提纲

龙芯中科
LOONGSON TECHNOLOGY

- CPU设计思路
- MIPS指令集
- 单周期CPU设计
- 流水线基本概念
- CPU流水线设计

提纲

龙芯中科
LOONGSON TECHNOLOGY

CPU设计思路

MIPS指令集

单周期CPU设计

流水线基本概念

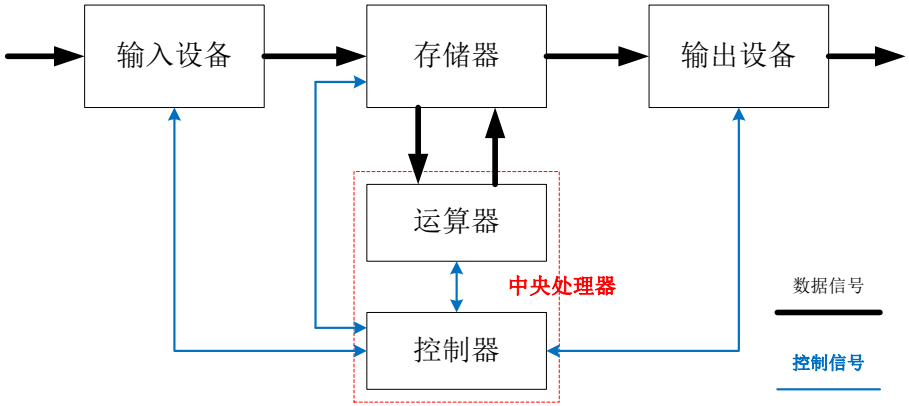
CPU流水线设计

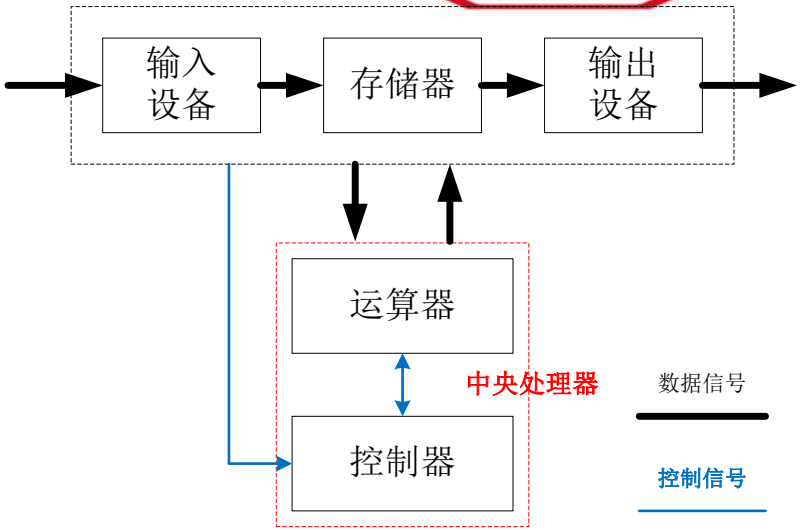
CPU设计思路之一

龙芯中科 LOONGSON TECHNOLOGY

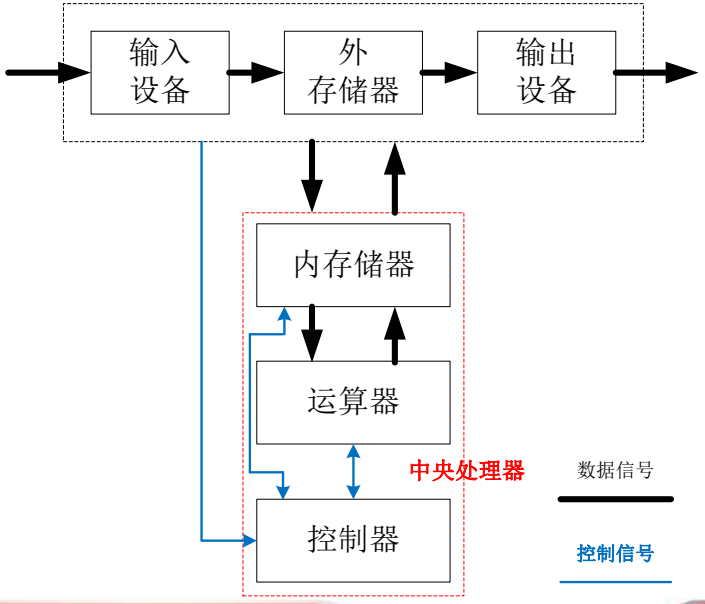
- **首先，搞清楚CPU这个数字逻辑系统的输入、输出是什么，要完成什么功能。**
  - 通过《数字逻辑》课程的学习和实验，每位学生对于从无到有设计一个数字逻辑系统可能有自己的一套解决思路。我的个人经验是：上来先要搞清楚这个数字系统的输入、输出是什么，要完成什么功能，简单来说就是对什么样的输入进行怎样的处理输出什么样的结果。
  - 要搞清楚CPU这个数字逻辑系统的输入、输出及其功能，我们必须掌握两个基本概念：
    - **冯•诺依曼体系结构**
    - **通用计算机系统的结构层次**

冯•诺依曼体系结构





- 几乎所有现代计算机都是遵循“冯诺依曼体系结构”的。左上图是冯诺依曼体系结构经典图示。具体实现中，I/O与存储器被统一编址，绝大部分控制信号（除中断信号）也被视作数据进行处理，形成如右上图的结构。由于存储器离运算器距离远、延迟大，所以进一步将一部分存储器内容（或内容镜像）存放到CPU内部，形成如右下图的结构。



通用计算机系统的结构层次

CPU设计思路之一

龙芯中科 LOONGSON TECHNOLOGY

- **在理解“冯•诺依曼体系结构”和“通用计算机系统结构层次”两个概念后，我们可以得出如下初步认识：**
  - CPU对外的接口通常包括：时钟输入、复位输入、中断信号输入、交互式总线（进一步包括命令、地址、数据信息）；
  - CPU的功能是：保证ISA中每条指令都能执行且执行正确
  - CPU是个“永动”的机器，其基本工作机制是：
    - 根据指令指针从存储器中取出指令
    - 控制器根据解析出的指令信息产生对于内部运算器、存储器和外设的控制信号
    - 内部运算器在控制信号的控制下，从存储器中取出源操作数，进行处理后，将结果写回到存储器中
    - 当前指令完成后，指令指针移到下一条指令的存储地址
  - 上面的描述中，请将存储器视作一个一般性的概念，即存储一切软件可见信息的媒介，它的具体对象可能是：硬盘、Flash Mem、DDR内存、Cache、Regfile、Control Register、PC、……

CPU设计思路之二

- **CPU是一个数字逻辑系统，因此在设计时可以按照“control path+data path”的思路进行考虑。**
- **优先考虑“data path”，随后根据data path的设计需要添加“control path”。**
- **何谓“data path”？就是“data”流经的路径。何谓“data”？就是指令中的“宾语”所包含的内容。下面我们以“data path”的视角重新看CPU的运作机制（红色部分是“data”）：**
  - 根据指令指针从存储器中取出指令
  - 控制器根据解析出的指令信息产生对于内部运算器、存储器和外设的控制信号
  - 内部运算器在控制信号的控制下，从存储器中取出源操作数，进行处理后，将结果写回到存储器中
  - 当前指令完成后，指令指针移到下一条指令的存储地址

