第五章 流水线处理器

- **流水线概述**
- 流水线数据通路和控制
- 数据冒险：前递与停顿
- 控制冒险
- 例外

单周期处理器的性能问题

指令时延

Instr	IF=200ps	ID=100ps	ALU=200ps	MEM=200ps	WB=100ps	Total
add	√	√	√		√	600ps
beq	√	√	√			500ps
jal	√	√	√		√	600ps
load	√	√	√	√	√	800ps
store	√	√	√	√		700ps

- 为了提高单周期处理器性能，**可通过流水线来提高性能**

单周期处理器的性能问题

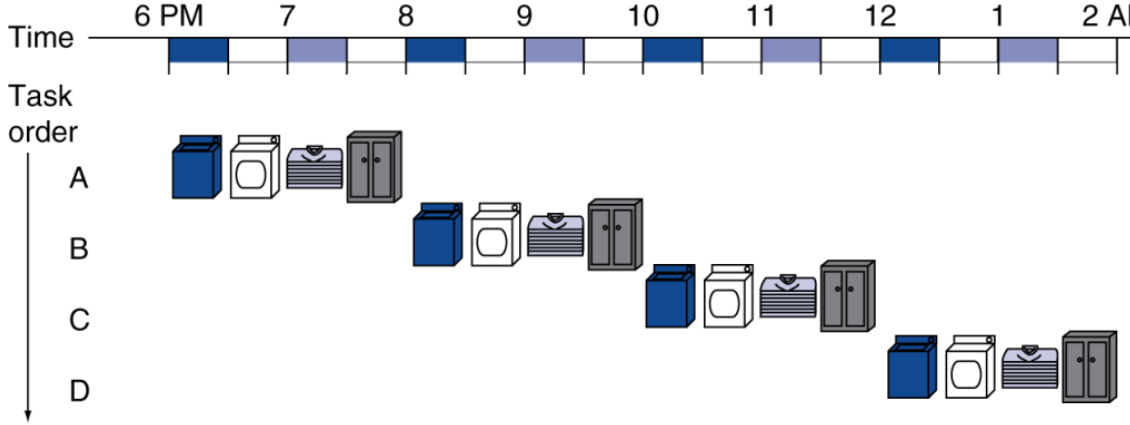
- 单周期设计中，时钟周期对于每条指令必须相等
- 因此，处理器中的关键路径决定了时钟周期
  - 这条路径很可能是一条load指令
  - 经历了完整的五个步骤：访问指令存储器、访问寄存器堆、ALU运算、访问数据存储器、写回寄存器堆

Instr	IF=200ps	ID=100ps	ALU=200ps	MEM=200ps	WB=100ps	Total
lw	X	X	X	X	X	800ps

- 违反了RISC-V指令集设计思想
  - 无法加速经常性事件：实际需要的执行时间比较短的指令无法加速
- 为了提高单周期处理器性能，**可通过流水线来提高性能**

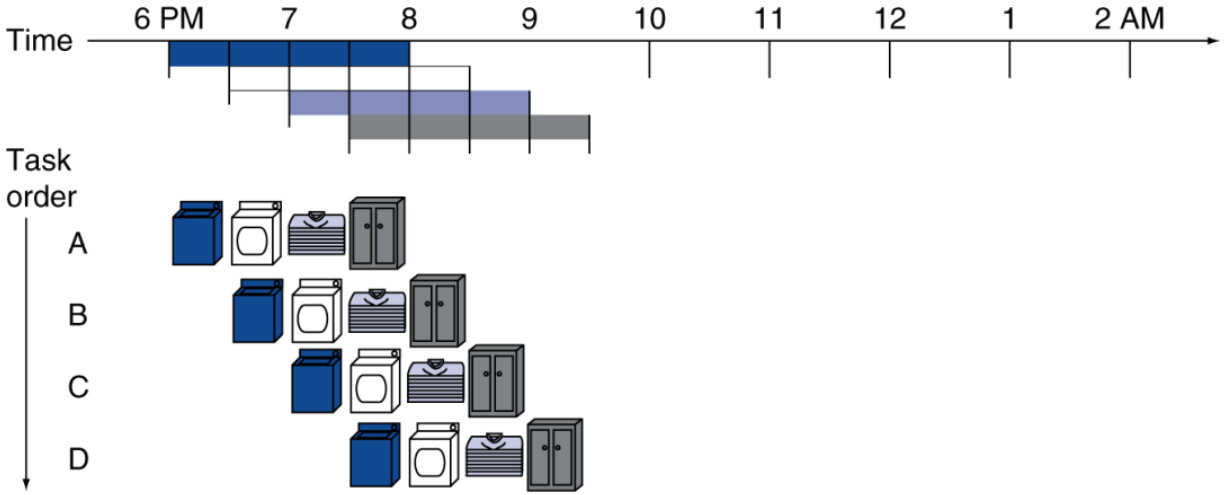
生活中的流水线

- 假设洗衣包括四个步骤：洗衣机中洗衣、烘干机中烘干、叠衣服、收纳到柜子中，每个步骤0.5小时。



- 洗衣任务为4，**加速比**

  = 2*4/(2+0.5*3) ≈ 2.3



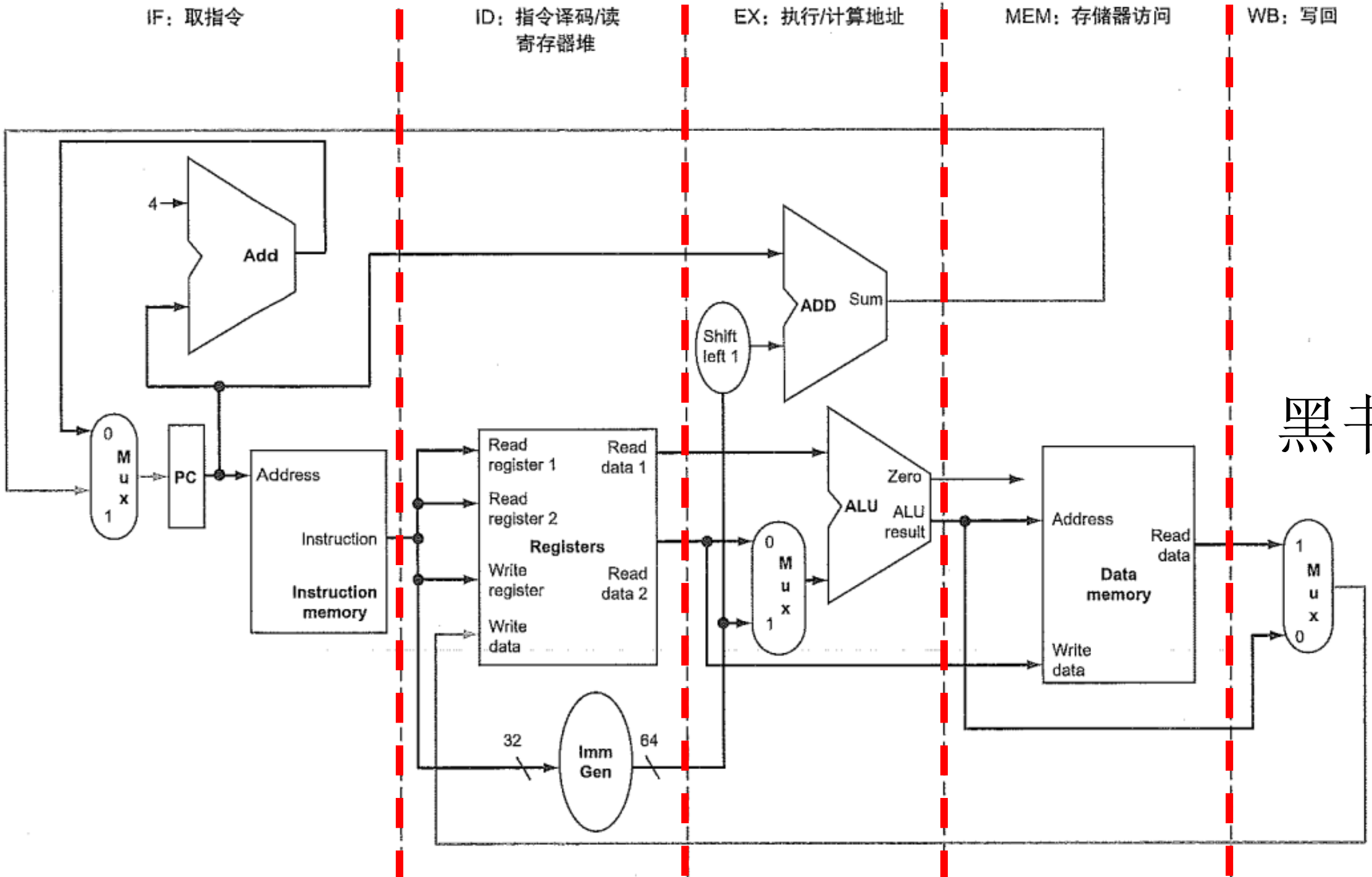
- 洗衣任务数为n，加速比

  = 2n/(2+0.5*(n-1))

  ≈ **4** = 流水线中的步骤数

RISC-V 指令执行的五个阶段

- IF (Instruction Fetch): 从指令存储器中取出指令
- ID (Instruction Decode): 读寄存器堆并译码指令
- EX (EXecute): ALU执行操作或计算地址
- MEM (MEMory access): 访问数据存储器（如有必要）
- WB (Write Back register): 将结果写入寄存器（如有必要）
- 让五个阶段重叠执行**可提高性能**


IF：取指令
ID：指令译码/读寄存器堆
EX：执行/计算地址
MEM：存储器访问
WB：写回
4
Add
ADD
Sum
Shift left 1
0
M
u
x
1
PC
Address
Instruction
Instruction memory
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
ALU
Zero
ALU result
Address
Read data
Data memory
Write data
32
Imm Gen
64

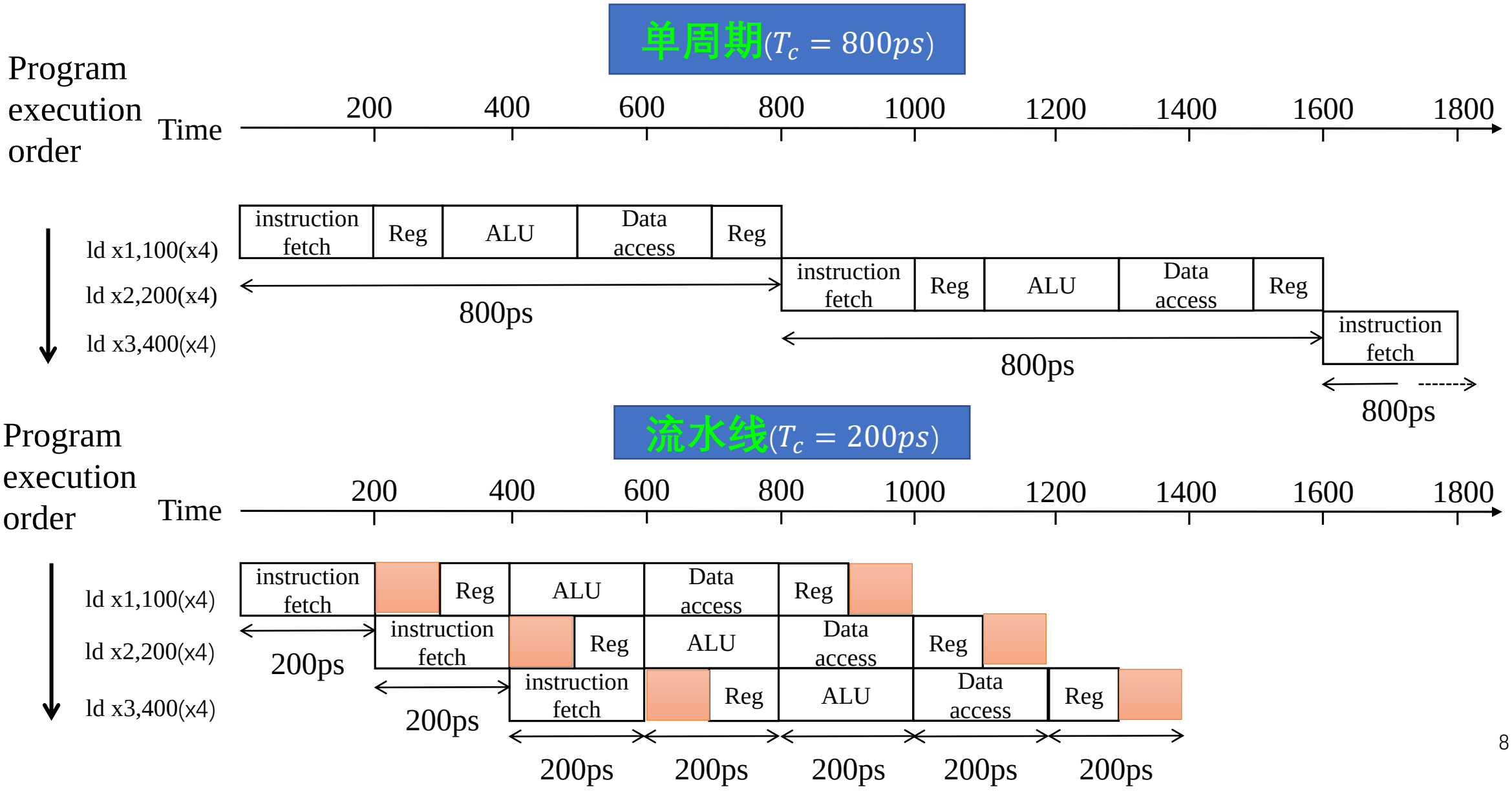
黑书P194

流水线性能分析

- 假设各个阶段的耗时：
  - 寄存器堆的读或写为100ps
  - 其他阶段为200ps
- 比较流水线指令执行与单周期指令执行的平均执行时间

指令类型	取指令	读寄存器	ALU 操作	数据存取	写寄存器	总时间
Load类指令	200ps	100 ps	200ps	200ps	100 ps	800ps
Store类指令	200ps	100 ps	200ps	200ps		700ps
R型指令	200ps	100 ps	200ps		100 ps	600ps
beq	200ps	100 ps	200ps			500ps

流水线性能


单周期($T_c = 800ps$)
Program execution order
Time
200
400
600
800
1000
1200
1400
1600
1800
ld x1,100(x4)
ld x2,200(x4)
ld x3,400(x4)
instruction fetch
Reg
ALU
Data access
Reg
800ps
800ps
800ps
流水线($T_c = 200ps$)
Program execution order
Time
200
400
600
800
1000
1200
1400
1600
1800
ld x1,100(x4)
ld x2,200(x4)
ld x3,400(x4)
instruction fetch
Reg
ALU
Data access
Reg
200ps
200ps
200ps
200ps
200ps
200ps
200ps

流水线加速比

- 如果流水线各阶段操作平衡
  - 例如：所有阶段需要相同的时间

则：$\text{指令执行时间}_{\text{流水线}} \approx \dfrac{\text{指令执行时间非流水线}}{\text{流水线级数}}$

- 若各阶段不完全平衡，加速比会变小

- 通过提高指令吞吐率来提高性能

注意：单个指令的执行时间没有减少

面向流水线的指令系统设计（RISC-V）

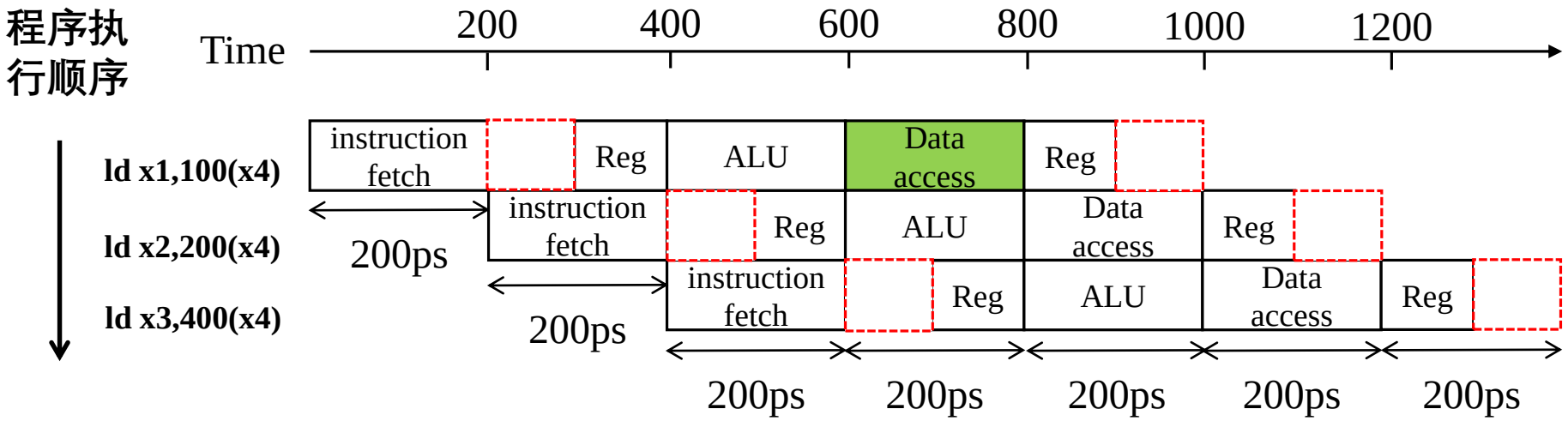
- 所有RISC-V指令长度相同，都是32 bits
  - 简化了流水线第一阶段取指令和第二阶段指令译码
  - x86: 1-17 bytes 的变长指令不利于流水线实现
- 只有6种指令格式，格式整齐
  - 能在一个阶段内完成译码和读寄存器
- 存储器操作只出现在load/store指令中
  - 利用执行阶段来计算存储器地址，然后在下一阶段访问存储器

流水线冒险(hazard)

- 流水线中，有时在下一个时钟周期中无法执行相应的指令
- **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情况
- **数据冒险**
  - 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情况
- **控制冒险（分支冒险）**
  - 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.

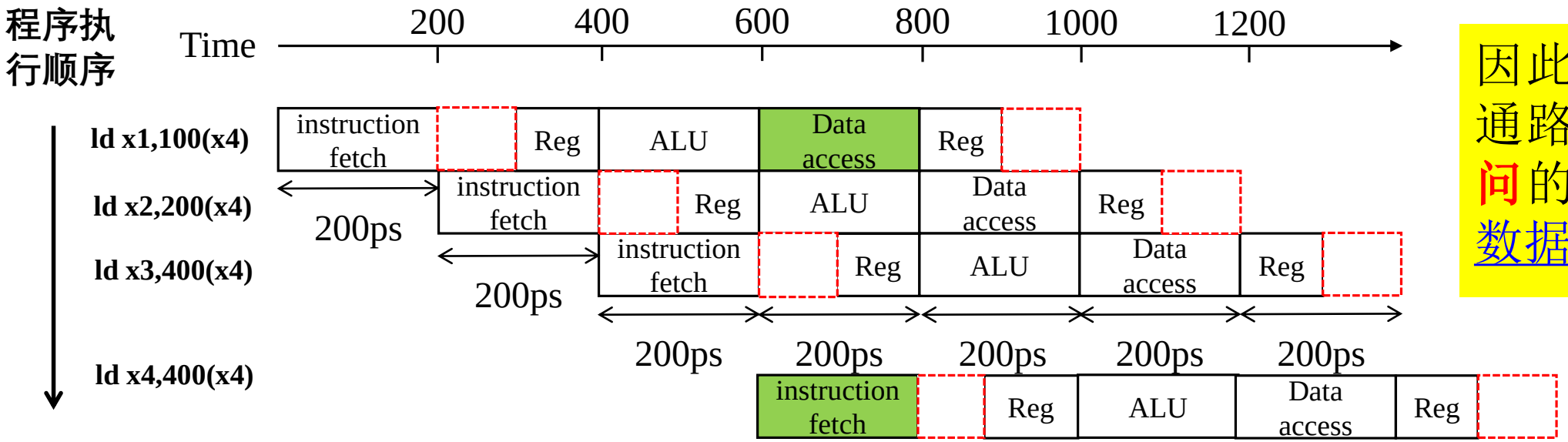
结构冒险

- **硬件资源**不支持多条指令在同一时钟周期执行
- **假设**下图RISC-V流水线结构**只有一个存储器**
  - load/store需要访问存储器
  - 如果有第四条指令，则第一条指令从存储器取数据的同时，第四条指令从同一存储器取指令，流水线发生结构冒险



结构冒险

- **硬件资源**不支持多条指令在同一时钟周期执行
- 假设下图RISC-V流水线结构只有一个存储器
  - load/store需要访问存储器
  - 如果有第四条指令，则第一条指令从存储器取数据的同时，第四条指令从同一存储器取指令，流水线发生结构冒险

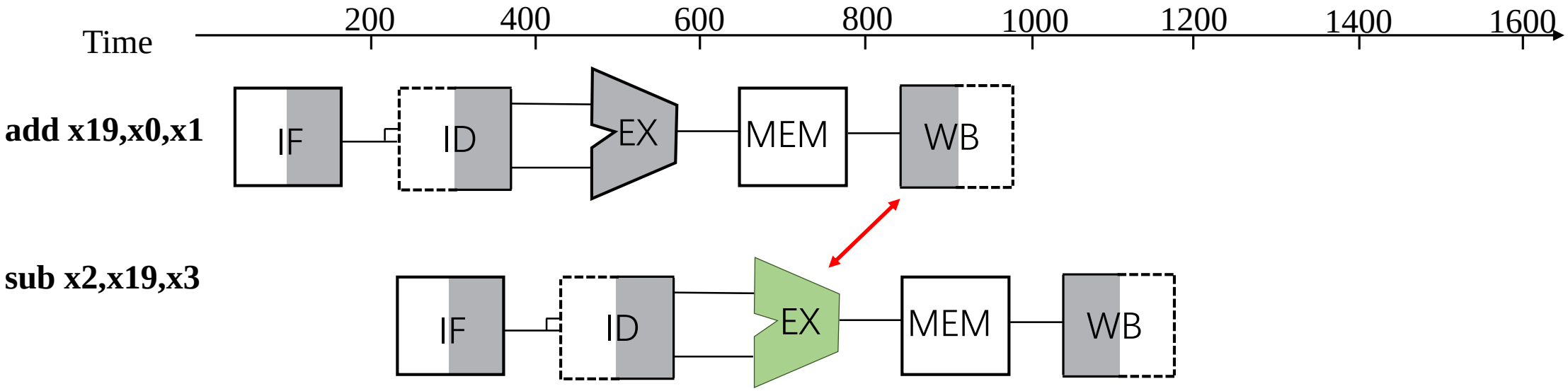


因此，流水线数据通路需要**可独立访问**的指令存储器和数据存储器

数据冒险

- 一条指令依赖于前面一条尚在流水线中的指令运行结果

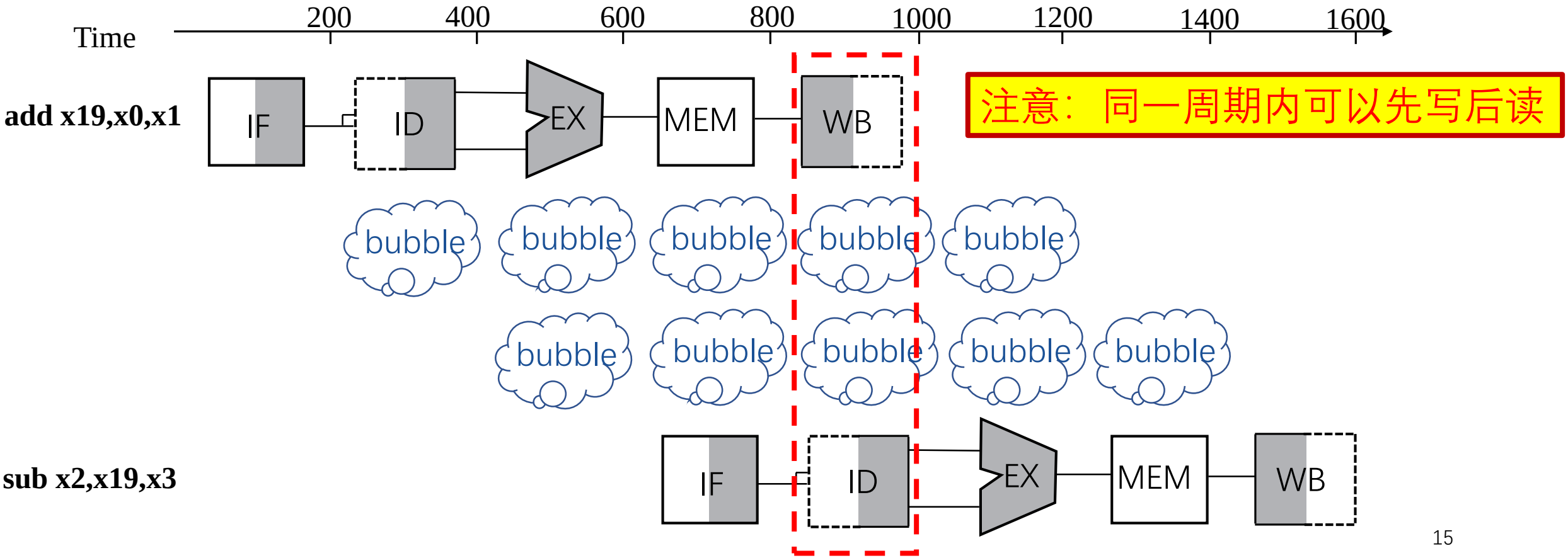
```
add    x19, x0, x1
sub    x2, x19, x3
```



数据冒险

- 一条指令依赖于前面一条尚在流水线中的指令运行结果

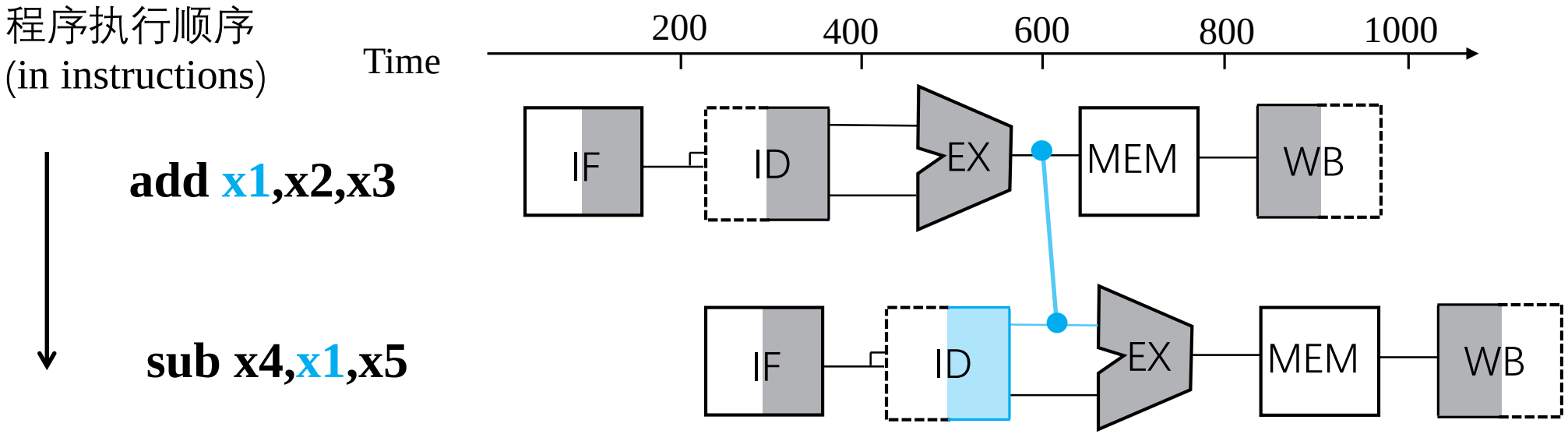
```
add    x19, x0, x1
sub    x2, x19, x3
```



前递(forwarding)

