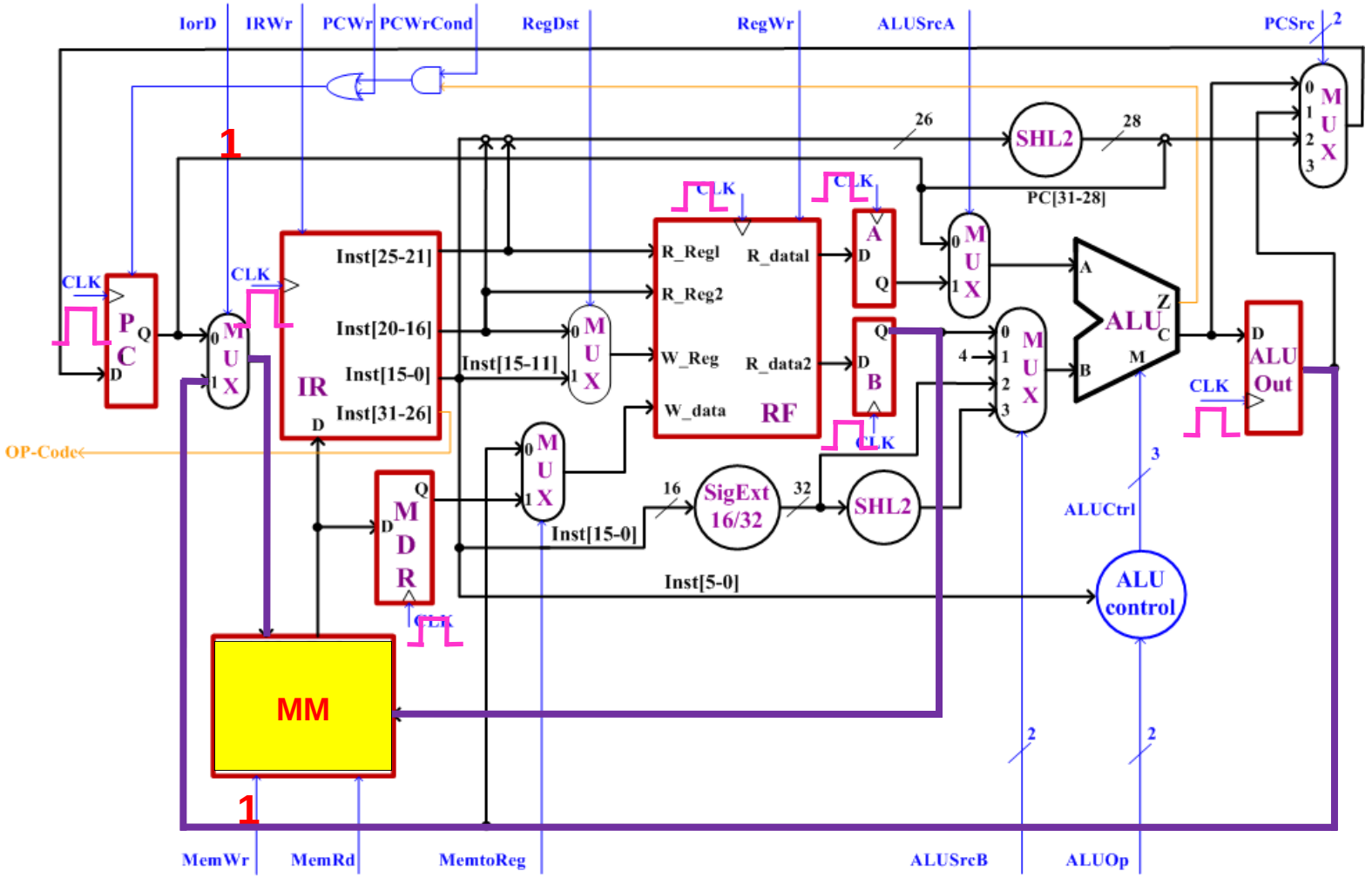
i指令第三个时钟周期



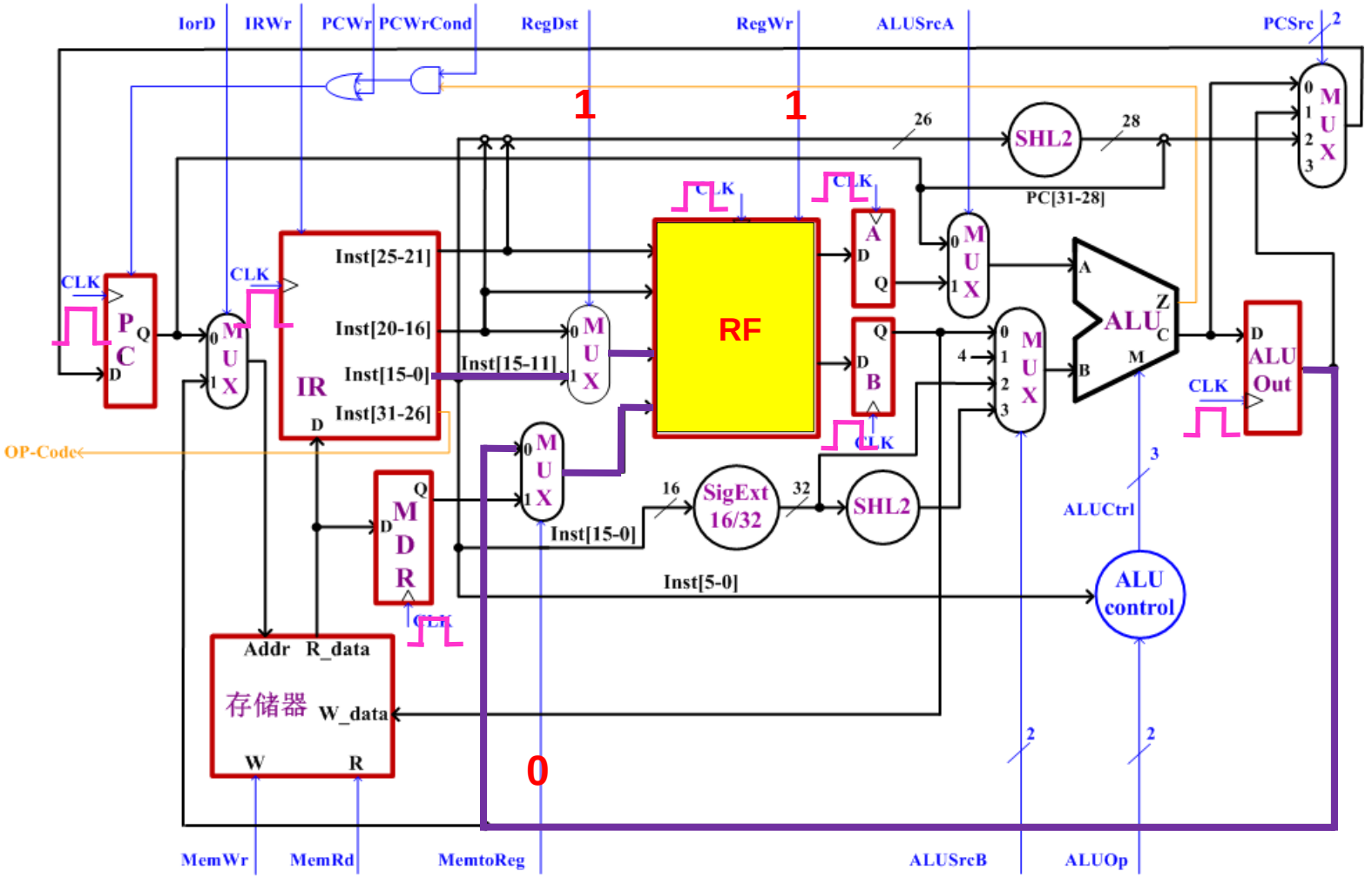
load指令第四个时钟周期



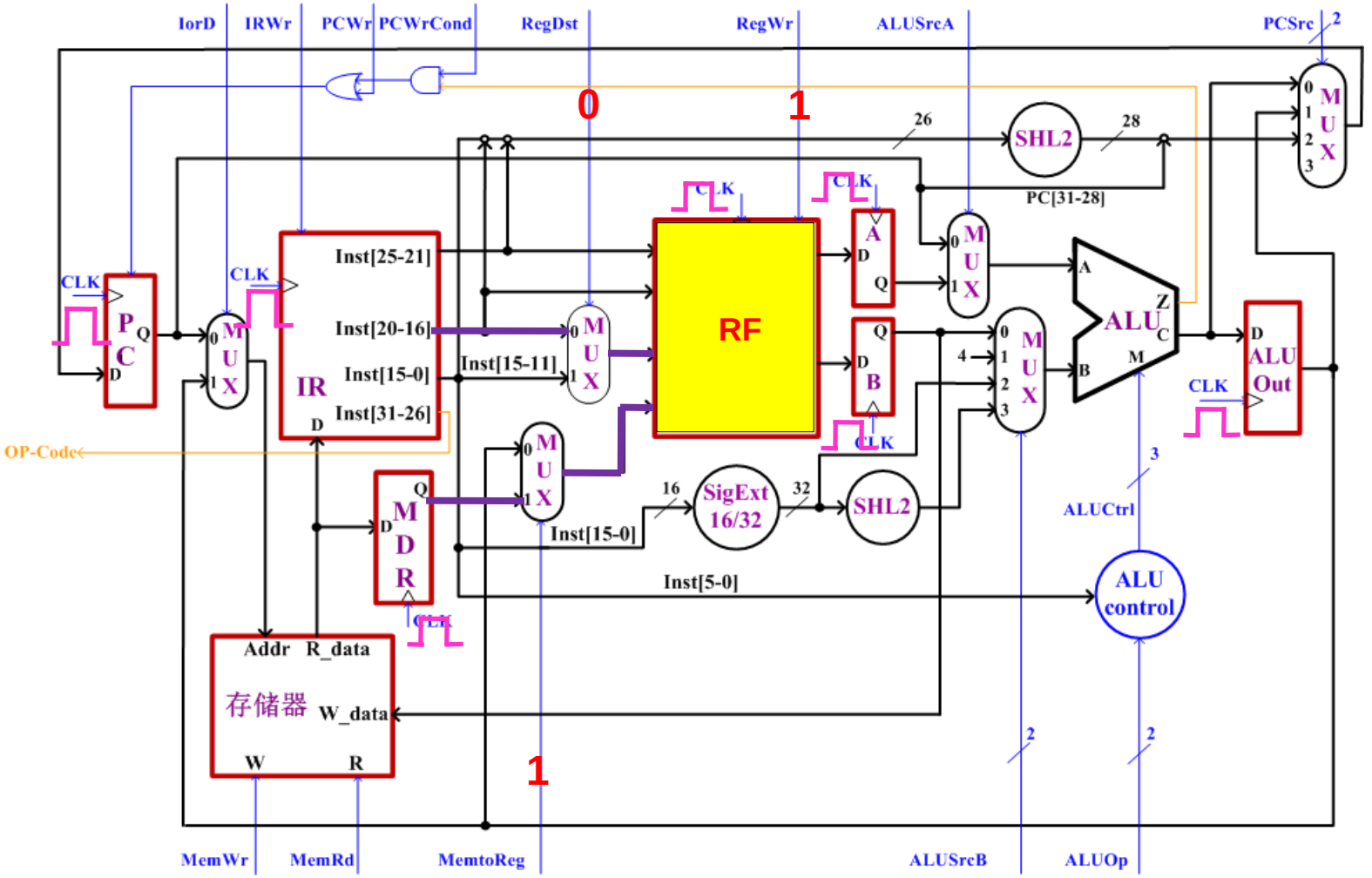
store指令第四个时钟周期



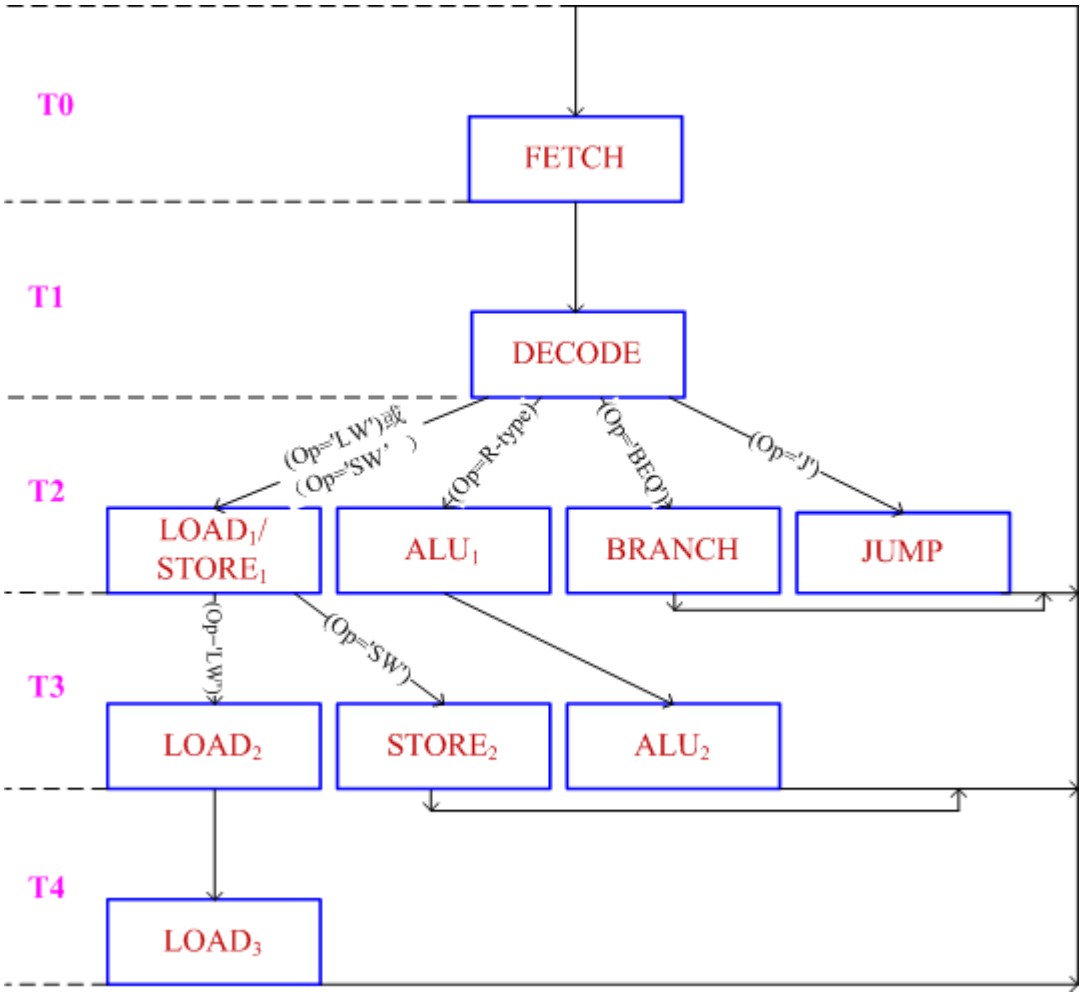
