计算机设计与实践

流水线CPU设计

2022·夏

HITSZ 实验与创新实践教育中心
Education Center of Experiments and Innovations, HITSZ

目录

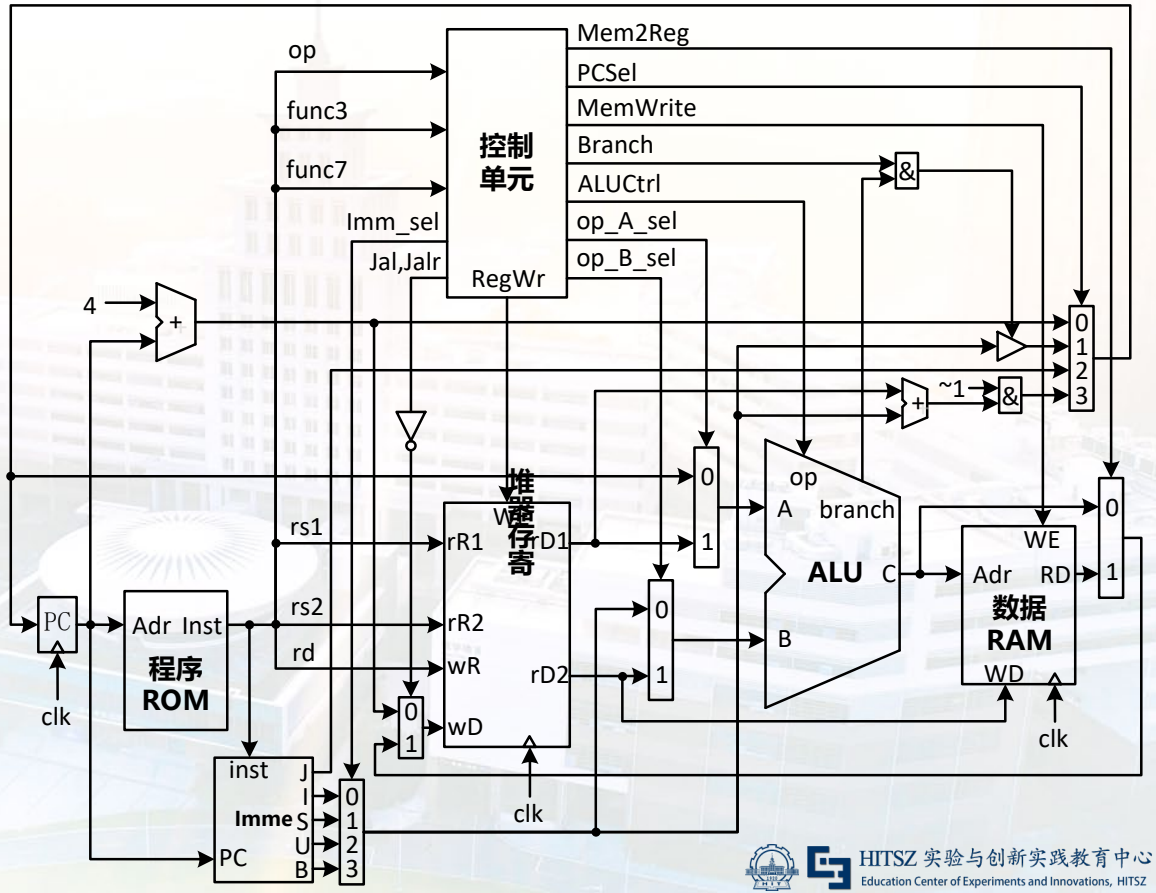
设计要求

- **单周期CPU回顾**

CPU频率：≥25MHz

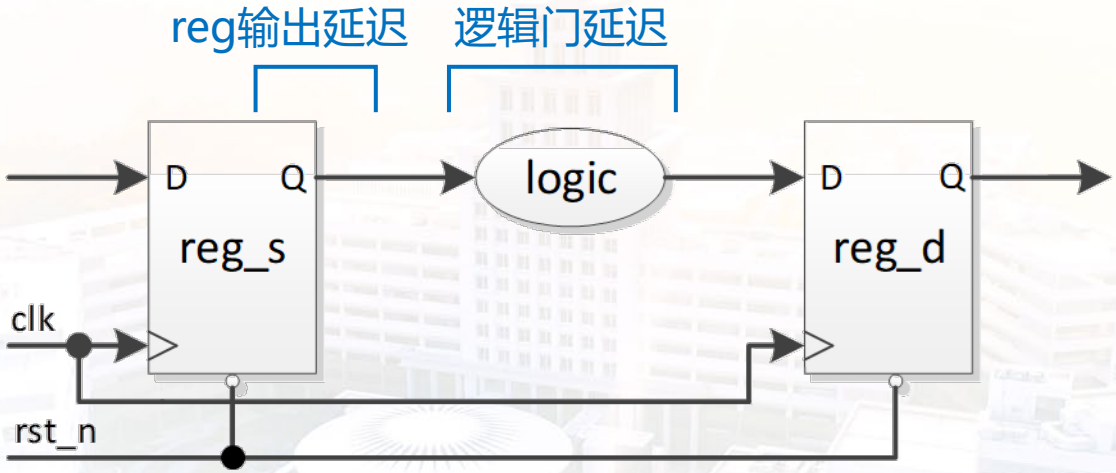
思考

如何提升CPU频率?



设计要求

- **CPU频率的决定因素**



$$\sum delay \leq T_{cycle}$$

设计要求

- 五级流水线 (至少)
- 最少支持24条常规指令 (可添加选做指令)
- 改造单周期CPU数据通路，实现理想流水线CPU
- 实现流水线暂停机制，解决数据冒险
- 实现数据前递机制，解决数据冒险
- 实现分支预测 (选做)，解决控制冒险
- 所实现的流水线CPU主频≥50MHz

目录

流水线概述

- 流水线：将指令处理过程**拆分**为多个步骤，并通过多个硬件处理单元**并行执行**来加快指令执行速度
- 理想条件下，流水线CPU的指令执行时间：

$T_{non-pipeline}$：非流水线指令执行时间

$T_{Pipeline}$：流水线指令执行时间

N：流水线级数

$$T_{Pipeline} = \frac{T_{non-pipeline}}{N}$$

流水线概述 – 流水线划分

- **流水线划分原则**
  - 按**功能**划分
    - 各段有明确的功能，或存在逻辑上具备先后关系的若干个阶段
  - 按**延时**划分
    - 为提高频率/性能，各段延时应尽量接近
  - 按**位宽**划分
    - 松耦合原则：各段之间的交互信号应尽量少

流水线概述 – 流水线划分案例

RISC经典五级流水（MIPS、RISC-V、ARM8/9）

IF -> ID -> EXE -> MEM -> WB

① **取指**：从指令存储器读取一条指令

② **译码**：分解指令、从寄存器堆取出操作数、产生控制信号

③ **执行**：根据控制信号完成相应的运算、计算出访存地址

④ **访存**：访问数据存储器

⑤ **写回**：将执行阶段的结果或访存取出的数据写回寄存器堆

流水线概述 – 流水线划分案例

ARM10六级流水

IF -> **ISS** -> ID -> EXE -> MEM -> WB

① **取指**：**分支预测**、从指令存储器读取一条指令

② **分派**：判断指令是否是协处理器指令

③ **译码**：分解指令、从寄存器堆取出操作数、产生控制信号

④ **执行**：根据控制信号完成相应的运算、计算出访存地址

⑤ **访存**：访问数据存储器

⑥ **写回**：将执行阶段的结果或访存取出的数据写回寄存器堆

流水线概述 – 流水线划分案例

ARM11八级流水

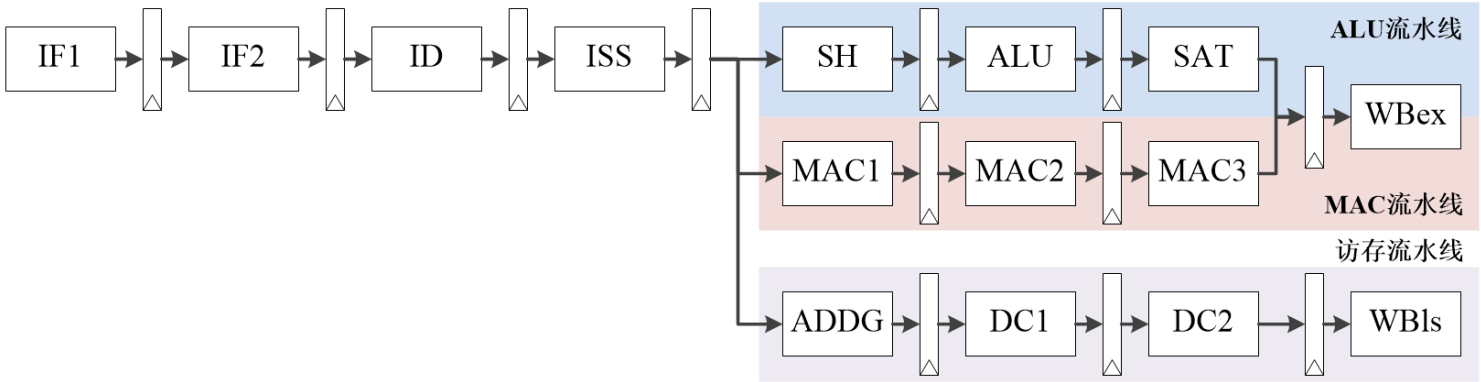
IF1 -> IF2 -> ID -> ISS -> ALU流水线/MAC流水线/访存流水线