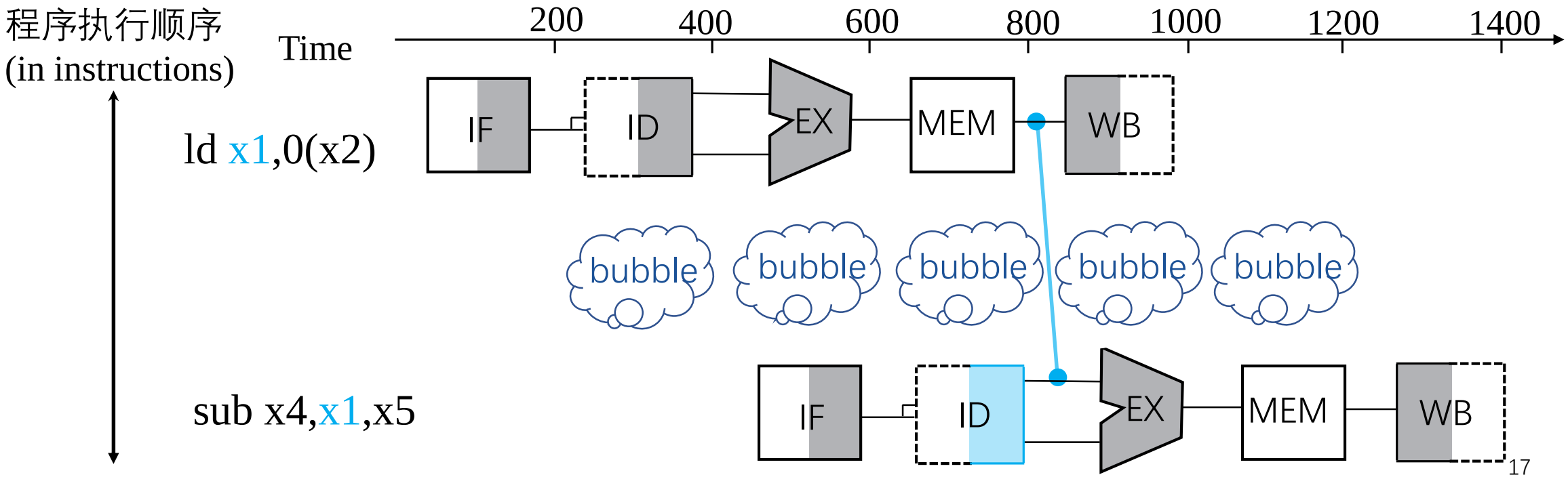
- 一种解决数据冒险的方法
  - 一旦ALU计算出结果，就提前取到数据，而不是等到数据到达寄存器或存储器
  - 需要向内部资源添加额外的硬件
    - 寄存器堆或存储器右半部分为阴影：正在被读
    - 反之，则正在被写

注意：寄存器堆或存储器
右半阴影表示正在被读；
左半阴影表示正在被写。
（即：先写后读划分）



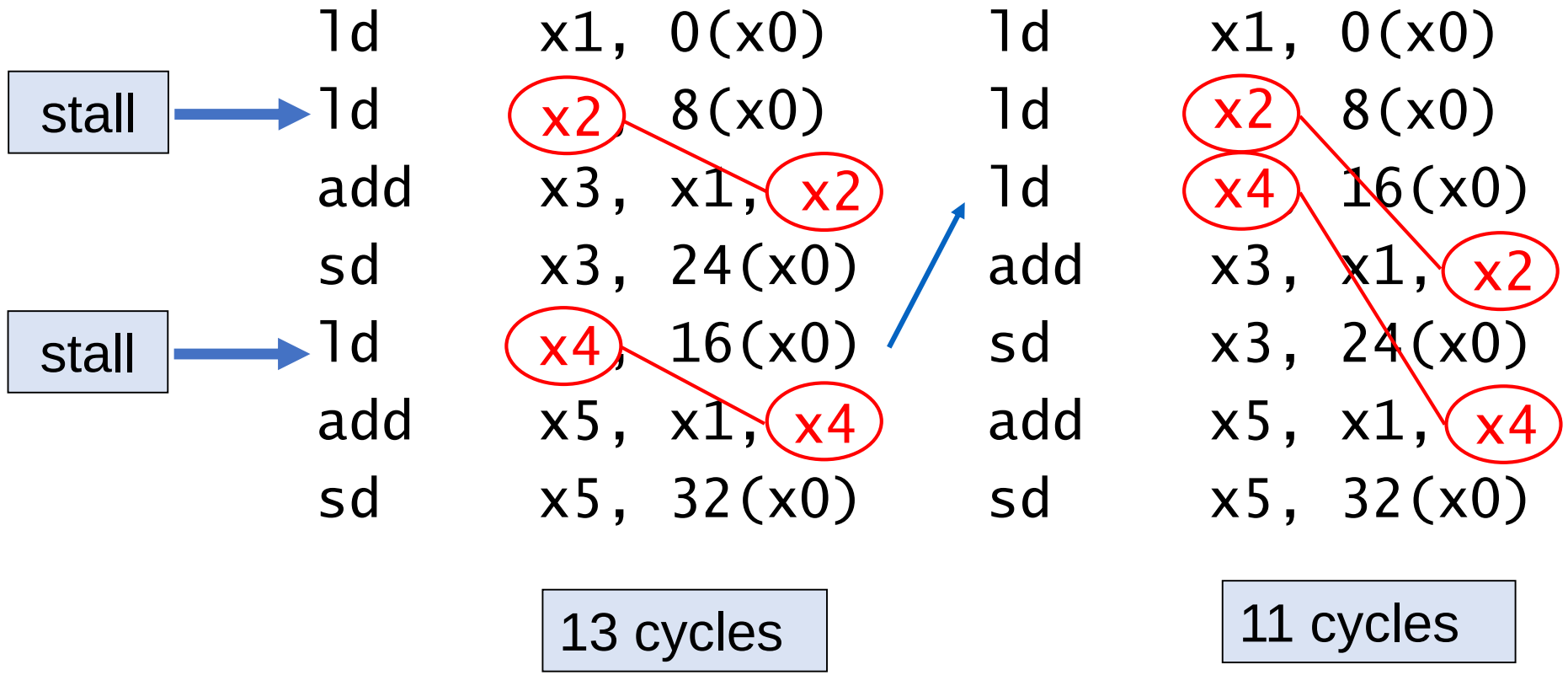
载入-使用型数据冒险(load-use data hazard)

- 前递不能避免所有的流水线停顿
  - 当载入指令要取的数据还没被取回，其他指令就需要该数据时，无法及时前递
  - 一种解决方案：**流水线停顿(stall)，也称为气泡(bubble)，为了解决冒险而实施的一种阻塞**



重排代码以避免流水线停顿（重要）

- 重排代码，避免在load指令之后立即使用取到的数据
- C code: a = b + e; c = b + f;

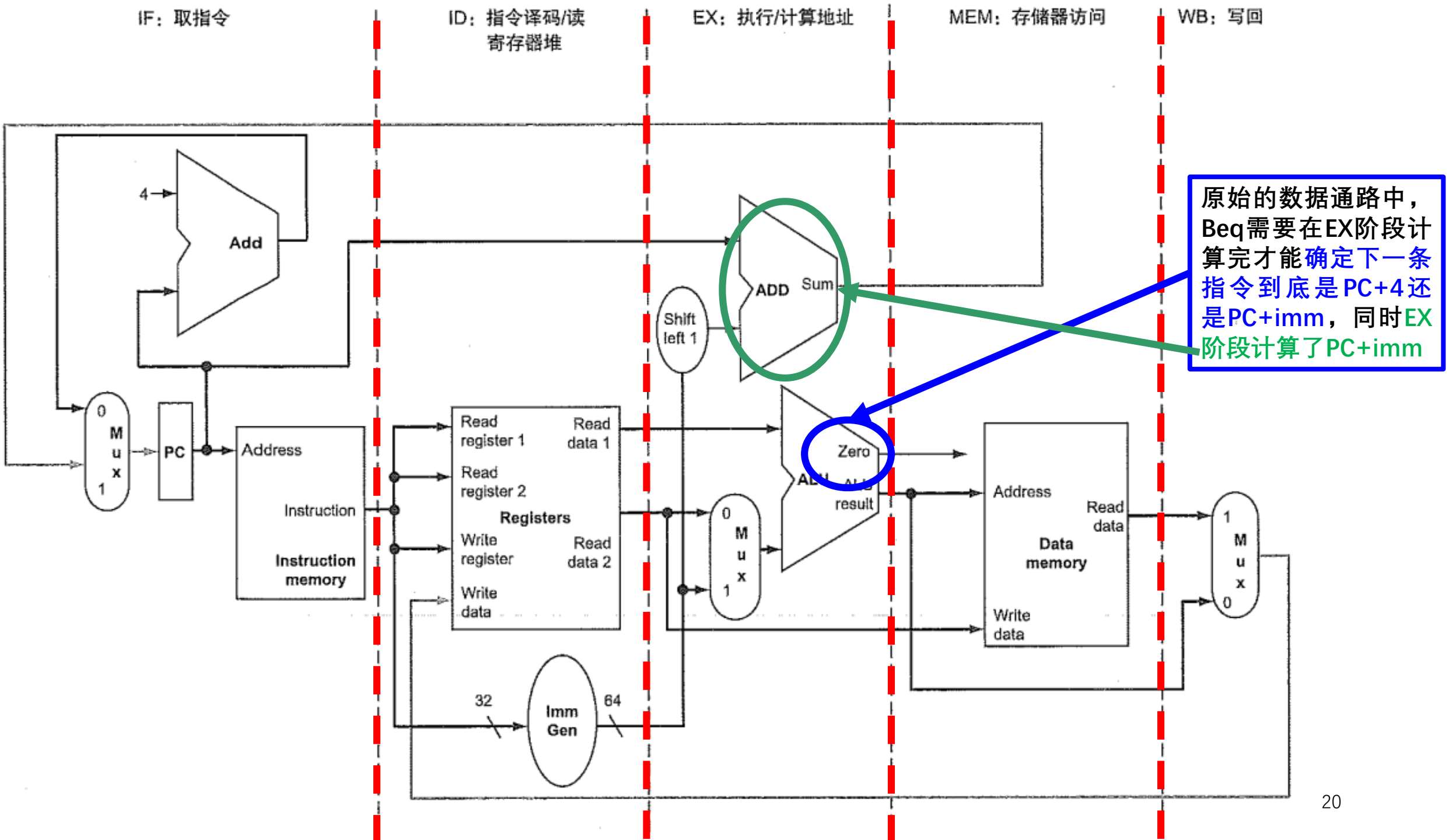


注意：此时x2仍然使用了前递

控制冒险（分支跳转）

- 分支决定了控制流：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
  - 由于分支指令之后的IF阶段依赖于分支的结果，因此流水线无法保证永远取到正确的指令
  - 假设分支指令在ID阶段，那么**其下一条指令**已经处在IF阶段

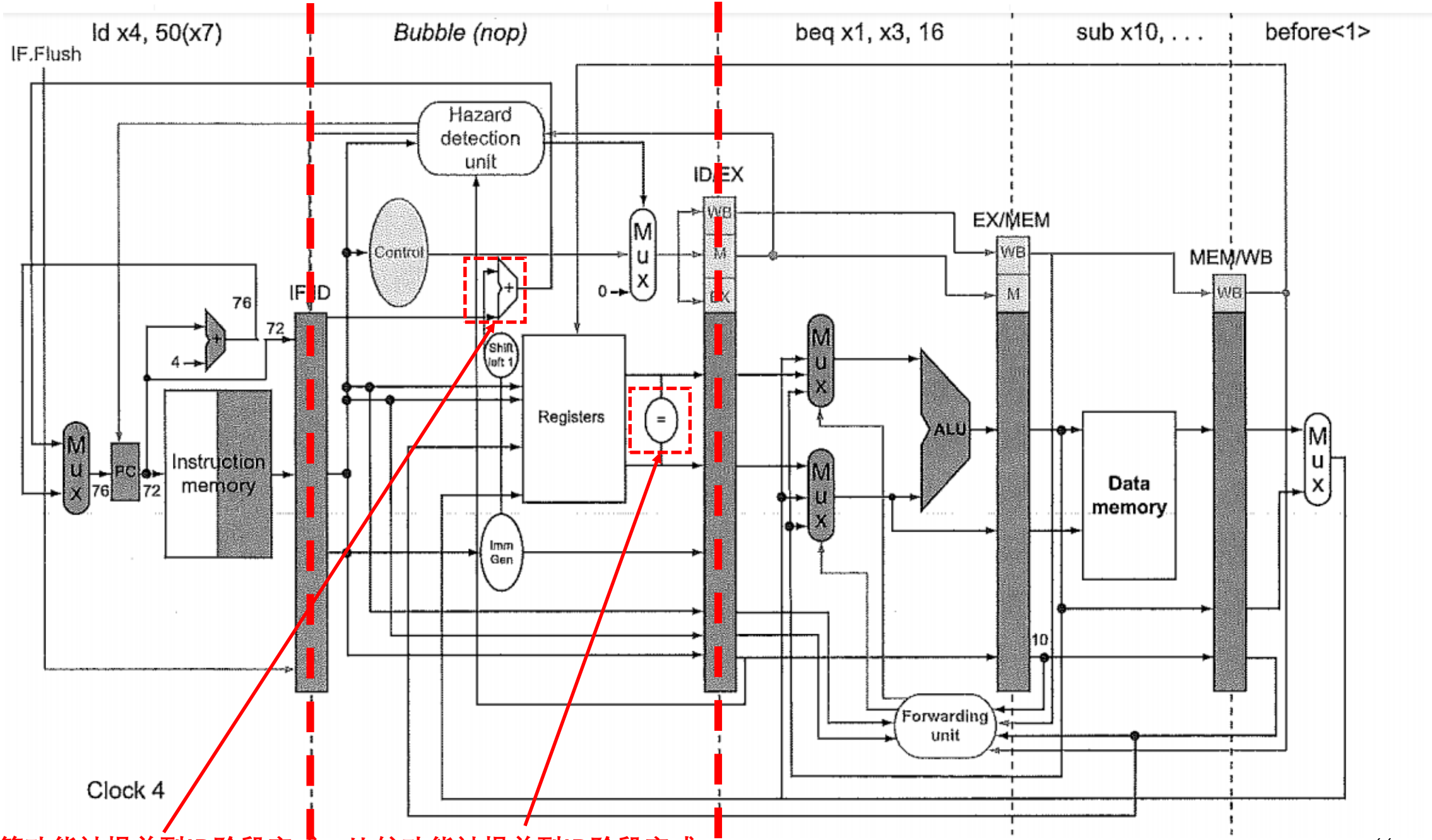
注意：“**其下一条指令**”有可能被错误地被加载进来，因为此时还不能计算出比较的结果，不知道是否跳转（**见下页PPT图示**）



原始的数据通路中，Beq需要在EX阶段计算完才能确定下一条指令到底是PC+4还是PC+imm，同时EX阶段计算了PC+imm

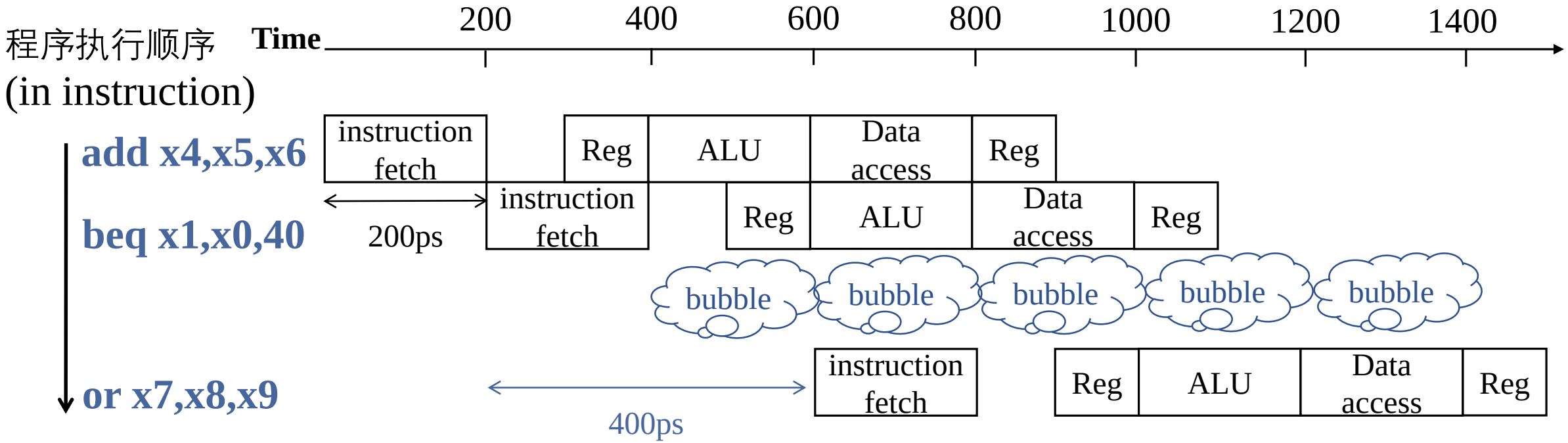
控制冒险（分支跳转）

- **解决方案一**
  - 在RISC-V流水线中尽可能早地完成寄存器比较、分支目标地址计算
  - 添加硬件，使得以上这两项功能可以提前到在ID阶段完成


ld x4, 50(x7)
Bubble (nop)
beq x1, x3, 16
sub x10, . . .
before<1>
IF.Flush
Hazard detection unit
Control
ID/EX
EX/MEM
MEM/WB
WB
M
EX
IF/ID
76
72
4
0
Shift left 1
Registers
=
Imm Gen
Instruction memory
PC
ALU
Data memory
Forwarding unit
10
Clock 4
计算功能被提前到ID阶段完成
比较功能被提前到ID阶段完成

解决方案二：每遇到条件分支指令就停顿的流水线

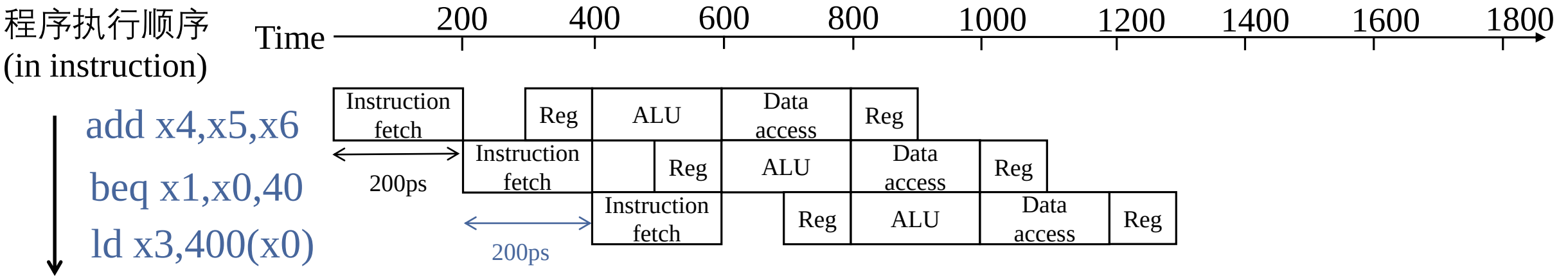
- 在取下一条指令之前，等待分支结果
  - beq指令的ID阶段“通过其他硬件电路”得到分支结果

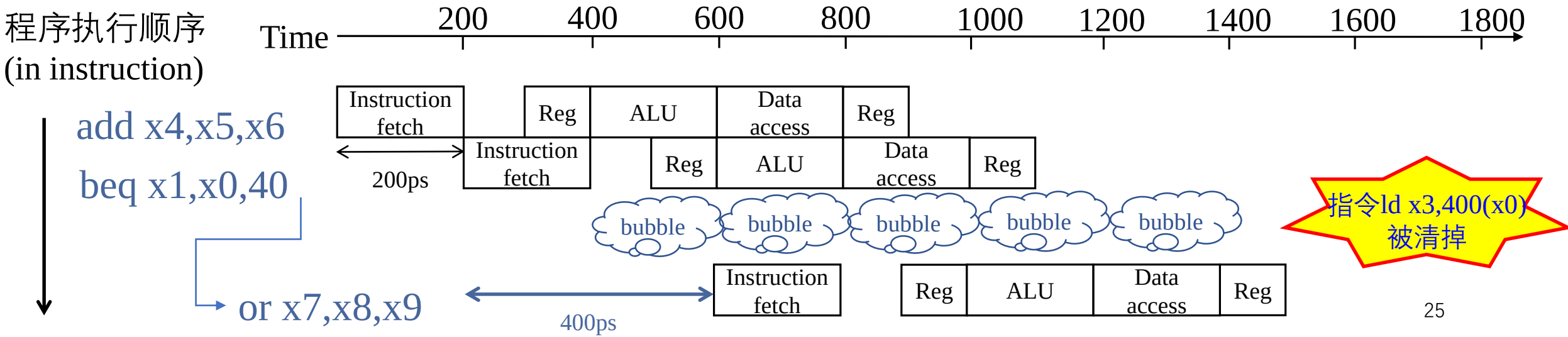


分支预测

- 对较长流水线而言，通常无法在第二阶段解决分支指令问题
  - 每个条件分支都停顿带来的代价太大，不可接受
- 采用**预测**来处理条件分支，并且当预测错误时，才引发停顿
  - 例如:可以总是预测分支指令不跳转。在分支指令之后，立即取指，流水线全速前进。如果后来判断出这条分支指令本应跳转，流水线此时要采用弥补措施，此时不得不耽搁时钟周期。

例子:判断提前到ID阶段+总是预测条件分支不发生跳转





两种更成熟的分支预测

- **静态分支预测**
  - 根据典型的分支行为来决定是否跳转
  - 例如：循环+条件指令产生的分支
    - 在计算机程序中，循环底部是条件分支指令，并会跳转回循环顶部
    - 很可能发生分支并向回跳转，所以可以预测发生分支并跳到靠前的地址处
- **动态分支预测**
  - 根据每个分支指令最近时间发生的行为进行预测
    - 一种常用实现方式：保存每个条件分支是否发生分支的历史纪录
  - 假设未来的行为会延续这个趋势
    - 当预测错误时，需要停顿、重新取指令，并更新历史记录

填空题 0分

设置

此题未设置答案，请点击右侧设置按钮

下面的程序在流水线处理器中执行时，程序 [3]无需前递和停顿，程序[1] 需要停顿、 [2] 可使用前递避免停顿。

程序1

```
ld x10, 0(x5)
add x11, x10, x10
```

程序2

```
add x11, x10, x10
addi x12, x10, 5
addi x14, x11, 5
```

程序3

```
addi x12, x10, 5
addi x14, x10, 5
addi x16, x10, 7
addi x13, x10, 9
addi x17, x10, 8
addi x15, x10, 15
```

作答

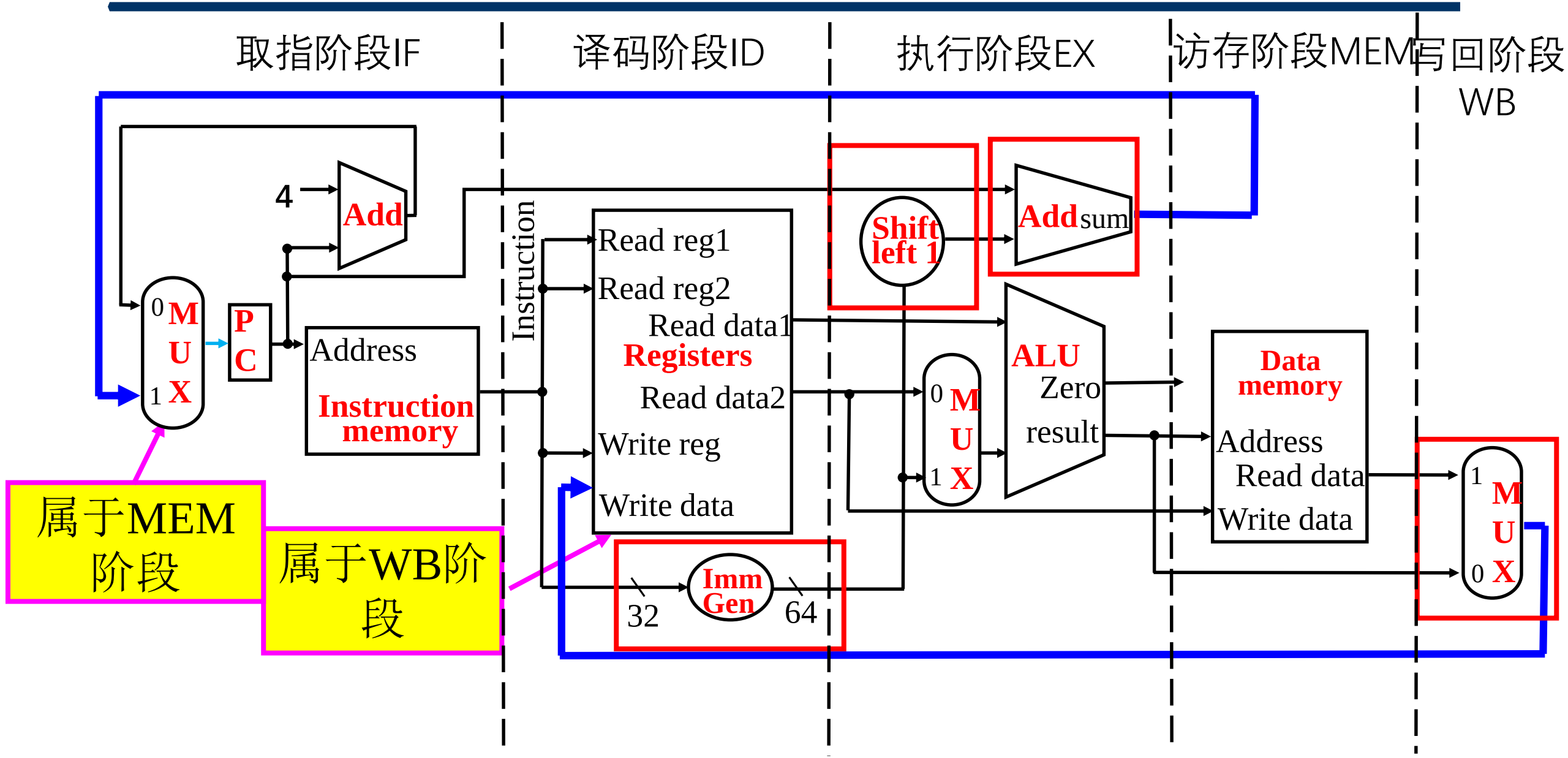
流水线总结

- 流水线通过提高吞吐率来提升性能
  - 并行执行多条指令
  - 每条指令拥有相同的延迟
    - 延迟：执行单条指令的时间
- 会受到冒险的影响
  - **结构、数据、控制**
- 指令集的设计影响到流水线实现的复杂度