CPU设计思路之二

龙芯中科 LOONGSON TECHNOLOGY

- **“control path”要施加怎样的控制呢?**
  - 根据当前指令有没有做完的信息决定是否取入下一条指令
  - 这一级流水信息能否在即将到来的时钟上升沿进入下一级流水
  - ALU中从adder、shifter、logic模块中选择哪个结果作为ALU计算的结果
  - ……

CPU设计思路之三

龙芯中科 LOONGSON TECHNOLOGY

- 每条指令在CPU中都需要完成取指、译码、操作数读取、执行、写结果等操作，显然我们不会对每个指令都各自独立设计一套用于上述操作的data path和control path，我们的思路是设计出一个具有取指功能的data path和control path能够支持所有指令的取指，设计出一个具有译码功能的data path和control path能够完成所有指令的译码，……
- 上述思路要求我们把所有指令的功能按照取指、译码、操作数读取、执行、写结果等方面进行分解，再将各个分项需要的所有功能总结归纳，提炼出功能点，进而根据功能点的并集设计出各个分项的data path和control path

提纲

龙芯中科
LOONGSON TECHNOLOGY

- CPU设计思路
- MIPS指令集
- 单周期CPU设计
- 流水线基本概念
- CPU流水线设计

CPU设计者眼中的MIPS指令集(1)

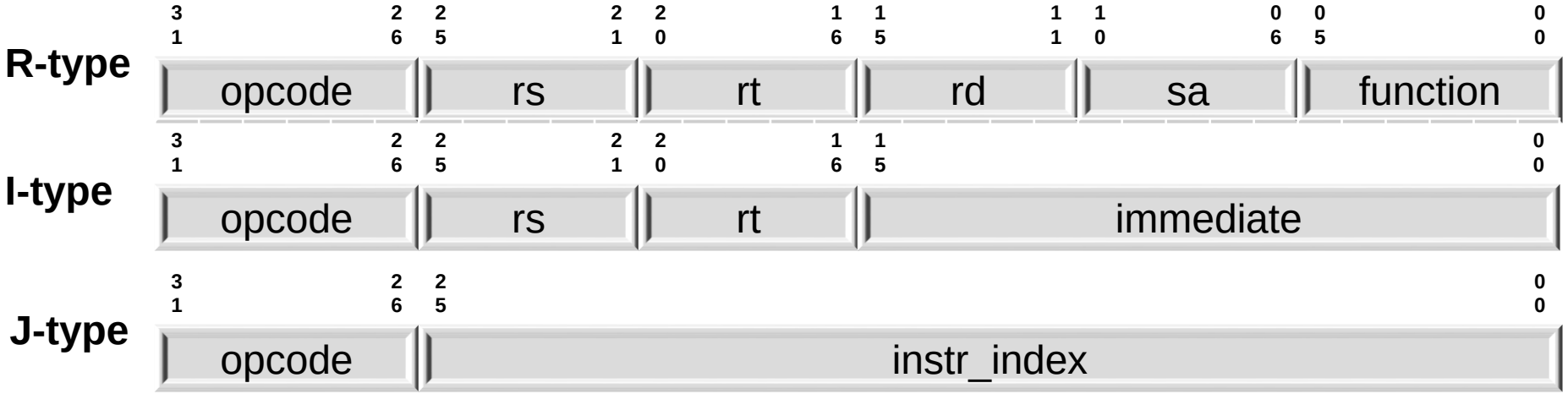
龙芯中科 LOONGSON TECHNOLOGY

- **MIPS指令集中每个软件可见的寄存器，CPU中一定需要有物理上的对应实现，这些寄存器有：**
  - PC寄存器
  - 通用寄存器(GPR)：32个×32bit
  - HI/LO寄存器：存放乘法和除法的结果
  - 控制寄存器(CP0寄存器)

CPU设计者眼中的MIPS指令集(2)

龙芯中科 LOONGSON TECHNOLOGY

- **MIPS指令集中每条指令都是32bit宽，且格式主要分为R-type、I-type、J-type三类，得到的额外信息是：**
  - 顺序取指令时，下一条指令的PC是当前指令PC+4
  - 可以通过对opcode、function几个位置固定的域译码来获得指令操作类型
  - 产生寄存器堆的读地址时，可以在不知道该指令的具体操作类型情况下直接从指令码的指定位置获取（实验中让学生填表，其中目的之一就是用穷举的方式证实这一特性对于所有指令均成立）



CPU设计者眼中的MIPS指令集(3)

龙芯中科 LOONGSON TECHNOLOGY

- **从功能角度看，所有MIPS指令大致分为三类**
  - 计算类：从通用寄存器（或指令立即数域）获得1～2个源操作数，运算后结果写入通用寄存器；
  - 转移类：从通用寄存器获得0-2个源操作数，运算后根据结果修改PC；
  - 访存类：从通用寄存器和指令立即数域分别获得2个源操作数，相加所得结果作为地址，进而对存储器进行访问
    - 如果是取数指令，则从存储器中读出，结果写入通用寄存器
    - 如果是存数指令，则从通用寄存器中另外读取一个值，写入到存储器中
- **对上面归纳出的特性进一步分析，又可以发现**
  - 每一类当中都有运算的操作，且这个运算的源操作数个数不超过2个；
  - 分支类指令为了修改PC，还需要额外的一个运算；
  - 访存类指令只有先计算出地址，才能进行存储器的访问；
  - 每条指令会需要从存储器中取出，但是如果是访存指令还会再访问一次存储器，两种访问间存在竞争（实验中我们将指令存储和数据存储分开）

CPU设计者眼中的MIPS指令集(4)