add指令第四个时钟周期



load指令第五个时钟周期



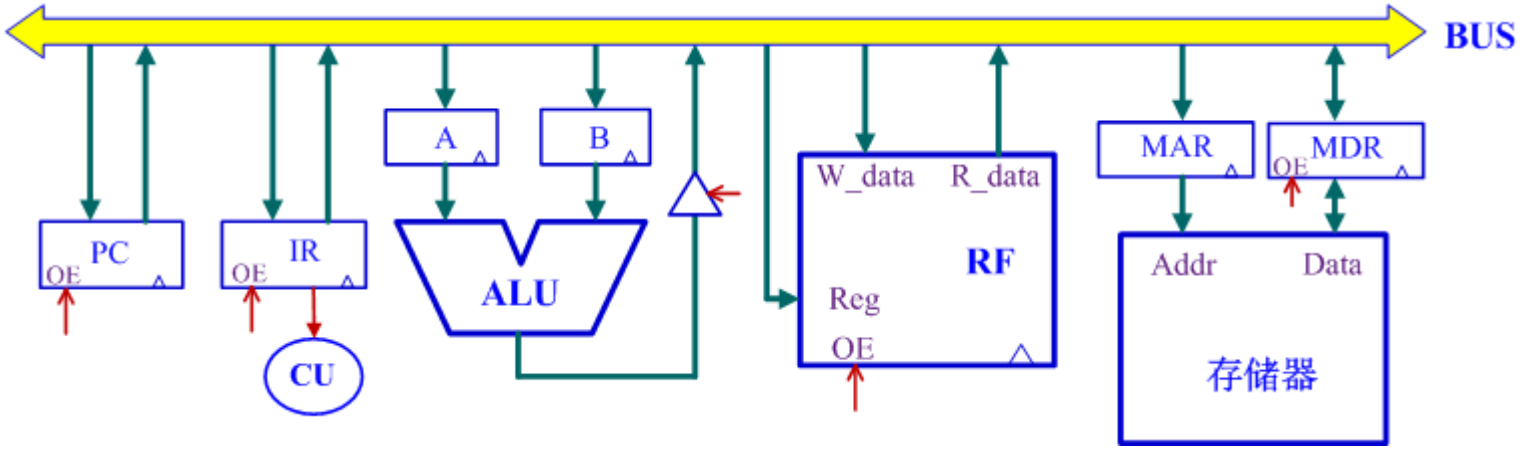
指令周期流程图



单总线互连结构

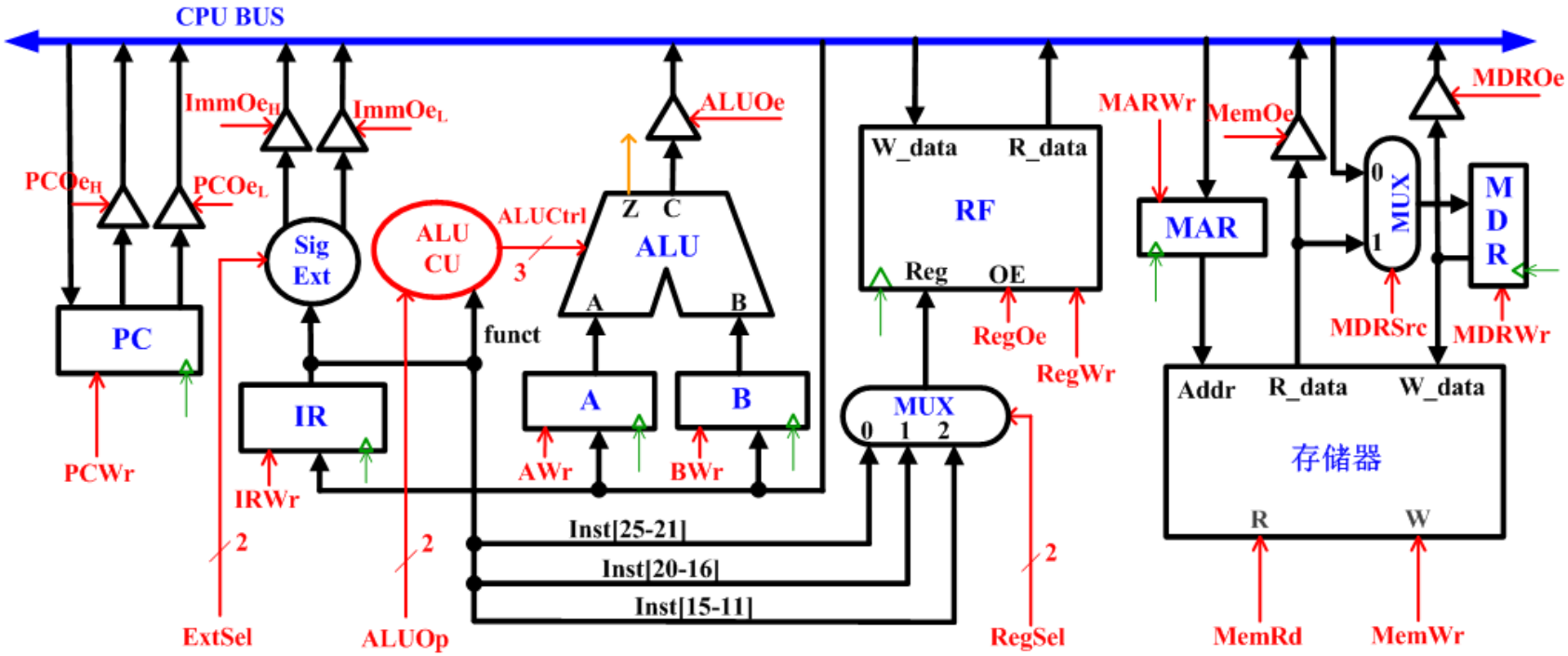
总线作为CPU多个部件间传输数据的公共通路，使得CPU内部结构更规整。但总线传输的“互斥性”决定了一条总线上只能进行串行的传送操作，降低了操作间的并行性，从而导致指令周期延长。