① **取指1**：动态分支预测、从指令存储器读取一条指令

② **取指2**：静态分支预测

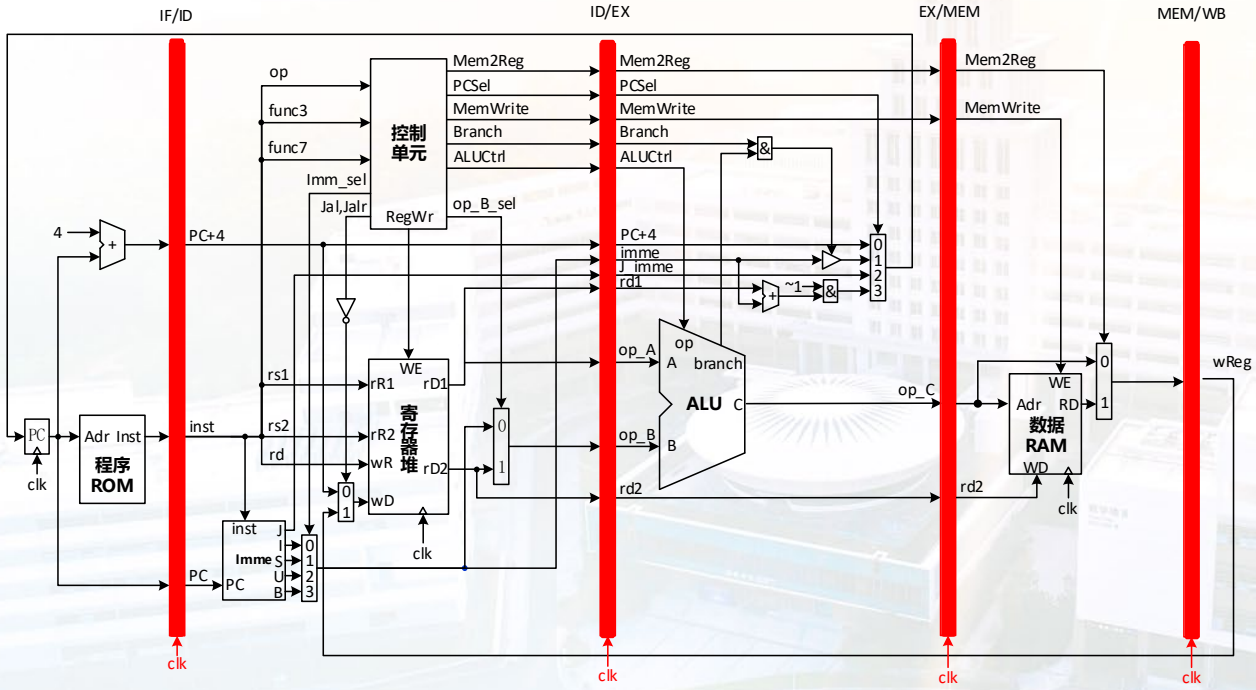
③ **译码**：分解指令、判断指令是否是协处理器指令

④ **分派**：从寄存器堆取出操作数，并分派指令到子流水线



流水线概述 – 流水线寄存器

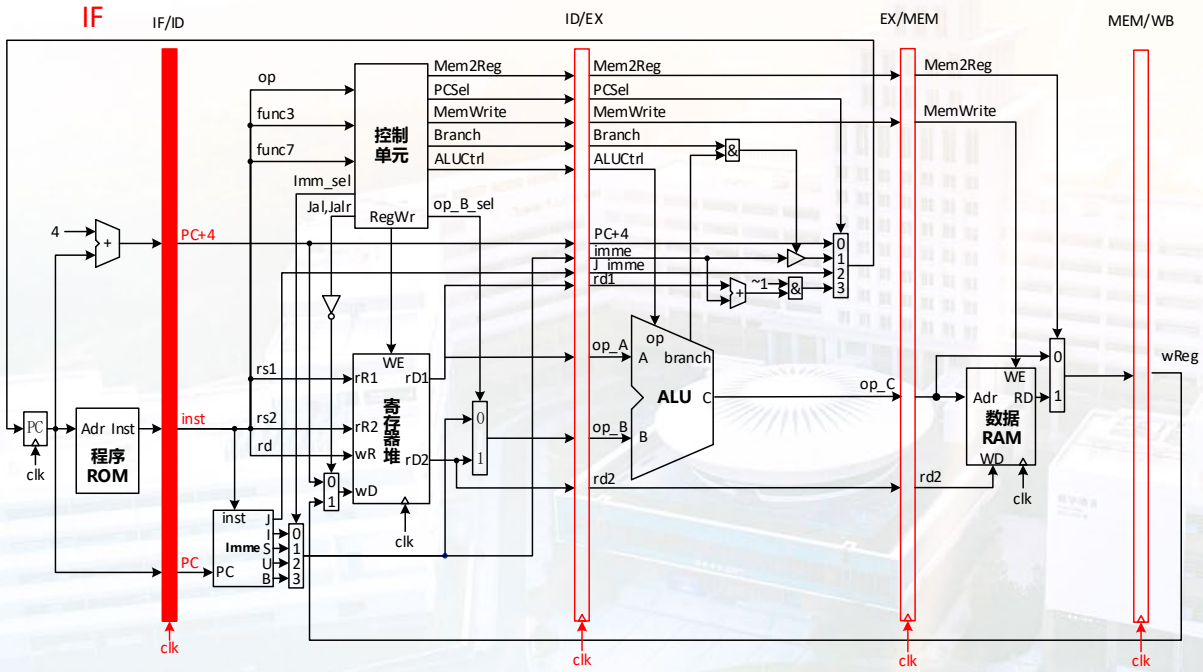
- **流水寄存器 / 段寄存器**



- 切割组合逻辑，提升系统频率
- 保存该流水级输出数据信息，交给下一级处理，并共享给其他指令

流水线概述 – 流水线寄存器

- 举例：**IF/ID流水寄存器**



- 段寄存器暂存什么信号，需根据具体设计而定
- 可以由多个“小”寄存器“拼”成完整的段寄存器

流水线概述 – 流水线寄存器

- **IF/ID流水寄存器**

PC：J型指令、B型指令

PC+4： J型指令

inst：所有指令

参考RTL实现

```
16 always @ (posedge clk or negedge rst_n) begin
17     if (~rst_n) id_pc <= 32'h0;
18     else        id_pc <= if_pc;
19 end
20
21 always @ (posedge clk or negedge rst_n) begin
22     if (~rst_n) id_inst <= 32'h0;
23     else        id_inst <= if_inst;
24 end
```

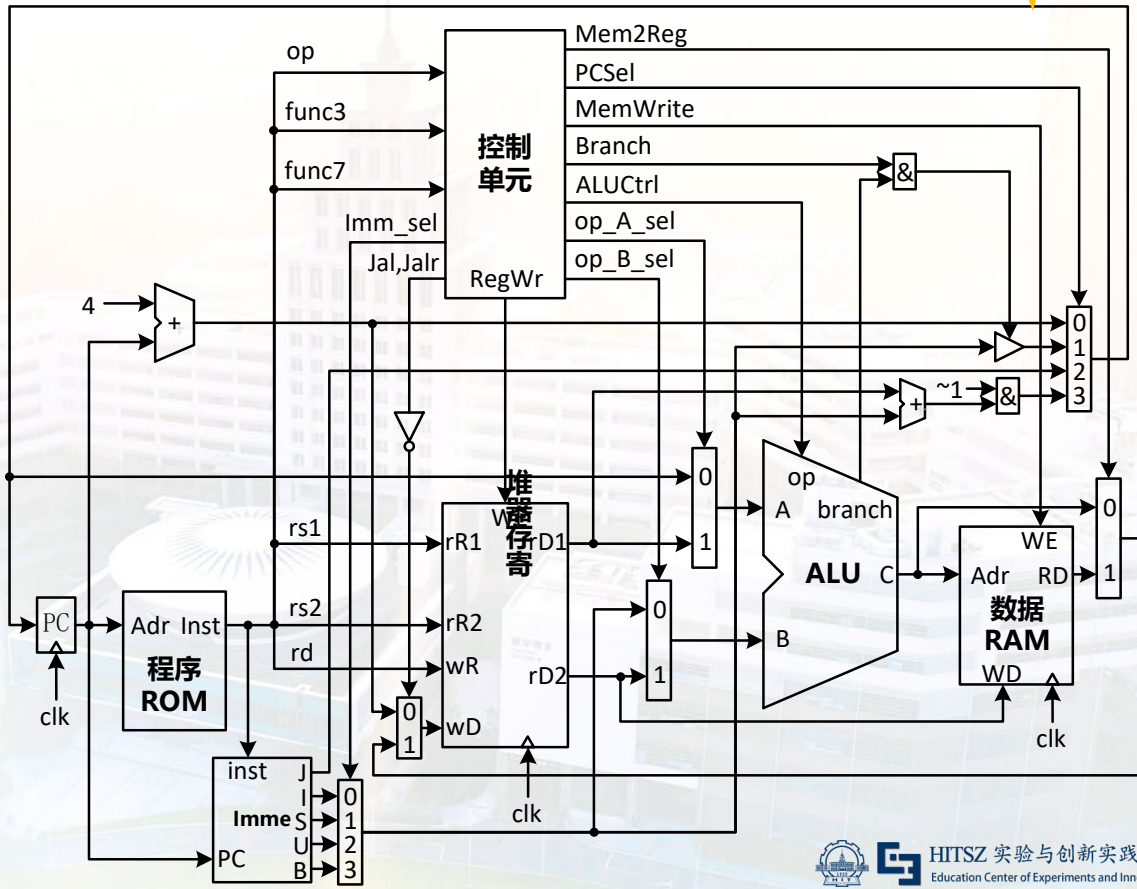
目录

流水线设计 – 单周期改造

单周期 → 流水线 How?