龙芯中科 LOONGSON TECHNOLOGY

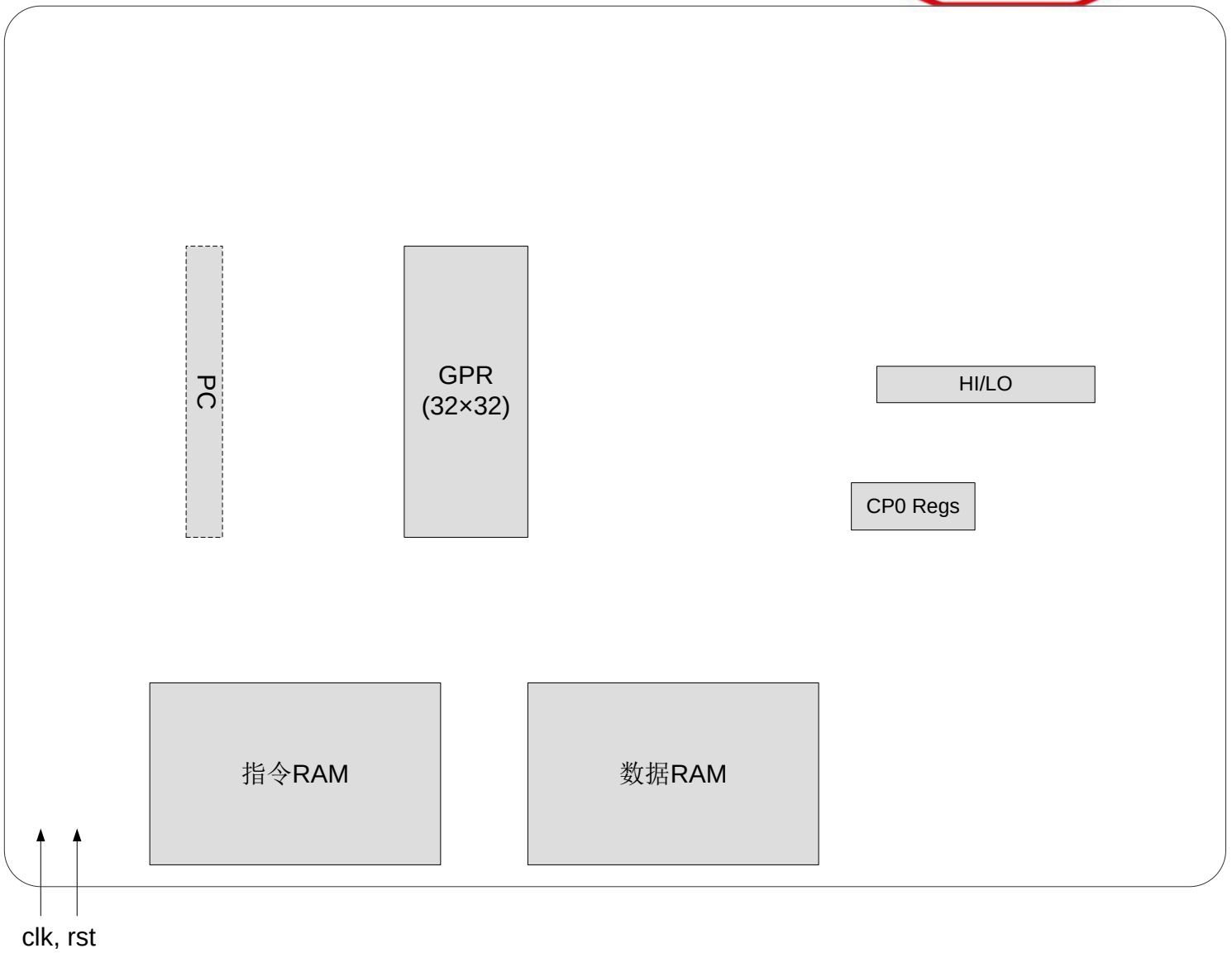
- **可以发现存在一些指令运算方式完全相同，只是源操作数的来源不同：**
  - ADDU和ADDIU、SLT和SLTI、SLTU和SLTIU、SLL和SLLV、SRL和SRLV、SRA和SRAV