❒ 数据通路基本结构

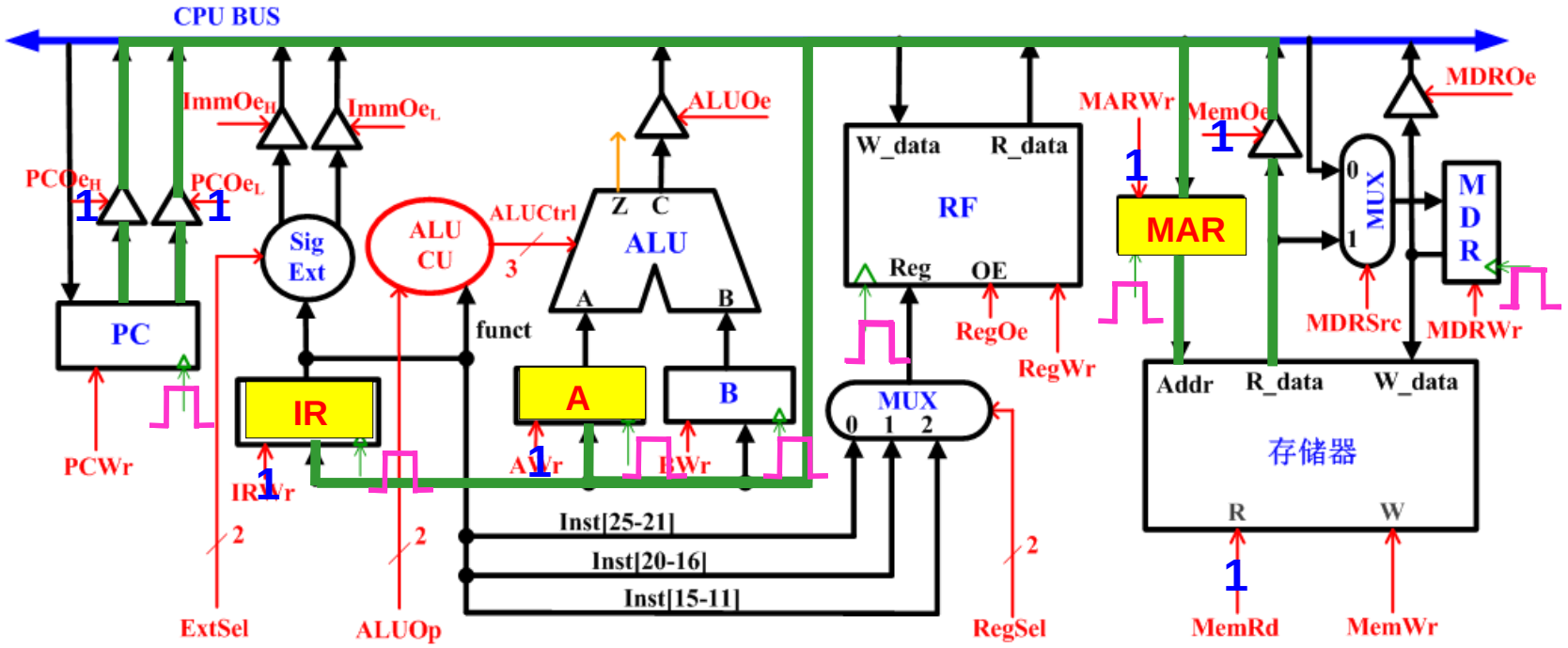


单总线结构（续）

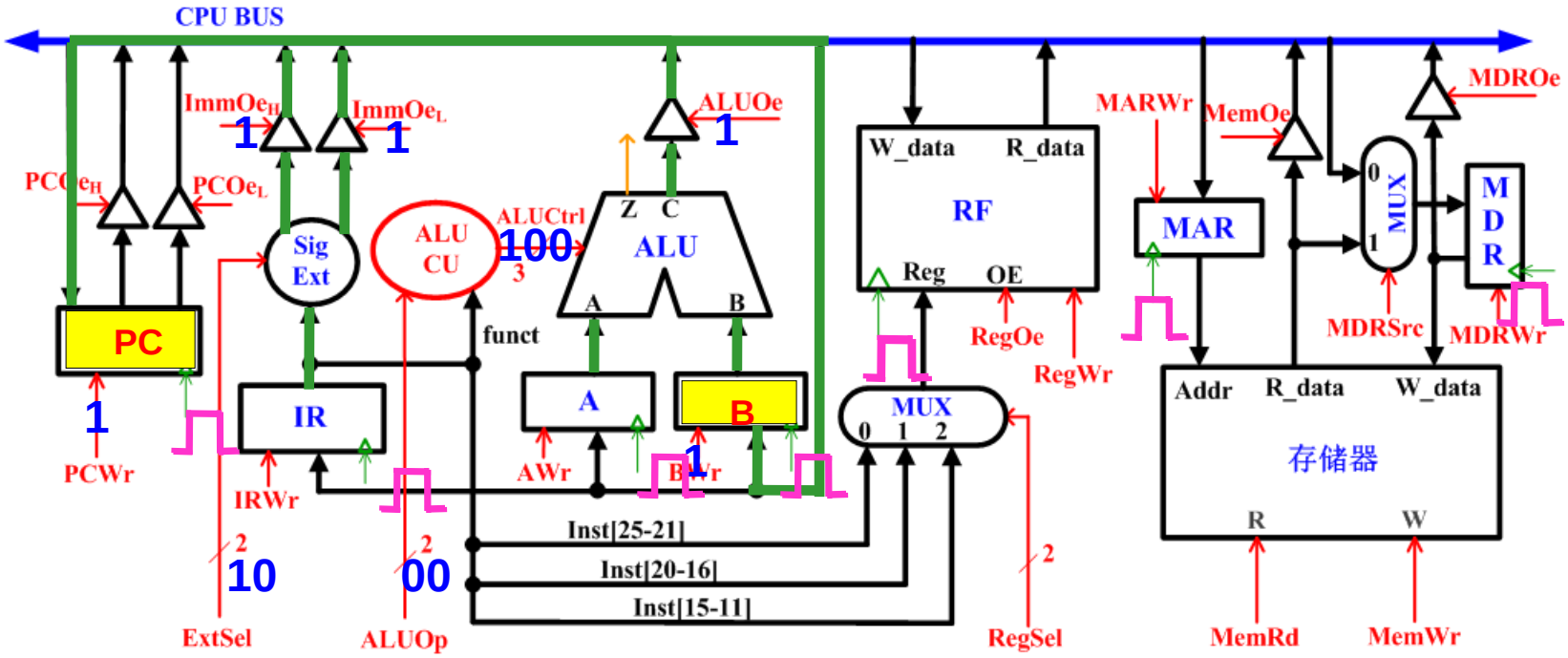
- 基于通用寄存器的单总线CPU数据通路



指令周期数据流图：第一、二个时钟周期

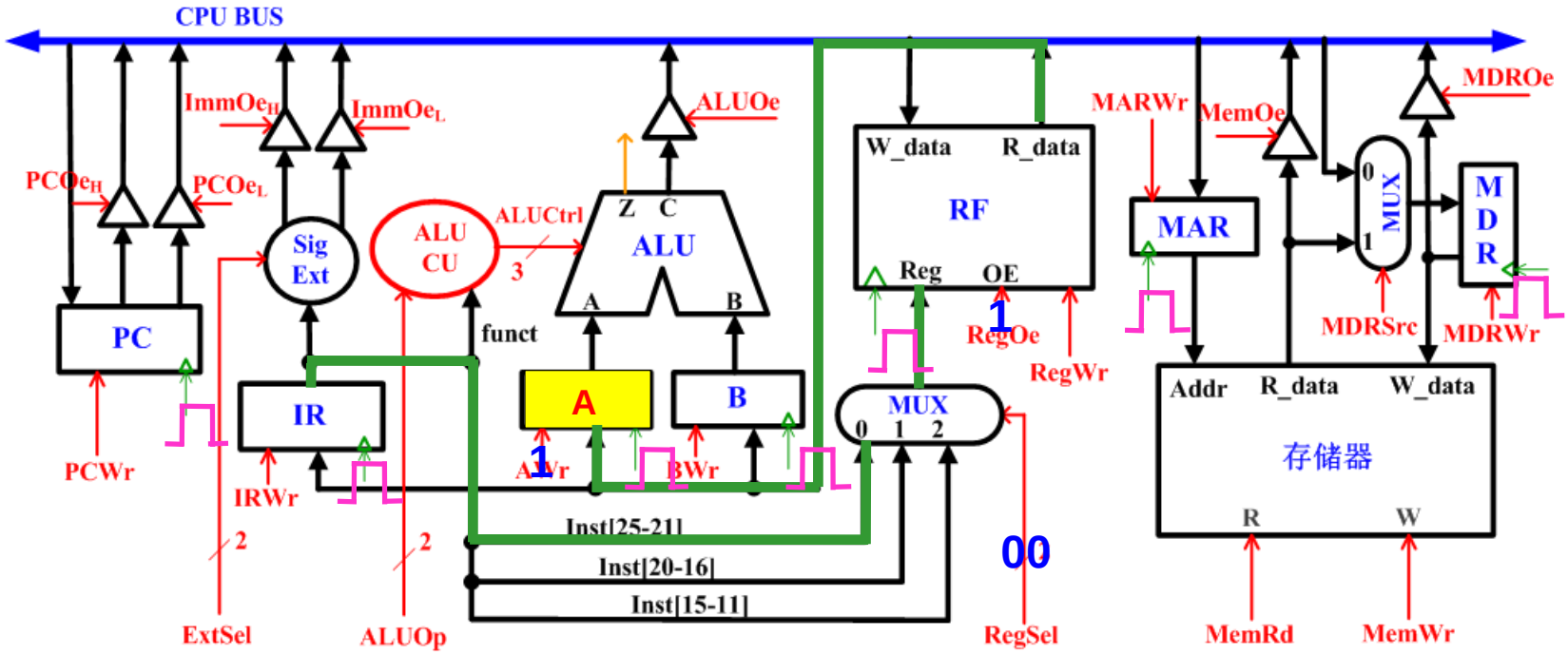


指令周期数据流图：第三、四个时钟周期

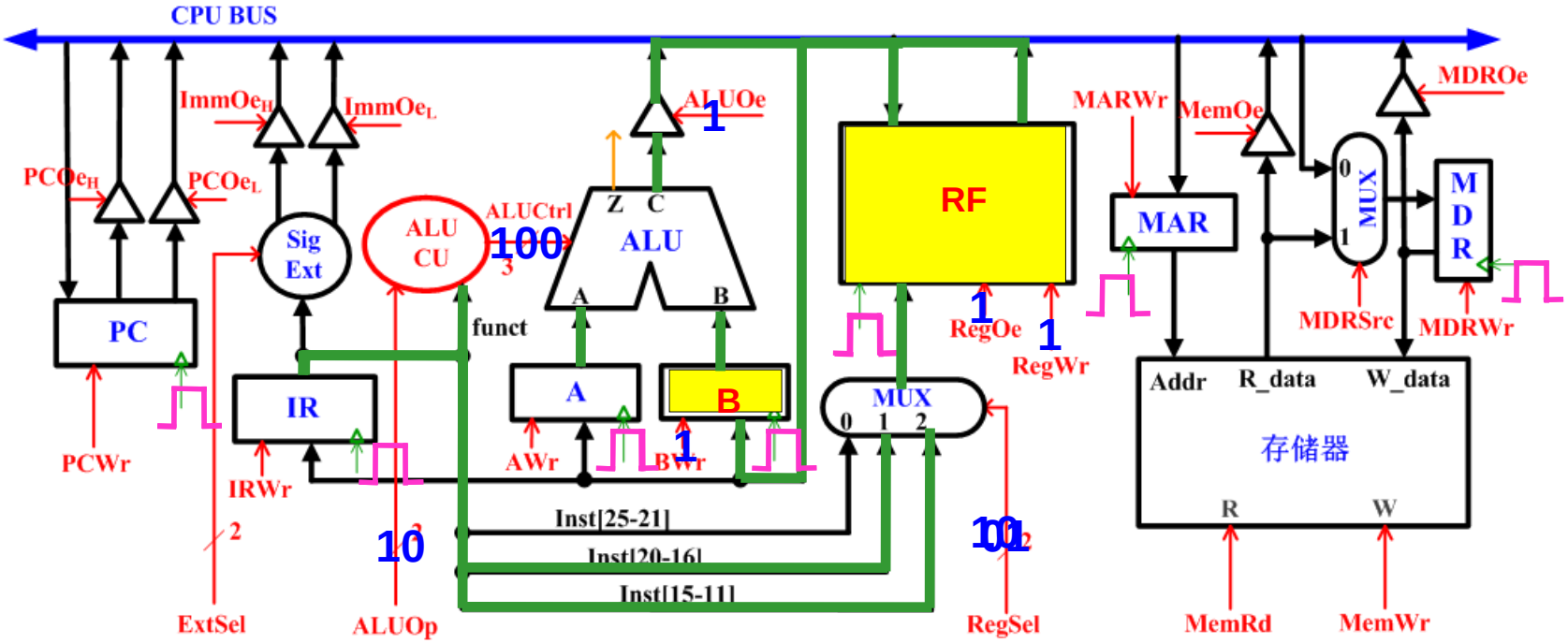


ExtSel	00	01	10	11
扩展方式	SigExt(IR[15-0])	SigExt (IR[15-0])<<2	立即数4	IR[25-0]<<2

指令周期数据流图：第五个时钟周期



ADD第六、七个时钟周期



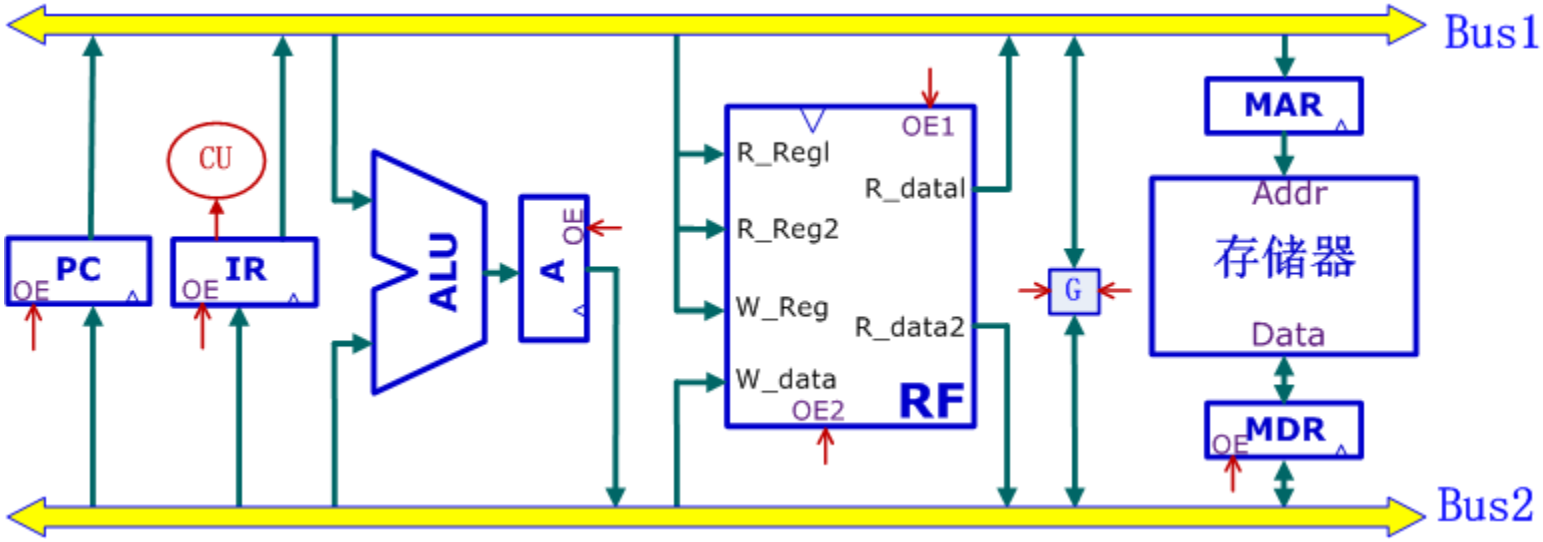
双总线结构

在单总线结构中，总线上操作非常繁忙，但部件间操作并行性较低。提高操作并行性的方法：

- 在单总线数据通路中建立一条或者多条专用通路
- 构造并行总线，允许总线事务并行地完成
  - 双总线结构
  - 三总线结构

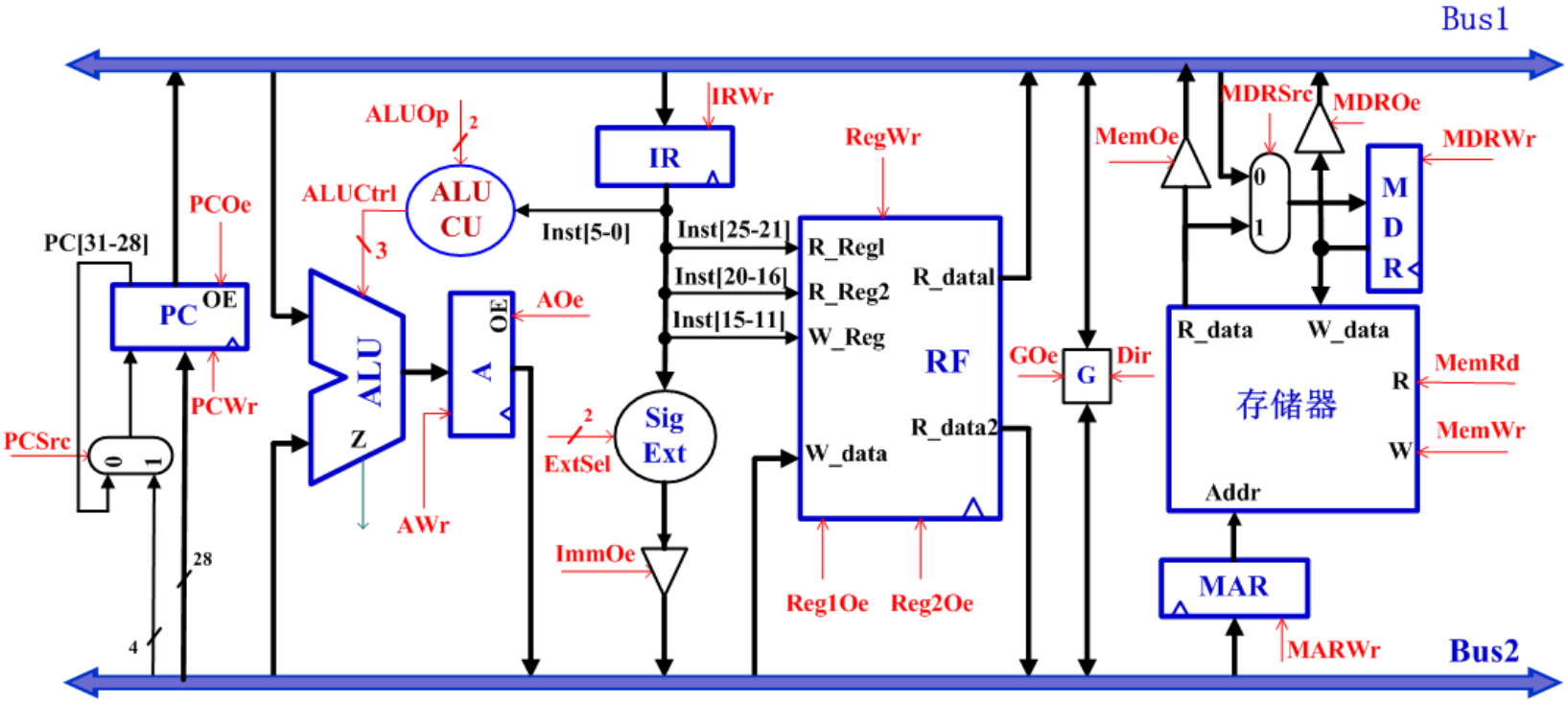
双总线结构（续）

❒数据通路基本结构



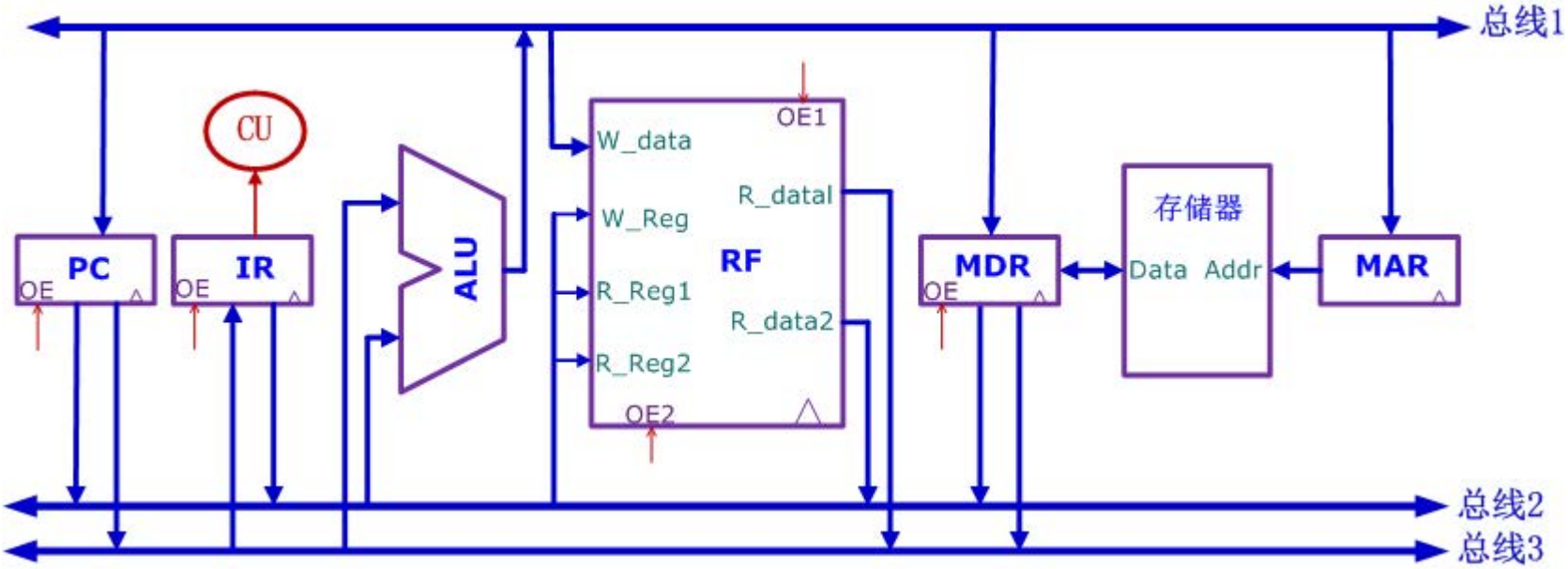
双总线结构（续）

❒完整数据通路



三总线结构

❒数据通路基本结构



不同互连结构性能比较

多周期CPU中不同类型指令的指令周期不同。可以用指令周期中所包含的时钟周期数 CPI 衡量指令周期的大小。而指令系统的平均CPI可以作为衡量CPU速度的一个指标。

一个程序平均CPI计算公式如下：

$$CPI=\frac{\text{CPU时钟周期数}}{\text{指令数}}=\frac{\sum(\text{指令数}_i\times CPI_i)}{\text{指令数}}=\sum(\frac{\text{指令数}_i}{\text{指令数}}\times CPI_i)$$