三级流水：

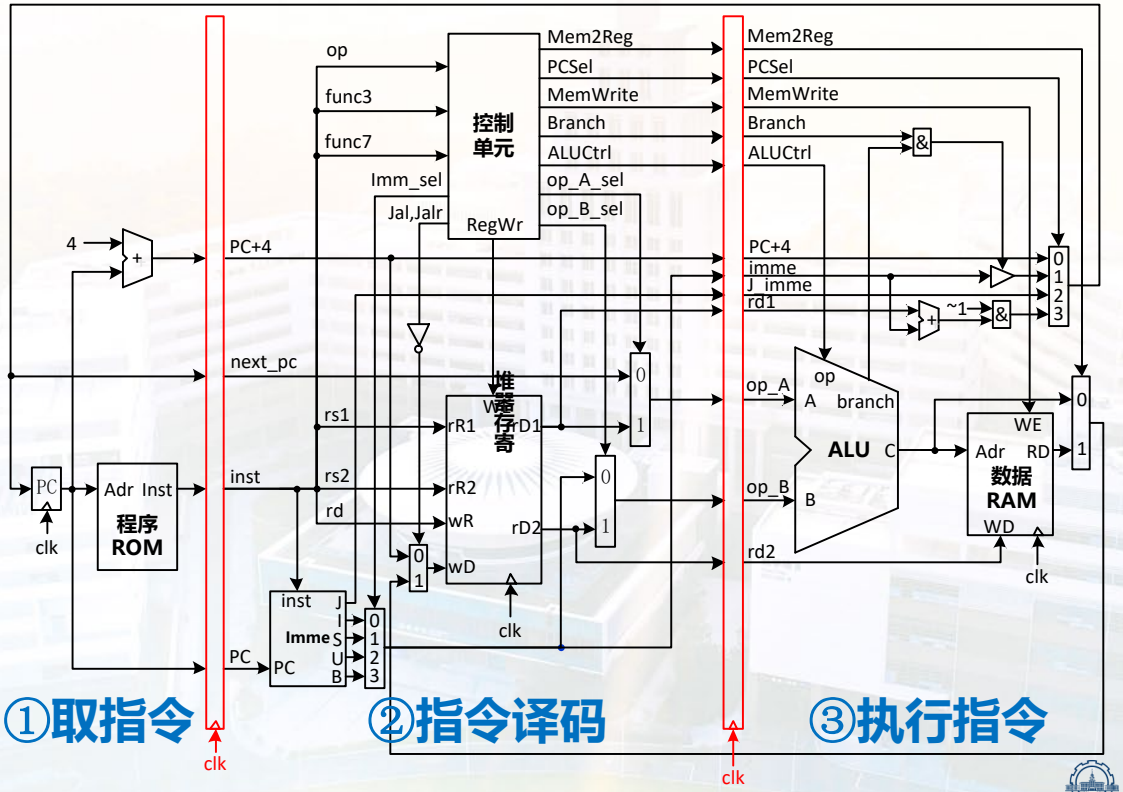
取指-译码-执行



流水线设计 – 单周期改造

单周期 → 理想流水线

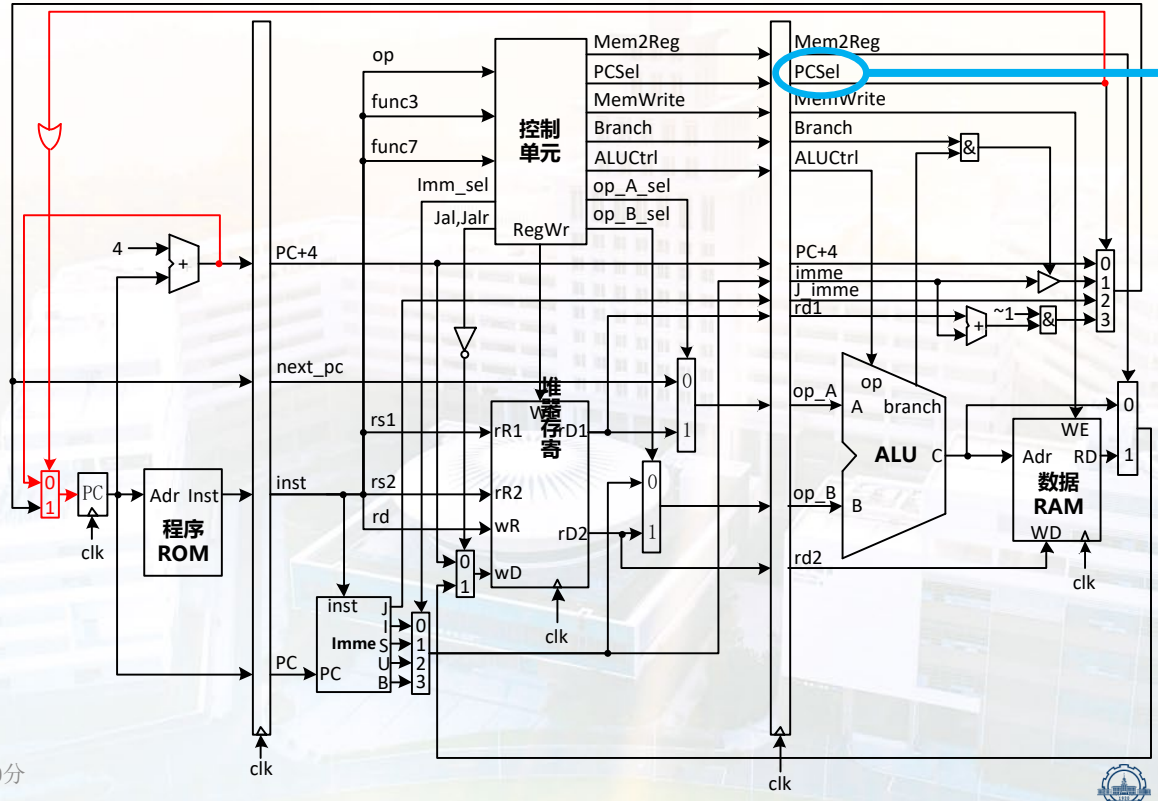
- **Step1：切分数据通路，添加段寄存器** (红色为新增)



流水线设计 – 单周期改造

单周期 → 理想流水线

- **Step2：按需修改写PC逻辑，让流水线“流动”起来** (红色为新增)

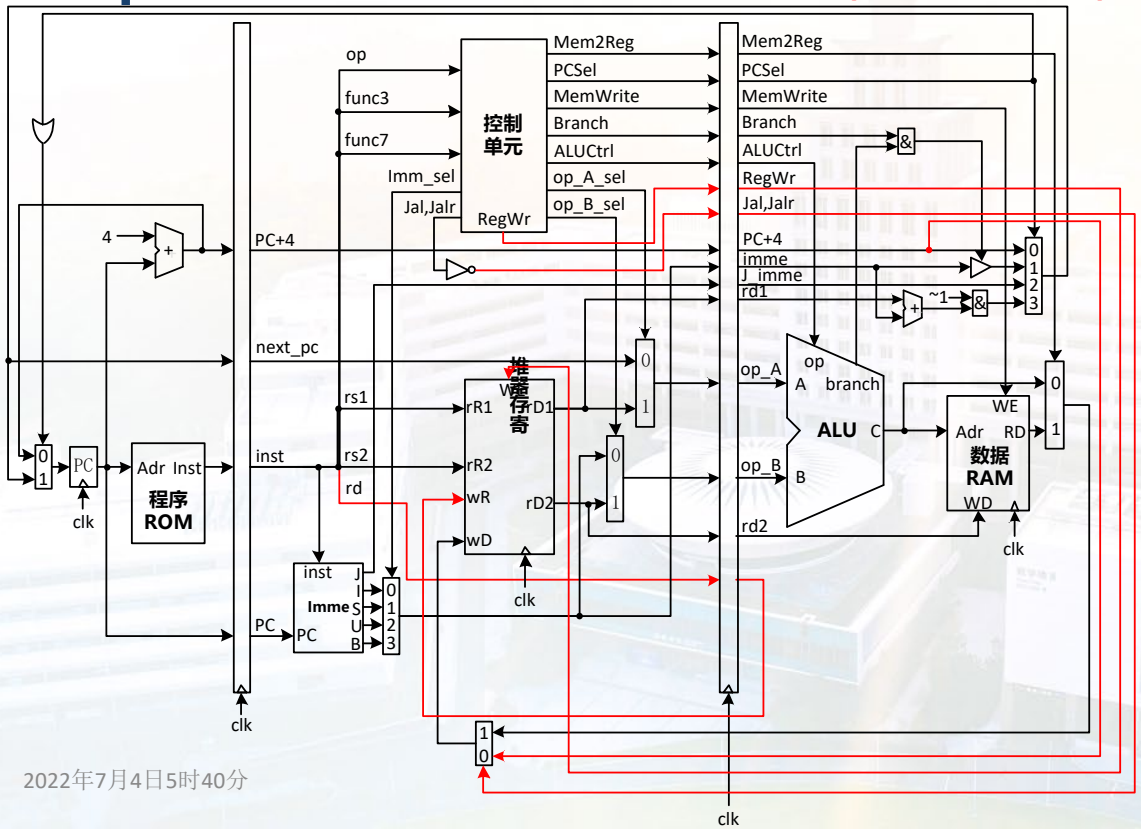


PCSel不为0
表示跳转

流水线设计 – 单周期改造

单周期 → 理想流水线

- **Step3：写回逻辑延后至最后阶段** (红色为新增)



- 段寄存器的必要性：存储指令的**执行状态**

执行状态：

指令执行时所有信号的取值构成的集合

- 延后写回逻辑，才能得到正确的执行结果

目录

流水线设计 – 冒险

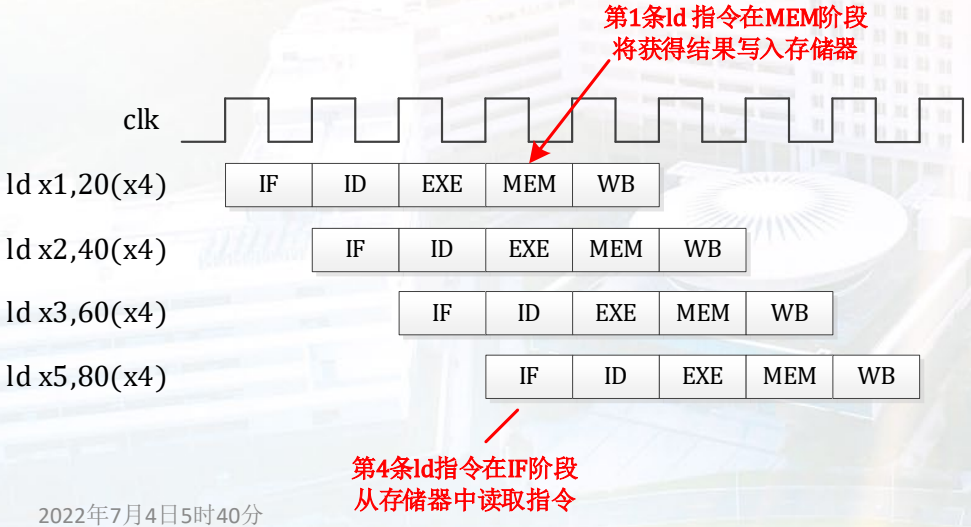
- **相关**（Dependence）：指令之间存在依赖关系
  - ① 数据相关 —— 当前指令使用前面指令的执行结果
  - ② 名相关
    - 多条指令使用相同的寄存器/存储单元，但不存在数据相关
    - 包括反相关（先读后写）、输出相关
  - ③ 控制相关 —— 分支指令
- **冒险** (Hazard)：指令相关引起流水线工作异常的现象