提纲

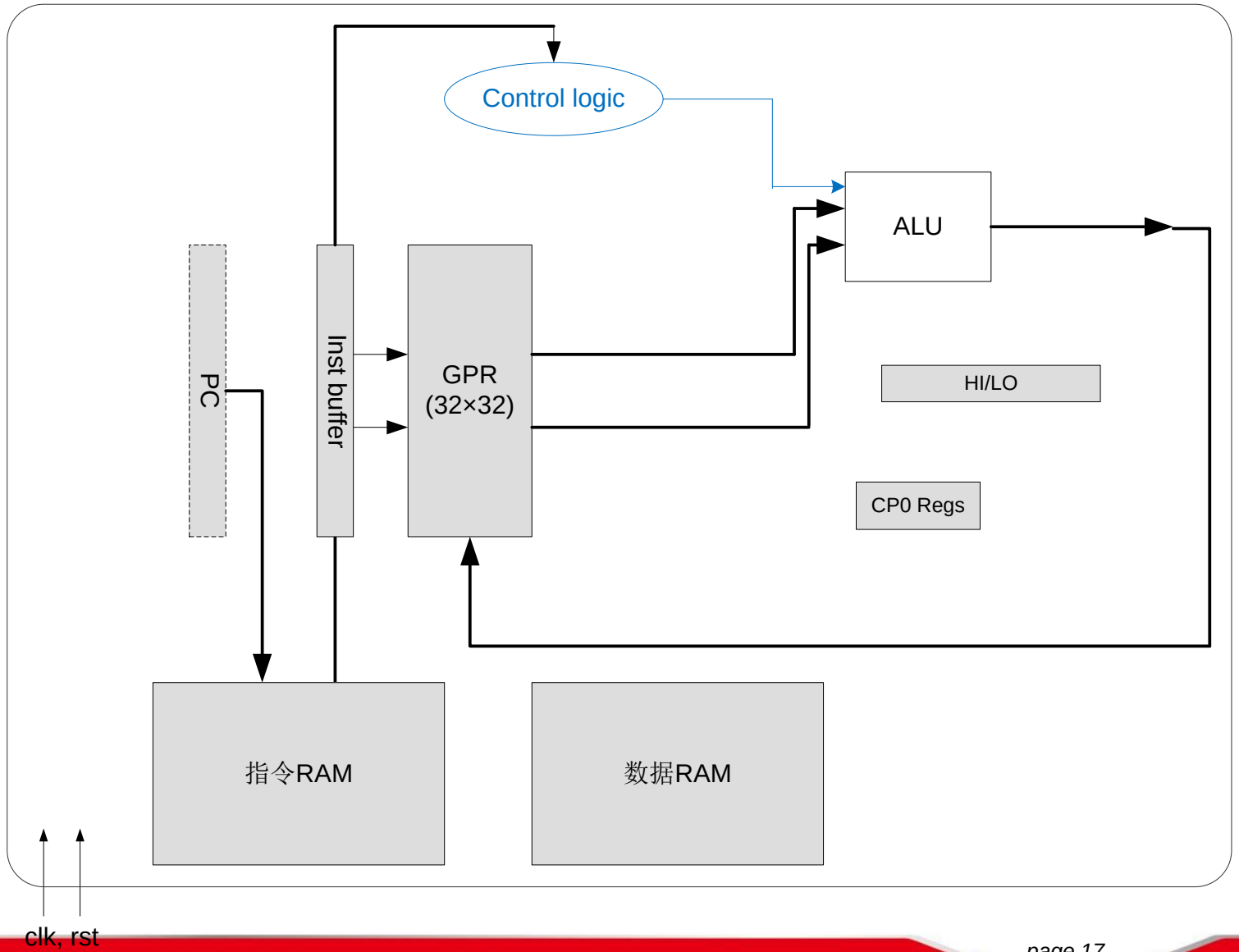
龙芯中科
LOONGSON TECHNOLOGY

- CPU设计思路
- MIPS指令集
- **单周期CPU设计**
- 流水线基本概念
- CPU流水线设计

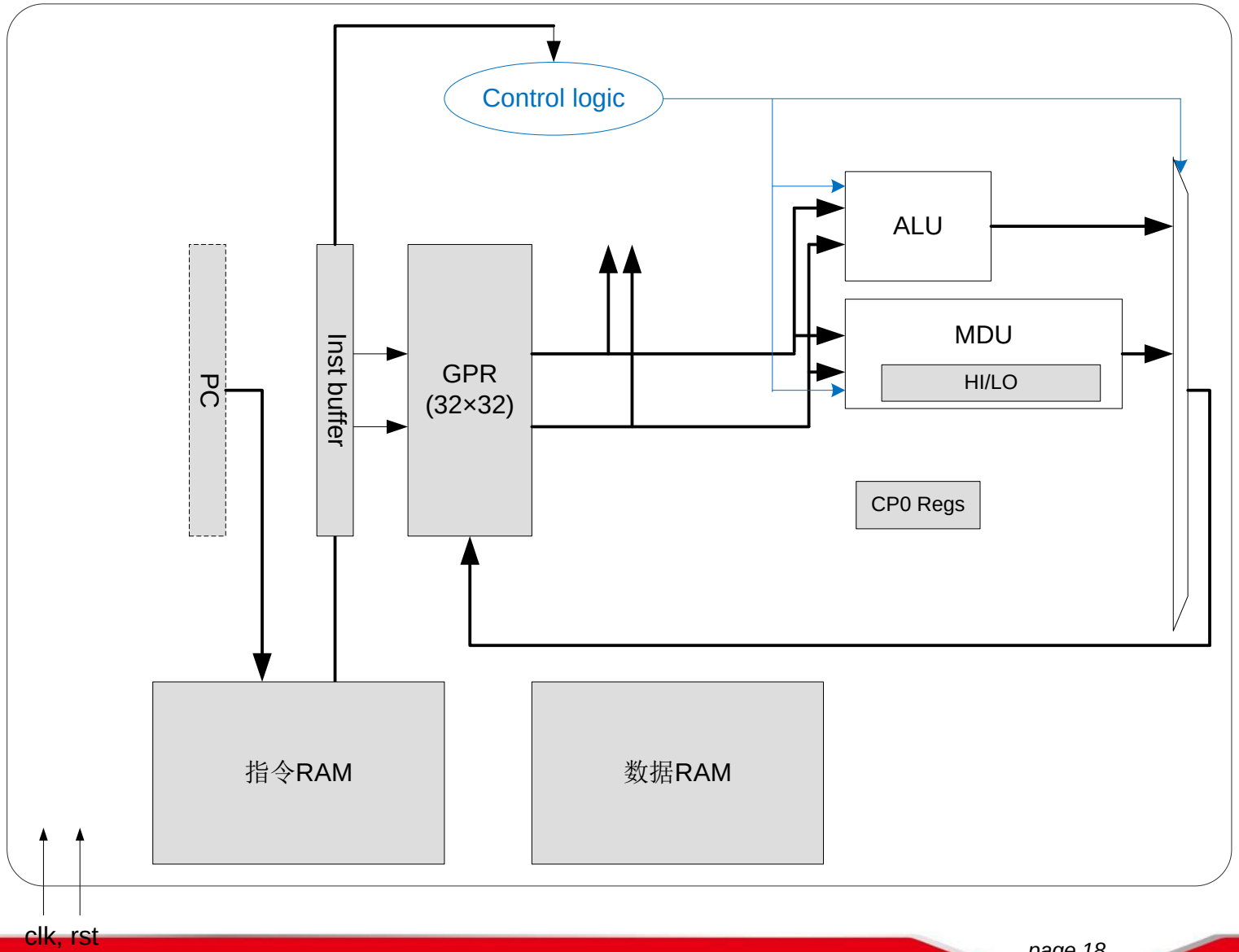
所有软件可见寄存器和内存



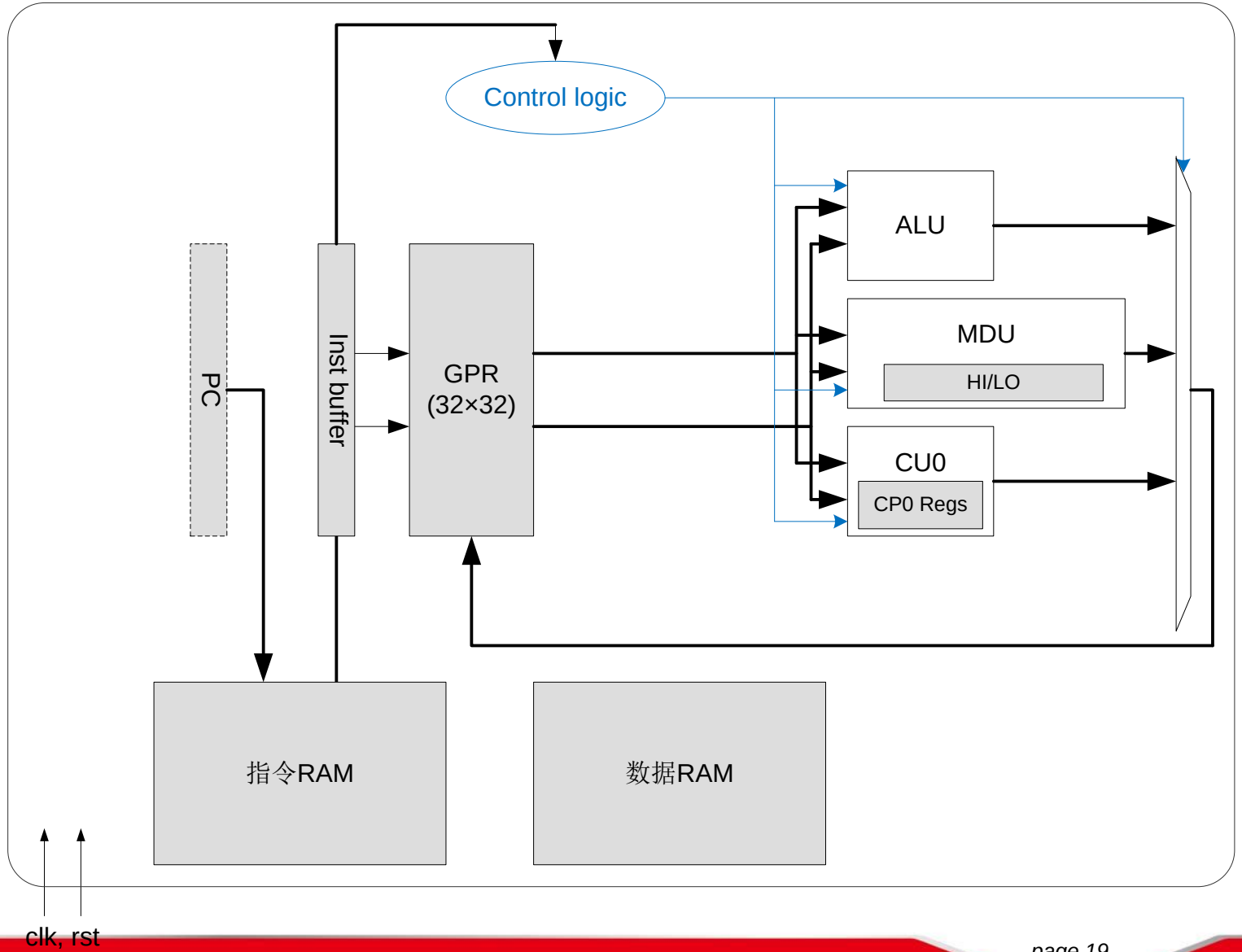
考虑运算类指令（ALU）


Control logic
ALU
PC
Inst buffer
GPR
(32×32)
HI/LO
CP0 Regs