- 指令数_i ——第i类指令在程序中出现的总条数；
- $\frac{\text{指令数}_i}{\text{指令数}}$ ——第i类指令在程序中出现的频度；
- CPI_i ——第i类指令的CPI。

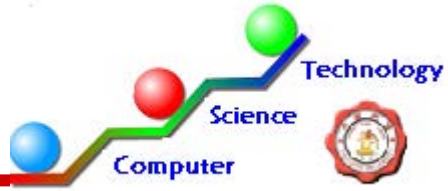
不同互连结构性能比较（续）

10条MIPS 32指令中，按照各自CPI可以将它们分为5类：取数指令、存数指令、算术/逻辑指令、分支指令和跳转指令。

假设这5类指令在程序中出现的频度依次为：30%、10%、40%、10%和10%，由不同多周期结构实现的该指令集的平均CPI如下：

指令类型 / 通路结构	load		store		算术/逻辑指令（R型）		branch		jump		平均CPI
	CPI	频度	CPI	频度	CPI	频度	CPI	频度	CPI	频度	
分散互连结构	5	30%	4	10%	4	40%	3	10%	3	10%	4.1
单总线结构	9		9		7		9		6		7.9
双总线结构	8		8		6		7		5		6.8

目录



第六章 中央处理器

指令流水原理

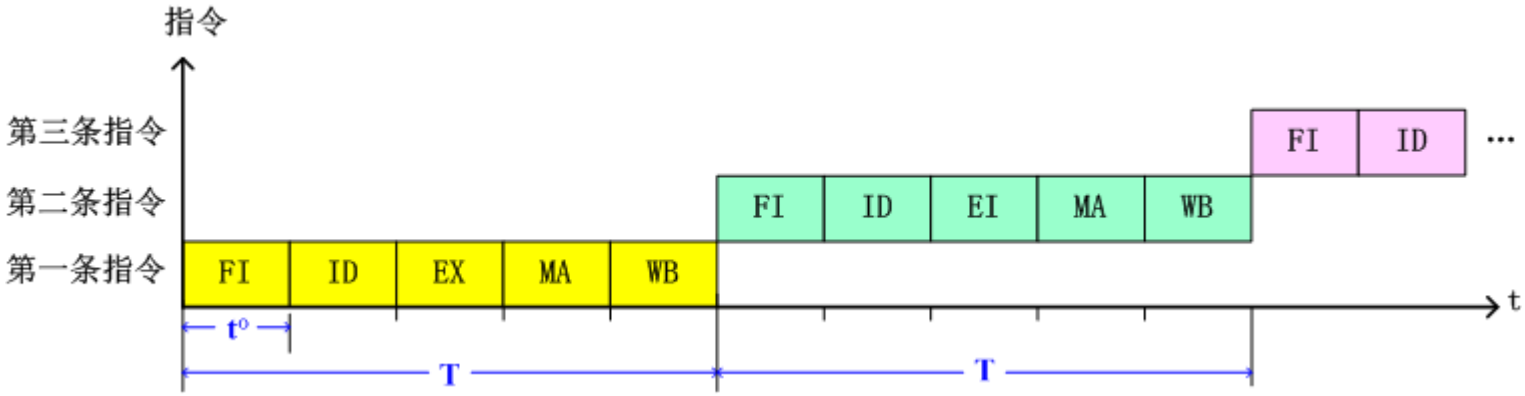
流水线CPU可以采用类似单周期数据通路。但是，通过重叠执行多条指令来提高硬件利用率，从而使CPU获得更高的效率。

- **指令流水**

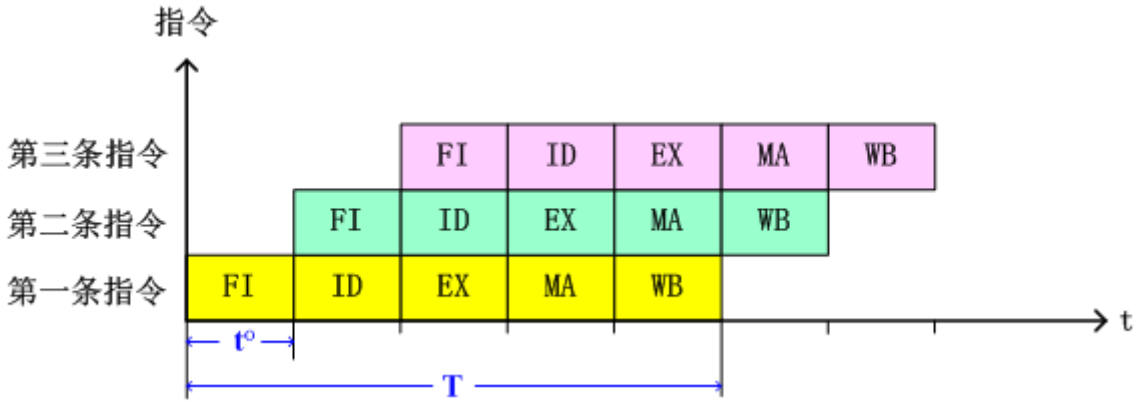
  将指令的执行过程分为几个阶段，每个阶段由数据通路中不同操作部件执行，不同指令多个阶段在时间上重叠。

  - 取指令（FI）：从指令存储器中读取指令。
  - 指令译码（ID）：分析当前指令，同时读取寄存器。
  - 执行指令（EX）：执行指令操作或计算地址。
  - 存储器访问（MA）：对数据存储器进行读写操作。
  - 数据写回（WB）：将操作结果写回寄存器。