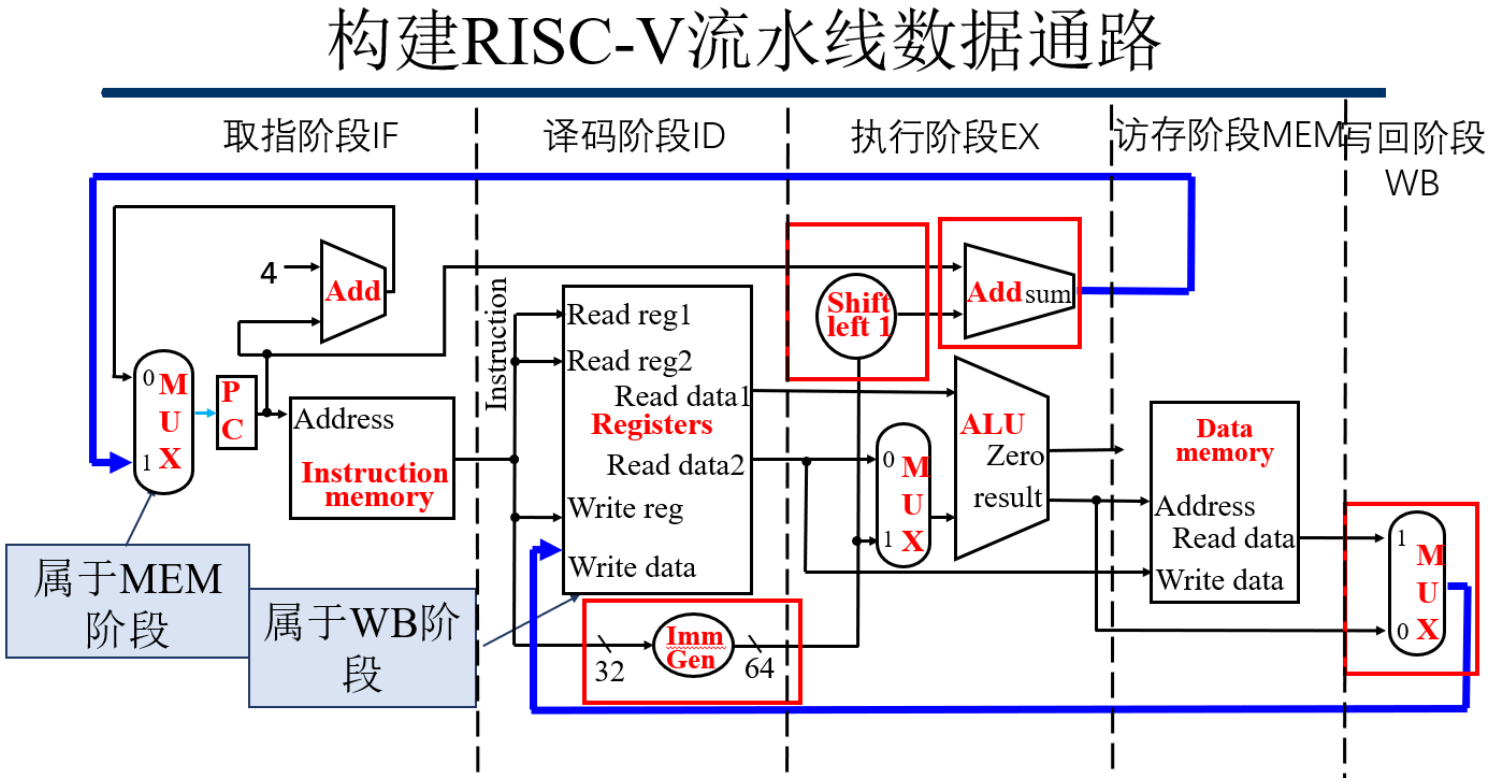
第五章

- 流水线概述
- **流水线数据通路和控制**
- 数据冒险：前递与停顿
- 控制冒险
- 例外

构建RISC-V流水线数据通路



RISC-V流水线数据通路中的特殊情况



1）下一条指令PC的选择：在自增PC值与MEM(数据存储器访问)阶段的分支地址之间进行选择；

2）写回阶段：将结果写回位于数据通路中的寄存器堆。

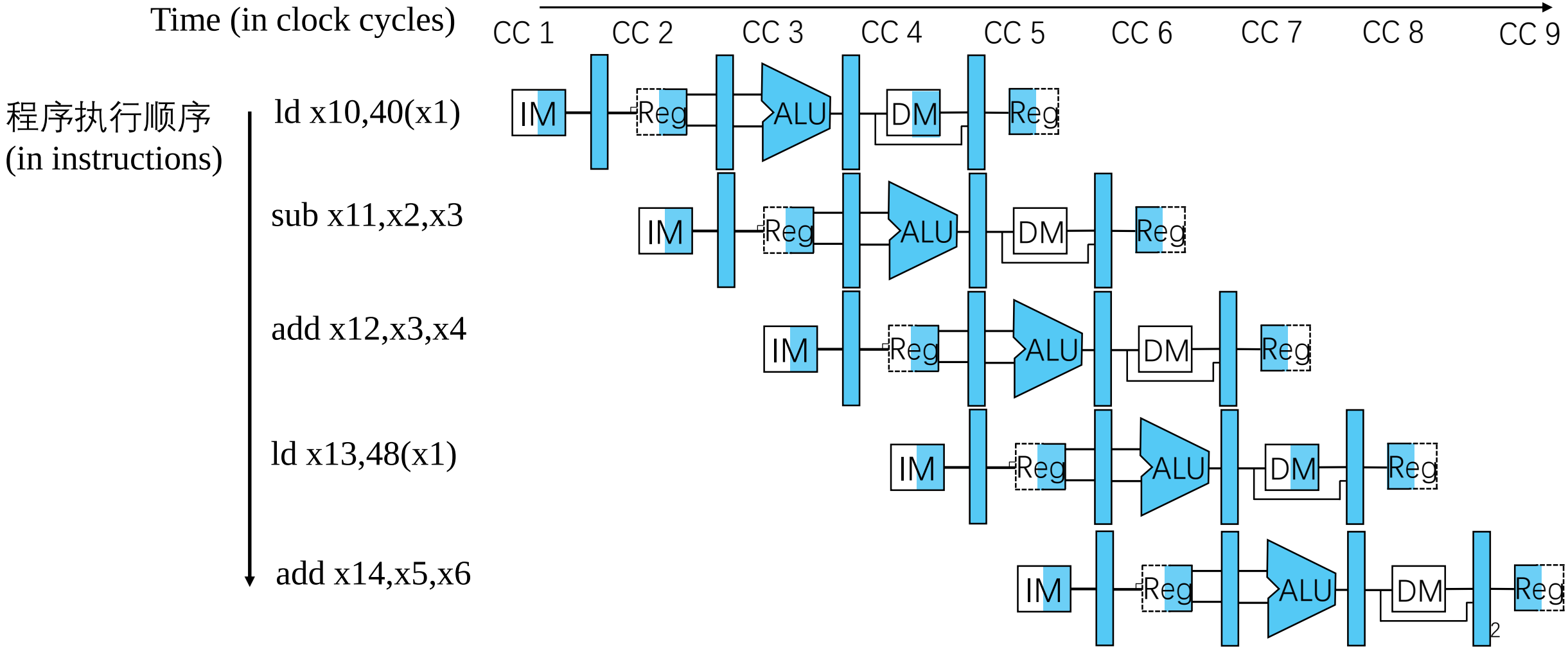
→ 控制冒险

→ 数据冒险

从右到左的数据流向可能引发冒险

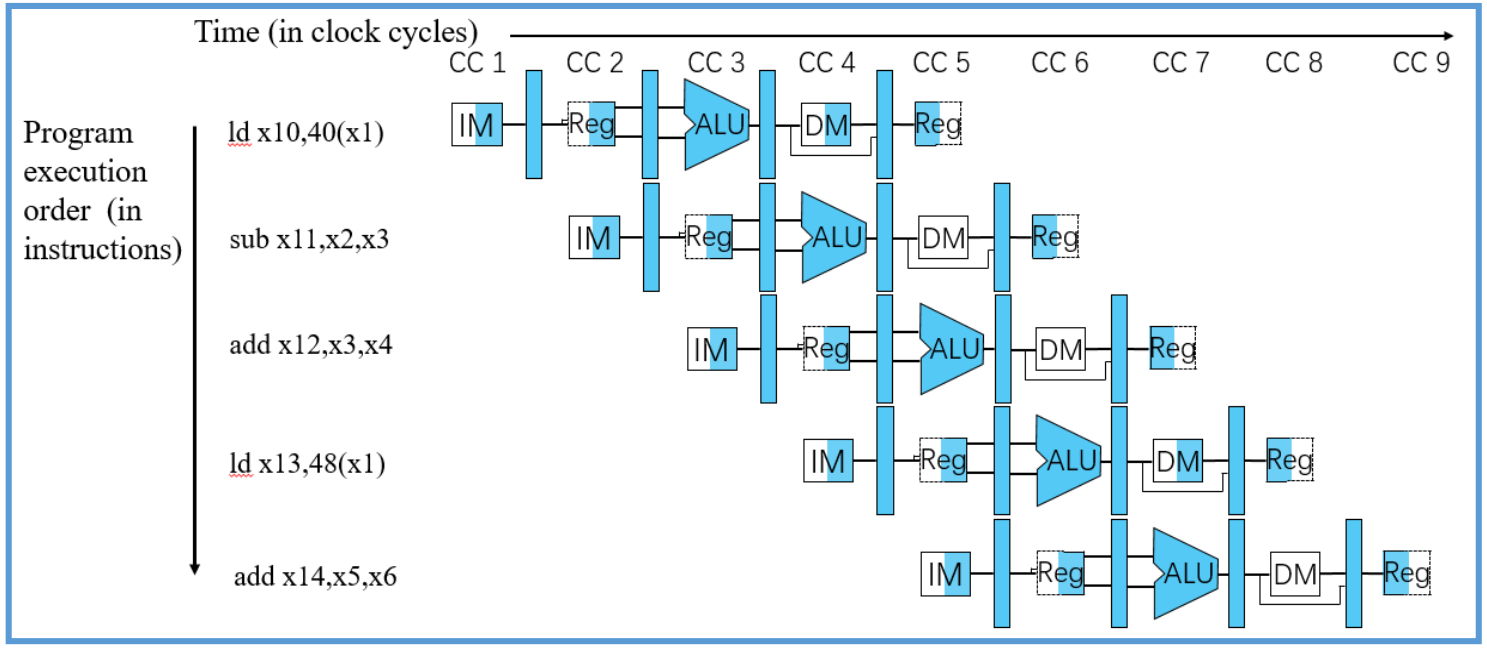
在流水线数据通路中观察指令的流动

Ripes：a graphical **processor simulator** and **assembly code editor** built for the RISC-V instruction set architecture （https://gitee.com/shzhou_admin/Ripes）



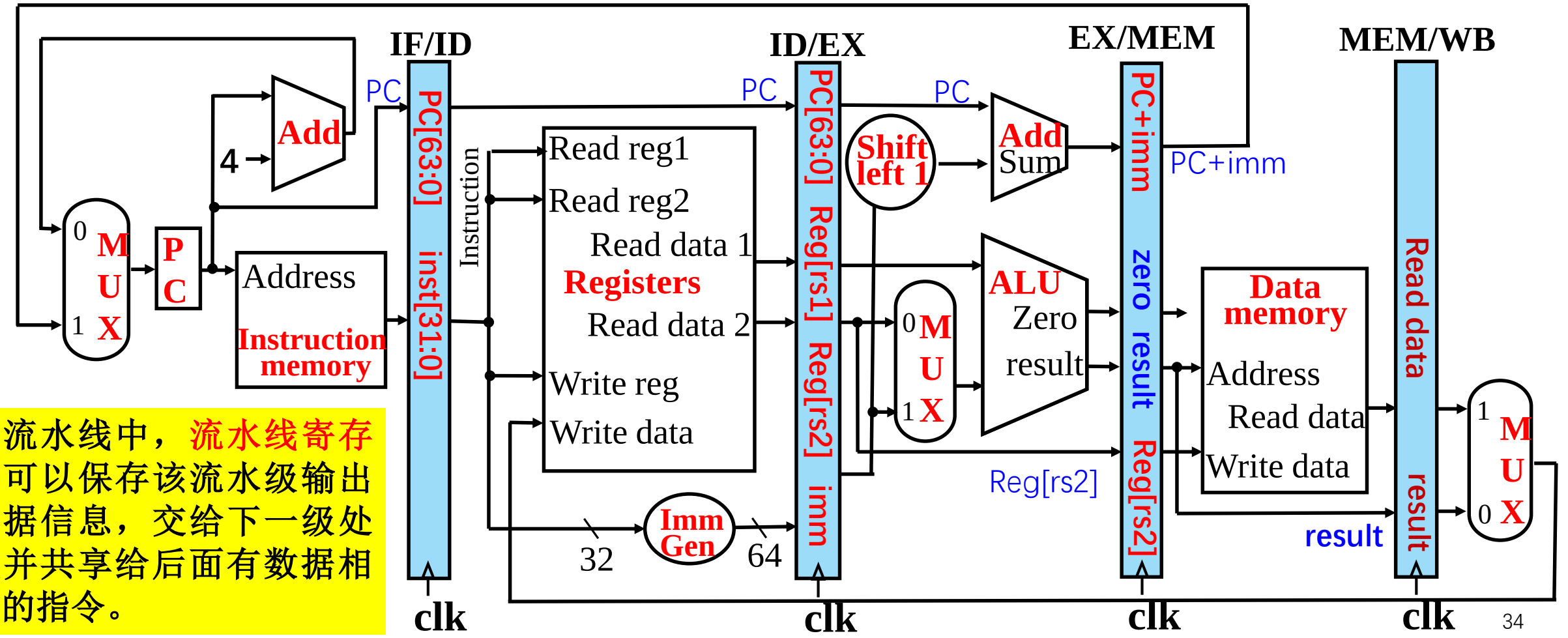
观察流水线操作

- 在流水线数据通路中，观察指令在不同周期的流动
  - **单时钟周期流水线图**（一个时钟周期的垂直切片）
    - 展现流水线在指定时钟周期上每条指令对数据通路的使用情况
    - 高亮表示使用到的硬件资源
  - **多时钟周期流水线图**
    - 展现一段时间的情况



加入流水线寄存器

- 在不同阶段之间需要寄存器——流水线寄存器
  - 保存前一个阶段产生的信息



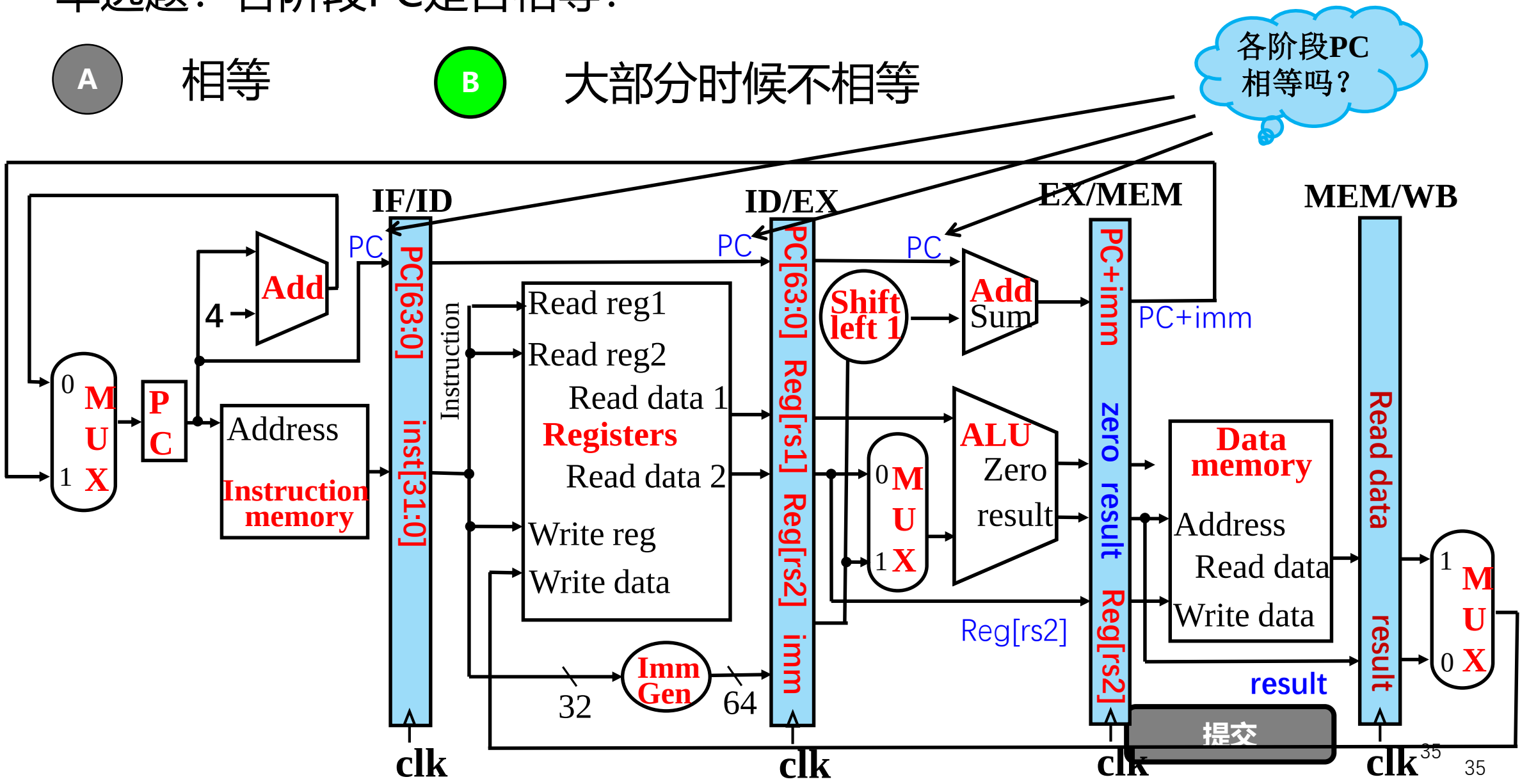
在流水线中，流水线寄存器可以保存该流水级输出数据信息，交给下一级处理并共享给后面有数据相关的指令。