流水线设计 – 冒险

- **结构冒险**

  因缺乏硬件支持而导致指令不能在预定的时钟周期内执行的现象

  比如流水线中只有一个存储器，数据与指令共享



解决方法：

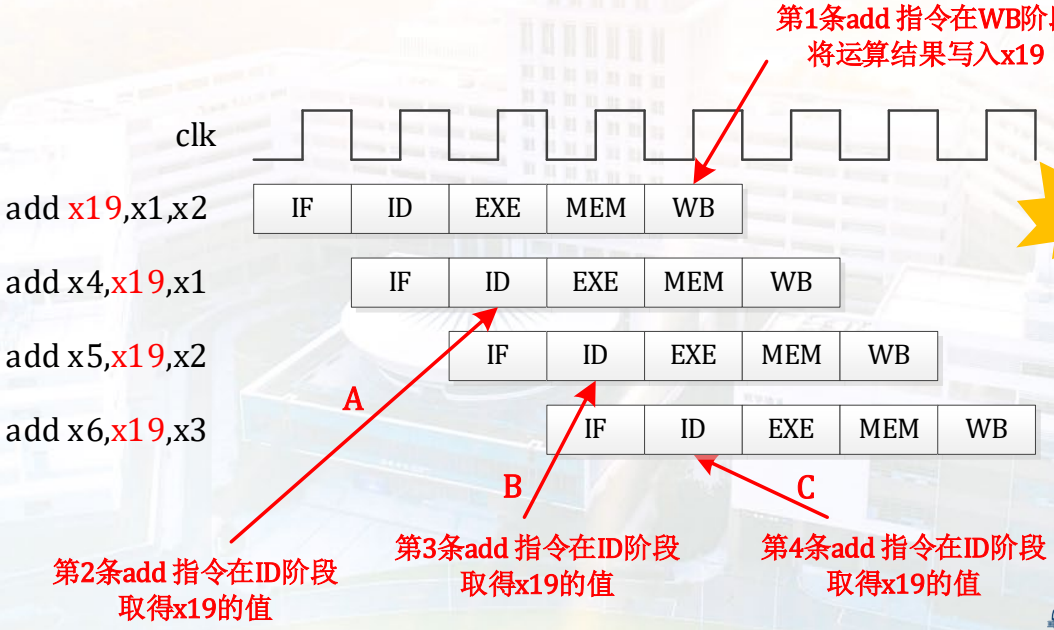
冯·诺依曼结构 -> 哈佛结构

流水线设计 – 冒险

- **数据冒险**

一条指令依赖于前面一条尚在流水线中的指令

因无法提供指令执行所需数据而导致指令不能在预期的时钟周期内执行的现象



思考

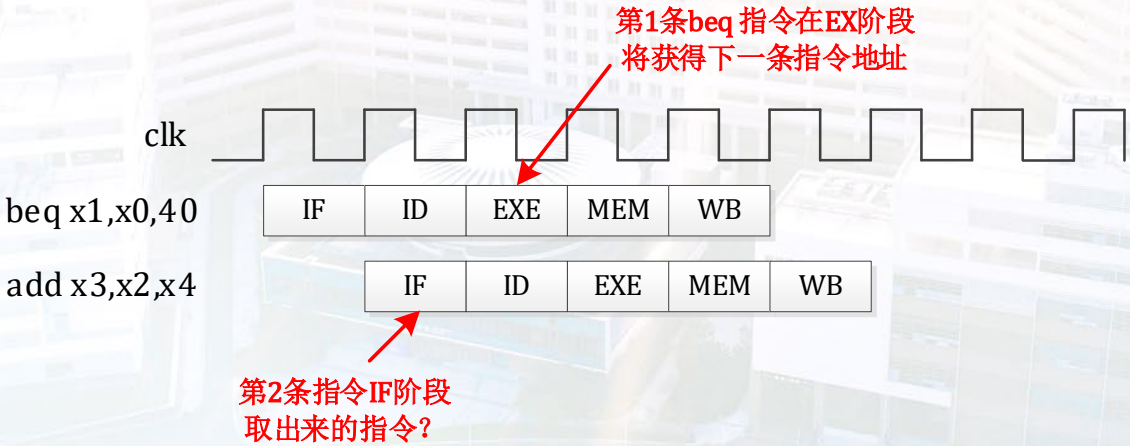
是否存在WAR和WAW？为什么？

流水线设计 – 冒险

- **控制冒险**

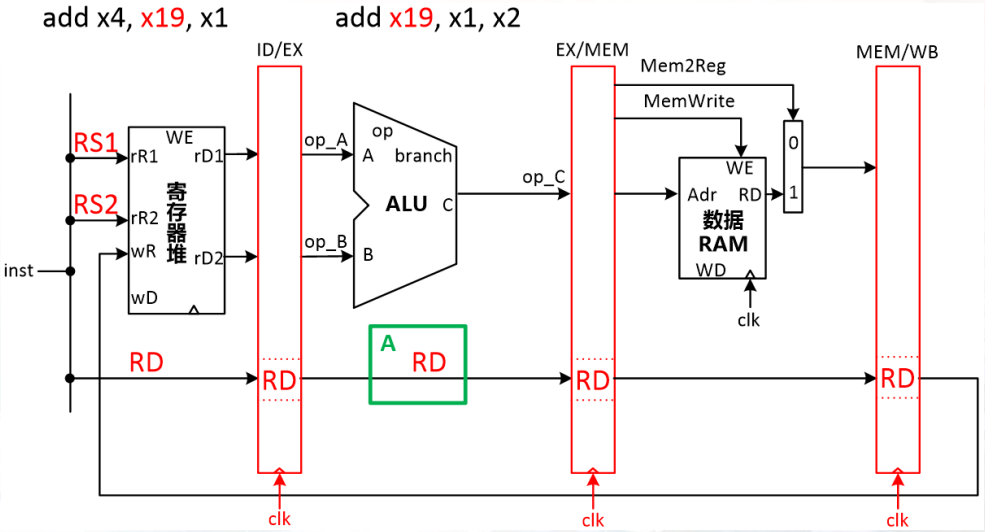
由于取到的指令并不是所需要的，或者指令地址的流向不是流水线所预期的，导致正确的指令无法在正确的时钟周期内执行的情况