指令流水原理（续）



（a）非流水线

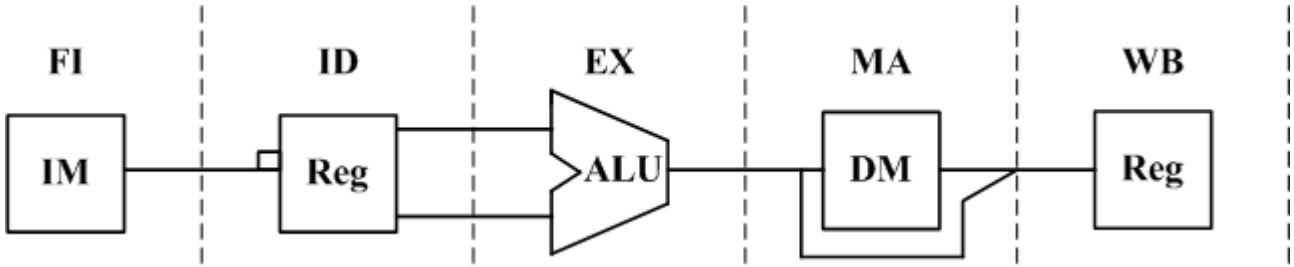


（b）流水线

流水线冒险

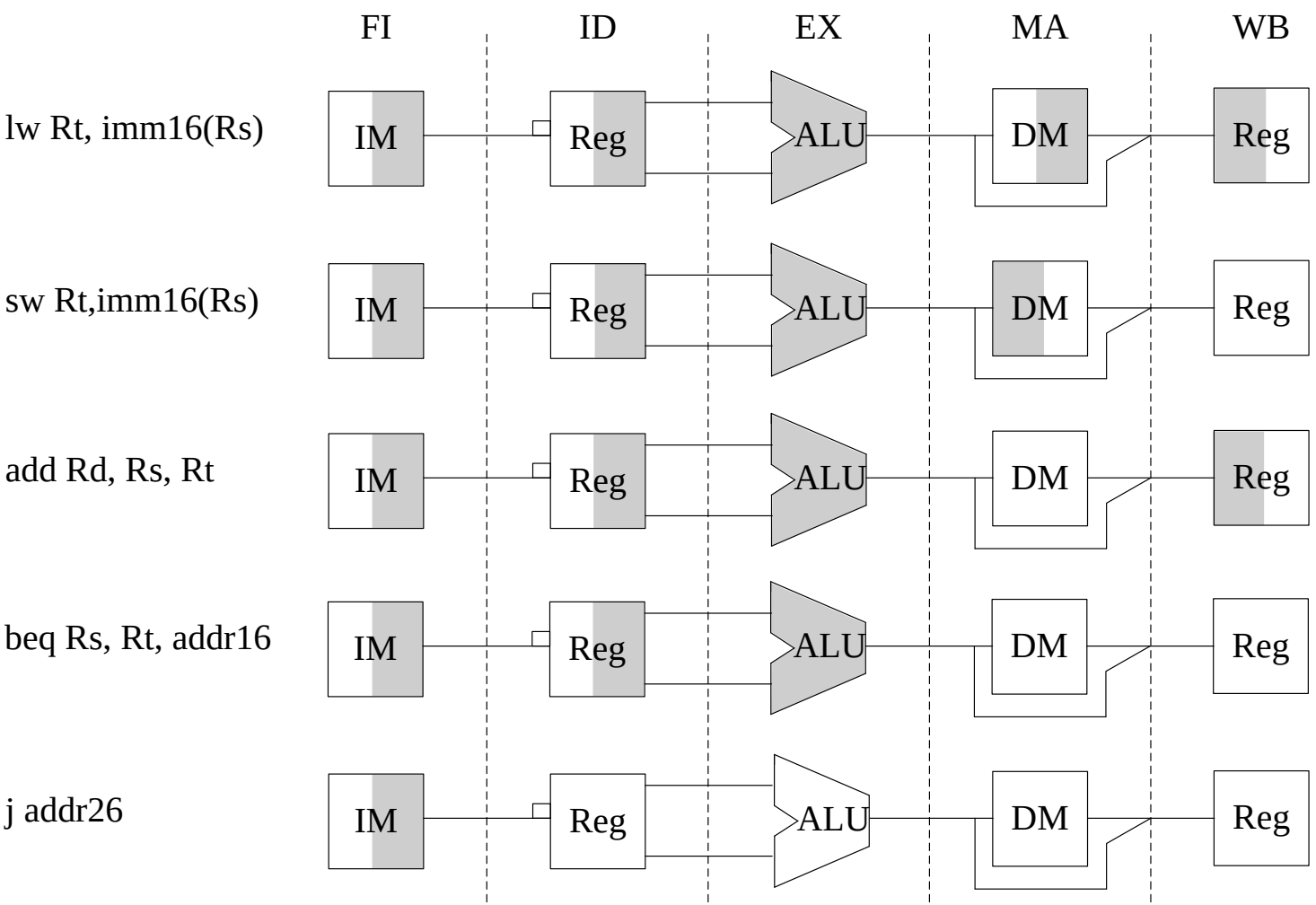
由于某些流水段不能正常工作，而引起流水线停顿。这种现象称为流水线冒险（hazard）。

❒ 指令流水线的图形表示



- ❍添加阴影表示该资源被指令所使用
- ❍为了区别寄存器堆的读操作、写操作，存储器的读操作、写操作，分别用左半边和右半边的阴影来表示，即右半边阴影表示读操作，左半边阴影表示写操作。

流水线冒险（续）



流水线冒险——结构冒险

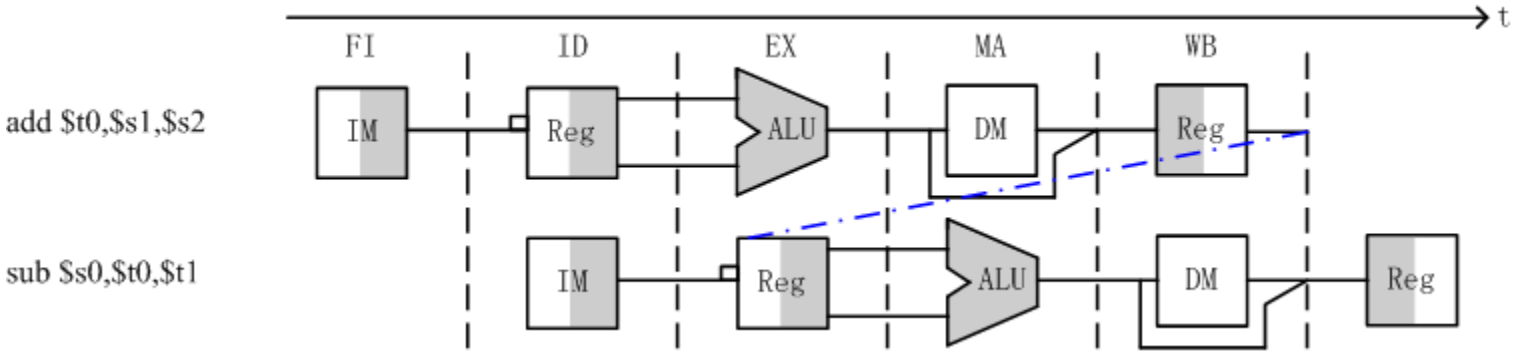
- 结构冒险
  - 结构冒险是由于指令间竞争硬件资源而引发的。
  - 单周期数据通路采用部件冗余技术实现，避免了结构冒险，所以，流水CPU数据通路可以通过改进单周期CPU数据通路来实现。

流水线冒险——数据冒险

❒ 数据冒险

生产数据的源指令和消费数据的目的指令重叠执行时，不能满足程序所要求的指令间数据处理的先后关系。

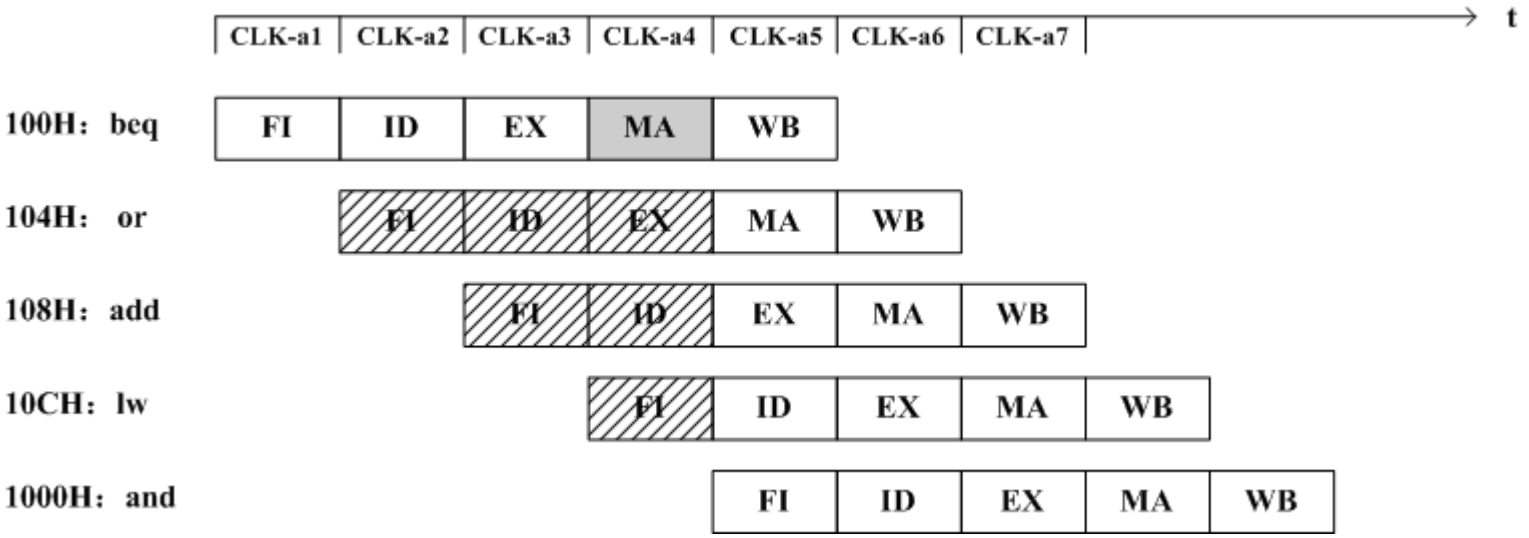
- ❍R-型指令间数据相关
- ❍R-型和I-型指令间数据相关
- ❍I-型和R-型指令间数据相关
- ❍I-型和I-型指令间数据相关



流水线冒险——控制冒险

❒ 控制冒险

条件转移指令做出决定之前，其后续指令已经出现在流水线中。这时，如果决定的结果与已经进入流水线的指令序列不同的话，条件转移之后指令的执行就是无效的。



指令流水线性能

通常，流水线的性能用吞吐率、加速比和效率三项指标来衡量。

❒ 吞吐率

- ❍ 吞吐率指单位时间内流水线所完成指令或输出结果的数量。
- ❍ 吞吐率又有最大吞吐率和实际吞吐率之分。
  - ✧ 最大吞吐率指流水线在连续流动达到稳定状态后所获得的吞吐率。

$$TP_{\max} = \frac{1}{t^0}$$

  - ✧ 实际吞吐率指流水线完成 n 条指令的实际吞吐率。

$$TP_a = \frac{n}{mt^0 + (n-1)t^0 + kt^0} = \frac{n}{t^0(n+m+k-1)} = \frac{n}{n+m+k-1} TP_{\max}$$

指令流水线性能（续）

❒ 加速比

- ❍ 加速比是某程序以流水线执行的速度与等功能的非流水线执行的速度之比。
- ❍ 加速比S

$$S = \frac{nmt^0}{mt^0 + (n-1)t^0} = \frac{nm}{m+n-1} = \frac{m}{1+(m-1)/n}$$

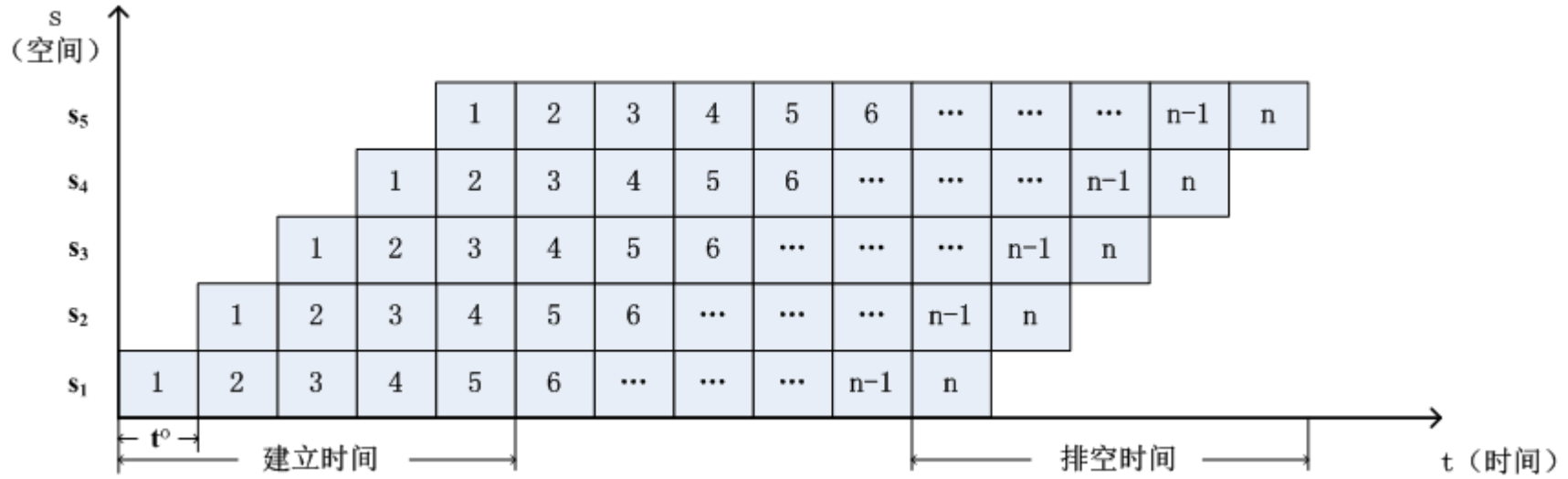
- ❍ 在$n>>m$时，$S_{max} \approx m$。即当流水线各段时间相等时，其最大加速比等于流水线的段数。