设置

单选题：各阶段PC是否相等？

A 相等

B 大部分时候不相等



IF for Load, Store, …（观察寄存器）

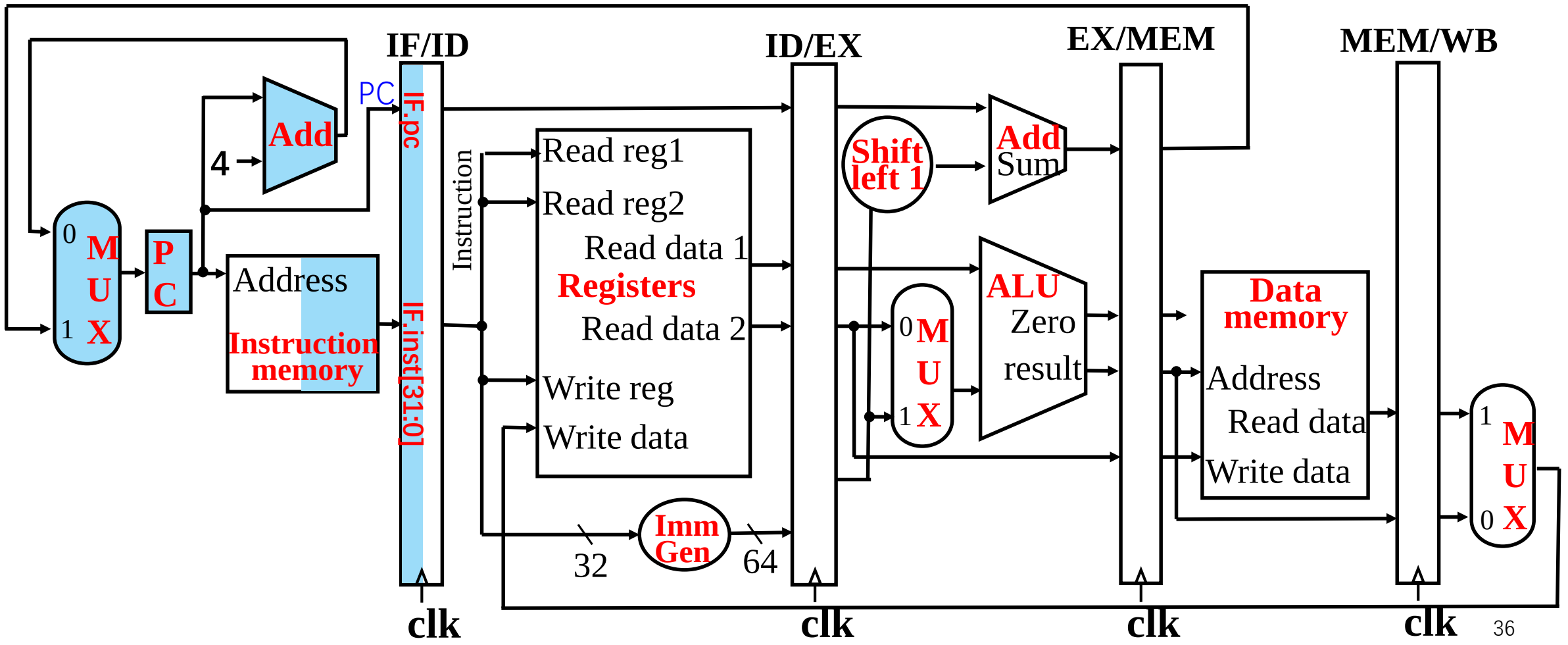
观察load和store指令的操作

ld

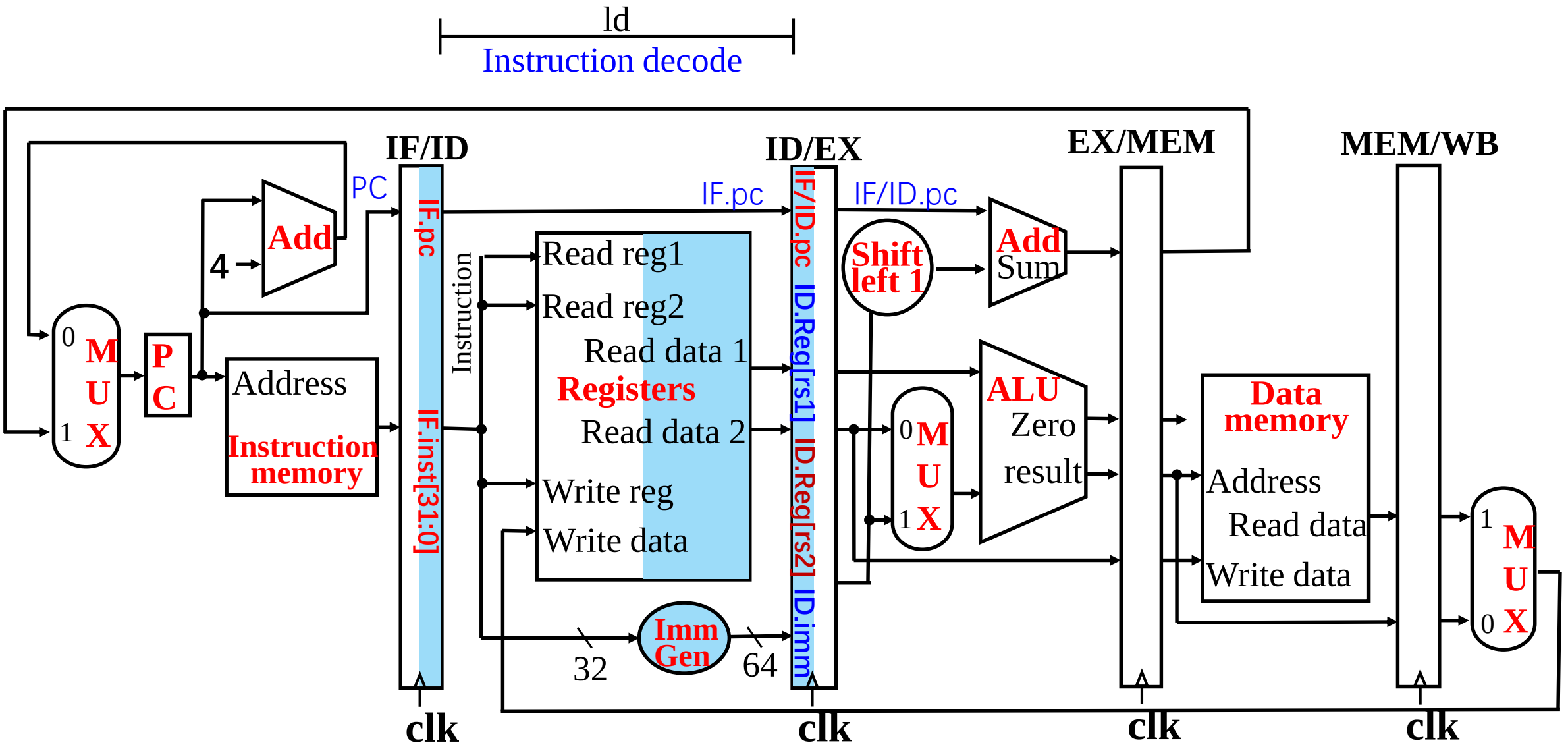
Instruction fetch

右半部分高亮：寄存器或存储器被读取

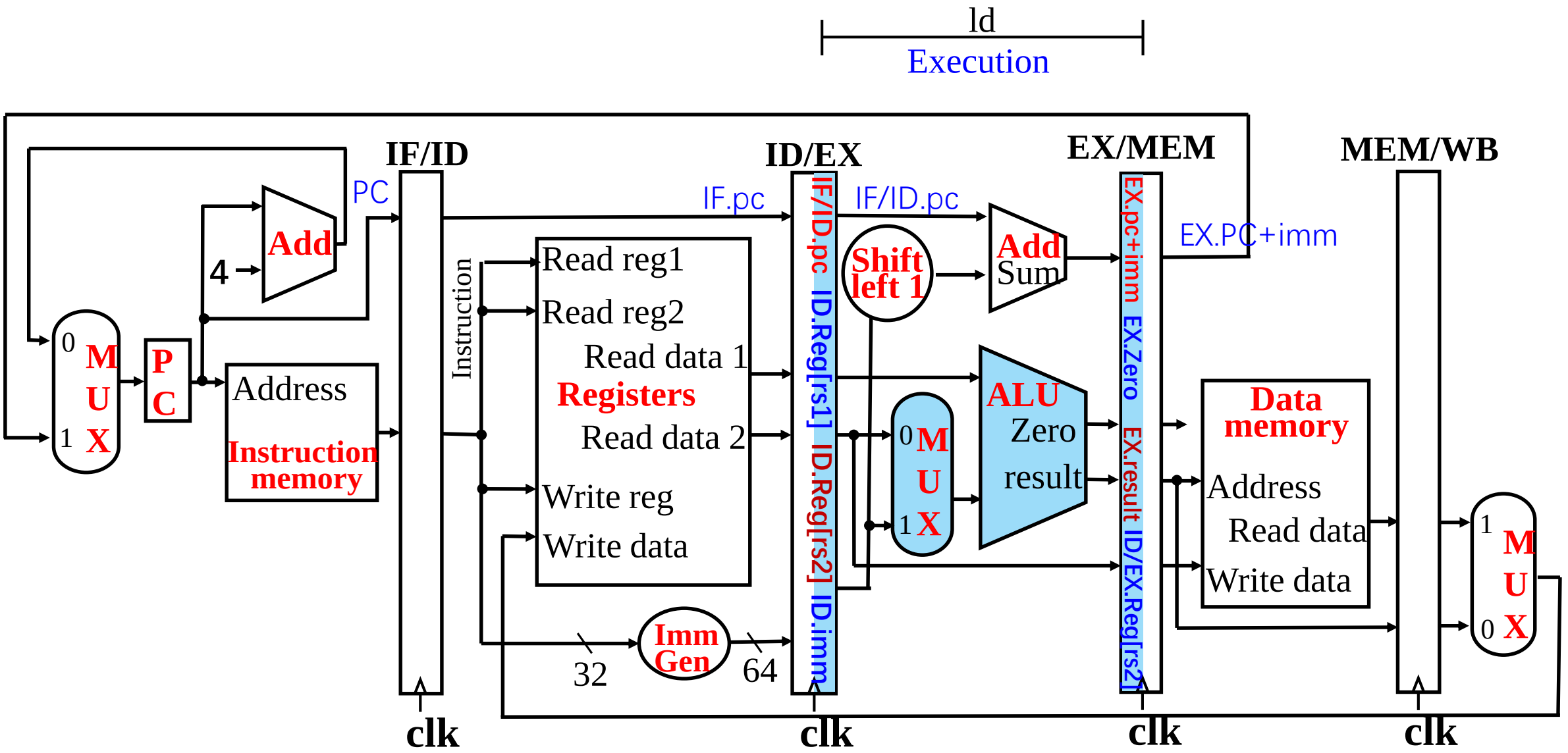
左半部分高亮：寄存器或存储器被写入



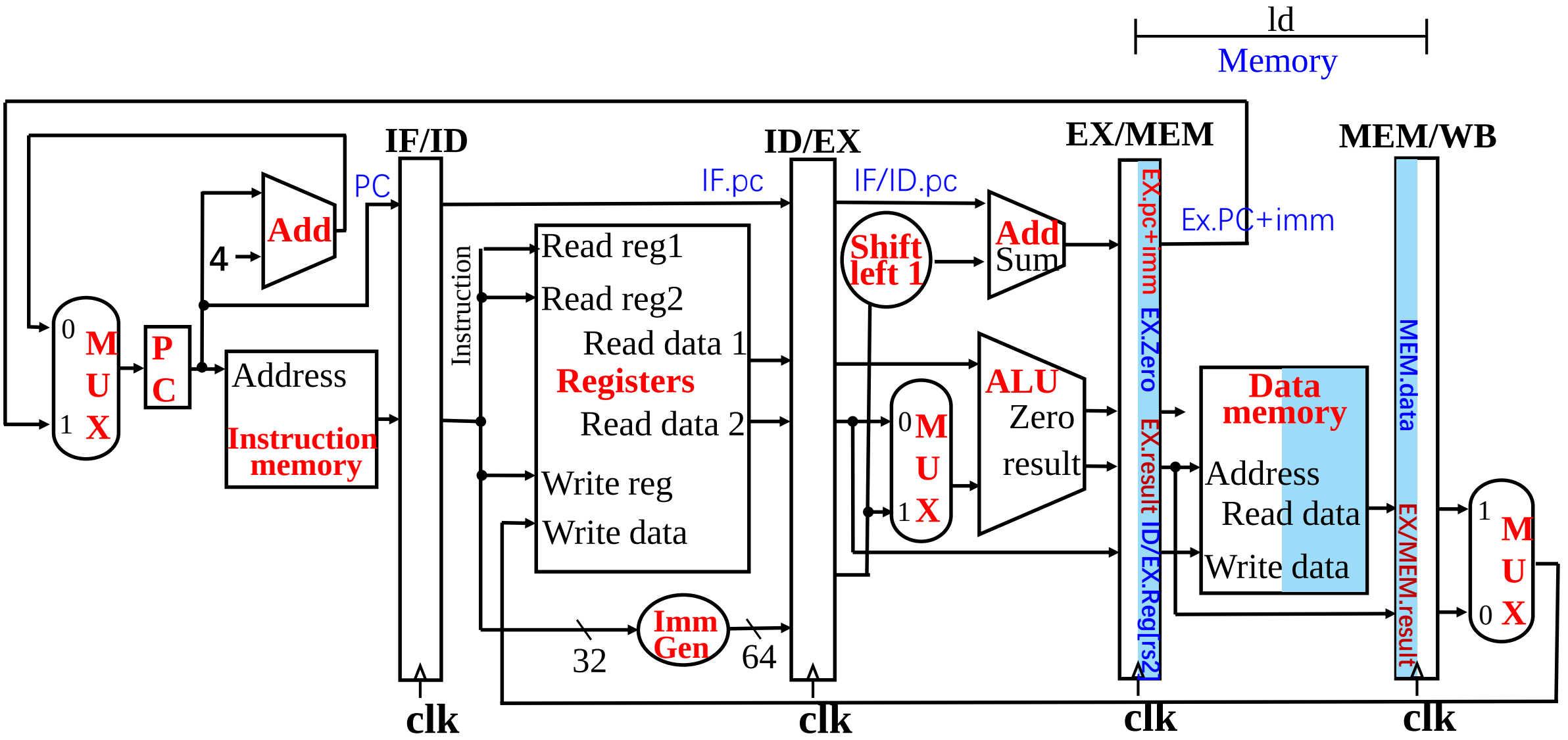
ID for Load, Store, …



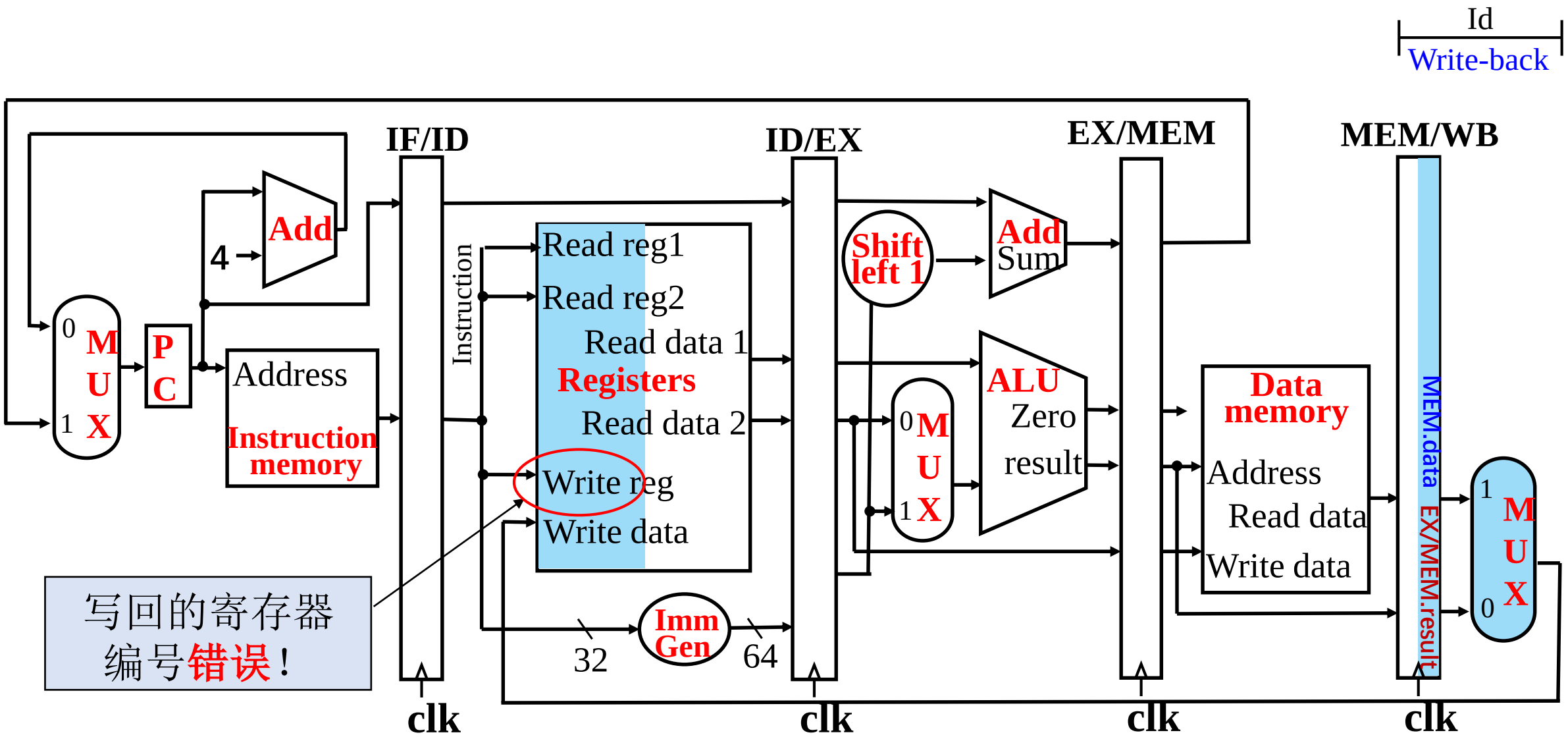
EX for Load


ld
Execution
IF/ID
ID/EX
EX/MEM
MEM/WB
PC
IF.pc
IF/ID.pc
EX.PC+imm
Add
4
Add
Sum
Shift
left 1
M
U
X
0
1
P
C
Address
Instruction
memory
Instruction
Read reg1
Read reg2
Registers
Read data 1
Read data 2
Write reg
Write data
IF/ID.pc ID.Reg[rs1] ID.Reg[rs2] ID.imm
ALU
Zero
result
EX.pc+imm EX.Zero EX.result ID/EX.Reg[rs2]
Data
memory
Address
Read data
Write data
Imm
Gen
32
64
clk
clk
clk
clk

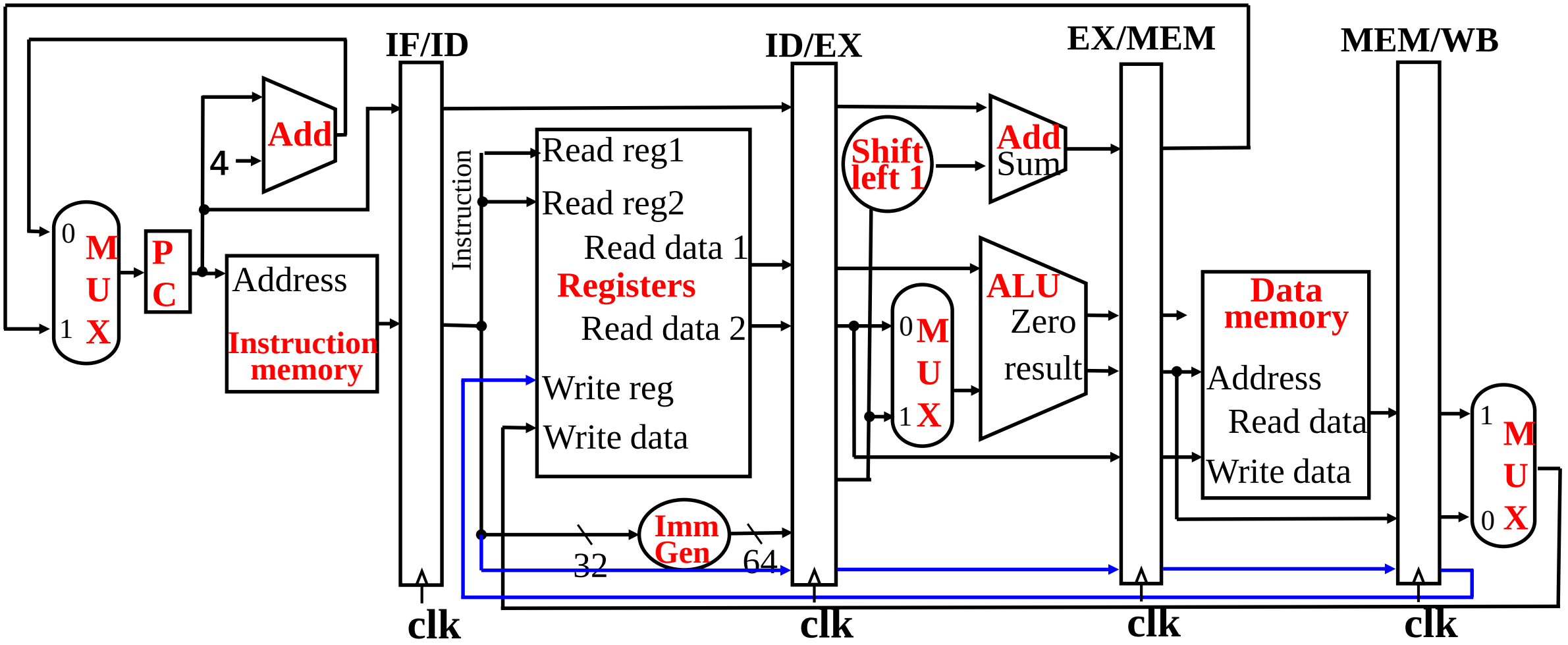
MEM for Load


ld
Memory
IF/ID
ID/EX
EX/MEM
MEM/WB
PC
IF.pc
IF/ID.pc
Add
4
0
M
U
X
1
P
C
Address
Instruction memory
Instruction
Read reg1
Read reg2
Read data 1
Registers
Read data 2
Write reg
Write data
Imm Gen
32
64
Shift left 1
Add
Sum
Ex.PC+imm
ALU
Zero
result
EX.pc+imm
EX.Zero
EX.result
ID/EX.Reg[rs2]
Data memory
Address
Read data
Write data
MEM.data
EX/MEM.result
clk
clk
clk
clk

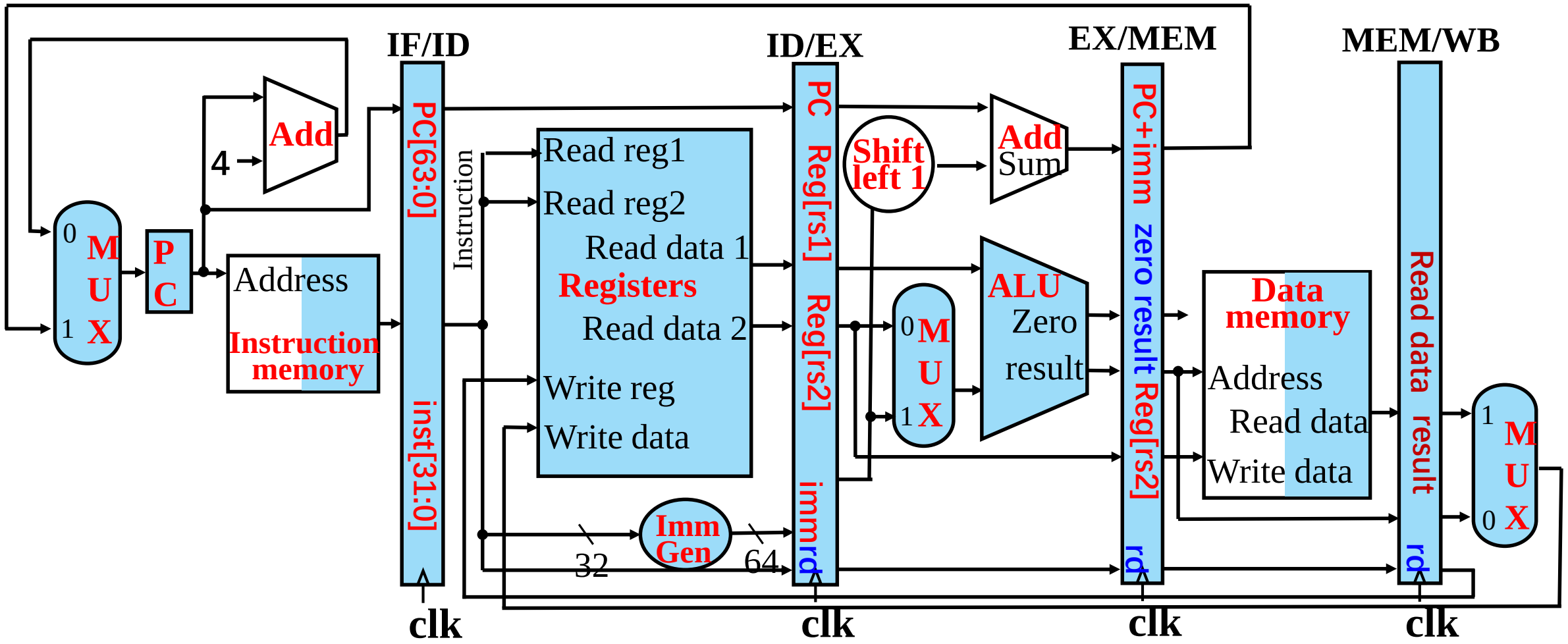
WB for Load


Id
Write-back
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
0
M
U
X
1
PC
Address
Instruction memory
Instruction
Read reg1
Read reg2
Read data 1
Registers
Read data 2
Write reg
Write data
Imm Gen
32
64
Shift left 1
Add
Sum
0
M
U
X
1
ALU
Zero
result
Data memory
Address
Read data
Write data
MEM.data
EX/MEM.result
1
M
U
X
0
clk
clk
clk
clk
写回的寄存器编号错误！

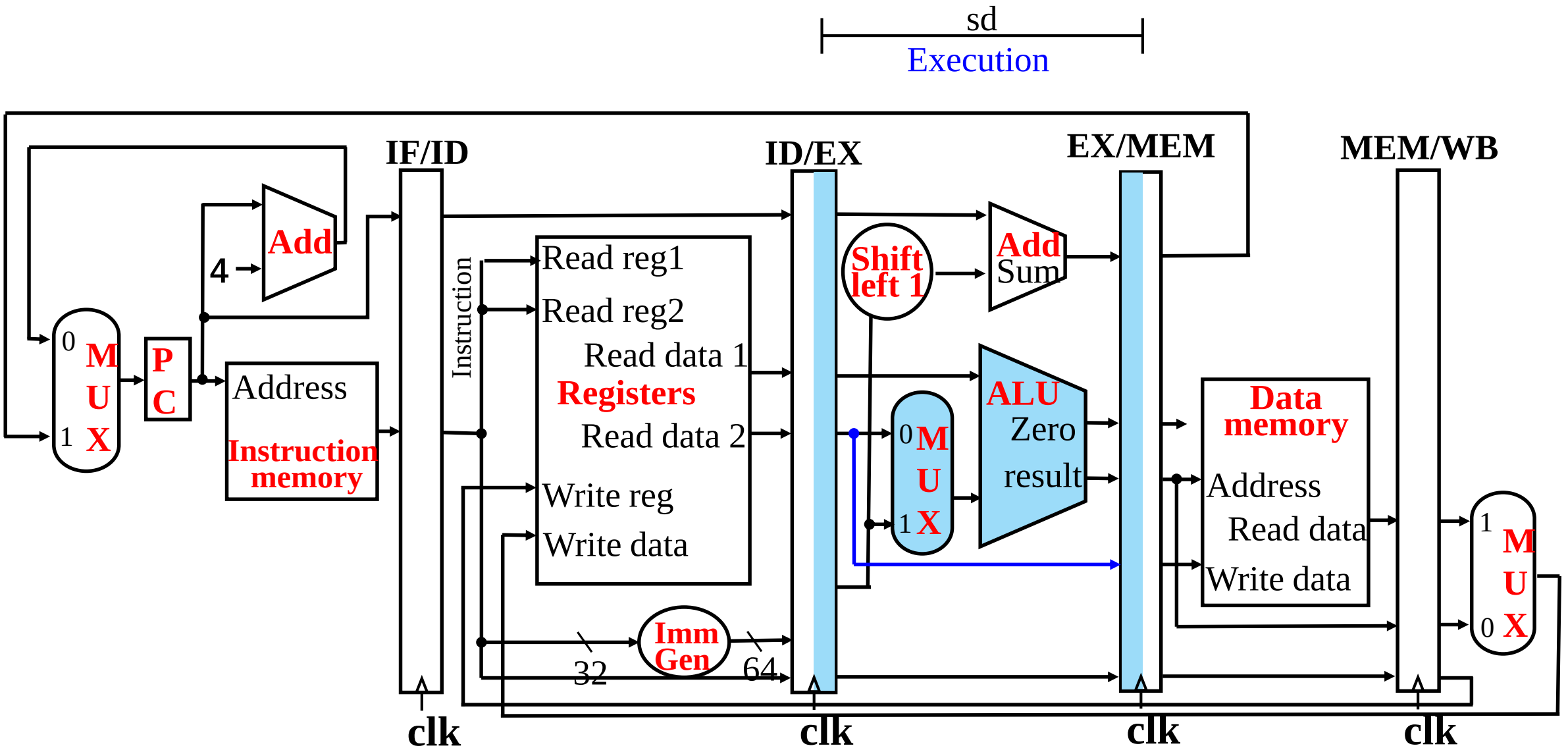
更正之后的数据通路


IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
M
U
X
0
1
P
C
Address
Instruction memory
Instruction
Read reg1
Read reg2
Read data 1
Registers
Read data 2
Write reg
Write data
Imm Gen
32
64
Shift left 1
Add
Sum
ALU
Zero
result
Data memory
Address
Read data
Write data
clk
clk
clk
clk

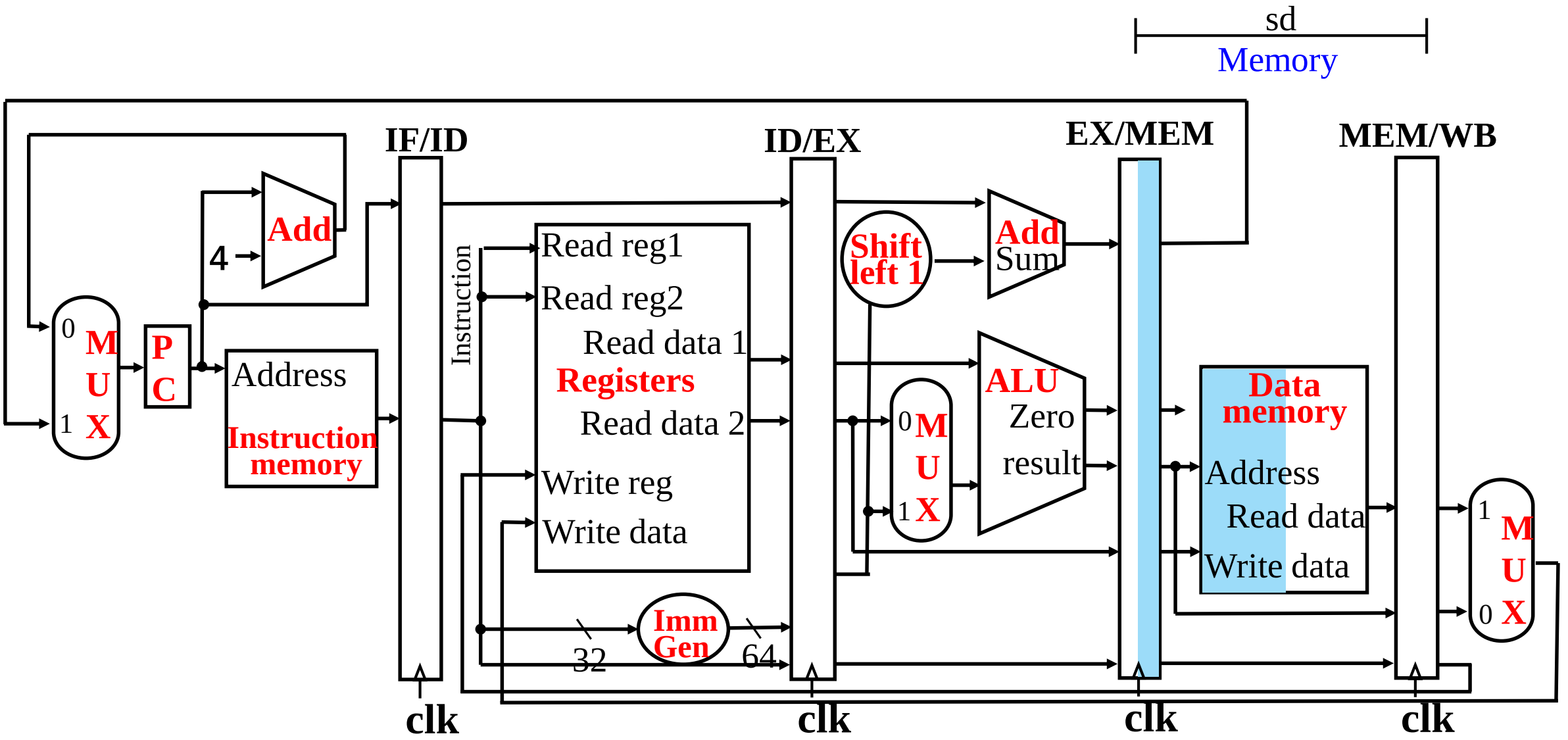
用于load类指令的全部五个流水线阶段部分



EX for Store


sd
Execution
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
M
U
X
0
1
P
C
Address
Instruction memory
Instruction
Read reg1
Read reg2
Read data 1
Registers
Read data 2
Write reg
Write data
Imm Gen
32
64
Shift left 1
Add
Sum
ALU
Zero
result
Data memory
Address
Read data
Write data
clk
clk
clk
clk

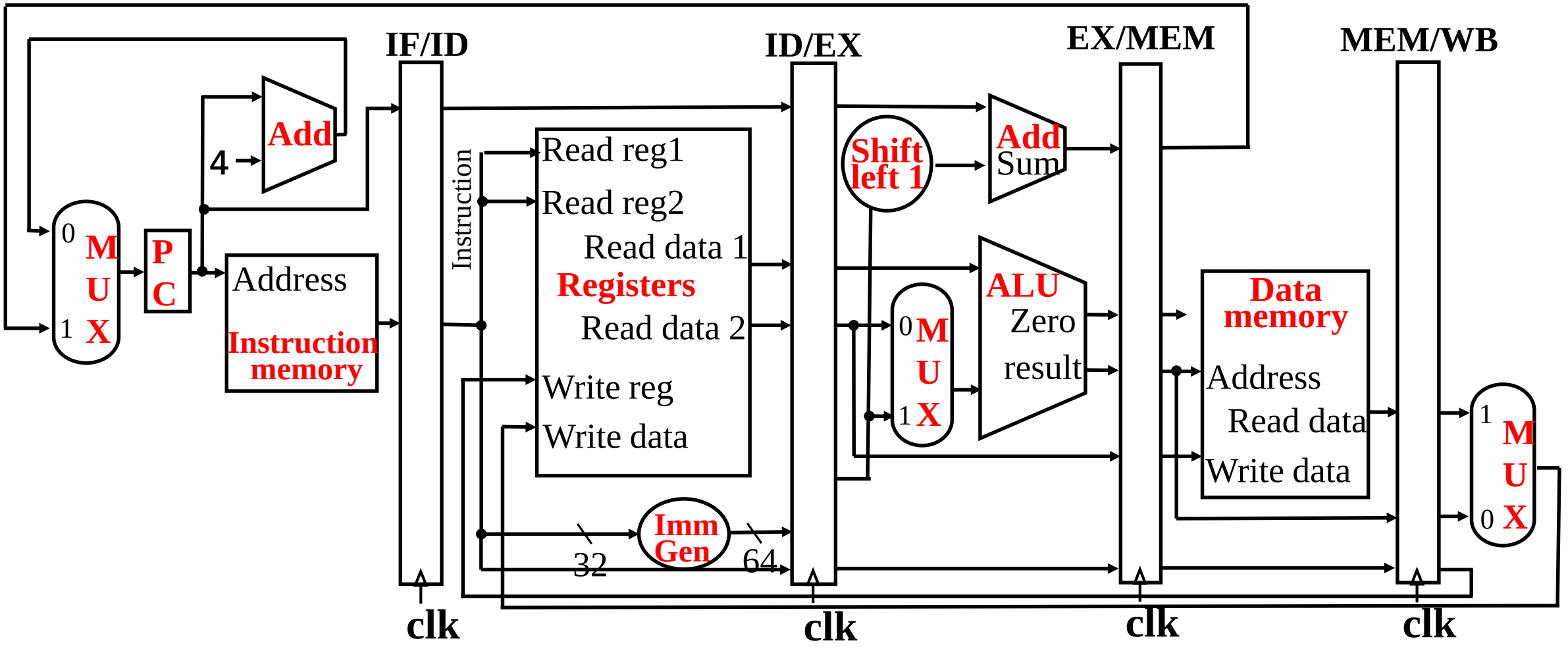
MEM for Store



WB for Store

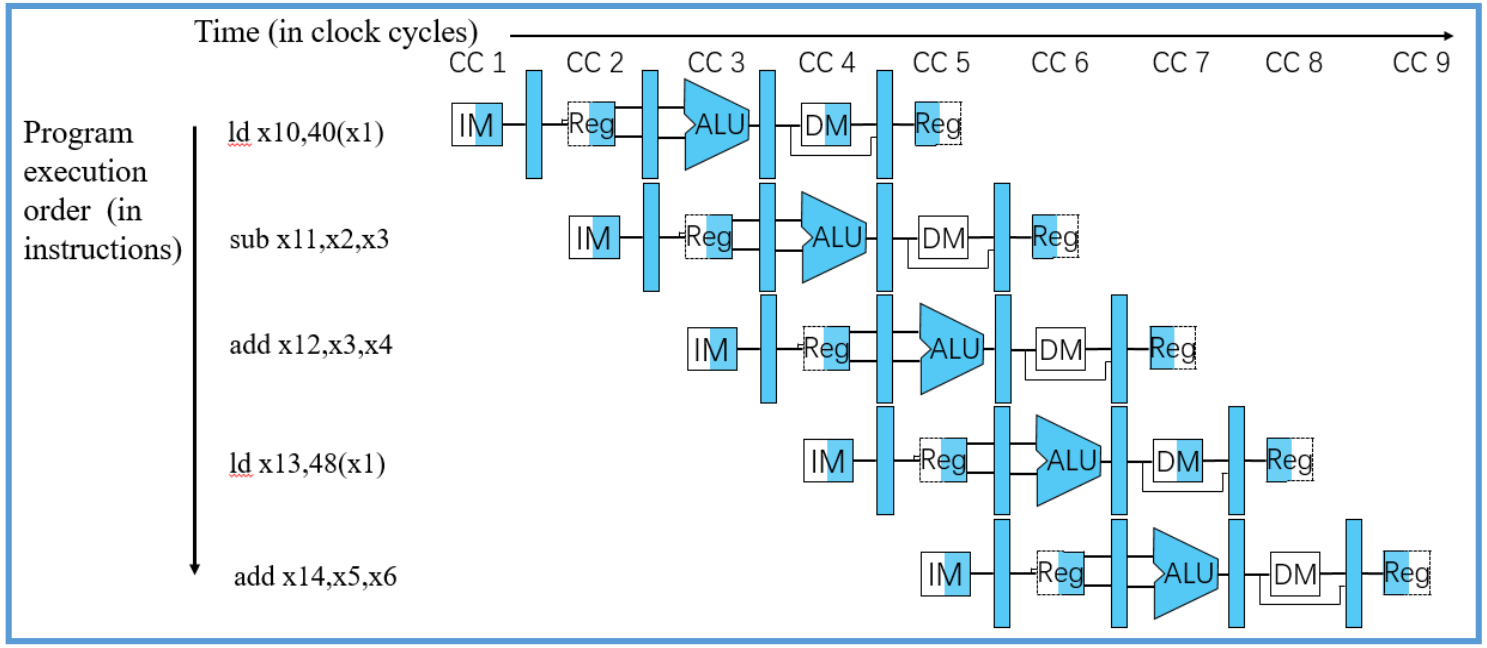
存储指令在“写回阶段”不执行任何操作。

sd
Write-back



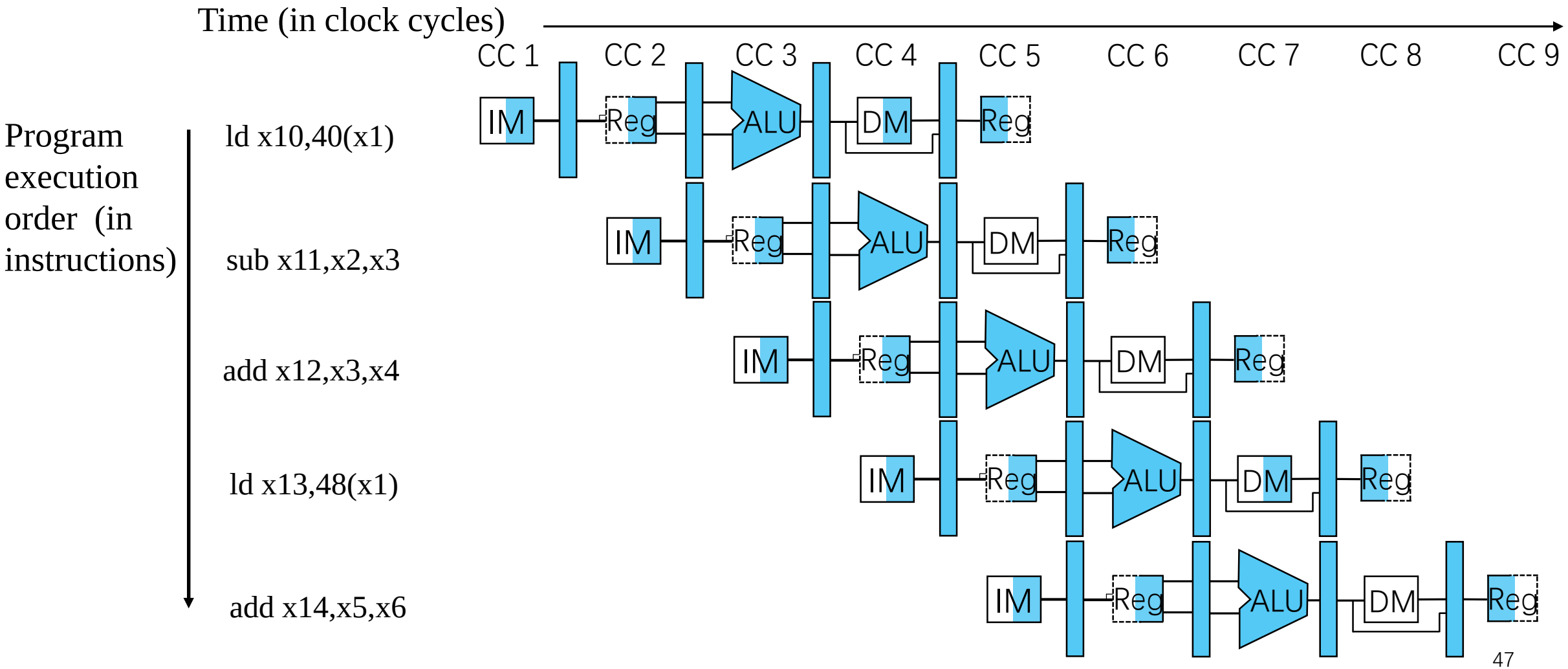
观察流水线操作

- 在流水线数据通路中，观察指令在不同周期的流动
  - **单时钟周期流水线图**（一个时钟周期的垂直切片）
    - 展现流水线在指定时钟周期上每条指令对数据通路的使用情况
    - 高亮表示使用到的硬件资源
  - **多时钟周期流水线图**
    - 展现一段时间的情况