指令RAM
数据RAM
clk, rst

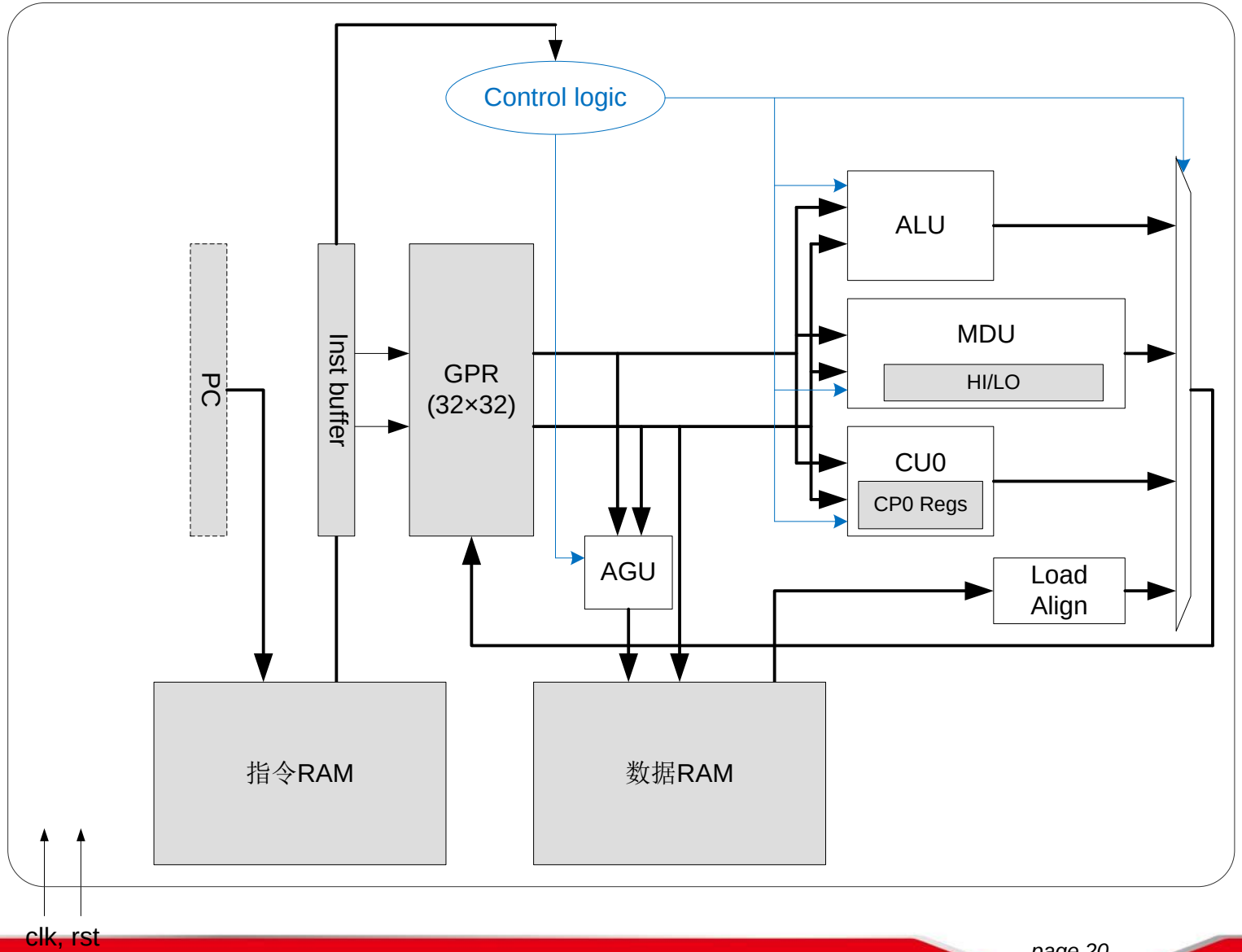
考虑运算类操作（MULT）



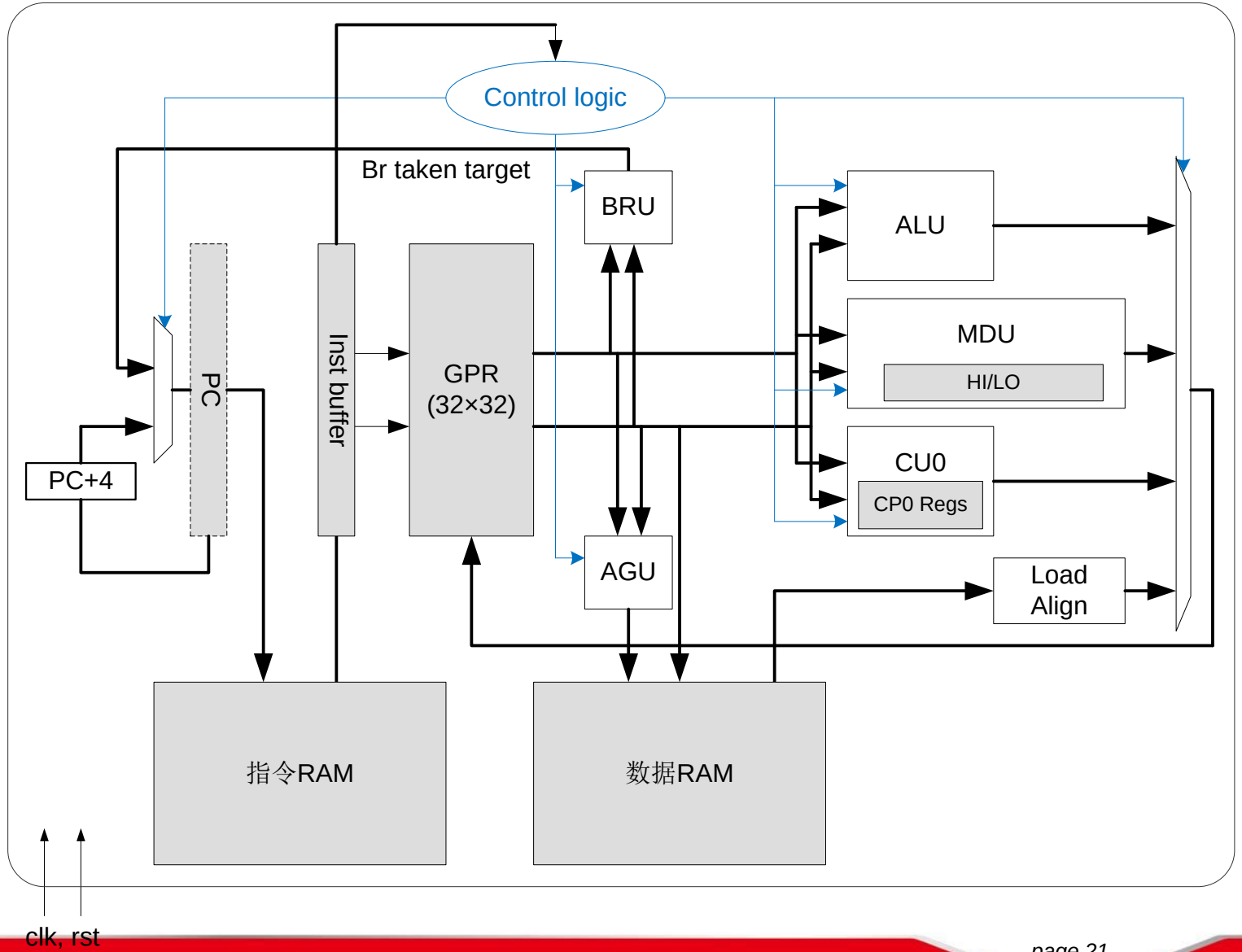
考虑与控制寄存器相关操作



考虑访存类操作


Control logic
ALU
MDU
HI/LO
CU0
CP0 Regs
Load
Align
PC
Inst buffer
GPR
(32×32)
AGU
指令RAM
数据RAM
clk, rst

考虑转移指令与PC维护



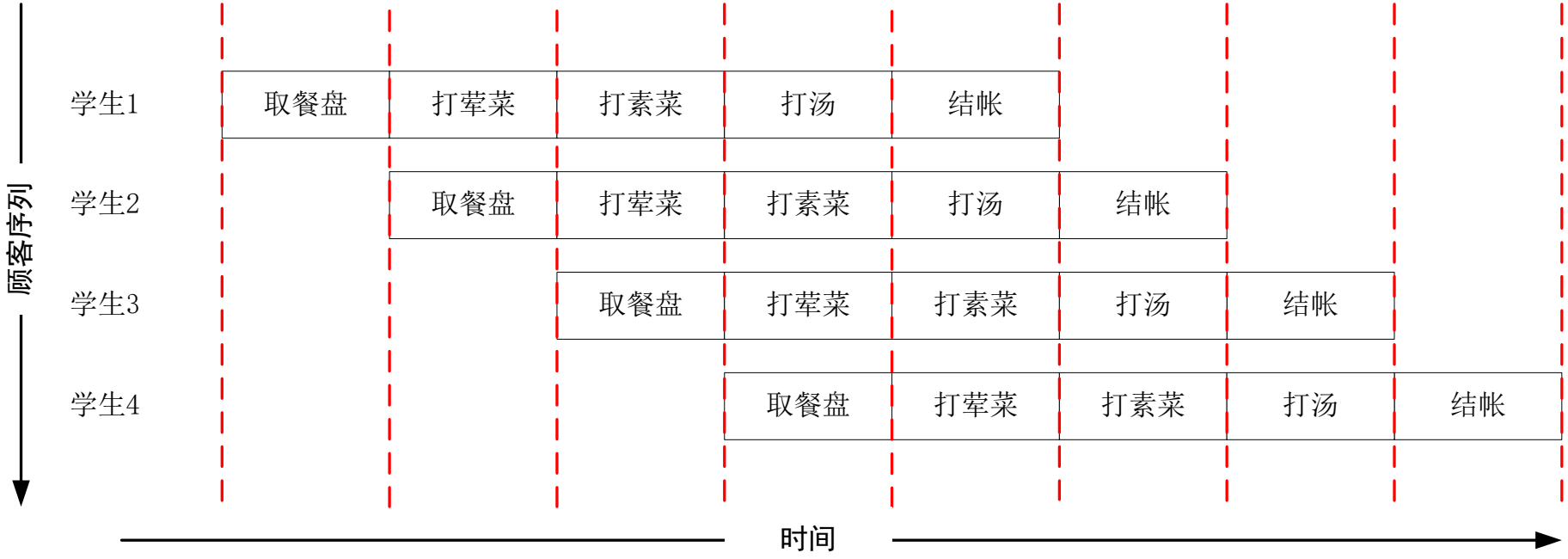