指令流水线性能（续）

❒ 效率

- ❍效率是指流水线中各功能段部件的利用率。
- ❍*m*段流水线连续执行*n*条指令的效率为：

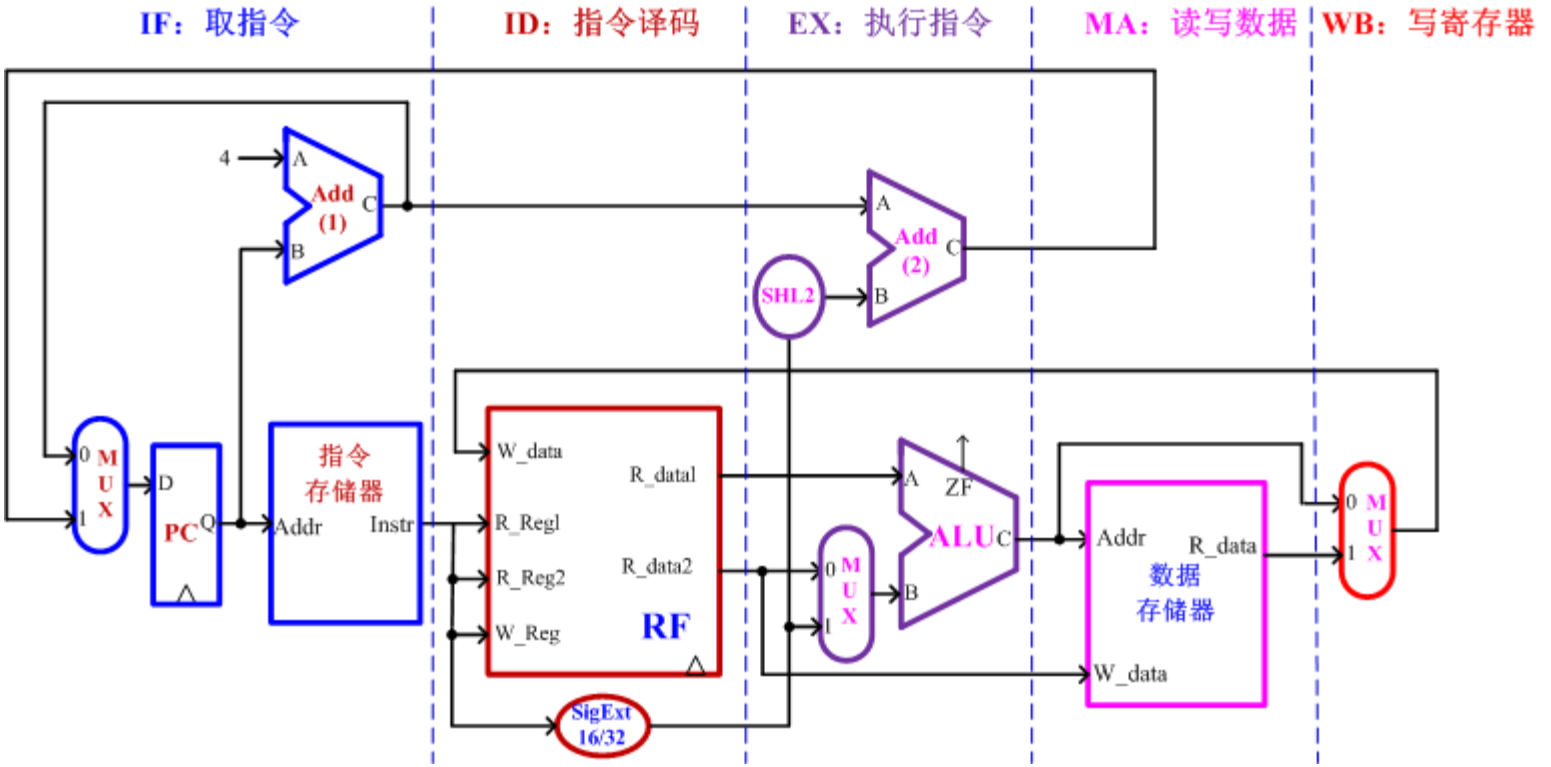
$$E = \frac{mnt^0}{m(m+n-1)t^0} = \frac{n}{m+n-1} = \frac{1}{m}S = TP_p \cdot t^0$$



同步流水线一般结构

单周期数据通路可以避免流水线的结构冒险，通常，通过改进单周期数据来实现流水线数据通路。

❒ 单周期数据通路操作分析



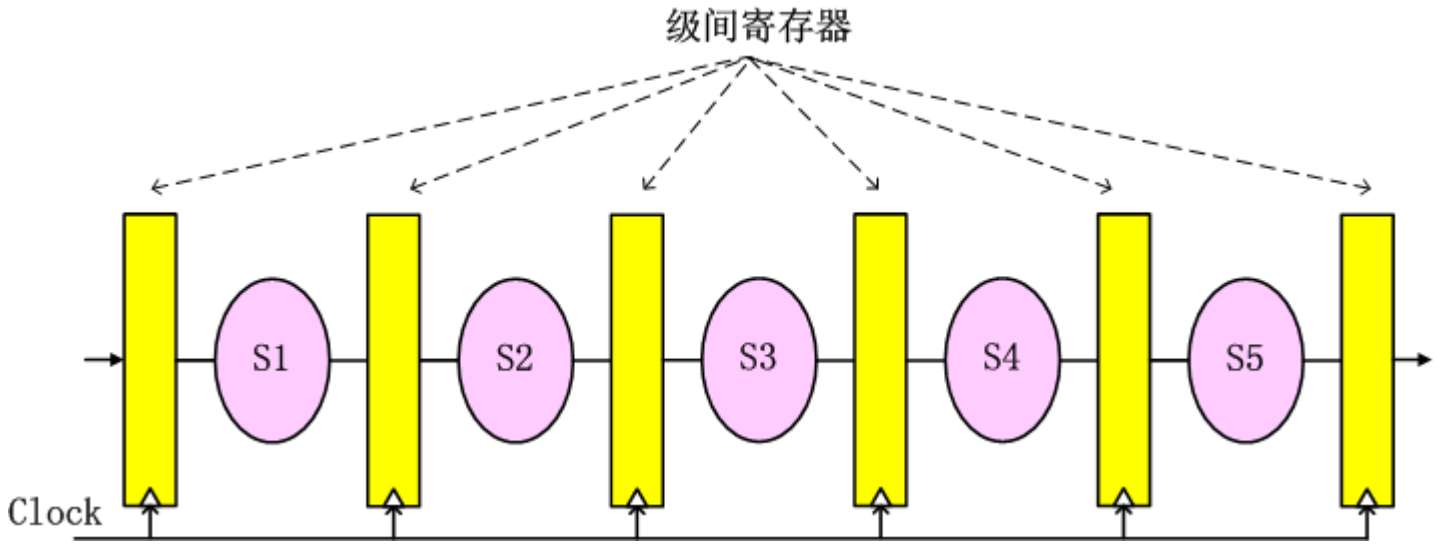
同步流水线一般结构（续）

- 取指令（FI）：先从指令存储器中取出指令，再由加法器I修正PC值（PC+4），然后写回到PC。
- 指令译码（ID）：由控制器对指令操作码进行译码确定指令类型，并且从寄存器堆读出数据。
- 执行指令（EX）：由ALU对数据进行加工或者计算操作数地址，并且由加法器II计算转移地址。
- 存储器访问（MA）：从数据存储器读取数据，或者向数据存储器写入数据。
- 写寄存器（WB）：将数据写入寄存器堆。

同步流水线一般结构（续）

- **同步流水线数据通路框架**

  为了使多条指令共享流水线数据通路，且保证流水线能够正确处理指令，必须在两个流水段之间设置缓存，称为**级间寄存器**。



流水CPU基本数据通路

- 设置级间寄存器
  - FI/ID级间
    - 指令寄存器（IR）：指令代码
    - 顺序程序计数器（NPC1）：顺序执行指令地址
  - ID/EX级间
    - 指令寄存器（IR）：指令代码
    - 顺序程序计数器（NPC1）：顺序执行指令地址
    - 源寄存器1（Rs）：Rs的内容
    - 源寄存器2（Rt）：Rt的内容
    - 立即数寄存器（Imm-32）： 16位立即数符号扩展后的32位立即数

流水CPU基本数据通路（续）

❍EX/MA级间

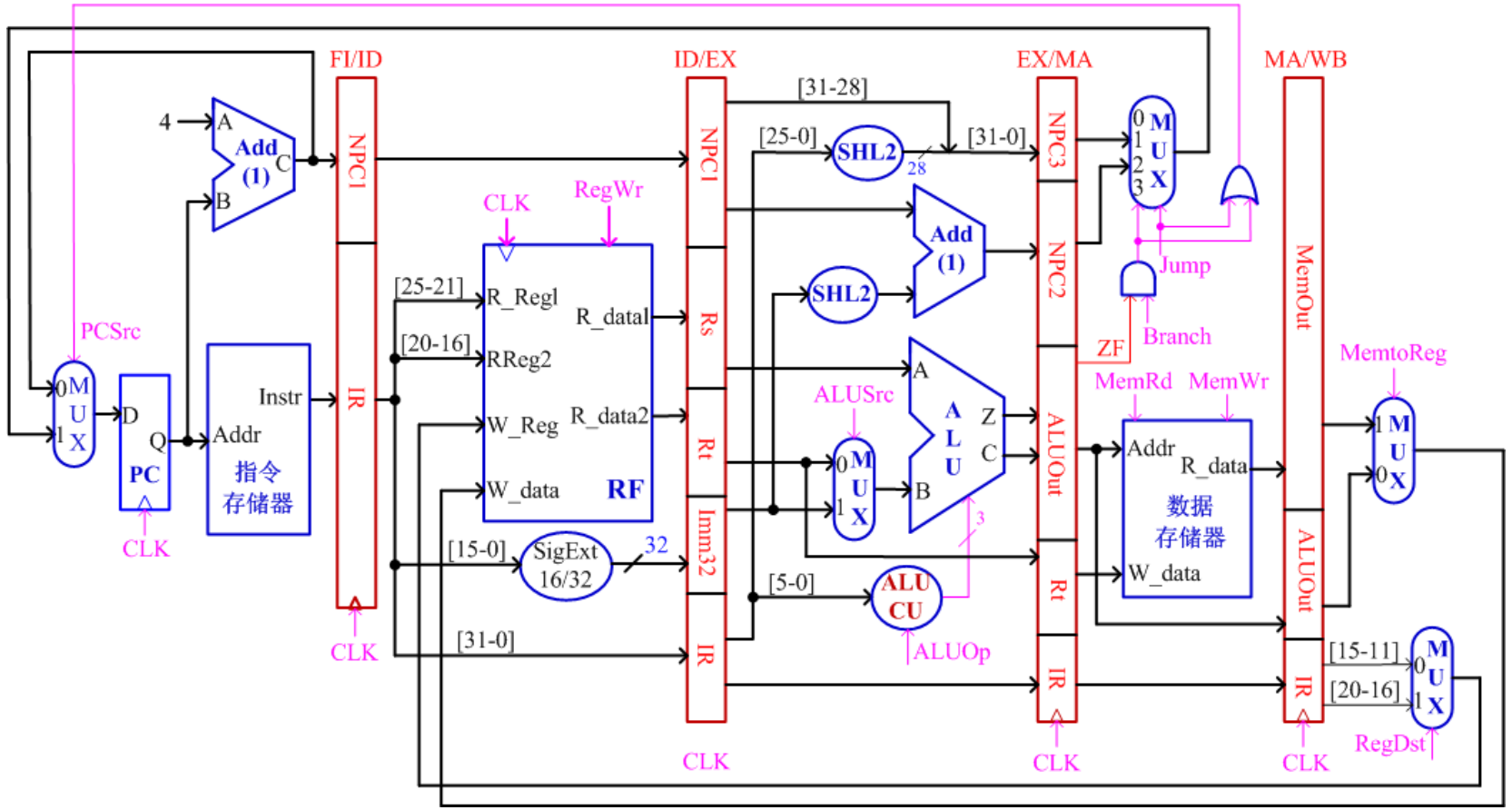
- ✧指令寄存器（IR）：指令代码
- ✧源寄存器2（Rt）：Rt的内容
- ✧分支地址寄存器（NPC2）：分支指令的目的地址
- ✧转移地址寄存器（NPC3）：转移指令的目的地址
- ✧ALU结果寄存器（ALU-Out）：ALU执行结果