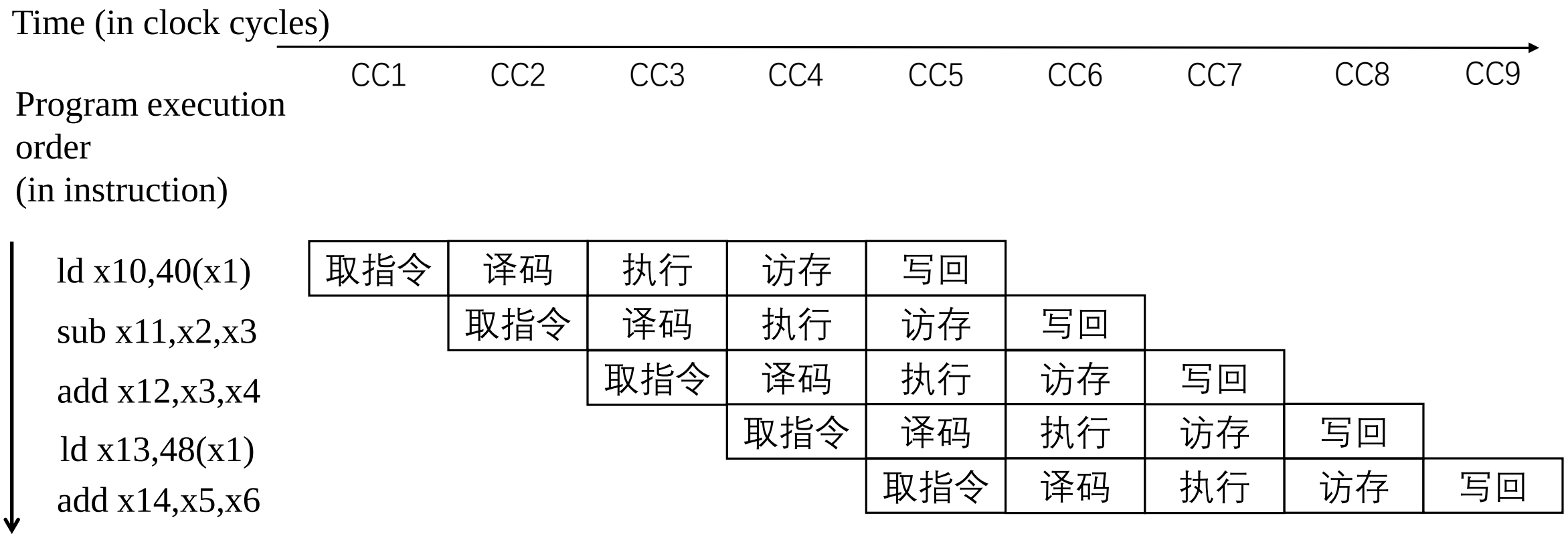
多时钟周期流水线图

- 显示了每个流水线阶段中使用的物理资源



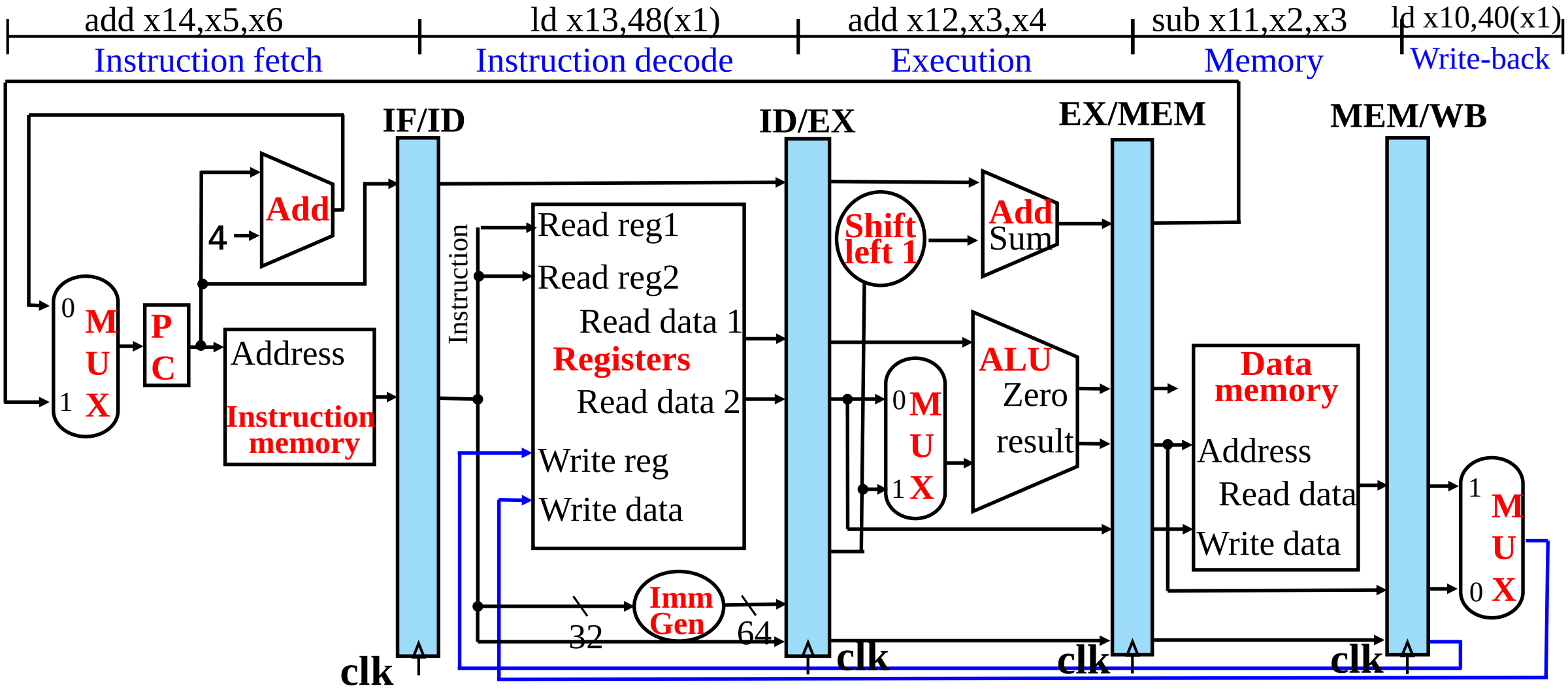
多时钟周期流水线图

用矩形块来命名每个流水线阶段

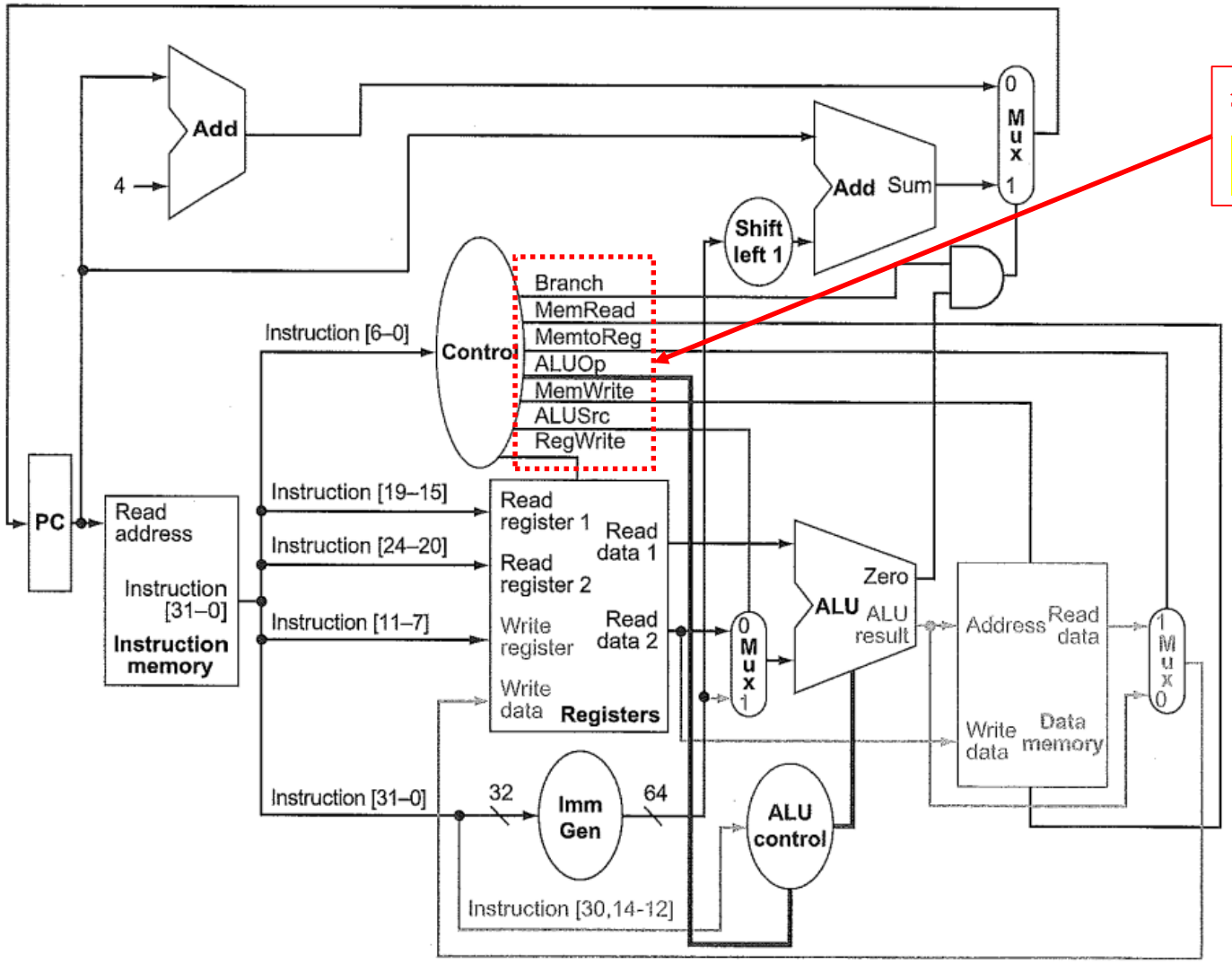


单时钟周期流水线图

- 显示了给定周期中流水线的状态
  - 单时钟周期图是多时钟周期图里一个时钟周期的垂直切片

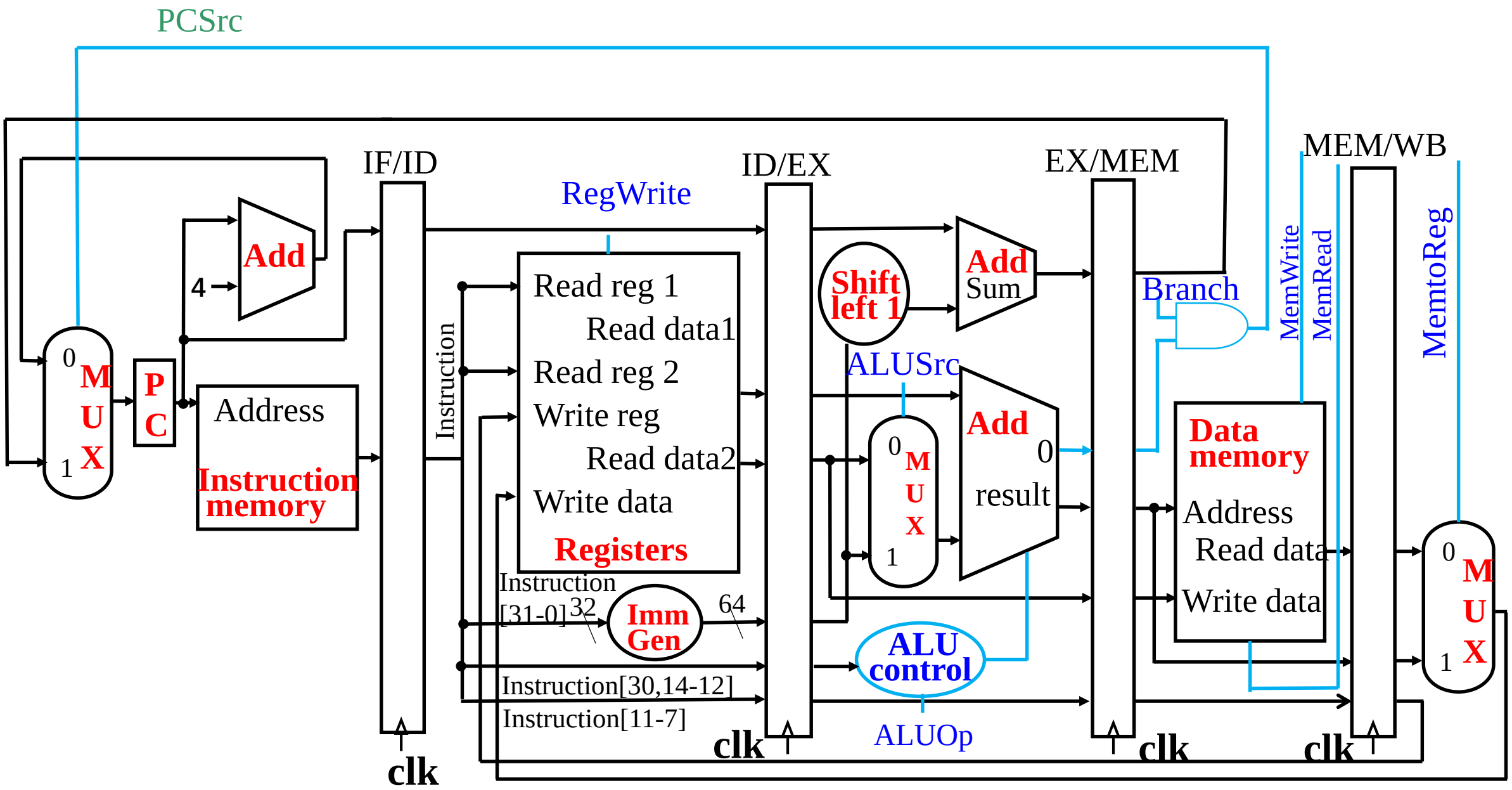


教材P182流水线CPU架构



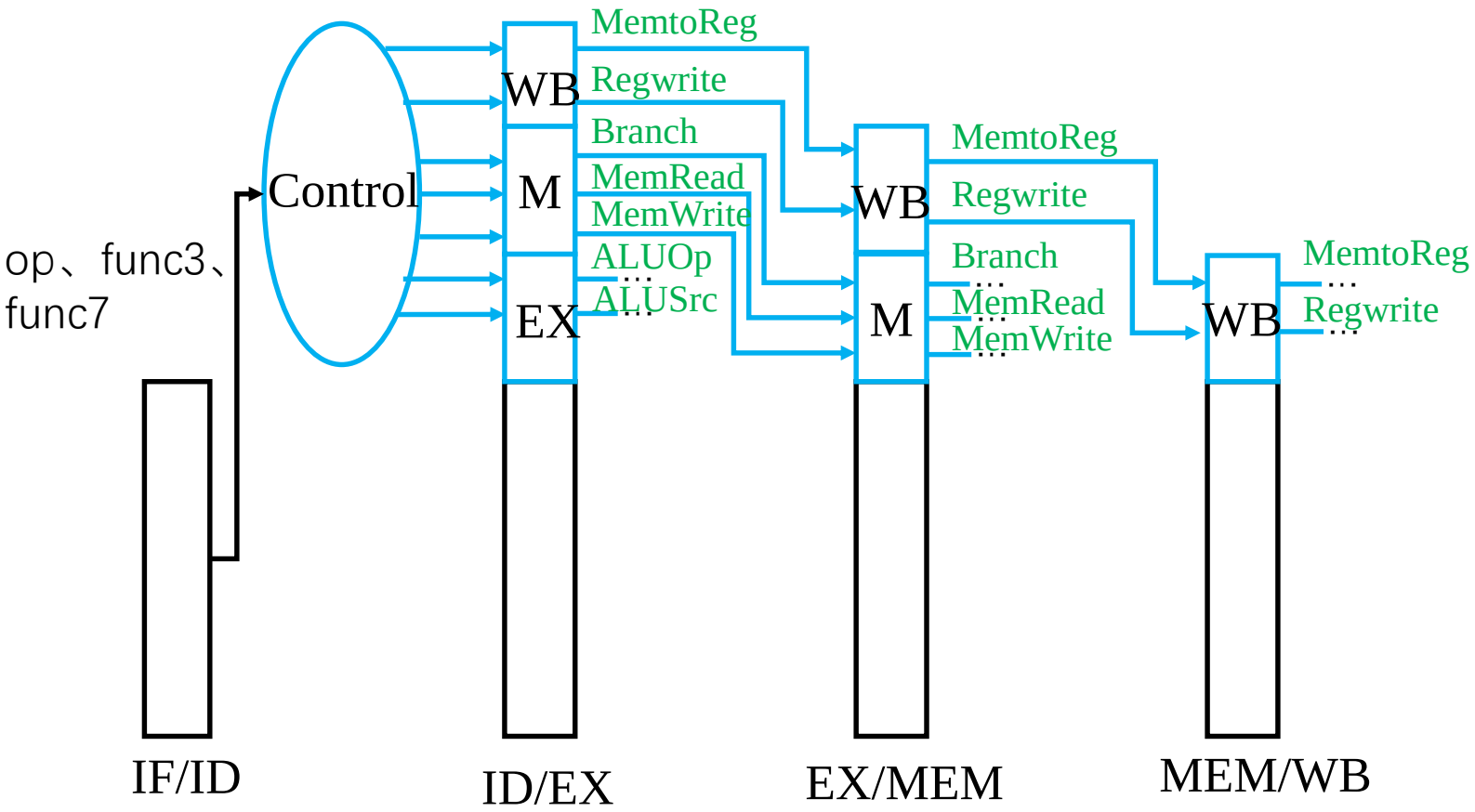
针对教材中的流水线CPU
共涉及7个控制信号

简单的流水线控制(见教材)



流水线控制

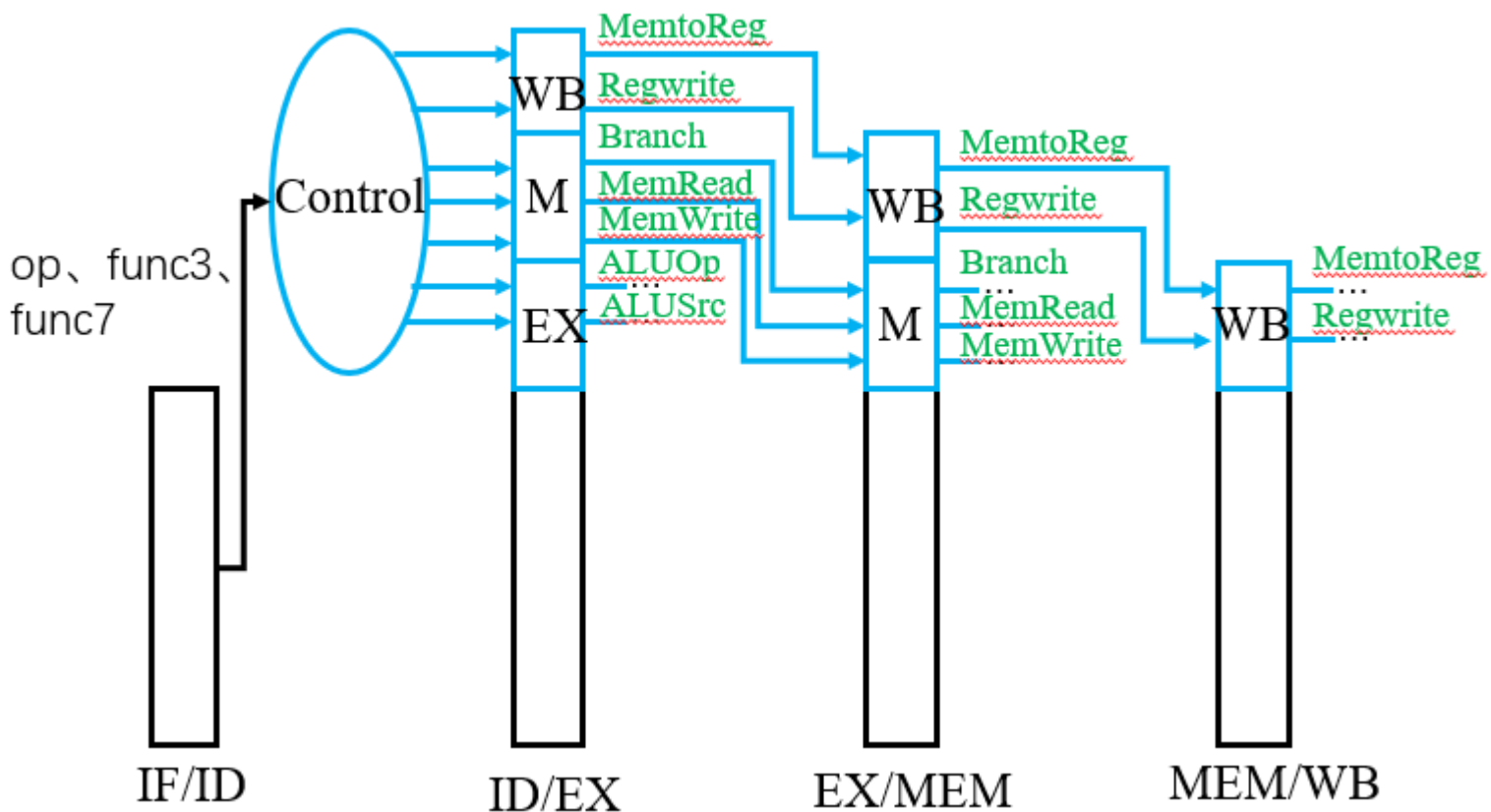
- 控制信号从指令中产生，与单周期实现相似
- 根据流水线阶段将控制线也对应进行分组
  - IF和ID阶段没有需要特别控制的内容

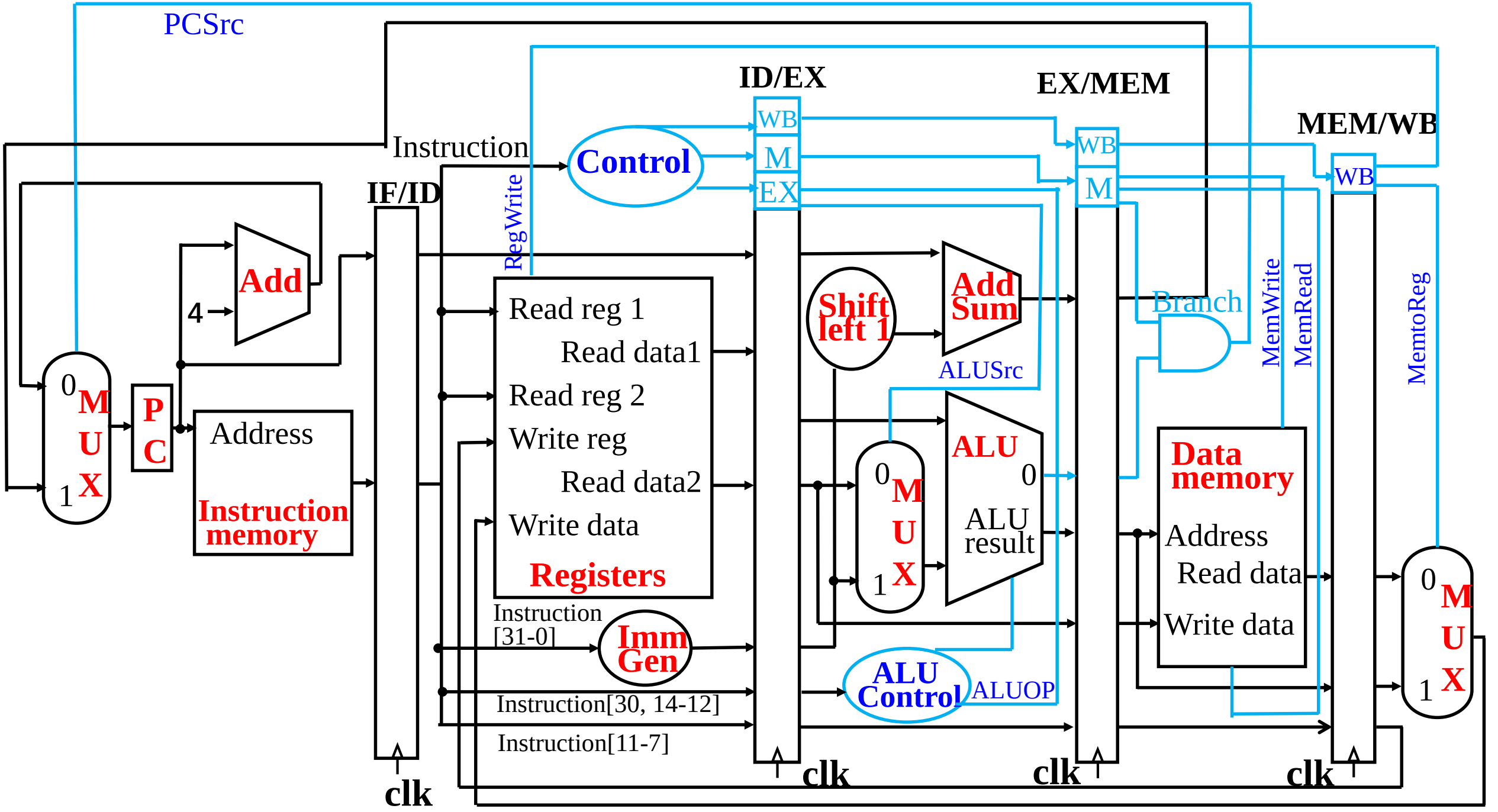


- MemtoReg选择ALU或数据存储器值写入到寄存器堆
- ALUSrc：ALU前的选择器控制信号，用于选择Reg[rs2]或立即数值
- Branch：beq指令的控制信号，相等则分支（**备注：本章所讲的数据通路对于分支指令仅实现了beq指令**）

见黑书P207页

Instruction	Execution/address calculation stage control lines		Memory access stage control lines			Write-back stage control lines	
	ALUOp	ALUSrc	Branch	Mem-Read	Mem-Write	Reg-Write	Memto-Reg
R-format	10	0	0	0	0	1	0
ld	00	1	0	1	0	1	1
sd	00	1	0	0	1	0	X
beq	01	0	1	0	0	0	X