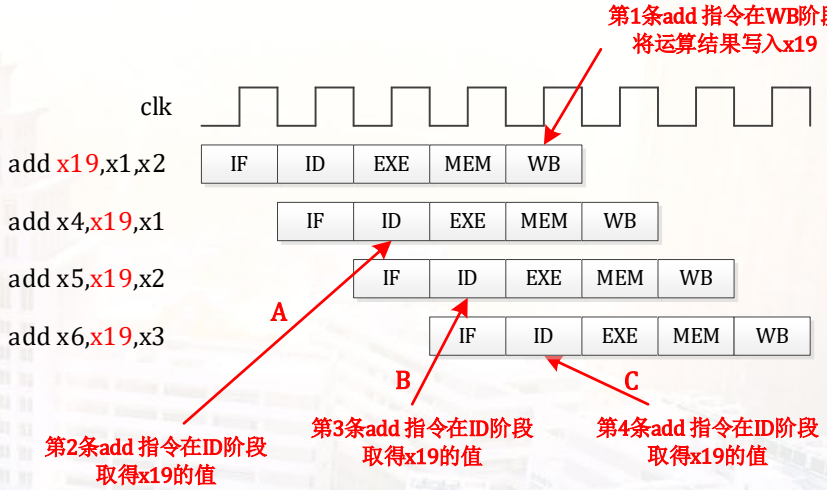
分支指令后的IF段依赖于分支的结果，无法保证永远取到正确的指令



流水线设计 – 数据冒险检测

- **RAW冒险检测 – 情形A**





判断条件：

$$REG_{ID/EX}.RD == ID.RS1$$

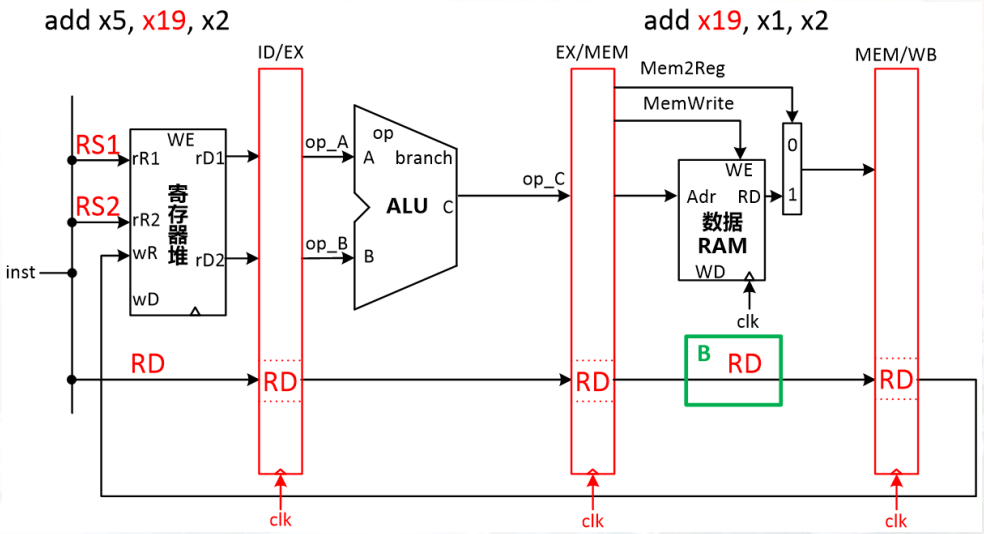
或 $$REG_{ID/EX}.RD == ID.RS2$$

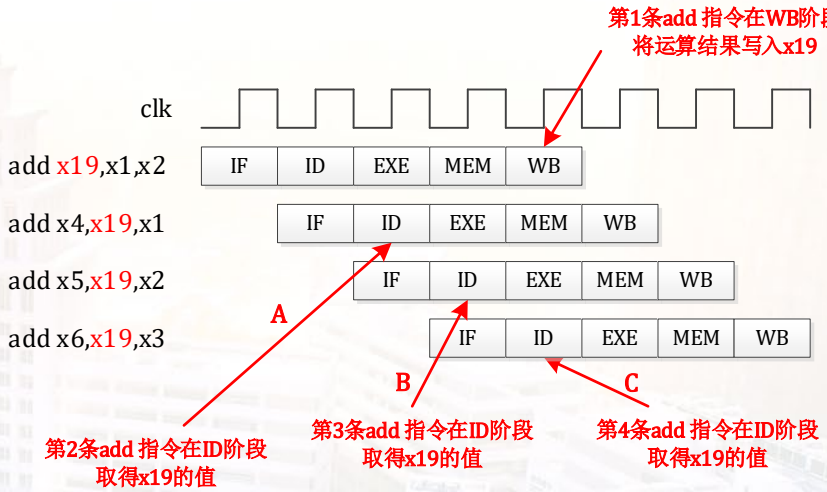
注意：$REG_{ID/EX}.RD$表示ID/EX寄存器的RD字段

ID.RSx表示ID阶段RSx寄存器的寄存器号

流水线设计 – 数据冒险检测

- **RAW冒险检测 – 情形B**





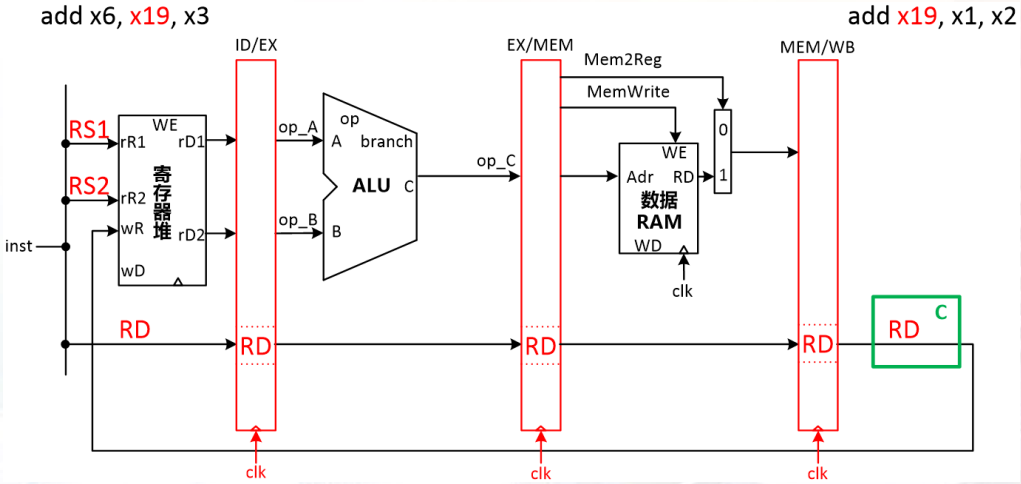
判断条件：

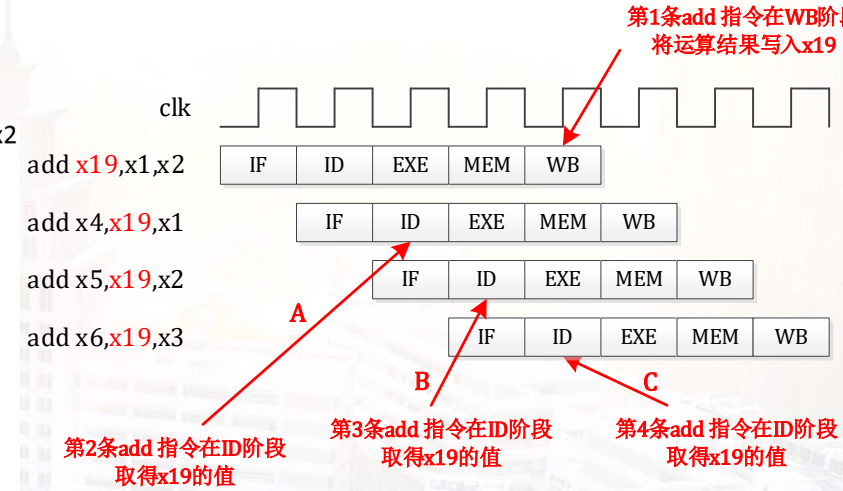
$$REG_{EX/MEM}.RD == ID.RS1$$

或 $$REG_{EX/MEM}.RD == ID.RS2$$

流水线设计 – 数据冒险检测

- **RAW冒险检测 – 情形C**





判断条件：

$$REG_{MEM/WB}.RD == ID.RS1$$

或 $REG_{MEM/WB}.RD == ID.RS2$

流水线设计 – 数据冒险检测

- **RAW冒险检测 – 情形C**

判断条件：

$$REG_{MEM/WB}.RD == ID.RS1$$

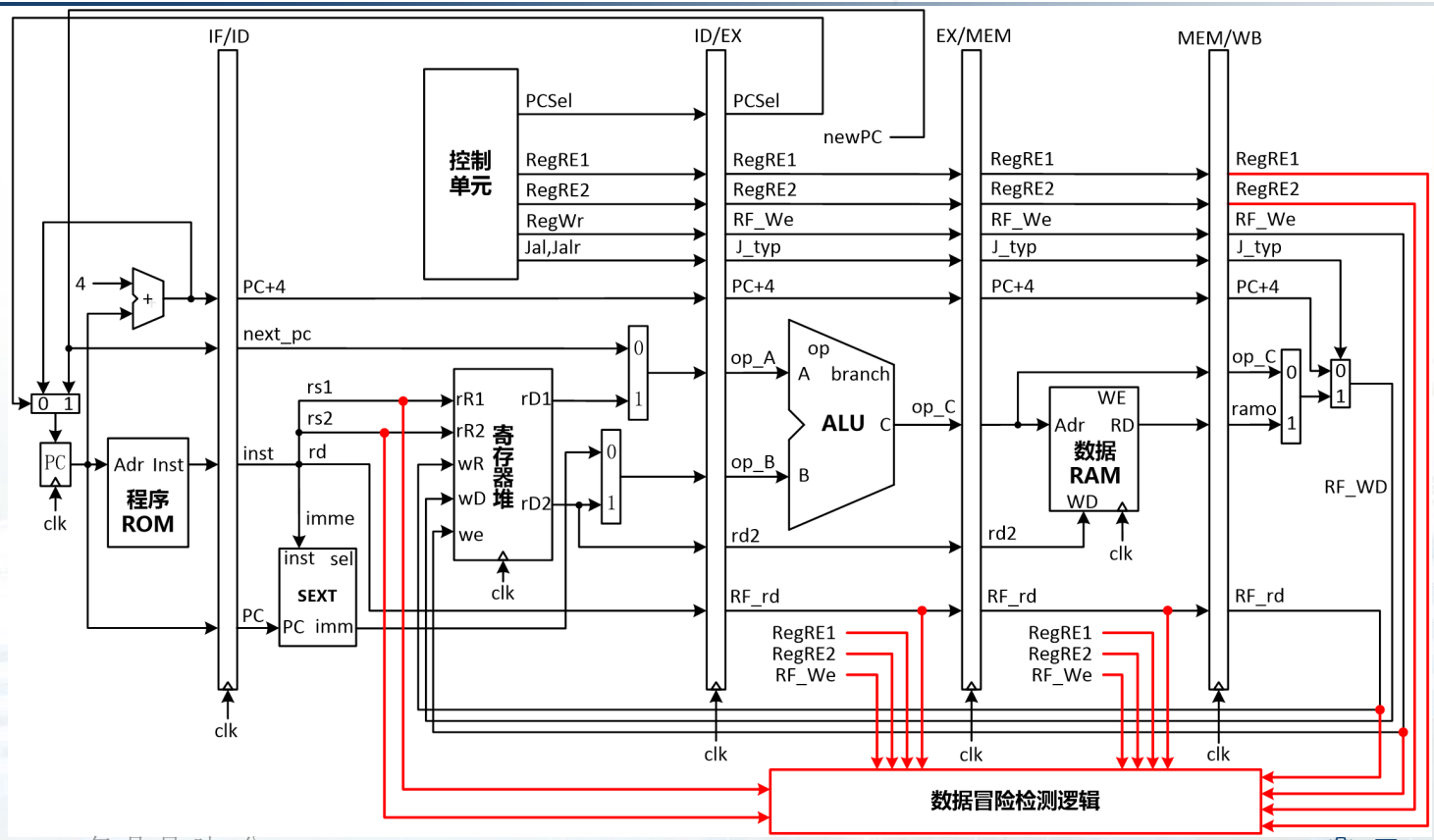
或 $$REG_{MEM/WB}.RD == ID.RS2$$

RTL参考实现

```
wire rs1_id_wb_hazard = (wb_rd == id_rs1) & wb_we & id_rf1;
wire rs2_id_wb_hazard = (wb_rd == id_rs2) & wb_we & id_rf2;
```