❍MA/WB级间

- ✧指令寄存器（IR）：指令代码
- ✧ALU结果寄存器（ALU-Out）：ALU执行结果
- ✧数据存储器读出数据寄存器（MEM-Out）：存储器数据

流水CPU基本数据通路（续）

❒ 流水CPU基本数据通路

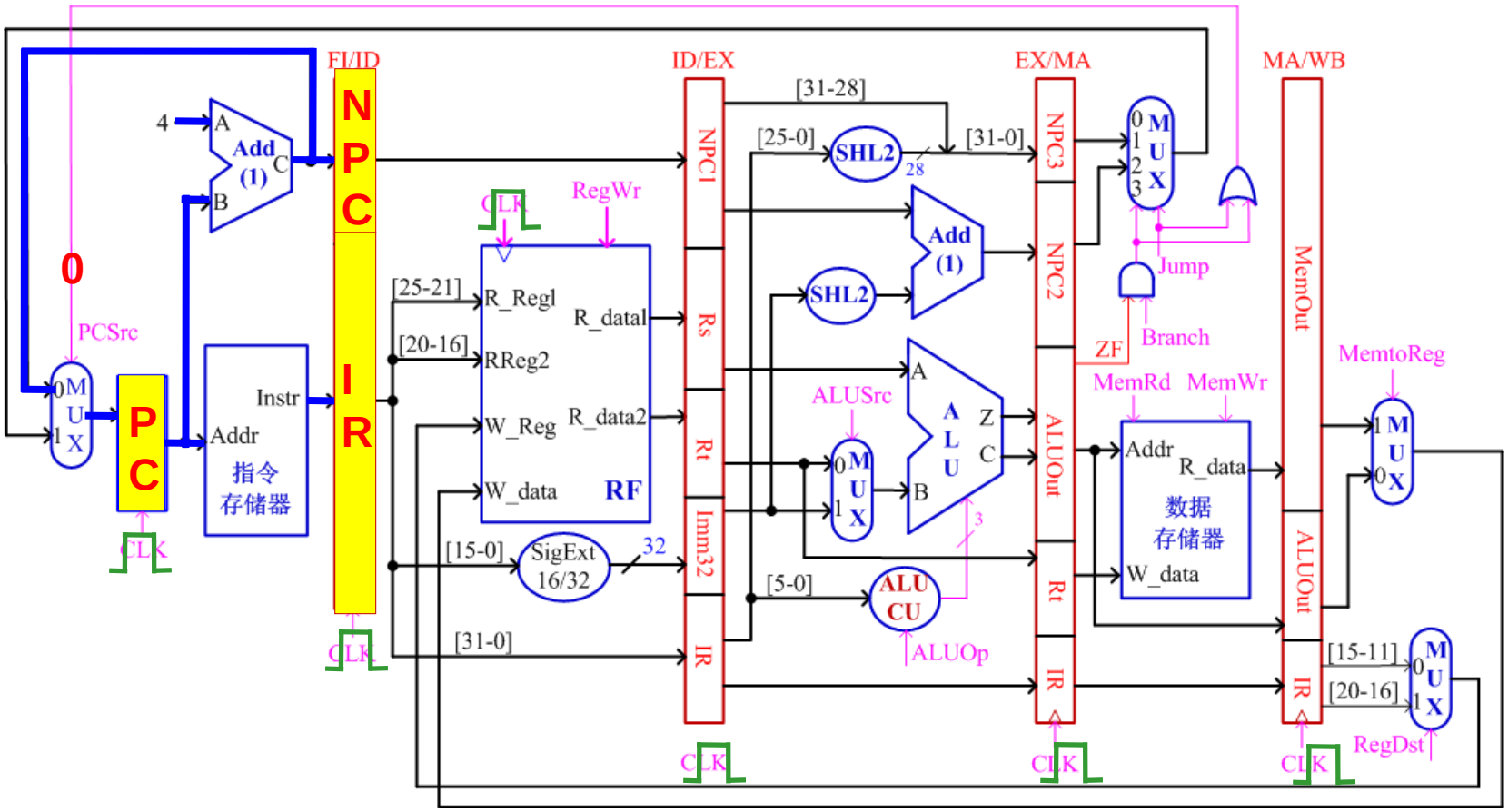


流水CPU指令周期

- 假设下列 5条指令在流水线上连续执行
  - lw $s0,100($s1)
  - sw $s2,200($s3)
  - add $t0,$t1,$t2
  - beq $t3,$t4,300
  - j 500

流水CPU指令周期数据流图

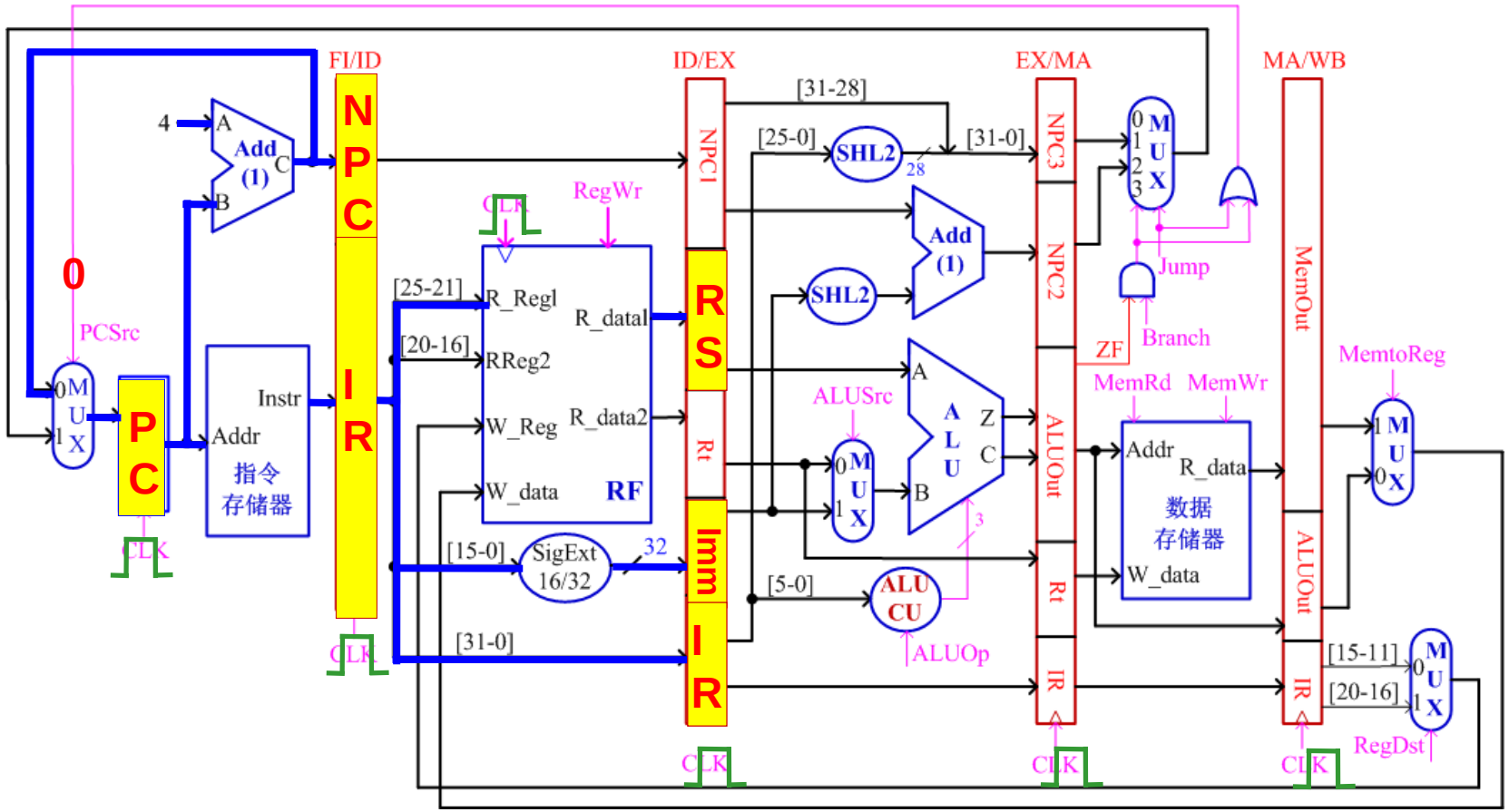
- 第一个时钟周期



lw $s0,100($s1)

流水CPU指令周期数据流程（续）

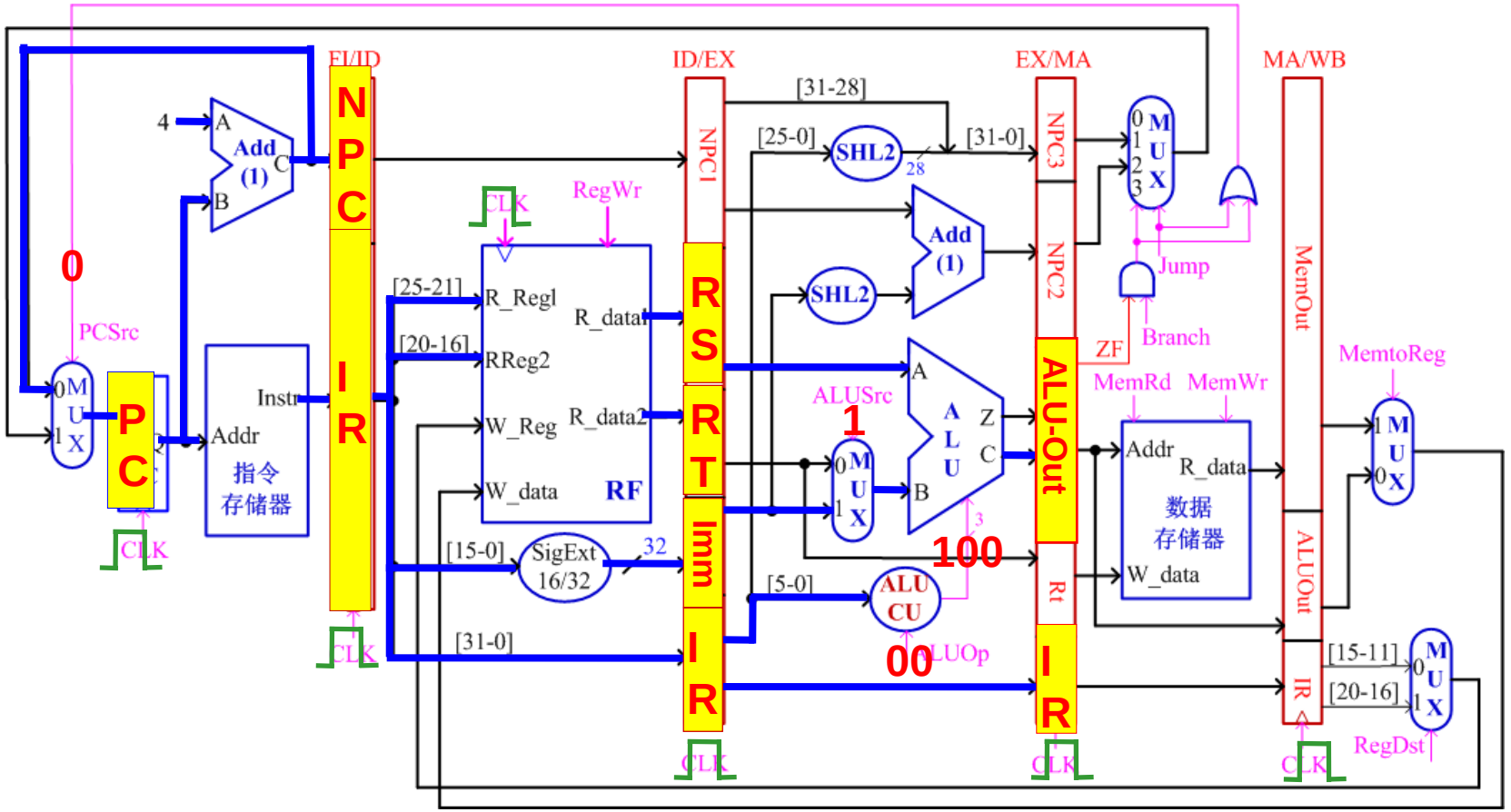
❍第二个时钟周期



sw $s2,200($s3)　　lw $s0,100($s1)