提纲

龙芯中科 LOONGSON TECHNOLOGY

- CPU设计思路
- MIPS指令集
- 单周期CPU设计
- **流水线基本概念**
- CPU流水线设计

流水线基本概念（1）

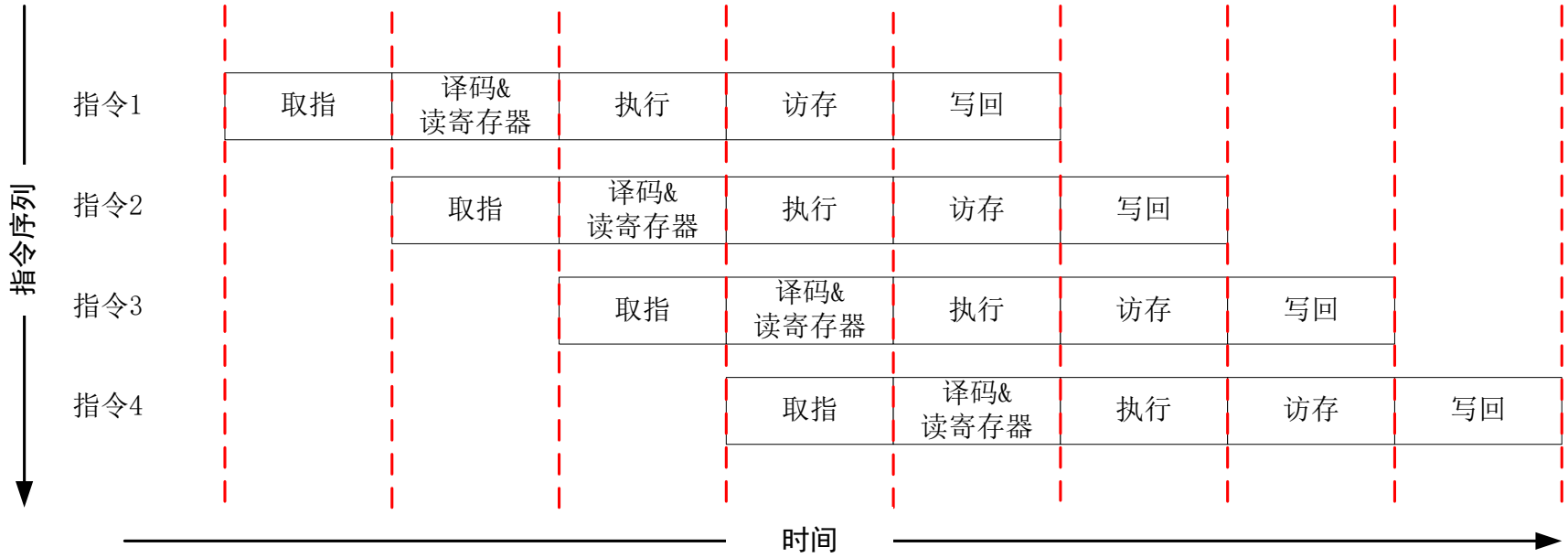
- **一个生活中的例子：学生食堂就餐**



- **问题1：后面的同学如果少打一个菜，能往前插队吗?**
- **问题2：前面的同学停下来，从它开始后面的队伍会被堵住吗?**
- **一个特点：打菜的师傅只根据当前在这个窗口前的那位同学的要求打菜，他/她不关心同一时刻其它窗口的同学的要求。**