- U型、J型指令**没有源操作数寄存器**，不需读取RF，故不存在RAW
- 单周期的RF采用**异步读取**，可能造成**误判**
- 为了保证冒险检测的正确性，需添加“读”标志信号加以区分

流水线设计 – 数据冒险检测逻辑



仅为**示例**，请在理解的基础上完成自己的设计

流水线设计 – 数据冒险解决

- **数据冒险解决方法**

1）停顿/暂停

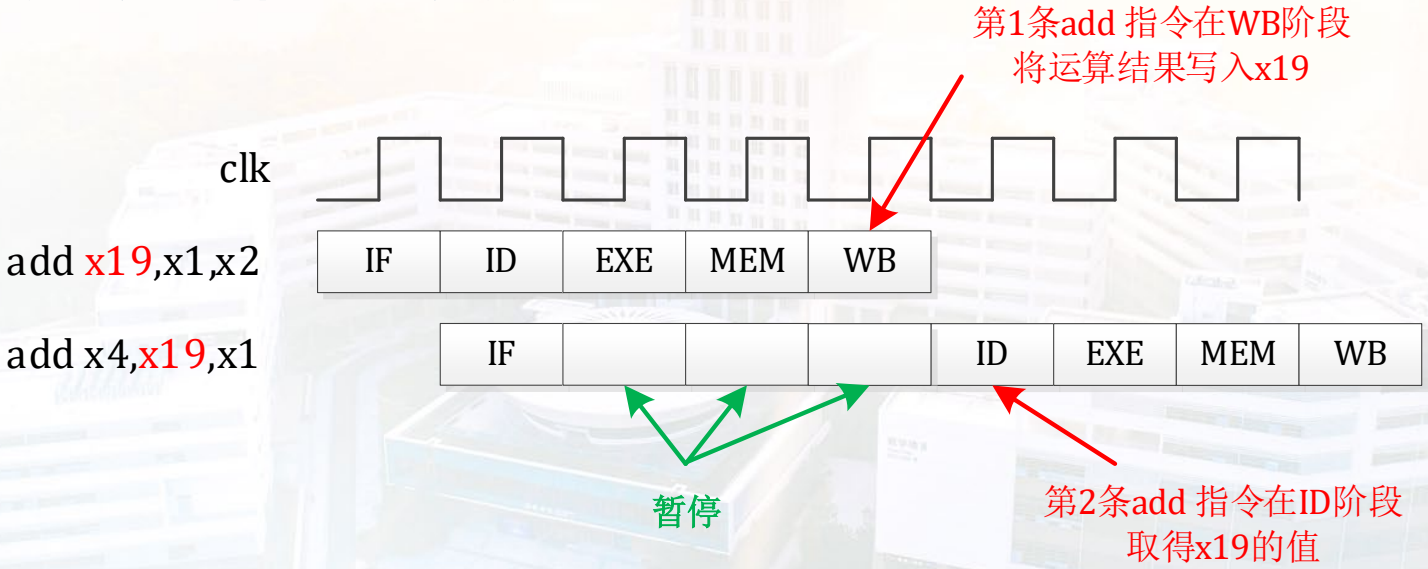
2）数据前递/前推/旁路/转发

3）乱序执行

流水线设计 – 暂停法

- **解决数据冒险**

解决方法1：插入暂停周期



流水线设计 – 暂停法

- **解决数据冒险**

  解决方法1：插入暂停周期

  实现方法：

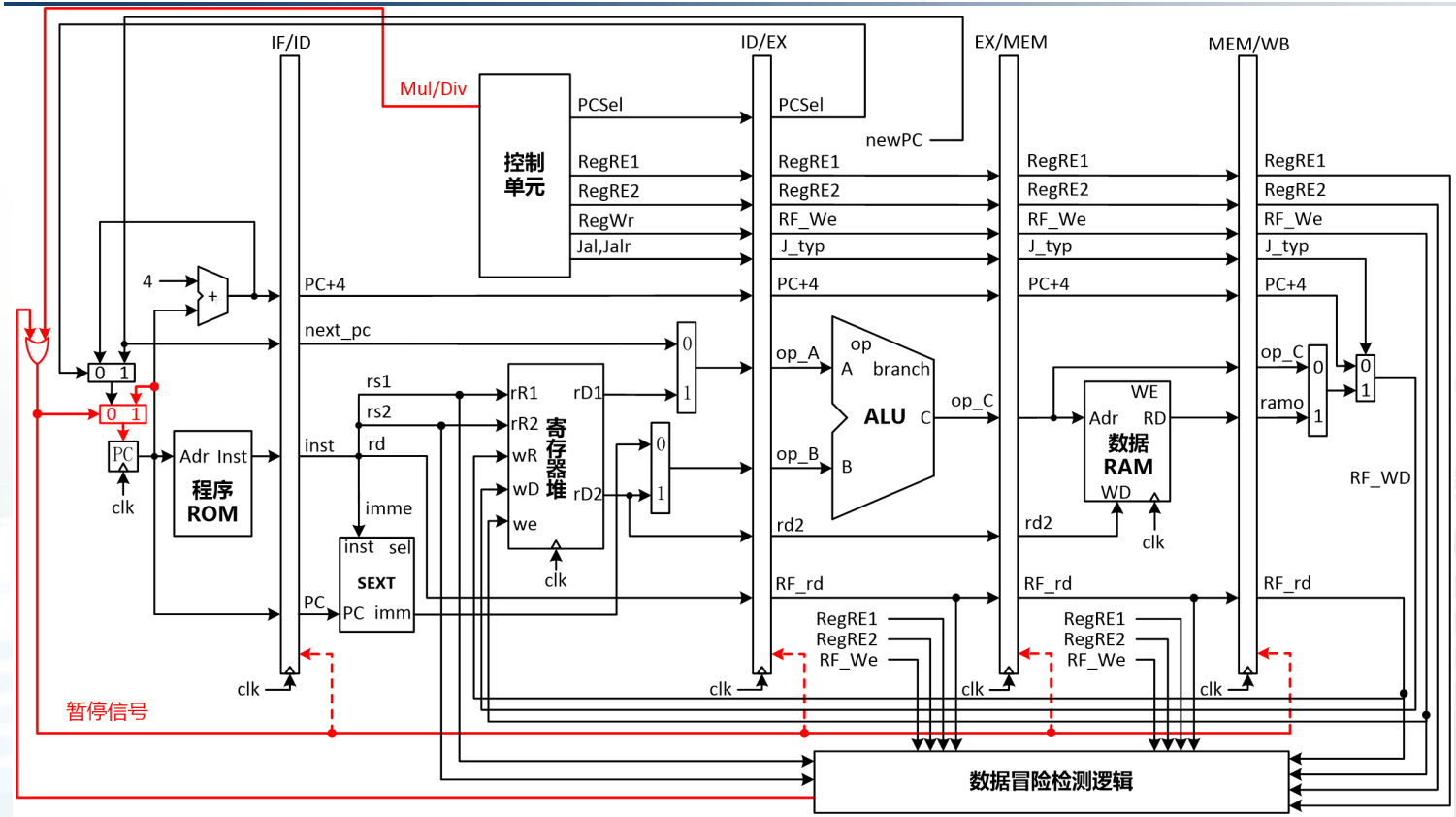
  保持PC的值不变

  保持流水线的各个段寄存器不变

  - IF/ID、ID/EX、EX/MEM、MEM/WB

  暂停寄存器：将寄存器的输入修改为寄存器自身的输出

流水线设计 – 暂停法设计

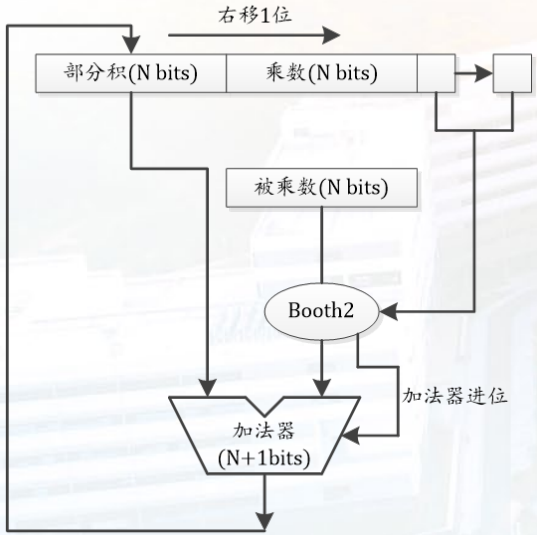


仅为**示例**，请在理解的基础上完成自己的设计

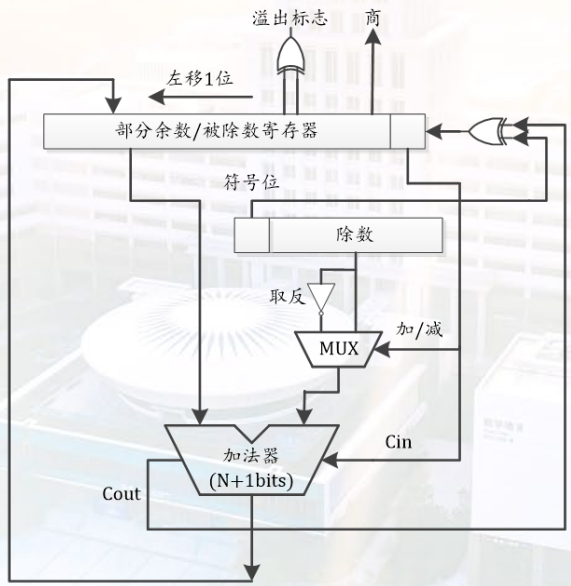
流水线设计 – 暂停法应用场景：MUL/DIV

- **特殊指令**：**乘/除**法指令

2位Booth编码算法乘法器



加减交替法除法器



操作	所需时钟数
除法	32
乘法	17
其余	1

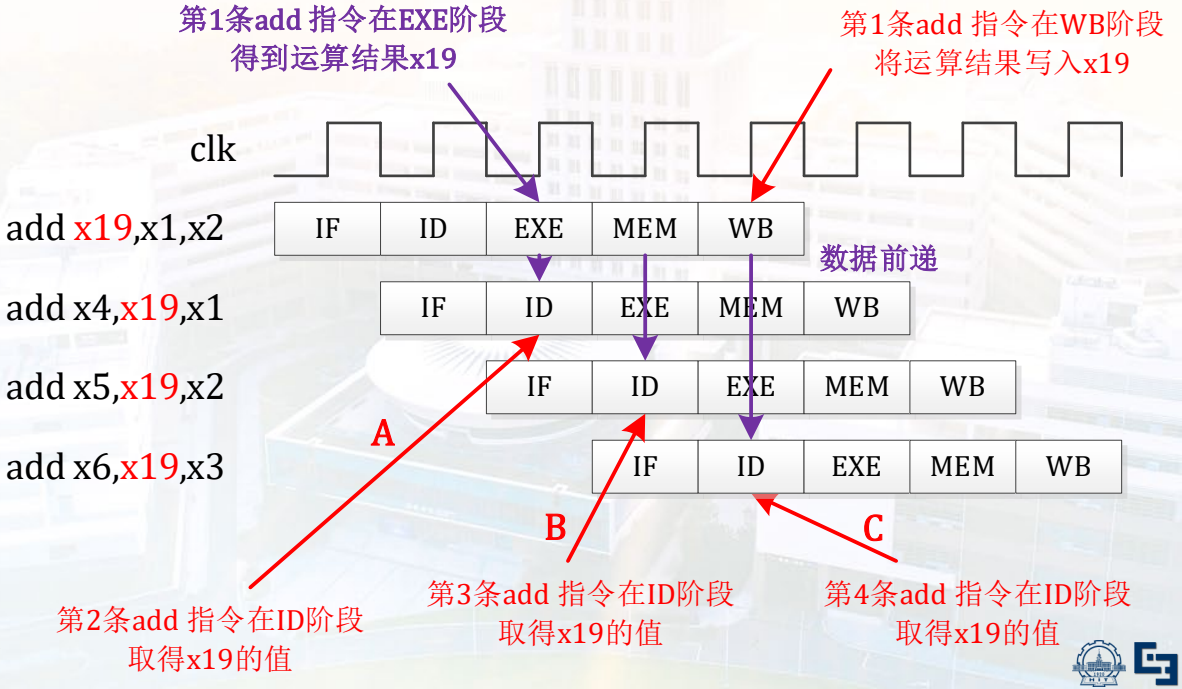
思考