PCSrc
ID/EX
EX/MEM
MEM/WB
Instruction
IF/ID
Control
WB
M
EX
RegWrite
Add
4
0
M
U
X
1
P
C
Address
Instruction memory
Read reg 1
Read data1
Read reg 2
Write reg
Read data2
Write data
Registers
Instruction [31-0]
Imm Gen
Instruction[30, 14-12]
Instruction[11-7]
clk
Shift left 1
Add Sum
ALUSrc
ALU
ALU result
ALU Control
ALUOP
Branch
MemWrite
MemRead
MemtoReg
Data memory
Address
Read data
Write data

第五章

- 流水线概述
- 流水线数据通路和控制
- **数据冒险：前递与停顿**
- 控制冒险
- 例外

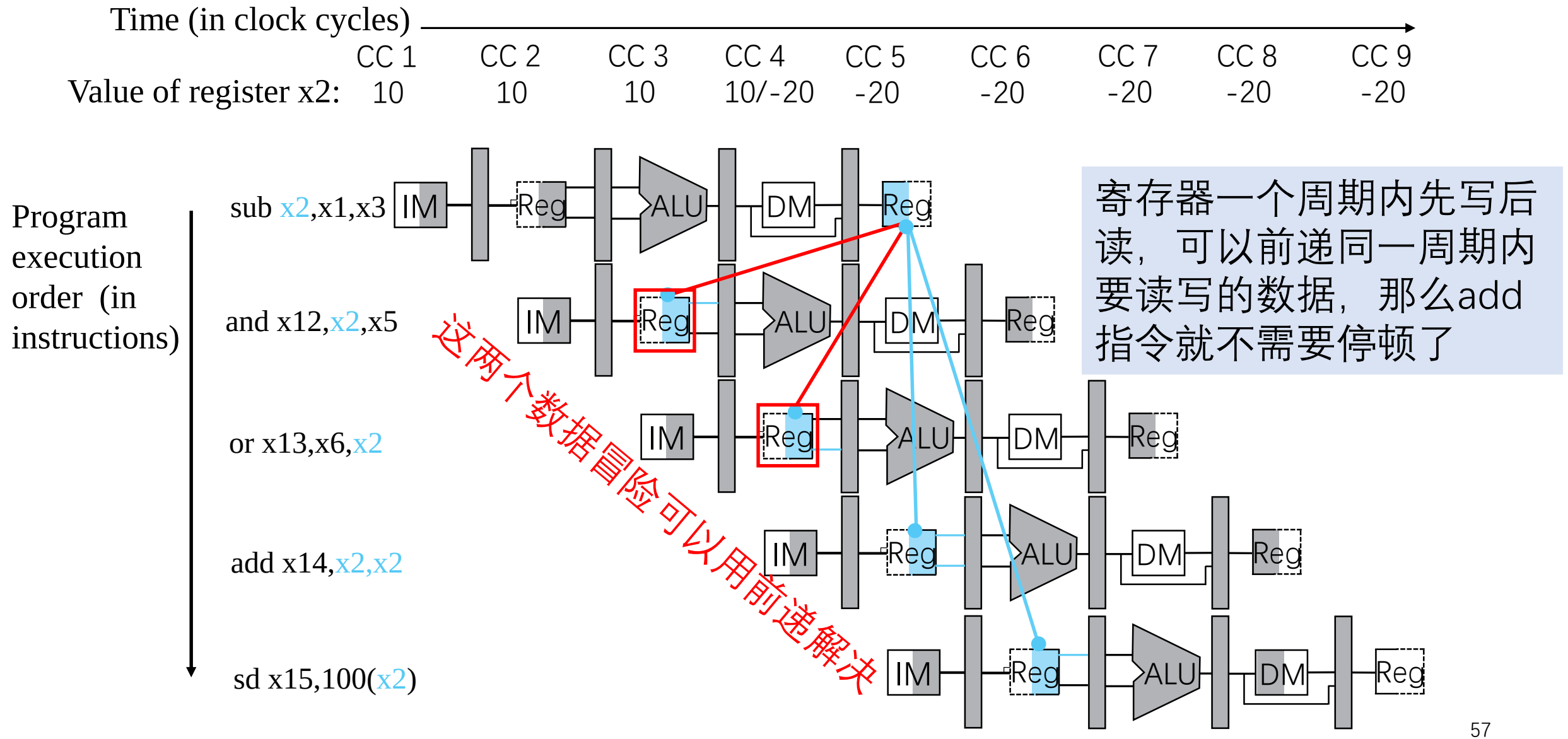
ALU相关指令中的数据冒险

- 考虑以下指令序列（**问题：都是x2，有几个冒险？**）

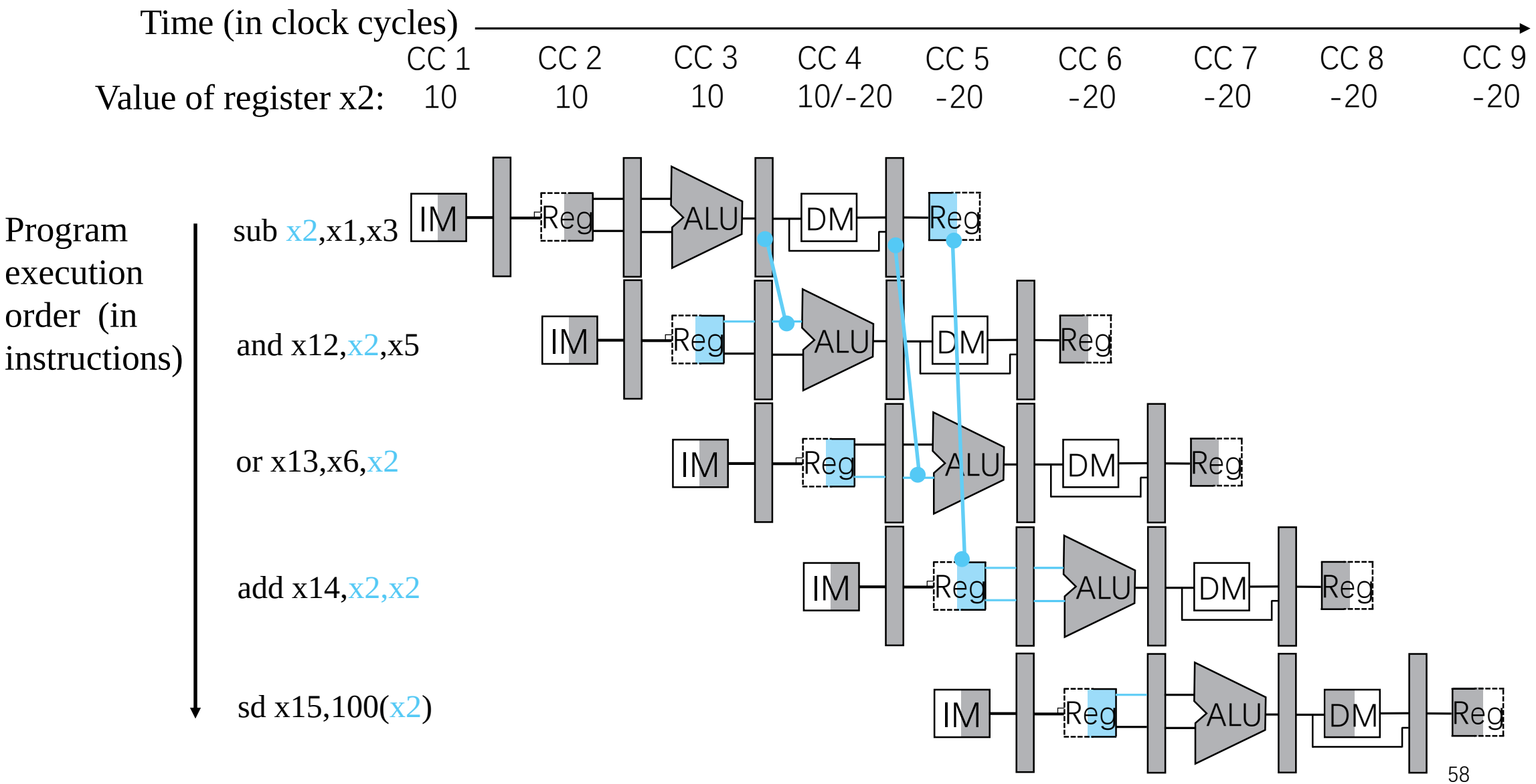
```
sub  x2,   x1, x3
and  x12, x2, x5
or    x13, x6,  x2
add  x14, x2,  x2
sd    x15, 100(x2)
```

- 我们能通过前递来解决这些冒险
  - 那么如何发现何时该前递？

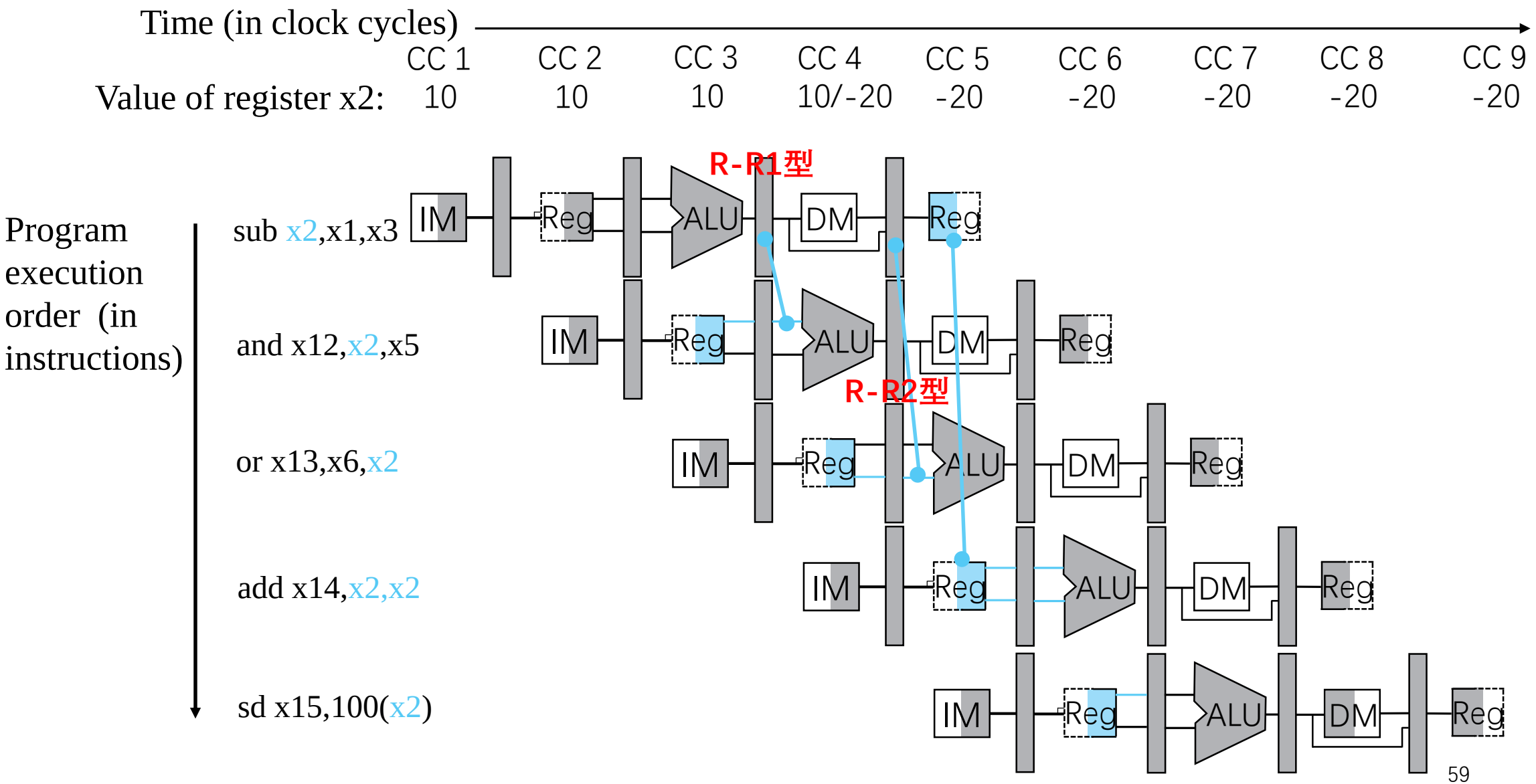
依赖关系与前递


Time (in clock cycles)
CC 1
CC 2
CC 3
CC 4
CC 5
CC 6
CC 7
CC 8
CC 9
Value of register x2:
10
10
10
10/-20
-20
-20
-20
-20
-20
Program execution order (in instructions)
sub x2,x1,x3
and x12,x2,x5
or x13,x6,x2
add x14,x2,x2
sd x15,100(x2)
IM
Reg
ALU
DM
Reg
这两个数据冒险可以用前递解决
寄存器一个周期内先写后读，可以前递同一周期内要读写的数据，那么add指令就不需要停顿了

前递的结果


Time (in clock cycles)
CC 1
CC 2
CC 3
CC 4
CC 5
CC 6
CC 7
CC 8
CC 9
Value of register x2:
10
10
10
10/-20
-20
-20
-20
-20
-20
Program execution order (in instructions)
sub x2,x1,x3
and x12,x2,x5
or x13,x6,x2
add x14,x2,x2
sd x15,100(x2)
IM
Reg
ALU
DM
Reg

前递的结果


Time (in clock cycles)
CC 1
CC 2
CC 3
CC 4
CC 5
CC 6
CC 7
CC 8
CC 9
Value of register x2:
10
10
10
10/-20
-20
-20
-20
-20
-20
Program execution order (in instructions)
sub x2,x1,x3
and x12,x2,x5
or x13,x6,x2
add x14,x2,x2
sd x15,100(x2)
IM
Reg
ALU
DM
Reg
R-R1型
R-R2型

填空题 0分

设置

此题未设置答案，请点击右侧设置按钮

问题1：

我们发现R-R类型和load-use类型这两种数据冒险都是具有相同的特点，就是RegW在被写入数据前，后面的指令就要使用该寄存器值。它们采用前递时，最初的数据来源分别是 [**ALU**] 和 [**DM**] 。

所以：R-R型与load-use两种数据冒险的主要区别：

(1)在前递最初数据来源不同；(2)处理冒险方式不同。

问题2：

R-R数据冒险具体还可以划分成两类，称为R-R1型和R-R2型。

作答

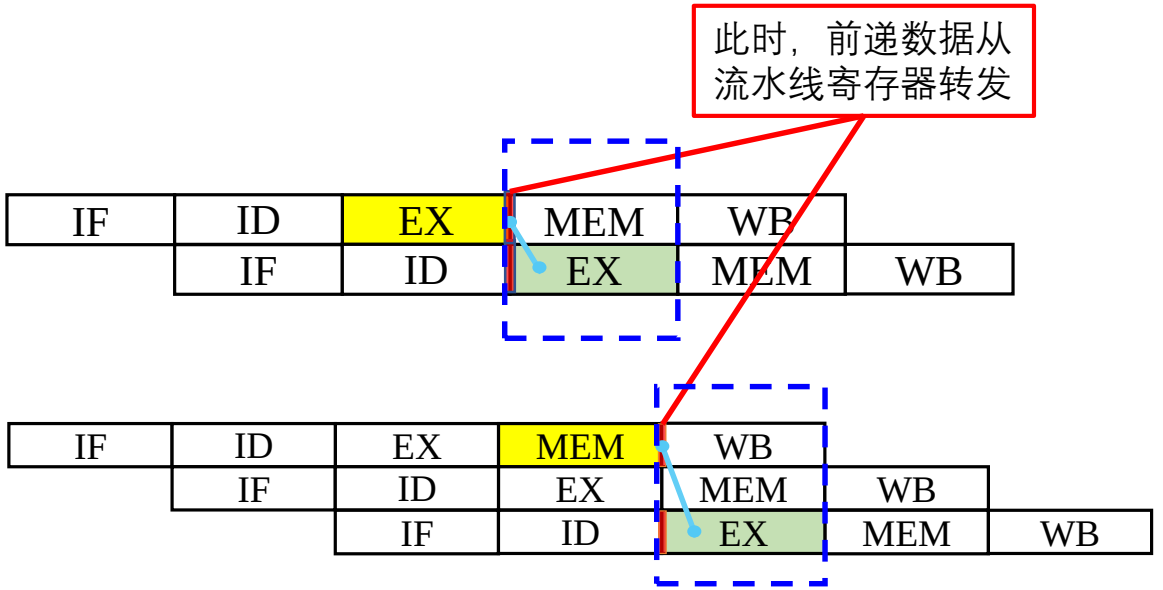
检测前递发生

- 命名流水线寄存器字段

  例如：ID/EX.RegisterRs1表示的是：一个寄存器号，它的值存在ID/EX流水线寄存器中。
  即这个寄存器堆中第一个读端口的值。

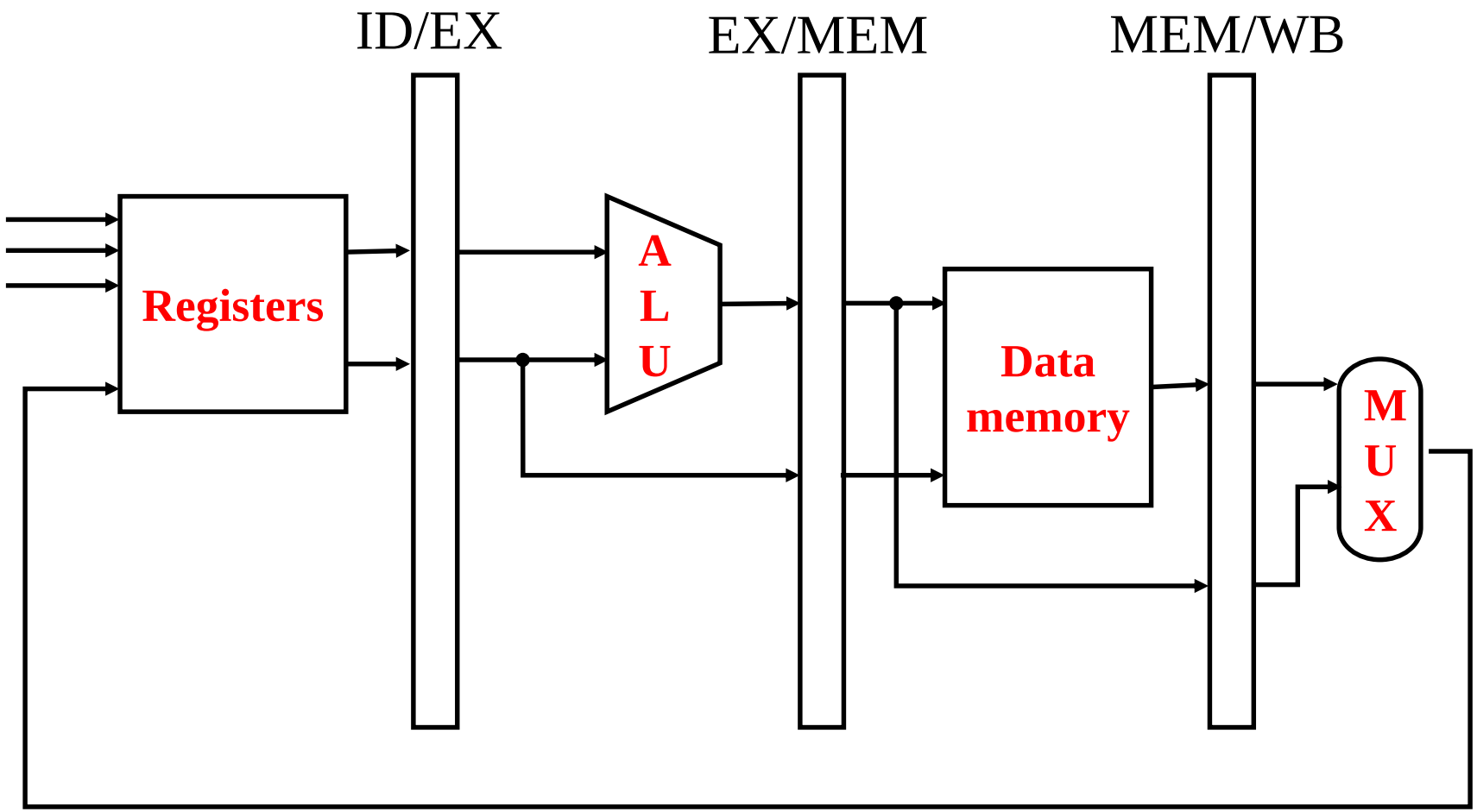
- EX阶段中，ALU操作数寄存器字段名称分别是：
  - ID/EX.RegisterRs1
  - ID/EX.RegisterRs2
- 两组数据冒险：EX冒险和MEM冒险
- 1a. EX/MEM.RegisterRd == ID/EX.RegisterRs1
- 1b. EX/MEM.RegisterRd == ID/EX.RegisterRs2

- 2a. MEM/WB.RegisterRd == ID/EX.RegisterRs1
- 2b. MEM/WB.RegisterRd == ID/EX.RegisterRs2

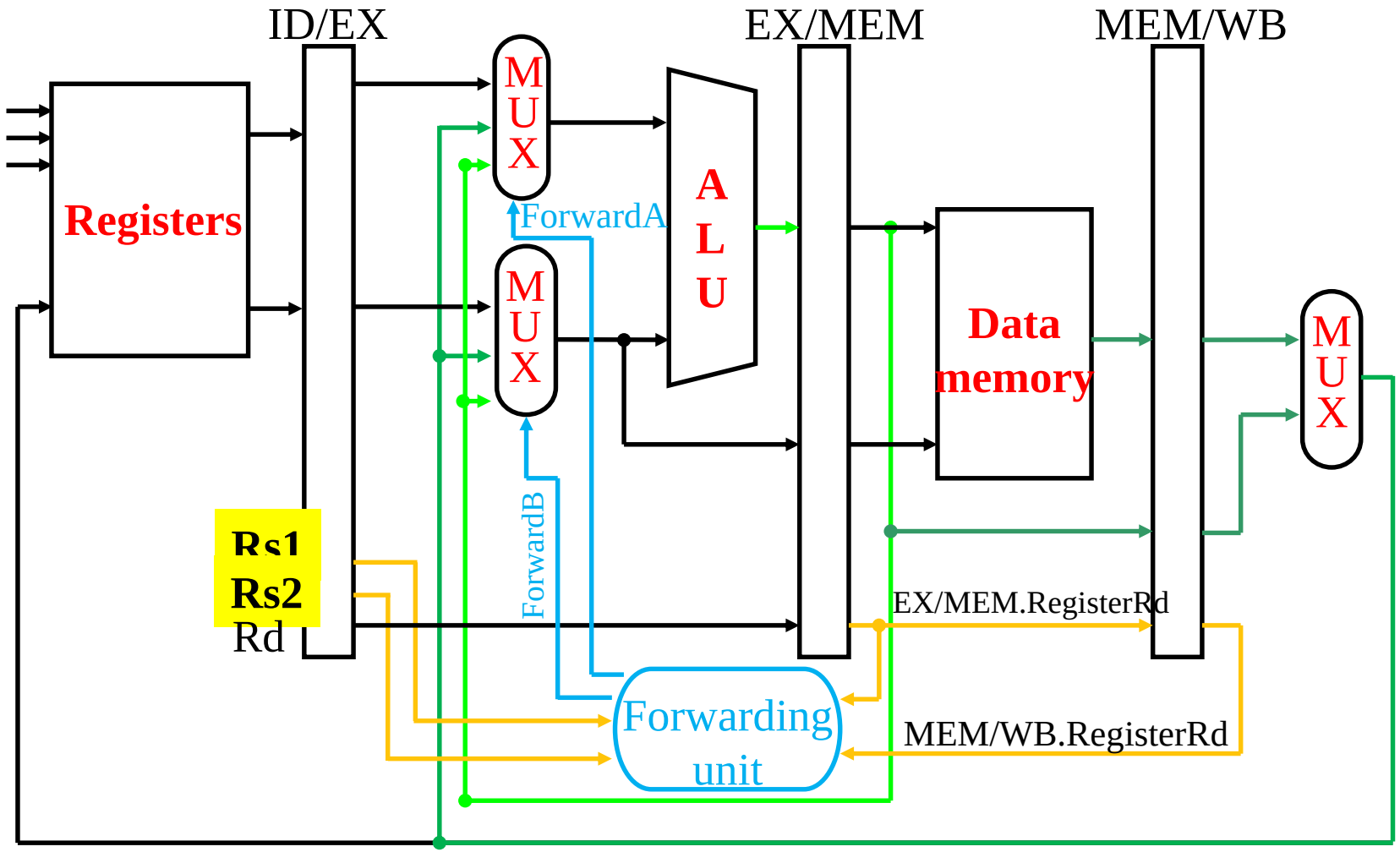


EX冒险（R-R1型）：从EX/MEM流水线寄存器中获取。
MEM冒险（R-R2型）：从MEM/WB流水线寄存器中获取。

添加前递前



添加前递后



加入前递之后，**Rs1**和**Rs2**字段也要添加到ID/EX流水线寄存器中

检测前递发生（进一步完善）

- 并不是所有指令都会写回寄存器。如果写回寄存器是x0，则不需要将数据前递出去，因此需要额外判断。
  - **EX**冒险判断（**对应R-R1型，3个条件要同时满足**）：

    EX/MEM.RegWrite and（EX/MEM.RegisterRd≠0）and

    EX/MEM.RegisterRd == ID/EX.RegisterRs
  - **MEM**冒险判断（**对应R-R2型，3个条件要同时满足**）：

    MEM/WB.RegWrite and（MEM/WB.RegisterRd≠0）and

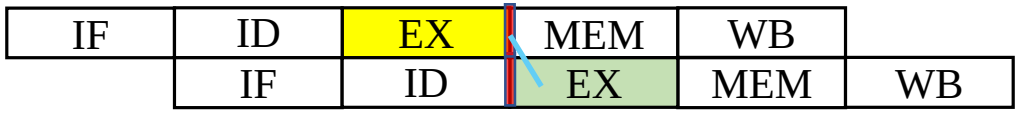
    MEM/WB.RegisterRd == ID/EX.RegisterRs

前递硬件中，多选器的控制值

Mux control	Source	Explanation
ForwardA = 00	ID/EX	ALU的第一个操作数来自寄存器堆
ForwardA = 10	EX/MEM	ALU的第一个操作数来自上一个ALU计算结果的前递
ForwardA = 01	MEM/WB	ALU的第一个操作数来自数据存储器或者更早的ALU计算结果的前递
ForwardB = 00	ID/EX	ALU的第二个操作数来自寄存器堆
ForwardB = 10	EX/MEM	ALU的第二个操作数来自上一个ALU计算结果的前递
ForwardB = 01	MEM/WB	ALU的第二个操作数来自数据存储器或者更早的ALU计算结果的前递

检测EX冒险的条件、相应的前递控制

- EX冒险（同时满足3个条件）



- if (EX/MEM.RegWrite
  and (EX/MEM.RegisterRd ≠ 0)
  and (EX/MEM.RegisterRd==ID/EX.RegisterRs1))

  则执行 ForwardA = 10

- if (EX/MEM.RegWrite
  and (EX/MEM.RegisterRd ≠ 0)
  and (EX/MEM.RegisterRd==ID/EX.RegisterRs2))

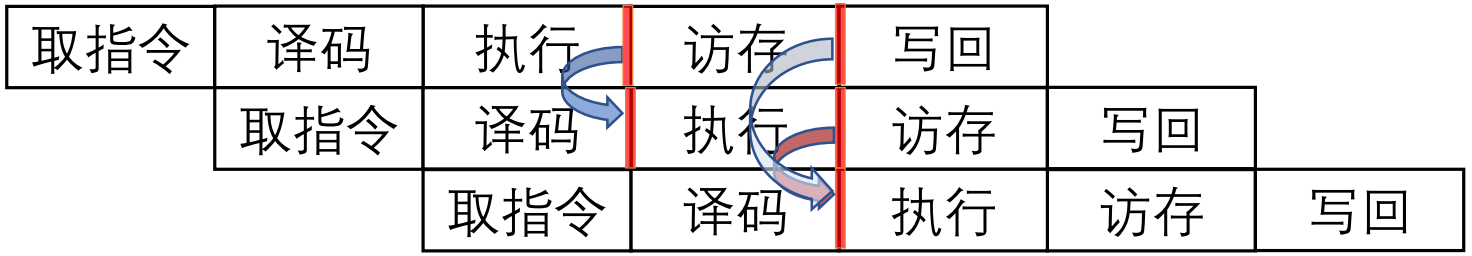
  则执行 ForwardB = 10

EX冒险：从EX/MEM流水线寄存器中获取（R-R1型）

MEM冒险检测仍有问题：双重数据冒险

例子： 考虑以下指令序列：

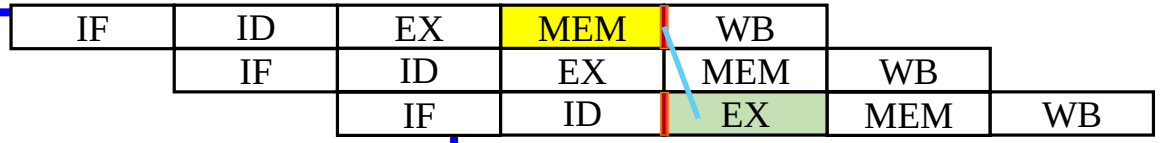
```
add x1,x1,x2
add x1,x1,x3
add x1,x1,x4
```