流水线基本概念（2）

- **具体到CPU的例子**



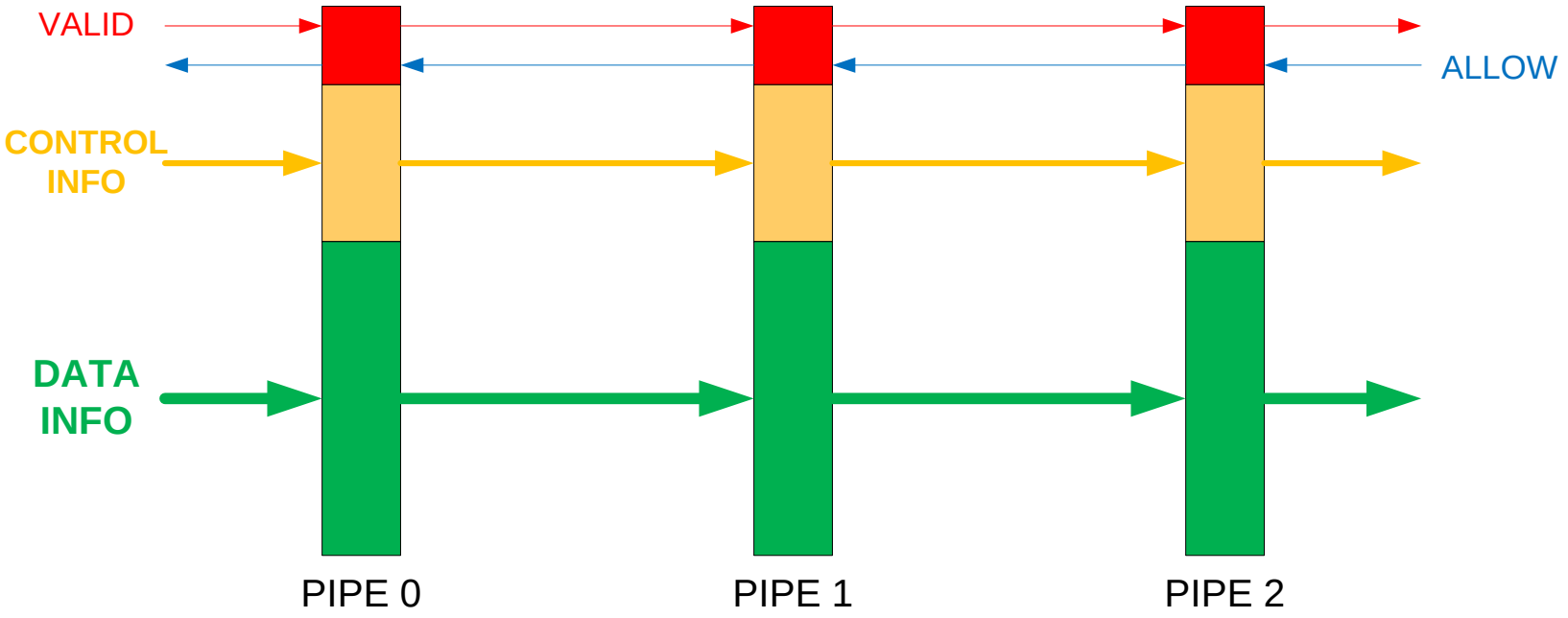
- **类似的问题：后面的指令可以越过前面的指令，提前进入后面的流水线吗?**
- **类似的特点：任何一级流水上的data path是受当前在这一级流水级上的那条指令控制的，与当前其它流水级的指令无关。**

设计流水线的一般性问题

龙芯中科 LOONGSON TECHNOLOGY

- **把一个动作拆分成若干个“分动作”，每个“分动作”对应一级流水，被操作的对象沿流水线逐级前进，进而依次完成每个“分动作”**
- **流水线的执行效率取决于最耗时的“分动作”，因此拆分时要考虑均衡性**
- **从实现角度看，拆分也要考虑拆分后“分动作”的功能界面是否清晰简洁**
- **流水线中会同时出现多个对象，这些对象在流水线能否前行取决于**
  - 该对象在当前流水级要做的事情完成了没有？
  - 该对象接下来需要进入的流水级，在“接下来”那个时间点上，是否有空位置？
    - 如果我们规定每个对象，无论其是否在每个流水级上都要做事情，都要按部就班的一级一级走下去，那么“需要进入的流水级”这个概念就可以简化为“下一级流水级”。