如果调用IP实现？

流水线设计 – 旁路法

- **解决数据冒险**

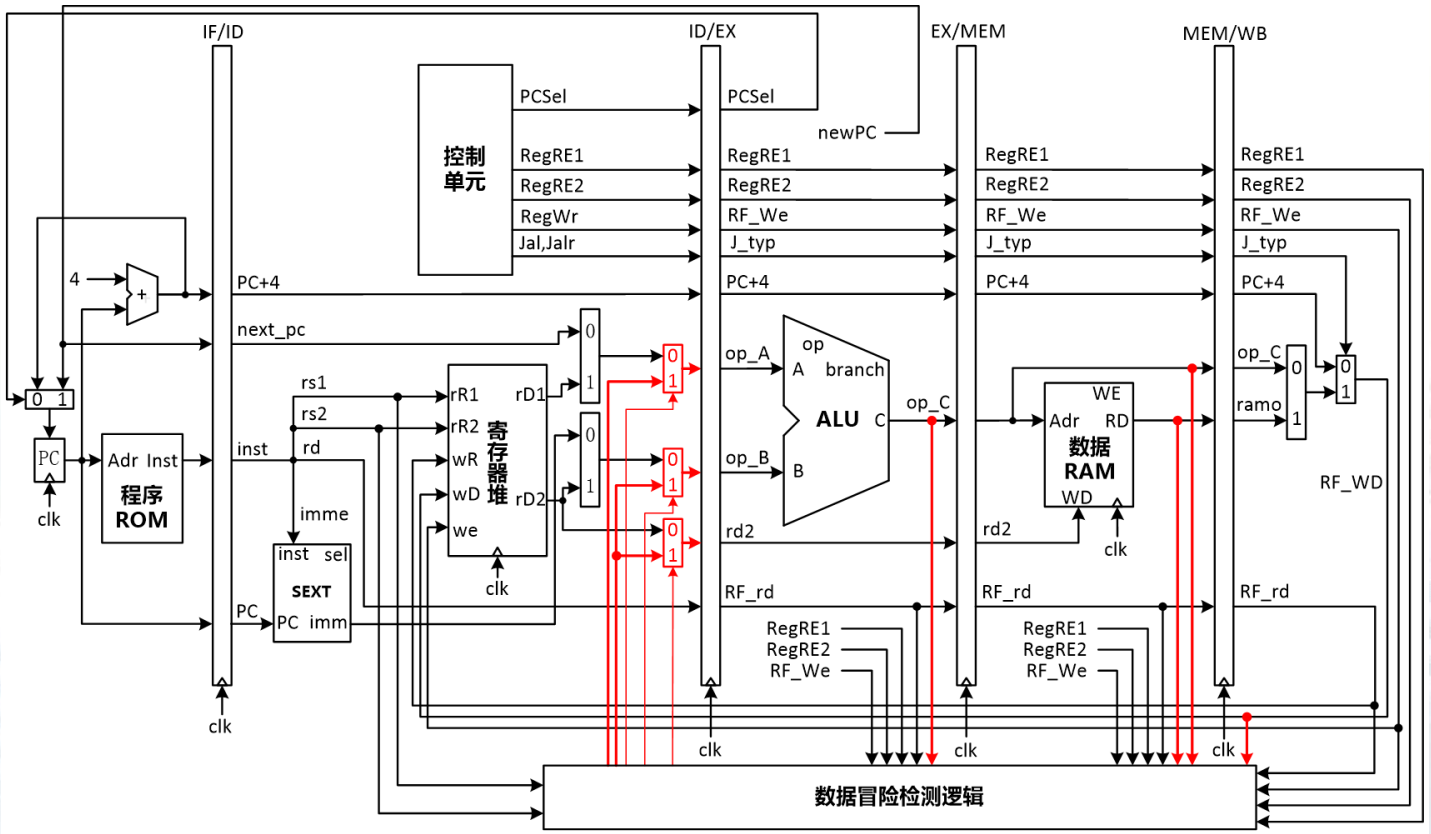
解决方法2：数据前推



流水线设计 – 旁路法

- **数据前推的逻辑**
  - 将EXE、MEM、WB阶段的结果前推到ID阶段，参与ID阶段运算源操作数的选择
  - **Pros**：重叠执行程度高、发生冒险时性能损失小
  - **Cons**：逻辑复杂，影响频率的提升

流水线设计 – 旁路法数据通路



仅为**示例**，请在理解的基础上完成自己的设计

流水线设计 – 乱序法

- **解决数据冒险**

  解决方法3：乱序执行法

存在数据冒险

```
add x2, x1, x6
add x3, x2, x7
add x4, x1, x8
add x5, x9, x7
add x6, x7, x1
```

无数据冒险

```
add x2, x1, x6
add x4, x1, x8
add x5, x9, x7
add x6, x7, x1
add x3, x2, x7
```

无相关指令

流水线设计 – 乱序法

- **解决数据冒险**

解决方法3：乱序执行法

实现方法：

① 软件调度 (汇编器、编译器)

② 硬件前瞻执行 (分支预测、Tomasulo算法)

参考资料：《计算机系统结构教程(第2版)》张晨曦

目录

流水线设计 – 控制冒险

- **控制冒险**
  - 分支指令执行结果出来之前，无法确定接下来要取哪条指令
  - 解决思路：
    ① 暂停法　　简单，但损失性能
    ② 延迟槽　　降低代码可读性、性能提升有限　RISC-V已舍弃不用
    ③ 分支预测