- 双重数据冒险都发生
  - 对于第三条指令而言，应该使用最近的结果
- **MEM冒险前递的条件：只有EX冒险不发生的时候才前递**

检测MEM冒险的条件、相应的前递控制

- MEM 冒险（同时满足6个条件，其中包括了EX的3个）



- **if** (MEM/WB.RegWrite
  and (MEM/WB.RegisterRd ≠ 0)
  and not(EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
  and (EX/MEM.RegisterRd==ID/EX.RegisterRs1))
  and (MEM/WB.RegisterRd==ID/EX.RegisterRs1))

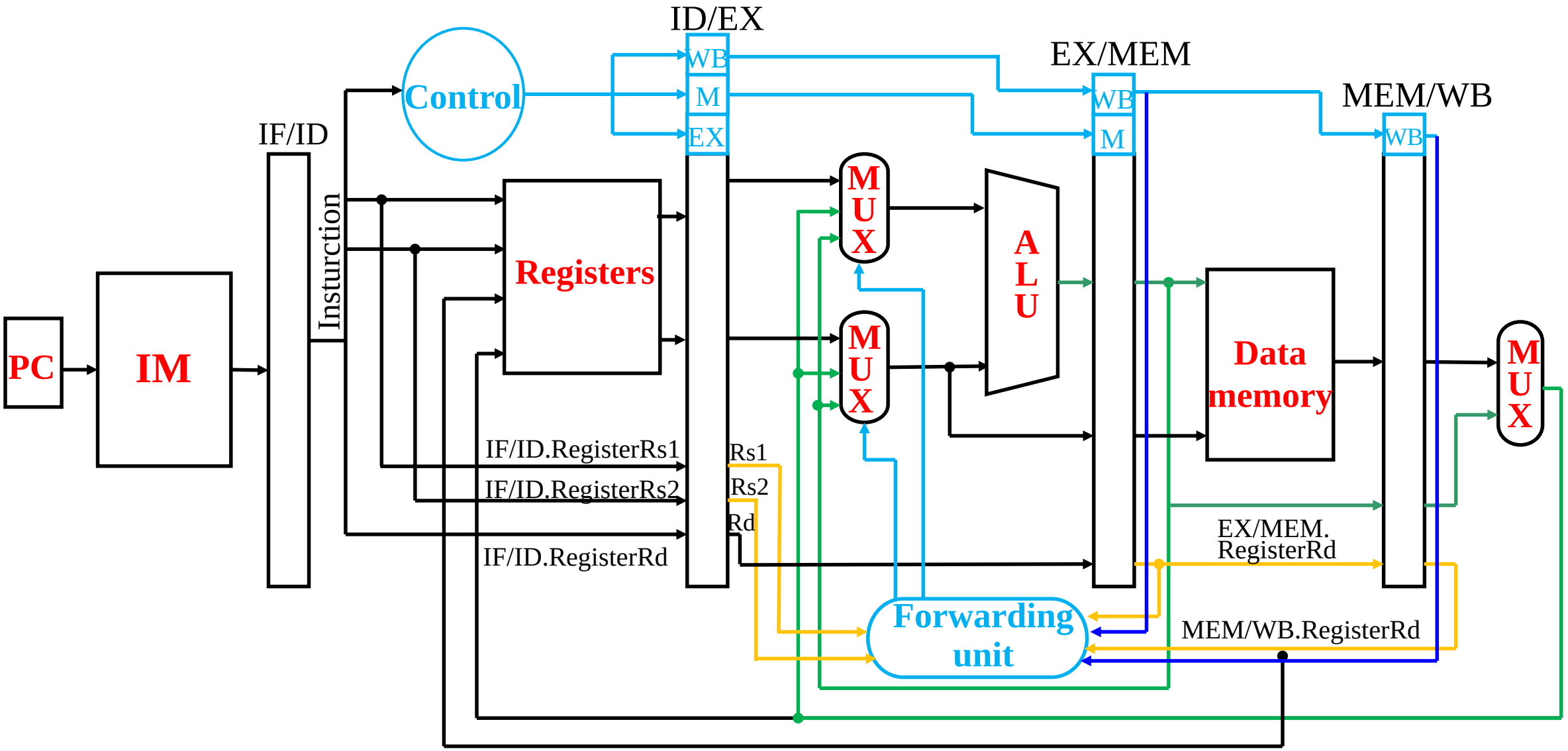
则执行
ForwardA = 01

- **if** (MEM/WB.RegWrite
  and (MEM/WB.RegisterRd ≠ 0)
  and not(EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
  and (EX/MEM.RegisterRd==ID/EX.RegisterRs2))
  and (MEM/WB.RegisterRd==ID/EX.RegisterRs2))

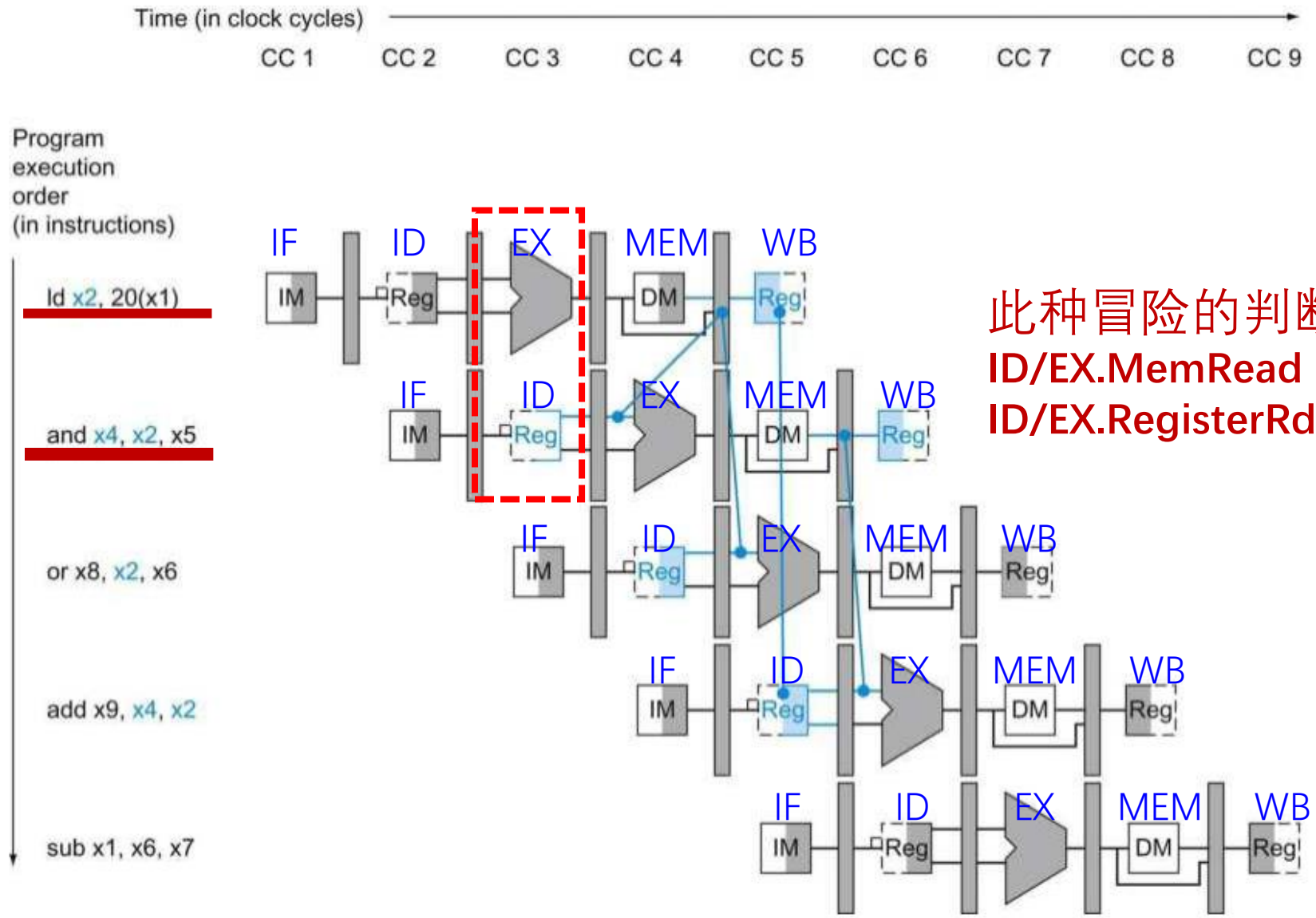
则执行
ForwardB = 01

MEM冒险：从MEM/WB流水线寄存器中获取（R-R2型）

通过前递解决冒险的数据通路



载入-使用型数据冒险



此种冒险的判断条件:

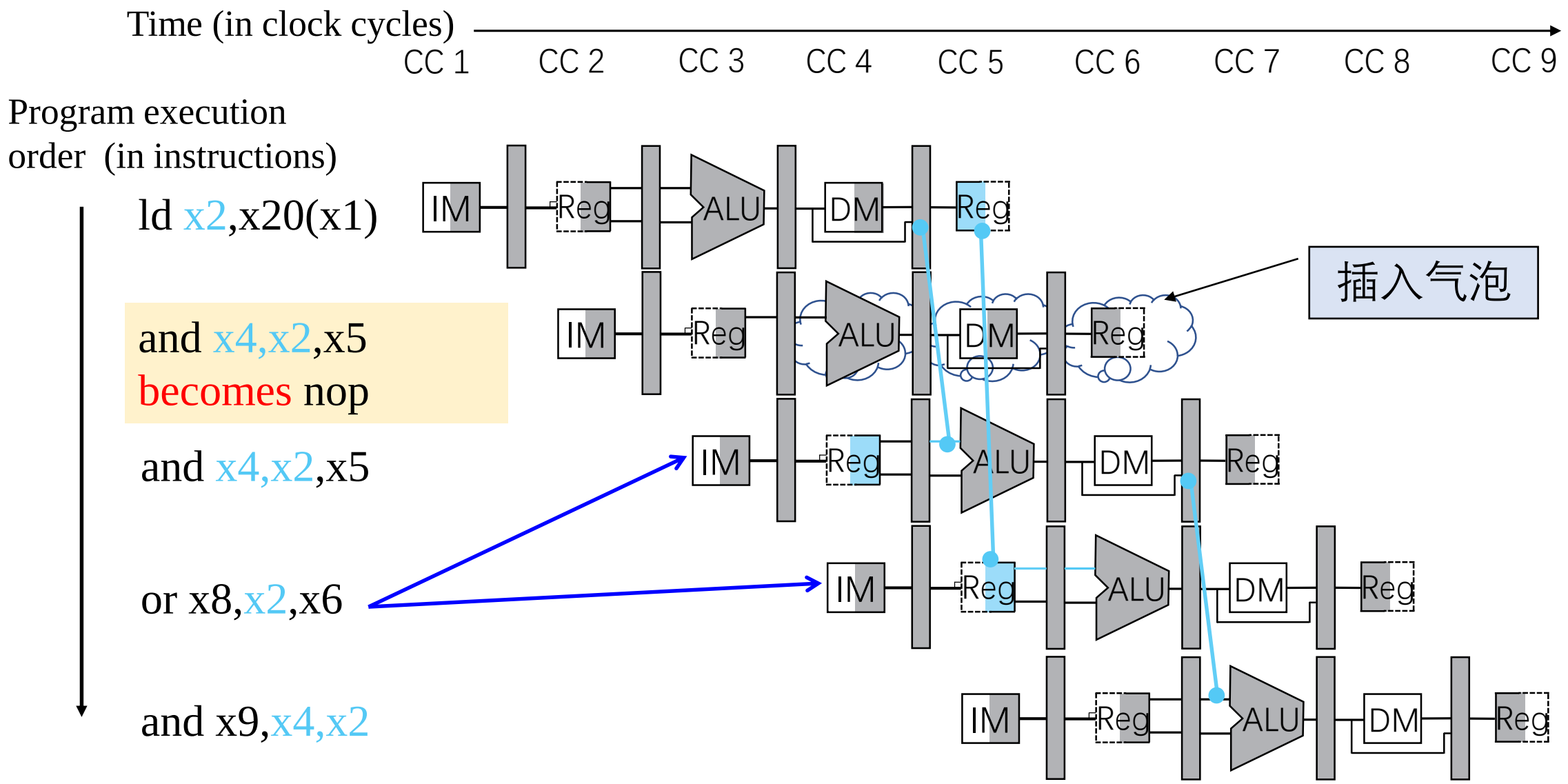
ID/EX.MemRead and
ID/EX.RegisterRd = IF/ID.RegisterRs

检测 载入-使用型数据冒险

- 任何指令进入到**ID阶段**都需要进行检测load-use是否发生
- ALU操作数寄存器字段名称分别是：
  - IF/ID.RegisterRs1, IF/ID.RegisterRs2
- 检测条件
  - ID/EX.MemRead and
    ((ID/EX.RegisterRd == IF/ID.RegisterRs1) or
    (ID/EX.RegisterRd == IF/ID.RegisterRs2))
- **如果检测到这种冒险，停顿流水线（插入气泡）**

还有一种数据冒险，存储指令紧跟在载入指令之后的情况（ld…, sd…）
例题: ld x10, 0(x11)，然后是sd x10, 8(x11), 相当于A[i+1]=A[i]
是否有冒险，是什么冒险？如何解决？ 答：数据冒险(M-M)，前递

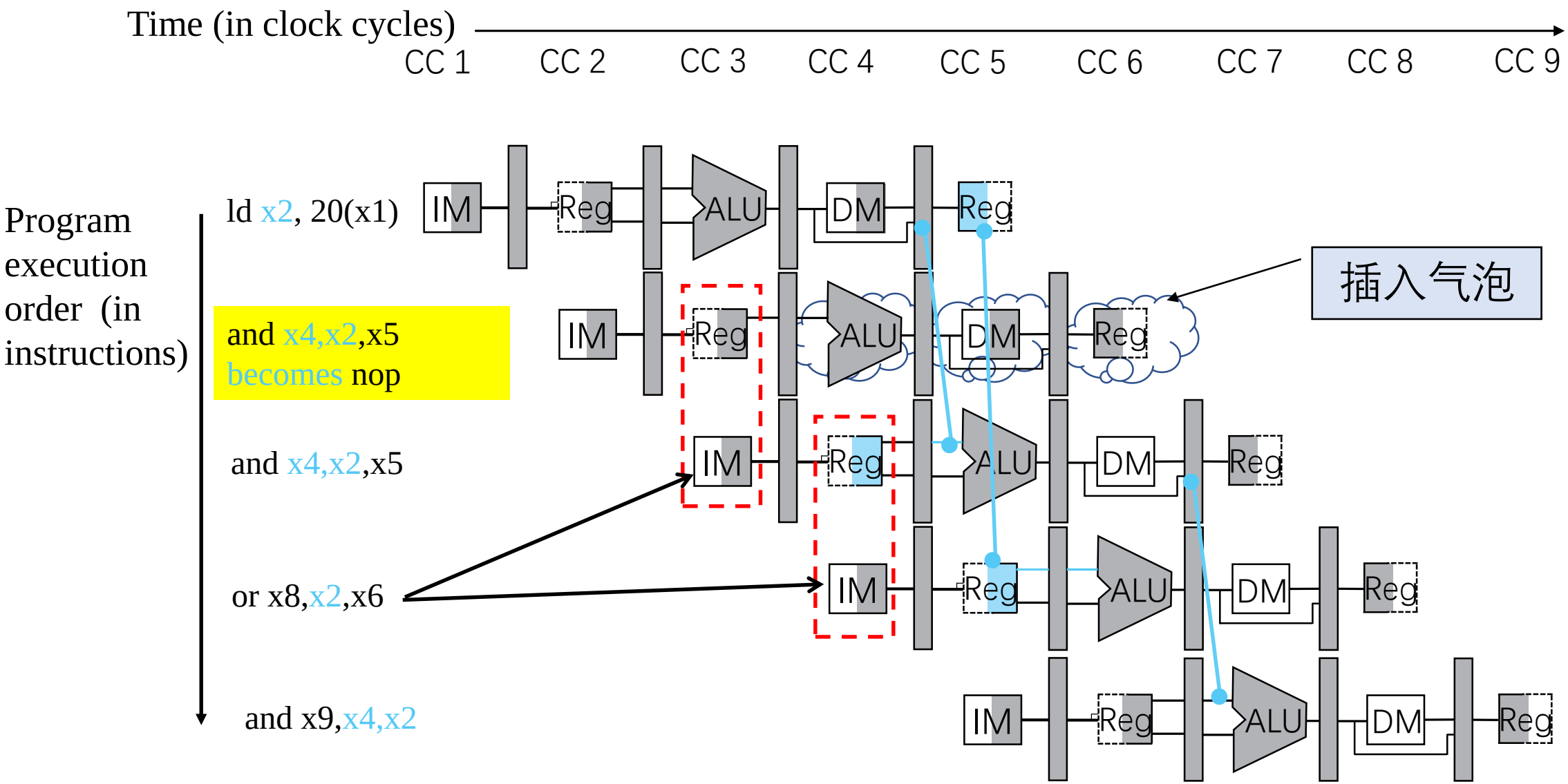
载入–使用型数据冒险


Time (in clock cycles)
CC 1
CC 2
CC 3
CC 4
CC 5
CC 6
CC 7
CC 8
CC 9
Program execution order (in instructions)
ld x2,x20(x1)
and x4,x2,x5 becomes nop
and x4,x2,x5
or x8,x2,x6
and x9,x4,x2
IM
Reg
ALU
DM
Reg
插入气泡

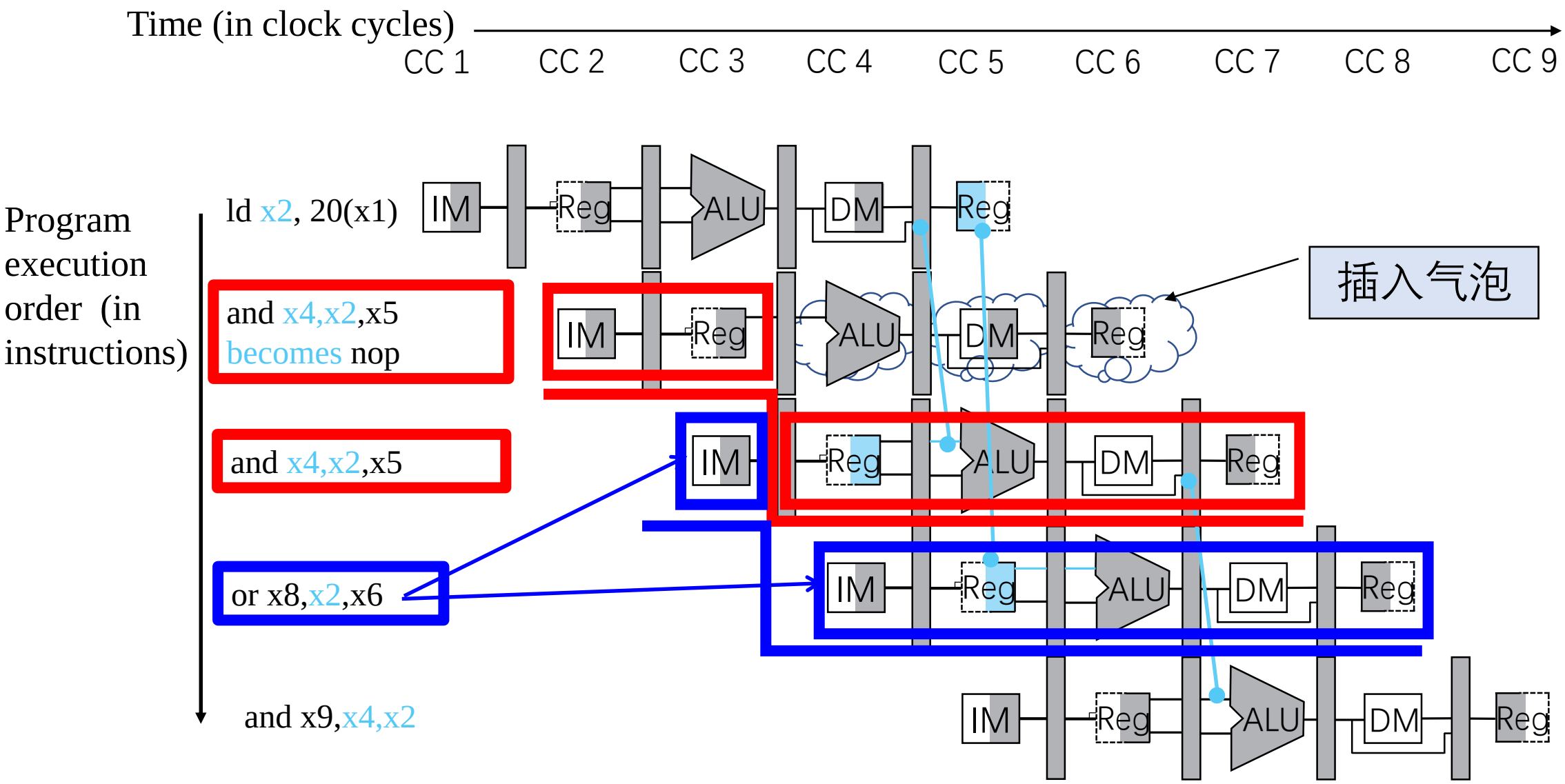
如何停顿流水线?

- 被停顿的指令正处于ID阶段
- 将ID/EX寄存器中控制信号全置为0(RegWrite, MemWrite...)
  - 被停顿的指令在EX, MEM, WB 阶段执行空指令 nop
  - 在控制值均为0时，不会有寄存器或者存储器被写入数据
- 禁止PC寄存器和IF/ID寄存器内容发生改变（寄存器加写使能）
  - ID阶段的寄存器会继续使用IF/ID寄存器中相同字段读寄存器
  - IF阶段的PC寄存器内容不变，下一条指令会重新取指
  - 1个时钟周期的停顿，能够让ld指令的MEM阶段完成
    - 就可以把取到的数据前递到EX阶段

载入–使用型数据冒险

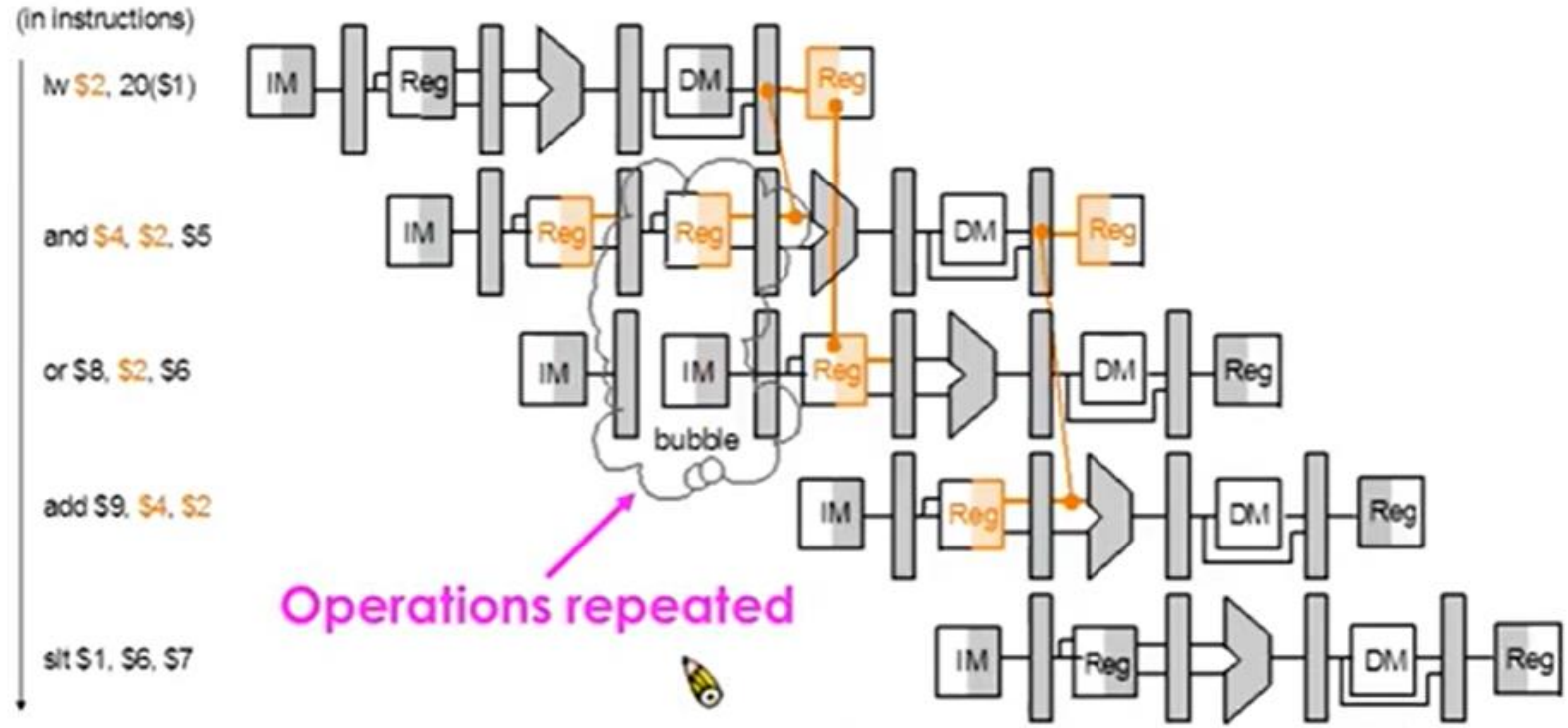

Time (in clock cycles)
CC 1
CC 2
CC 3
CC 4
CC 5
CC 6
CC 7
CC 8
CC 9
Program execution order (in instructions)
ld x2, 20(x1)
and x4,x2,x5 becomes nop
and x4,x2,x5
or x8,x2,x6
and x9,x4,x2
IM
Reg
ALU
DM
Reg
插入气泡