流水CPU指令周期数据流图（续）

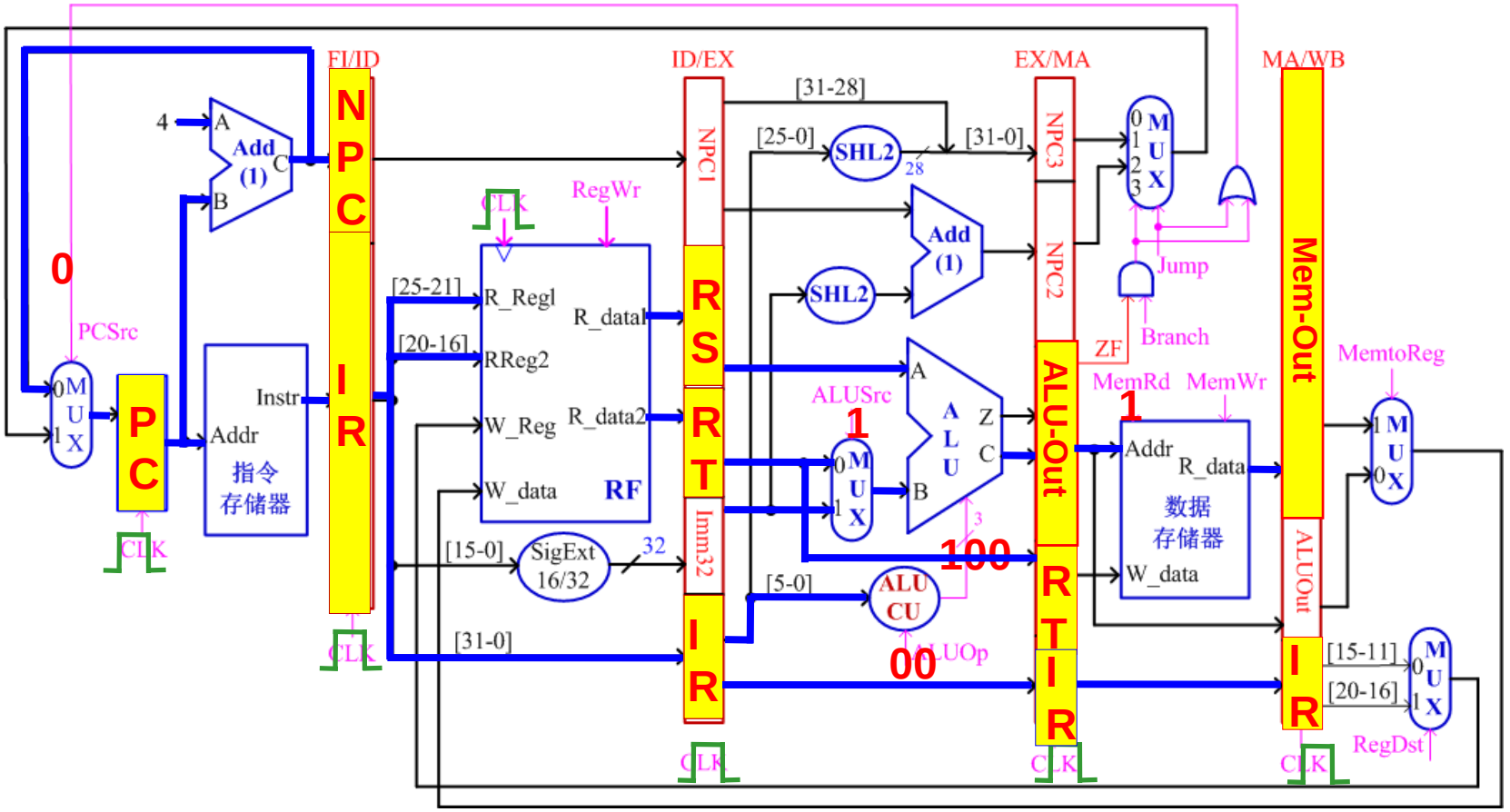
- 第三个时钟周期



add $t0,$t1,$t2　　sw $s2,200($s3)　　lw $s0,100($s1)

流水CPU指令周期数据流图（续）

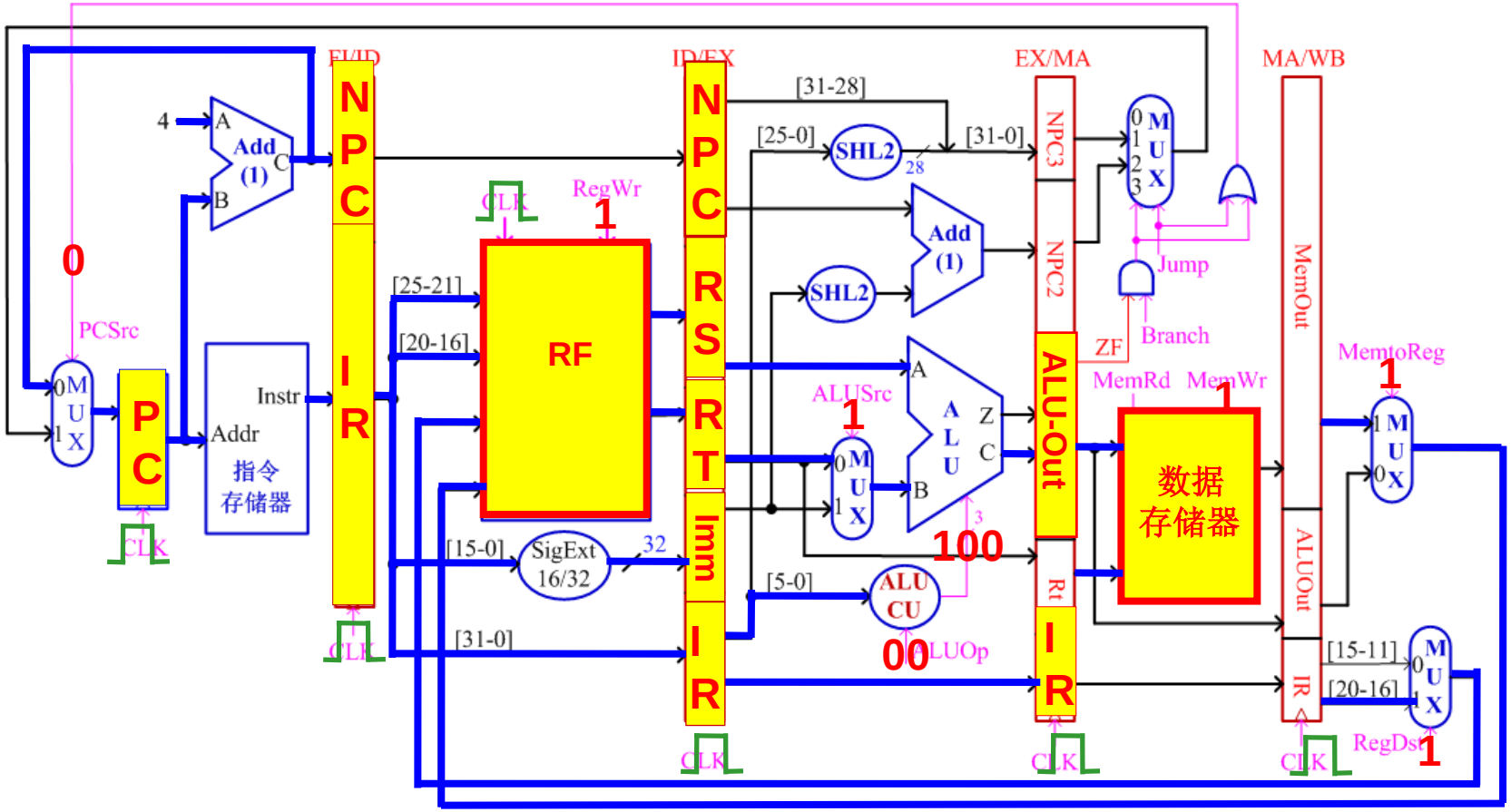
- 第四个时钟周期



beq $t3,$t4,300　add $t0,$t1,$t2　sw $s2,200($s3)　lw $s0,100($s1)

流水CPU指令周期数据流图（续）

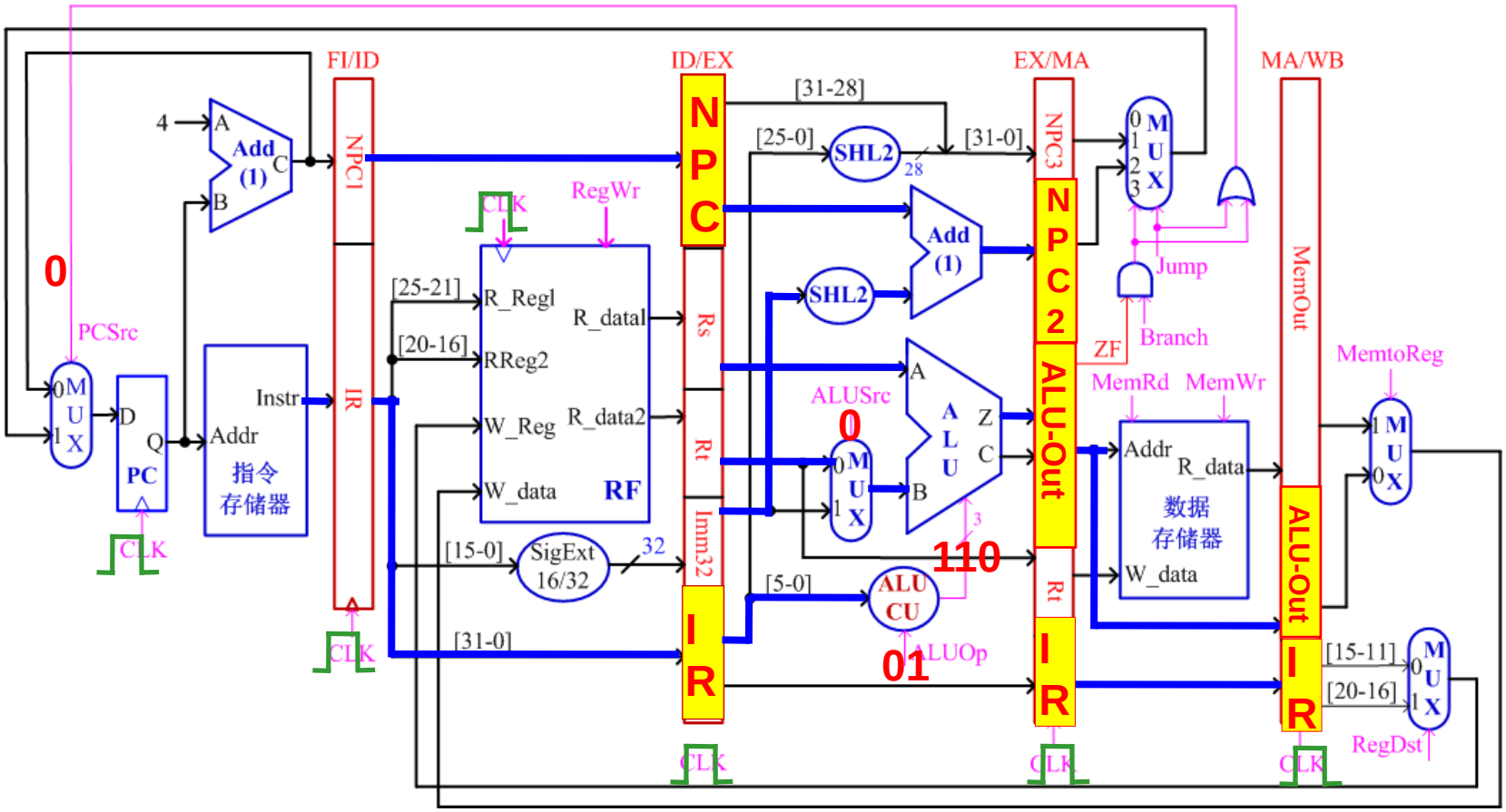
- 第五个时钟周期



j 500　　beq $t3,$t4,300　　add $t0,$t1,$t2　　sw $s2,200($s3)　　lw $s0,100($s1)

流水CPU指令周期数据流图（续）

- 第六个时钟周期



本章作业（总第10次作业）

6.3

6.15

6.20