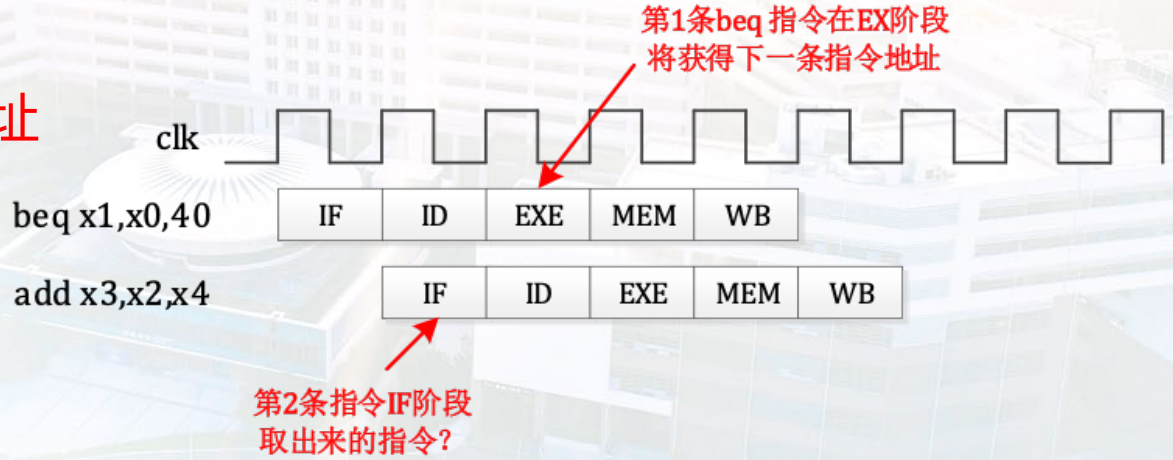
流水线设计 – 控制冒险

- **解决控制冒险**

- 移动硬件，使得分支决定提前到ID阶段
- 需要两个操作提早发生

  1）预测方向

  2）计算分支目标地址



流水线设计-冒险

- **解决控制冒险**

1）静态预测方法

- 总是预测跳转 or 总是预测不跳转
- BTFN预测（Back Taken, Forward Not Taken ）

Pros：易实现、开销小

Cons：准确度较低

流水线设计-冒险

2）动态预测方法

a. 基于BHT的分支预测

采用分支历史表记录分支历史，并以此预测分支行为

BHT：

Tag	分支历史(2bit)
A_0	H_0
A_1	H_1
…	…
A_{k-1}	H_{k-1}

Tag —— 分支指令地址的一部分，类似于Cache的Tag

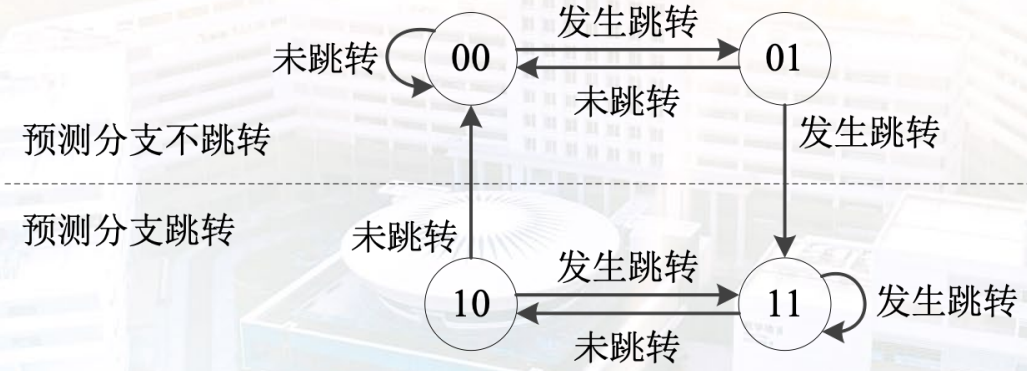
分支历史 —— 2bit饱和计数器

流水线设计-冒险

2）动态预测方法

a. 基于BHT的分支预测

采用分支历史表记录分支历史，并以此预测分支行为



先用指令地址查BHT，再根据分支历史预测是否跳转

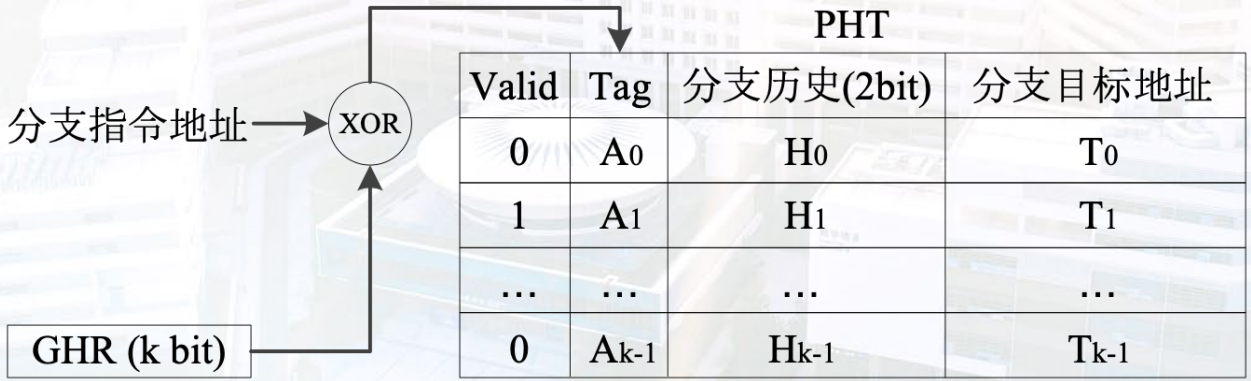
流水线设计-冒险

2）动态预测方法

b. 基于全局历史的分支预测

BHT方法忽视了分支指令之间的关联性

使用GHR关联所有分支指令，使用PHT记录分支历史行为



Valid	Tag	分支历史(2bit)	分支目标地址
0	A_0	H_0	T_0
1	A_1	H_1	T_1
…	…	…	…
0	A_{k-1}	H_{k-1}	T_{k-1}

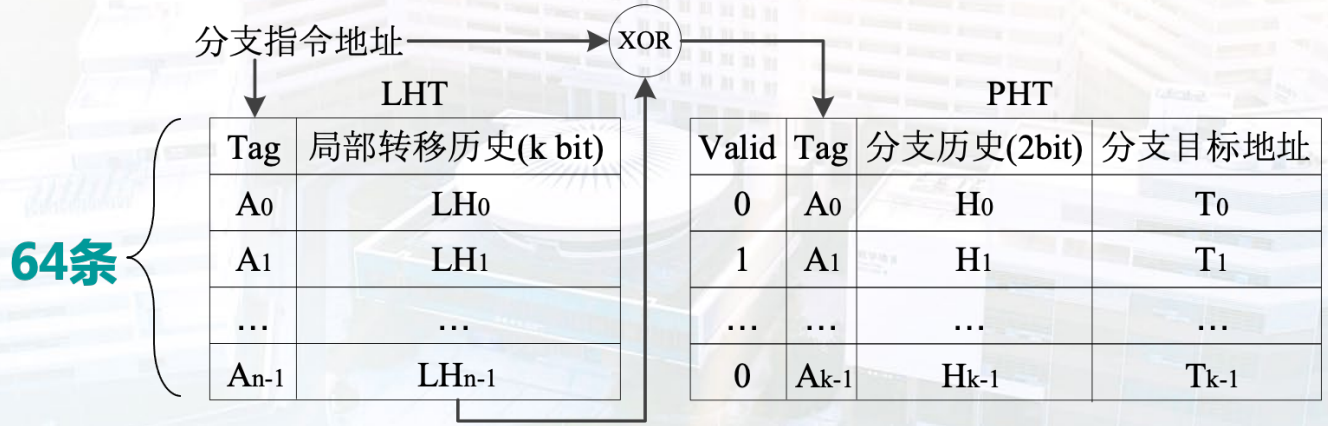
流水线设计-冒险

2）动态预测方法

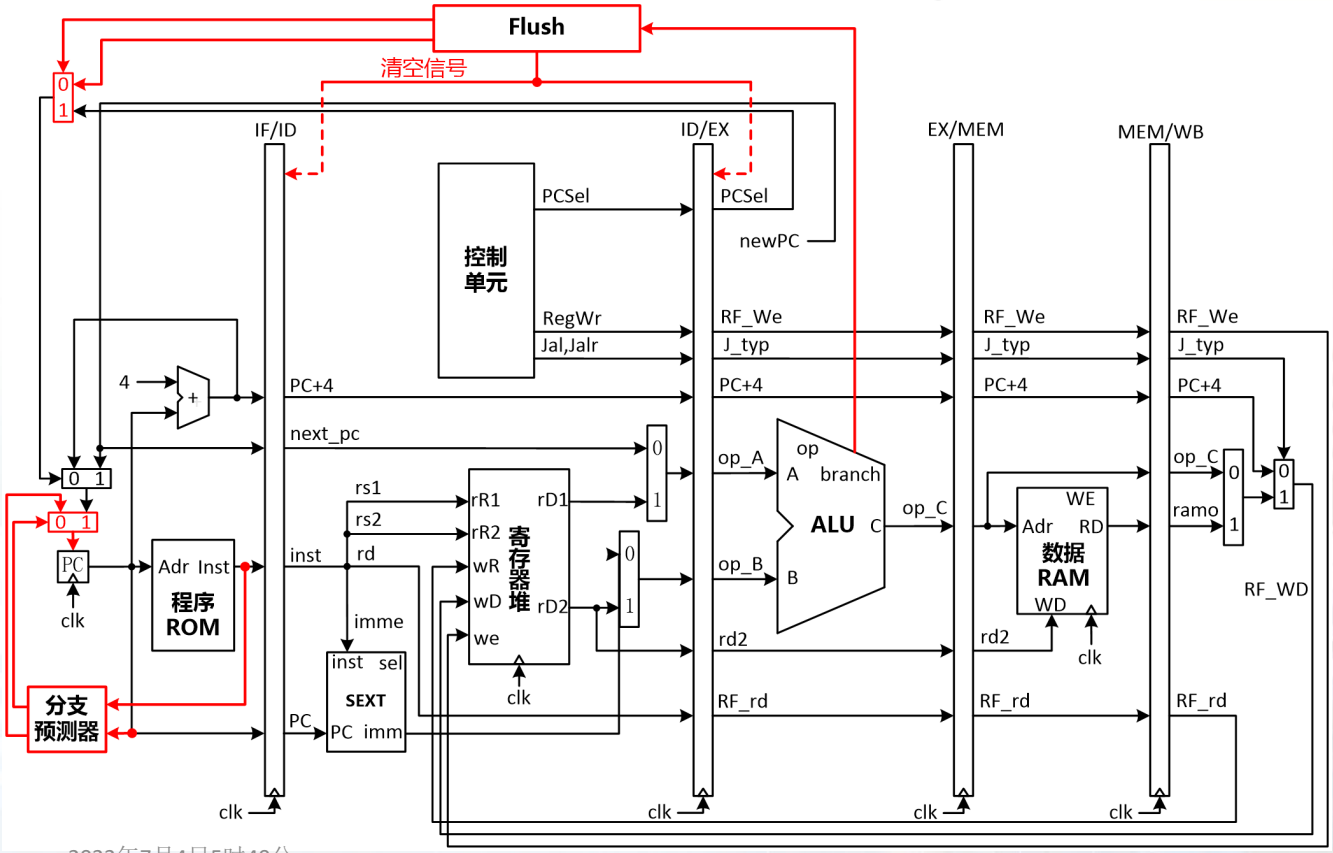
c. 基于局部历史的分支预测

全局方法仅用一个GHR关联所有分支指令，关联范围太广

使用LHT关联局部的分支指令，使用PHT记录分支历史行为



流水线设计-冒险



预测失败时，需排空IF、ID

仅为示例，请在理解的基础上完成自己的设计

谢 谢！

HITSZ 实验与创新实践教育中心

Education Center of Experiments and Innovations, HITSZ