最简单流水线的结构图模板



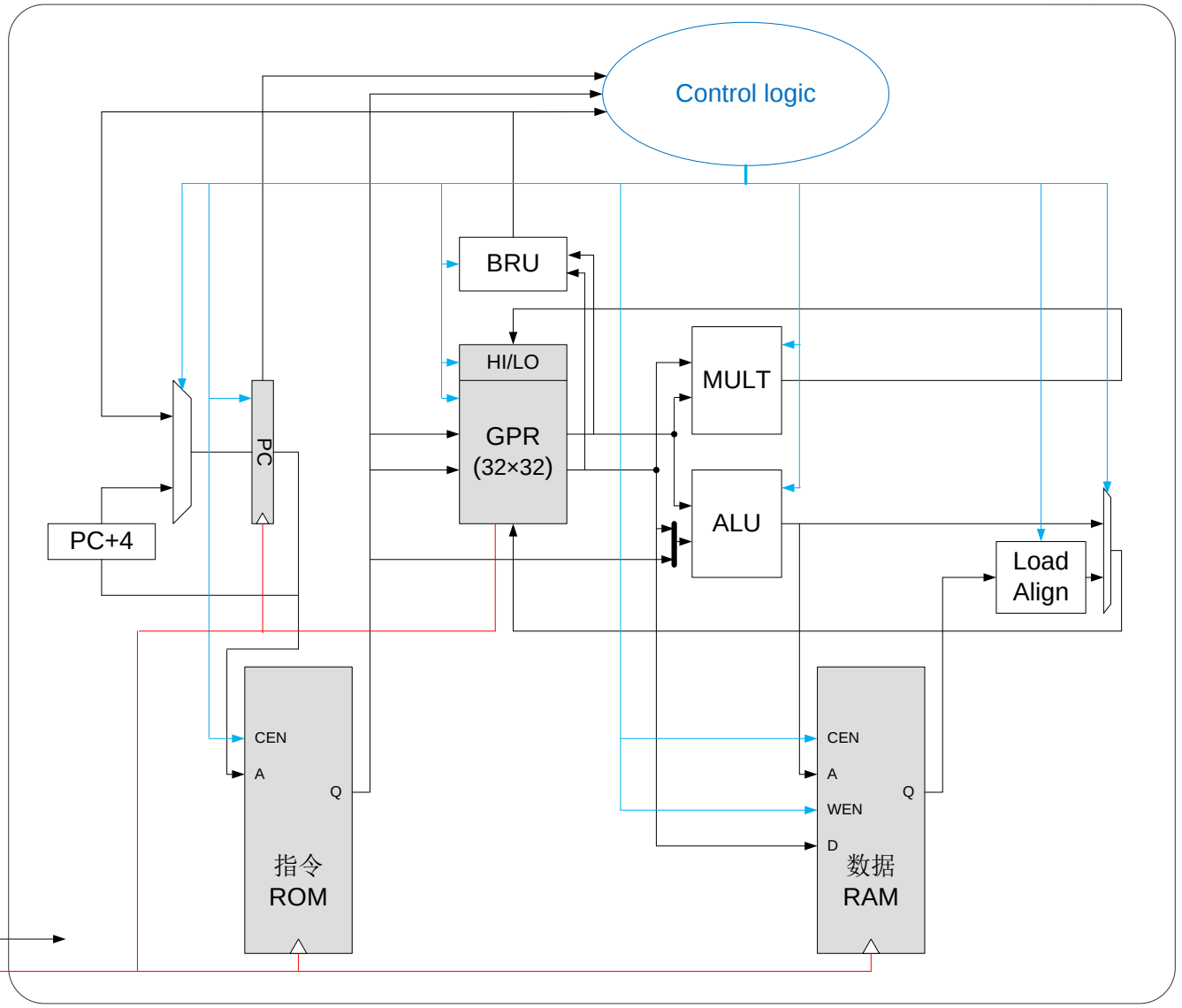
- **VALID：表示输入、输出或这一级流水上有没有有效内容**
- **ALLOW：表示是否允许进入该级流水**
- **CONTROL INFO：用于产生流水级内部状态控制信号的信息**
- **DATA INFO：纯数据型信息**

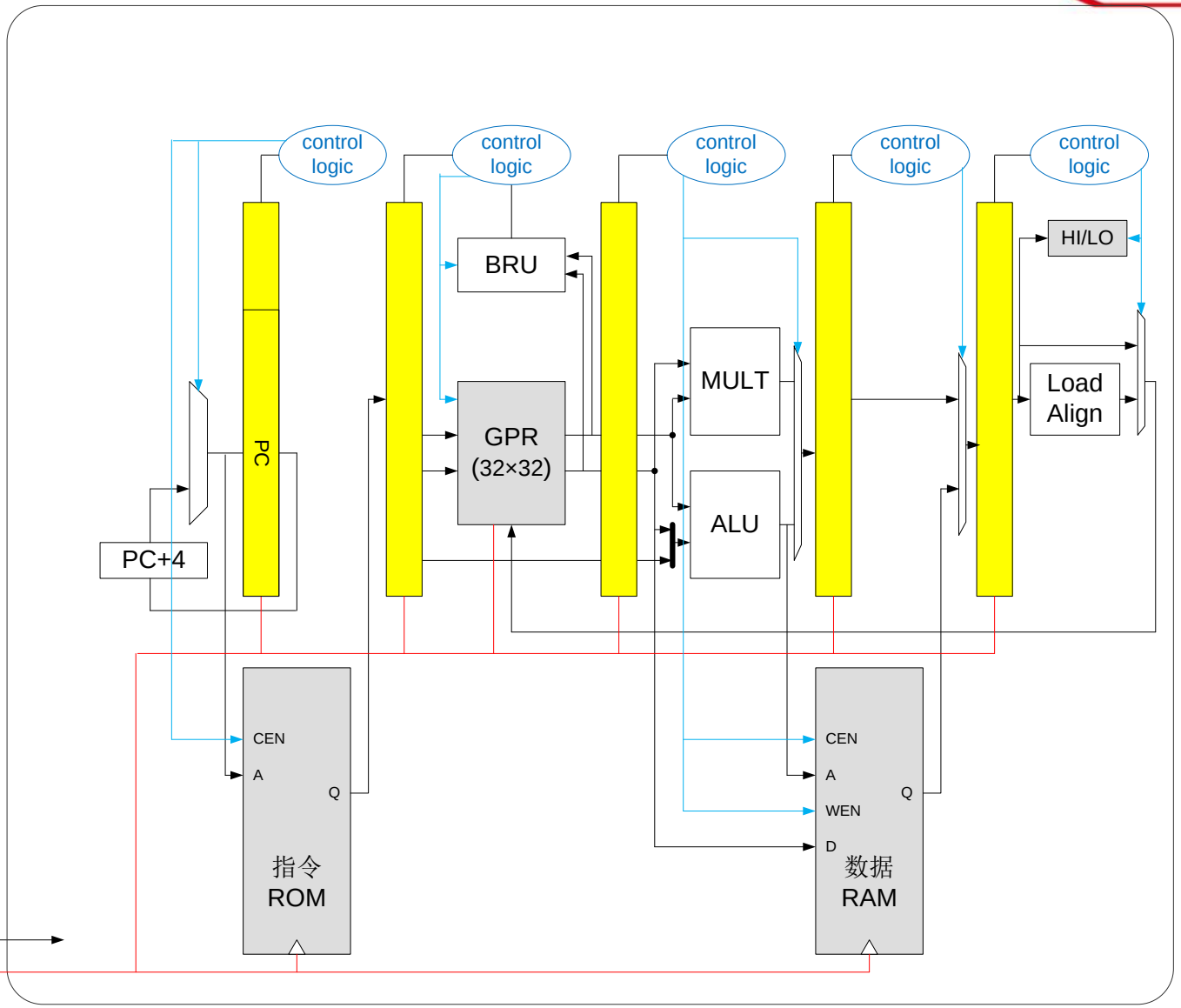
提纲

龙芯中科
LOONGSON TECHNOLOGY

- CPU设计思路
- MIPS指令集
- 单周期CPU设计
- 流水线基本概念
- CPU流水线设计

单周期CPU结构回顾


Control logic
BRU
HI/LO
GPR
(32×32)
MULT
ALU
PC
PC+4
Load
Align
CEN
A
Q
指令
ROM
CEN
A
WEN
D
Q
数据
RAM
rst
clk


control logic
control logic
control logic
control logic
control logic
PC
BRU
GPR (32×32)
MULT
ALU
HI/LO
Load Align
PC+4
CEN
A
Q
指令 ROM
CEN
A
WEN
D
Q
数据 RAM
rst
clk

其他

- **例外与中断：**

  例外与中断分为标记和报出两步。

  推荐都汇聚到一个流水级报出，汇聚点之后不会产生新的例外与中断。

- **控制相关：**
- **数据相关: forward**
- **结构相关：**

龙芯中科
LOONGSON TECHNOLOGY

Any Questions?