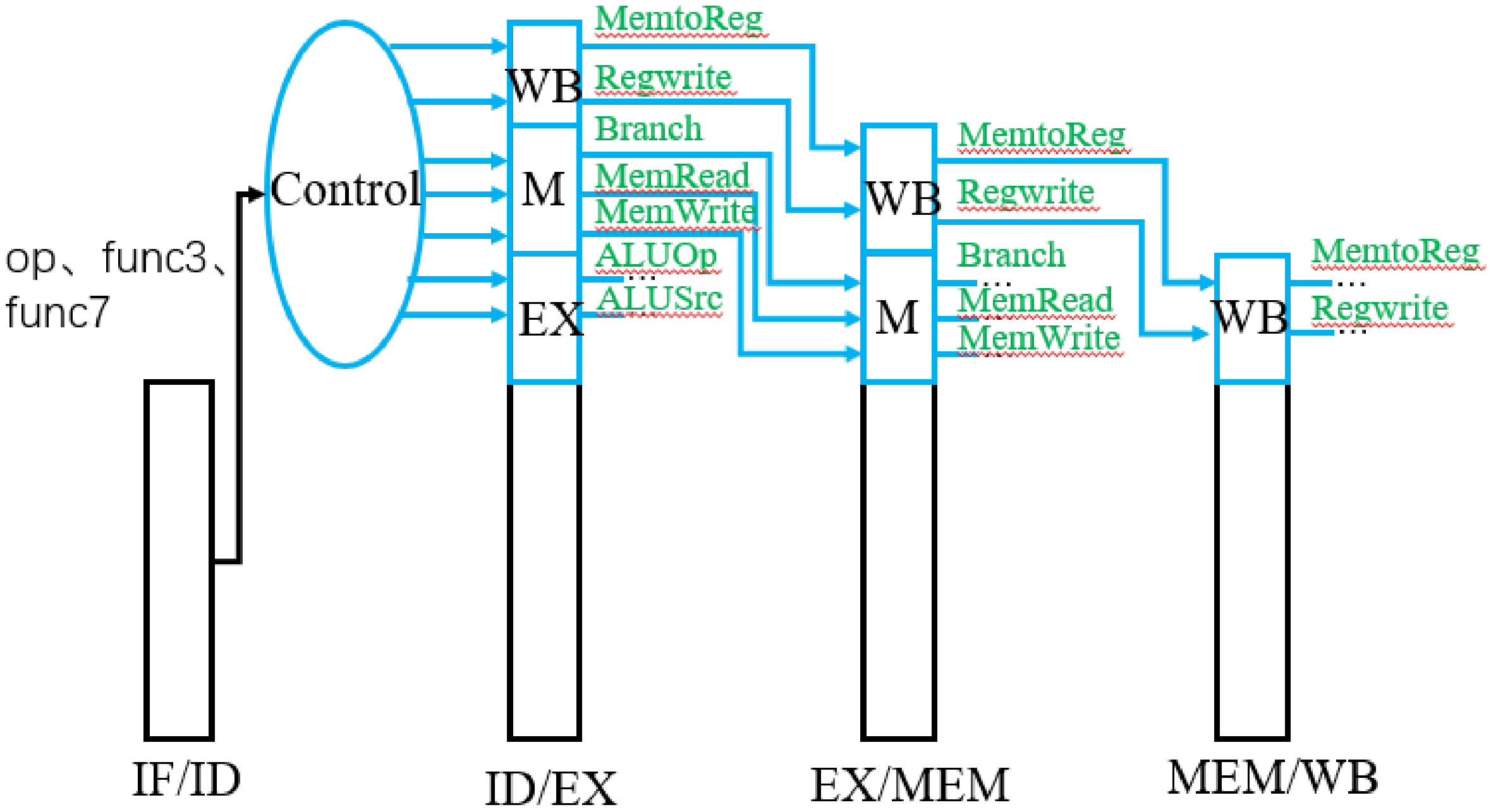
载入–使用型数据冒险

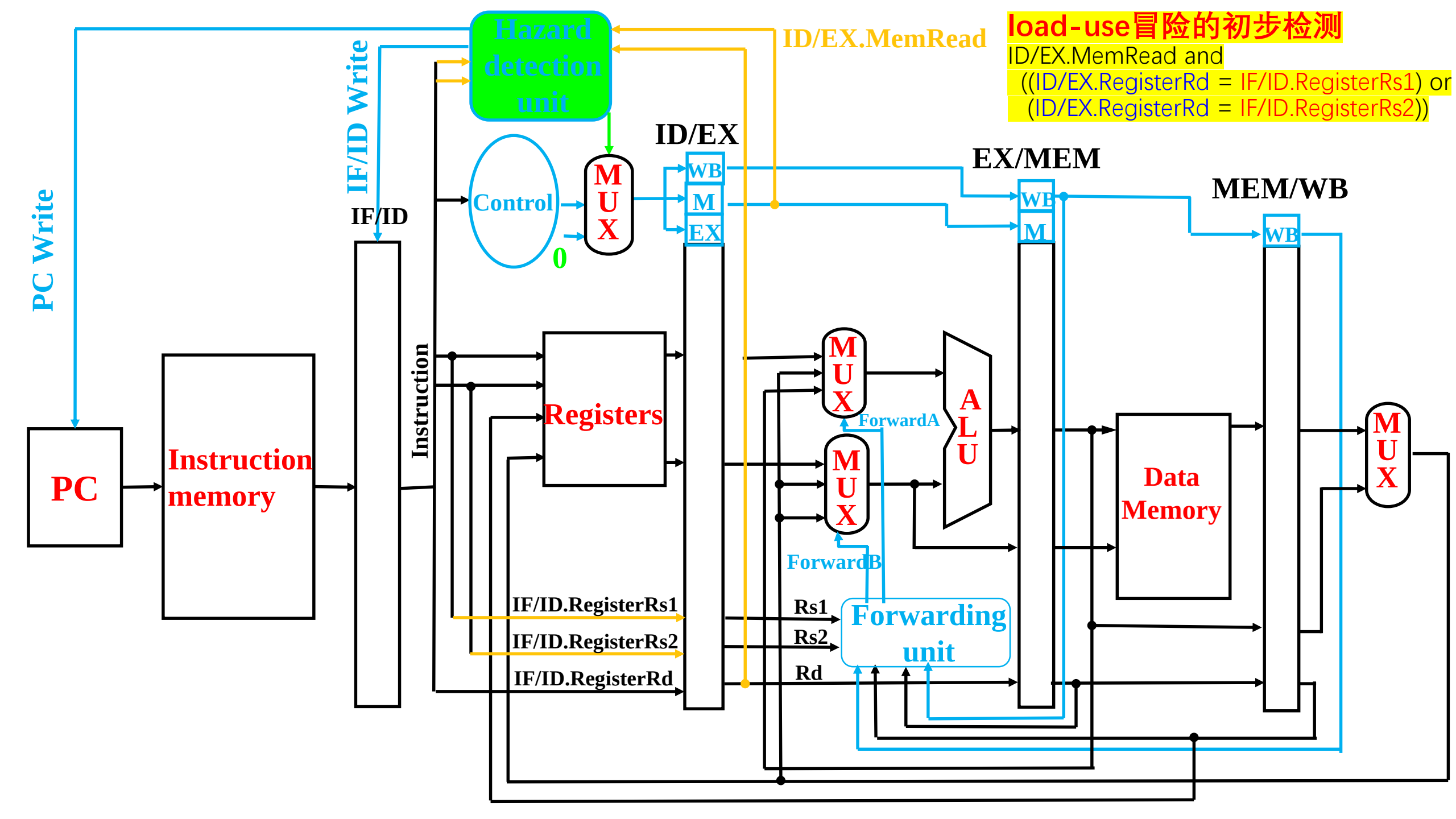

Time (in clock cycles)
CC 1
CC 2
CC 3
CC 4
CC 5
CC 6
CC 7
CC 8
CC 9
Program execution order (in instructions)
ld x2, 20(x1)
and x4,x2,x5 becomes nop
and x4,x2,x5
or x8,x2,x6
and x9,x4,x2
IM
Reg
ALU
DM
Reg
插入气泡

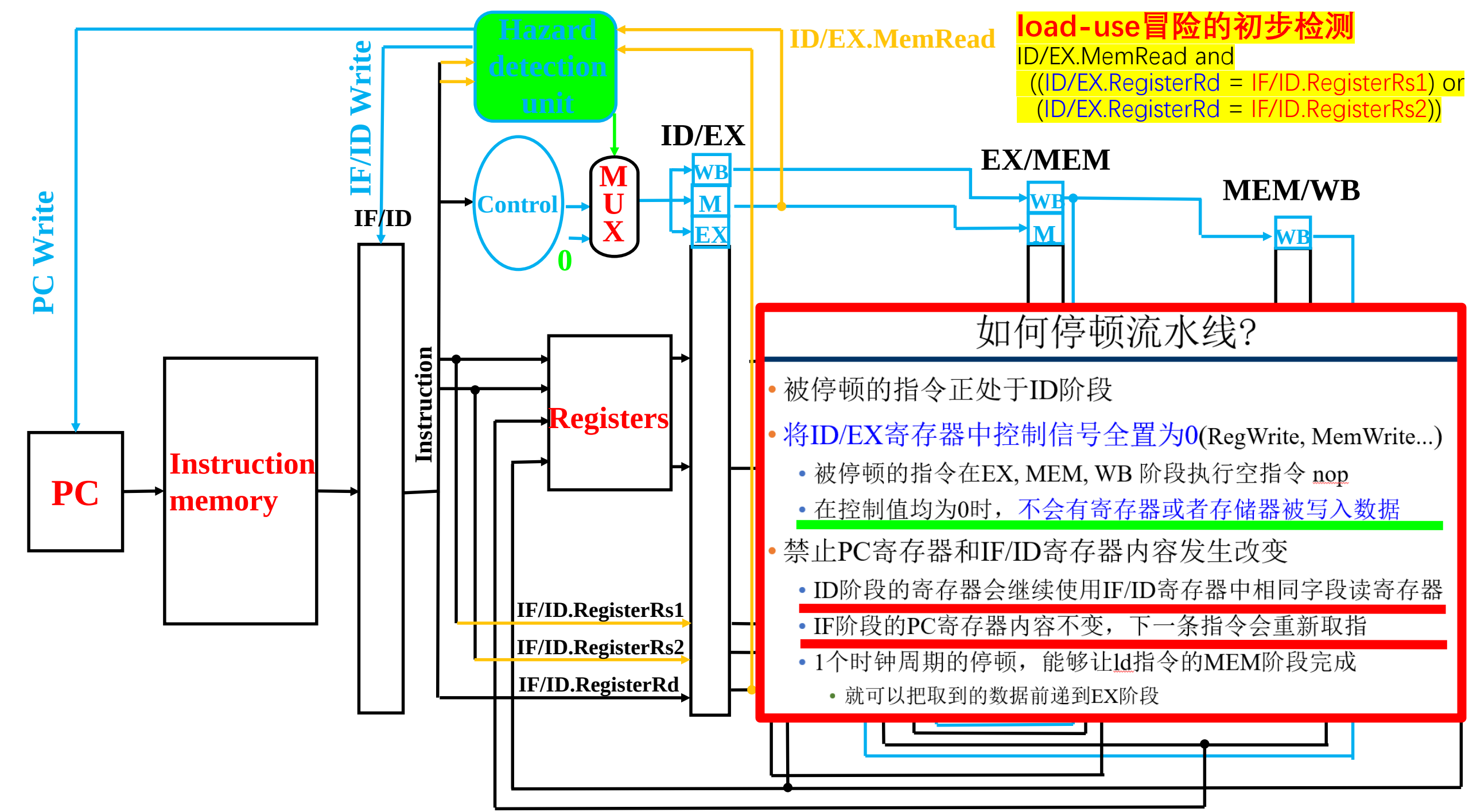
Stalling

- Stall pipeline by keeping instructions in same stage and inserting an NOP instead

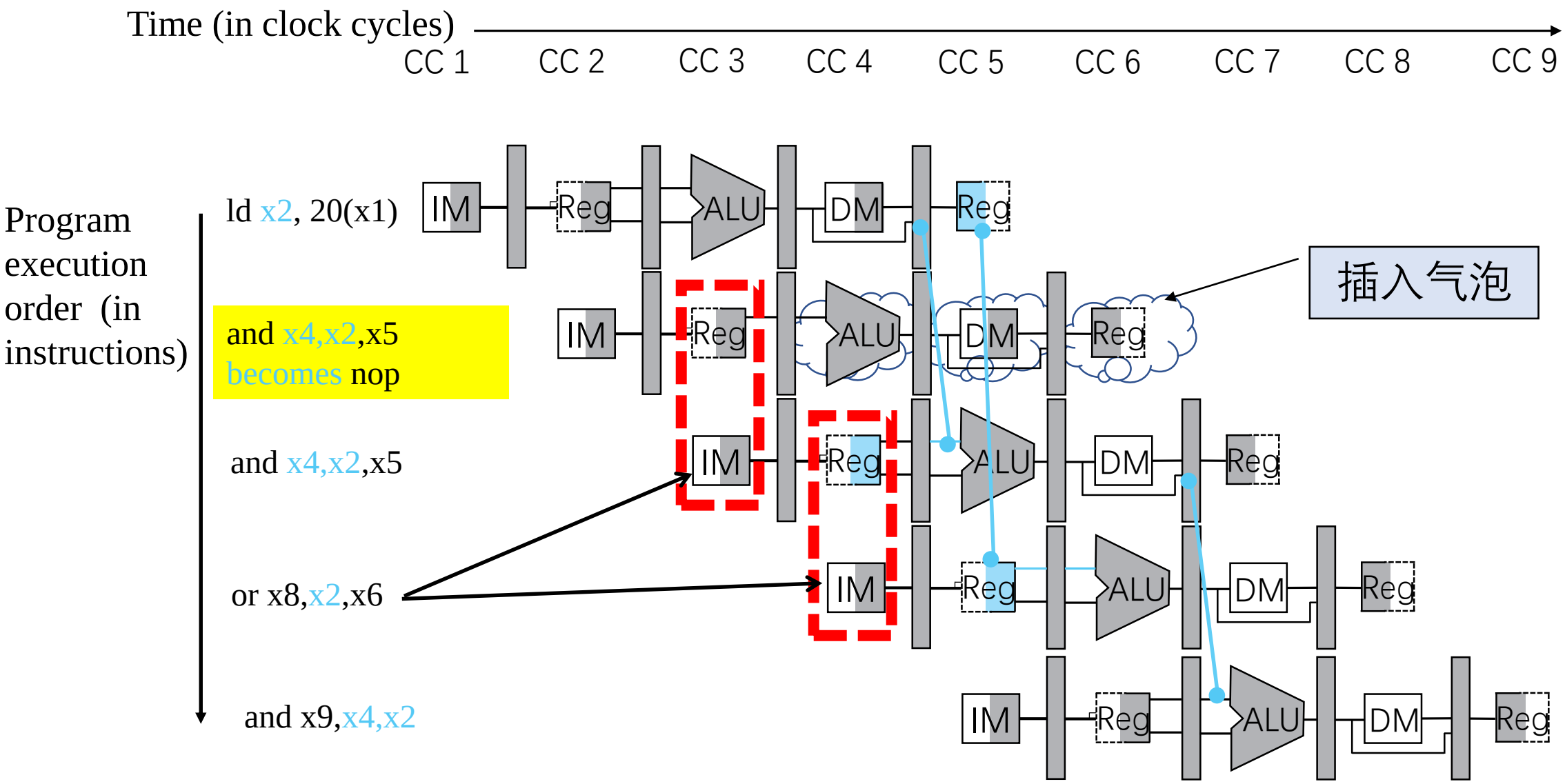



op、func3、
func7
Control
WB
M
EX
MemtoReg
Regwrite
Branch
MemRead
MemWrite
ALUOp
…
ALUSrc
WB
M
MemtoReg
Regwrite
Branch
…
MemRead
MemWrite
WB
MemtoReg
…
Regwrite
…
IF/ID
ID/EX
EX/MEM
MEM/WB


load-use冒险的初步检测
ID/EX.MemRead and
((ID/EX.RegisterRd = IF/ID.RegisterRs1) or
(ID/EX.RegisterRd = IF/ID.RegisterRs2))
Hazard detection unit
ID/EX.MemRead
IF/ID Write
PC Write
Control
0
MUX
ID/EX
WB
M
EX
EX/MEM
MEM/WB
IF/ID
Instruction
PC
Instruction memory
Registers
ALU
ForwardA
ForwardB
Data Memory
IF/ID.RegisterRs1
IF/ID.RegisterRs2
IF/ID.RegisterRd
Rs1
Rs2
Rd
Forwarding unit


load-use冒险的初步检测
ID/EX.MemRead and
((ID/EX.RegisterRd = IF/ID.RegisterRs1) or
(ID/EX.RegisterRd = IF/ID.RegisterRs2))
Hazard detection unit
ID/EX.MemRead
IF/ID Write
PC Write
Control
0
M U X
ID/EX
WB
M
EX
EX/MEM
MEM/WB
IF/ID
Instruction
PC
Instruction memory
Registers
IF/ID.RegisterRs1
IF/ID.RegisterRs2
IF/ID.RegisterRd
如何停顿流水线?
被停顿的指令正处于ID阶段
将ID/EX寄存器中控制信号全置为0(RegWrite, MemWrite...)
被停顿的指令在EX, MEM, WB 阶段执行空指令 nop
在控制值均为0时，不会有寄存器或者存储器被写入数据
禁止PC寄存器和IF/ID寄存器内容发生改变
ID阶段的寄存器会继续使用IF/ID寄存器中相同字段读寄存器
IF阶段的PC寄存器内容不变，下一条指令会重新取指
1个时钟周期的停顿，能够让ld指令的MEM阶段完成
就可以把取到的数据前递到EX阶段

载入–使用型数据冒险


Time (in clock cycles)
CC 1
CC 2
CC 3
CC 4
CC 5
CC 6
CC 7
CC 8
CC 9
Program execution order (in instructions)
ld x2, 20(x1)
and x4,x2,x5 becomes nop
and x4,x2,x5
or x8,x2,x6
and x9,x4,x2
IM
Reg
ALU
DM
Reg
插入气泡

停顿与性能

- 停顿会导致性能下降
  - 但为了得到正确的结果，需要停顿

- 编译器能通过**重排代码**，尽量避免冒险和停顿
  - 这需要流水线结构的知识

1、通常来说，一条新进入的指令需要在流水线数据通路的[ ]阶段检测是否是load-use冒险（hazard），即检测是否需要停顿。
（从IF、ID、EX、MEM、WB选择一个填空）

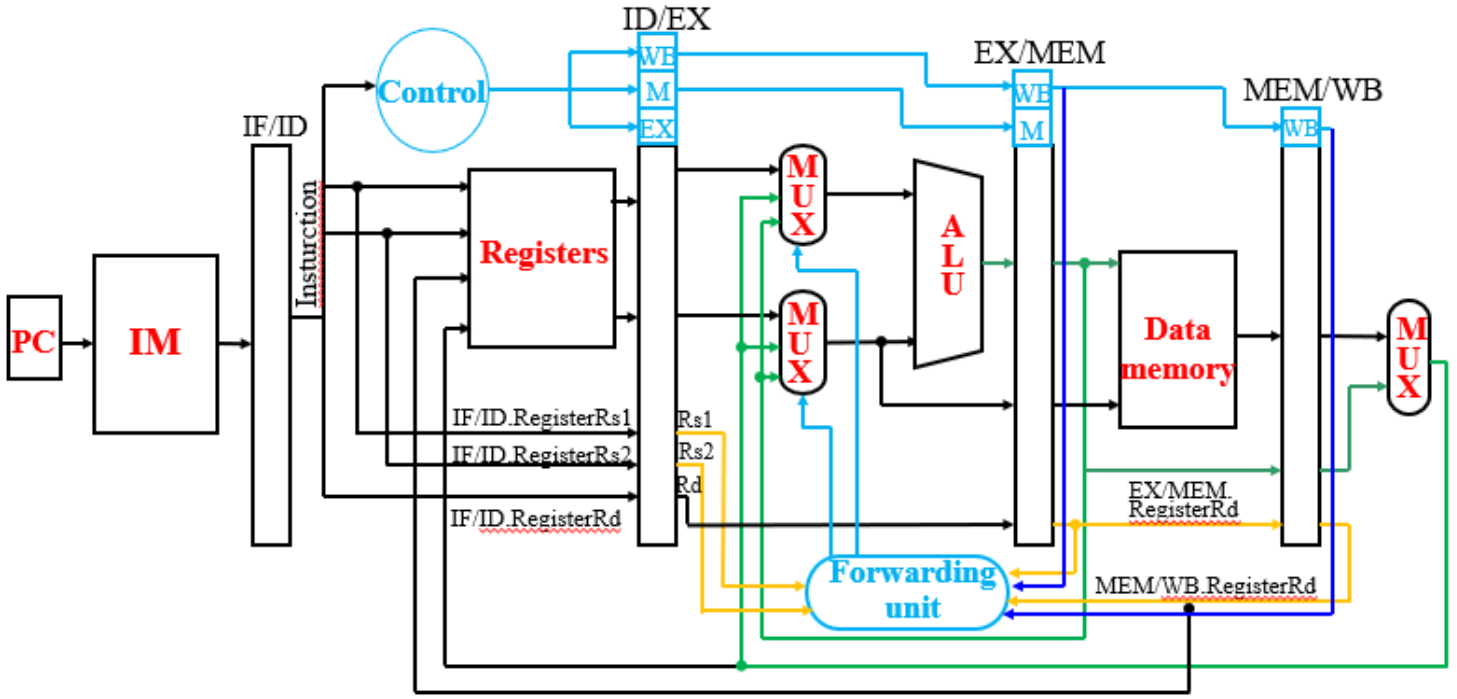
2、通常来说，一条新进入的指令需要先在流水线数据通路的[ ]阶段检测是否需要前递。
（从IF、ID、EX、MEM、WB选择一个填空）

1、通常来说，一条新进入的指令需要在流水线数据通路的[ID]阶段检测是否是load-use冒险（hazard），即检测是否需要停顿。
（从IF、ID、EX、MEM、WB选择一个填空）

2、通常来说，一条新进入的指令需要先在流水线数据通路的[ ]阶段检测是否需要前递。
（从IF、ID、EX、MEM、WB选择一个填空）

1、通常来说，一条新进入的指令需要在流水线数据通路的[ID]阶段检测是否是load-use冒险（hazard），即检测是否需要停顿。
（从IF、ID、EX、MEM、WB选择一个填空）

2、通常来说，一条新进入的指令需要先在流水线数据通路的[EX]阶段检测是否需要前递。
（从IF、ID、EX、MEM、WB选择一个填空）



检测前递发生（进一步完善）

- 并不是所有指令都会写回寄存器。如果写回寄存器是x0，则不需要将数据前递出去，因此需要额外判断。
  - EX冒险判断（对应R-R1型，3个条件要同时满足）：

    EX/MEM.RegWrite and（EX/MEM.RegisterRd≠0）and

    EX/MEM.RegisterRd == ID/EX.RegisterRs
  - MEM冒险判断（对应R-R2型，3个条件要同时满足）：

    MEM/WB.RegWrite and（MEM/WB.RegisterRd≠0）and

    MEM/WB.RegisterRd == ID/EX.RegisterRs

题目2：通常来说，一条新进入的指令需要先在流水线数据通路的[EX]阶段检测是否需要前递。

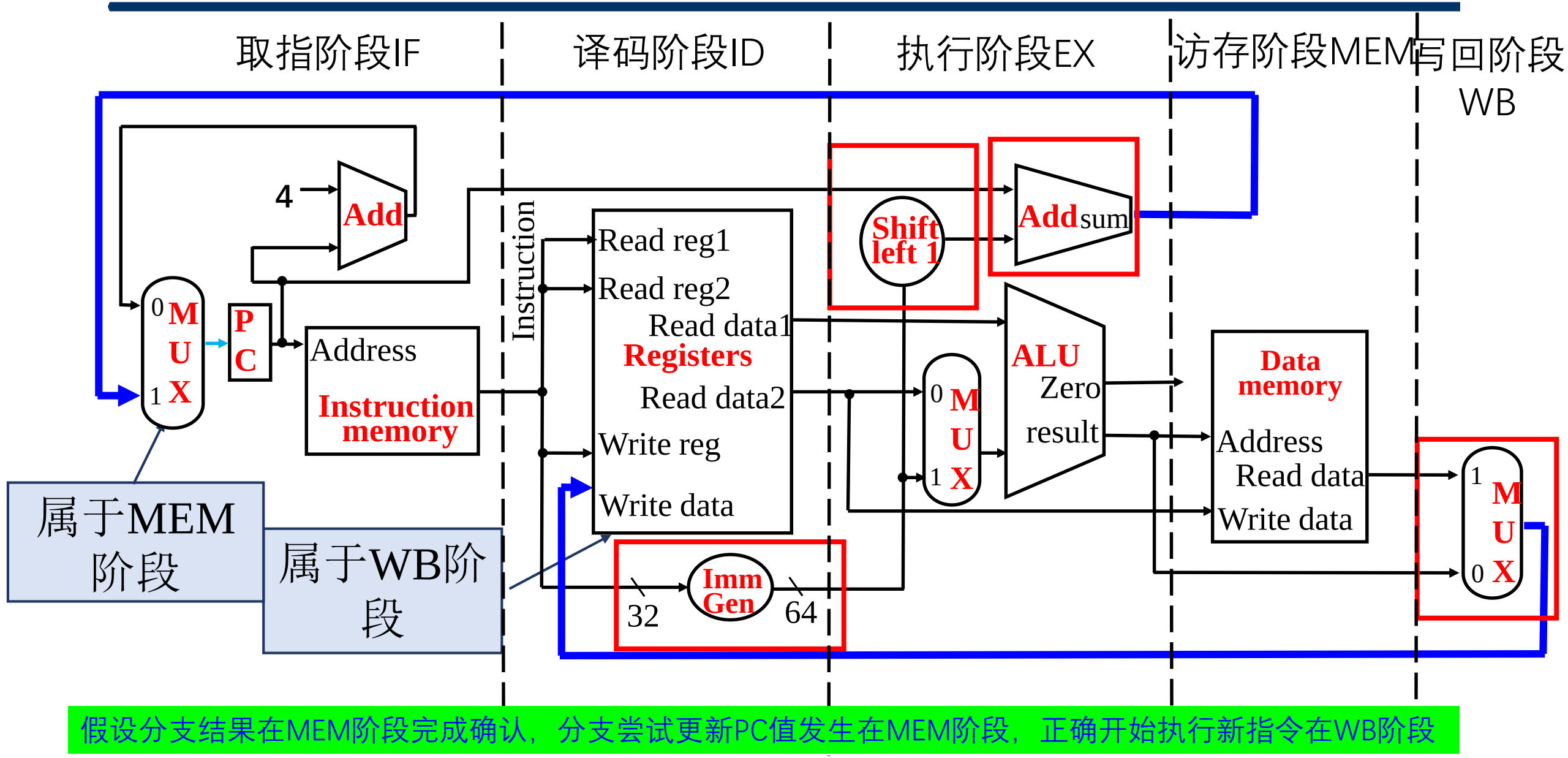
注意：ID/EX寄存器内容与EX阶段指令是一起考虑，也意味着从ID/EX里读取数据时，该指令必然处于EX阶段。同理，读取EX/MEM寄存器数据时，当前指令处于MEM阶段。

另外，前递的电路设计中，最佳的的（后进来的指令的Rs和先进来指令的Rd）比较时间是当前时钟上升沿发生后的时间，从电路图中可以看到，R-R1型和R-R2型在EX阶段进行前递检测是比较合理的。

第五章

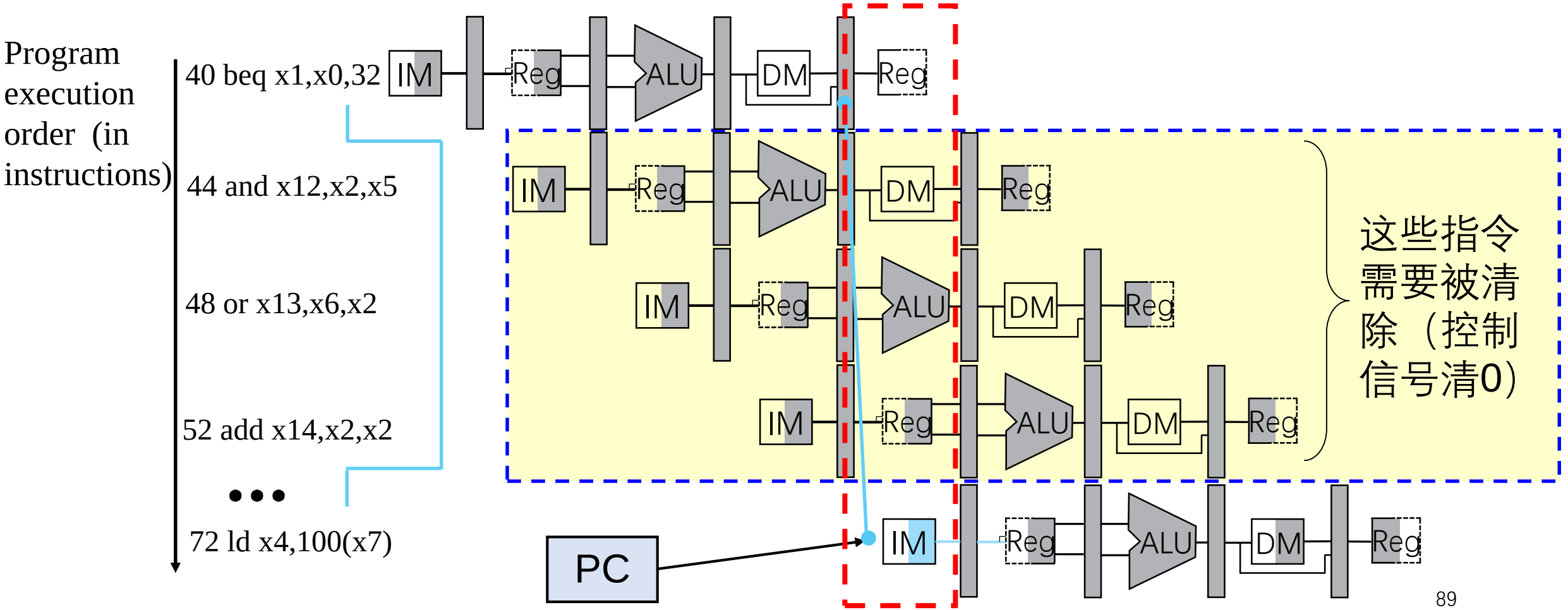
- 流水线概述
- 流水线数据通路和控制
- 数据冒险：前递与停顿
- **控制冒险**
- 例外

构建RISC-V流水线数据通路



假设分支不发生

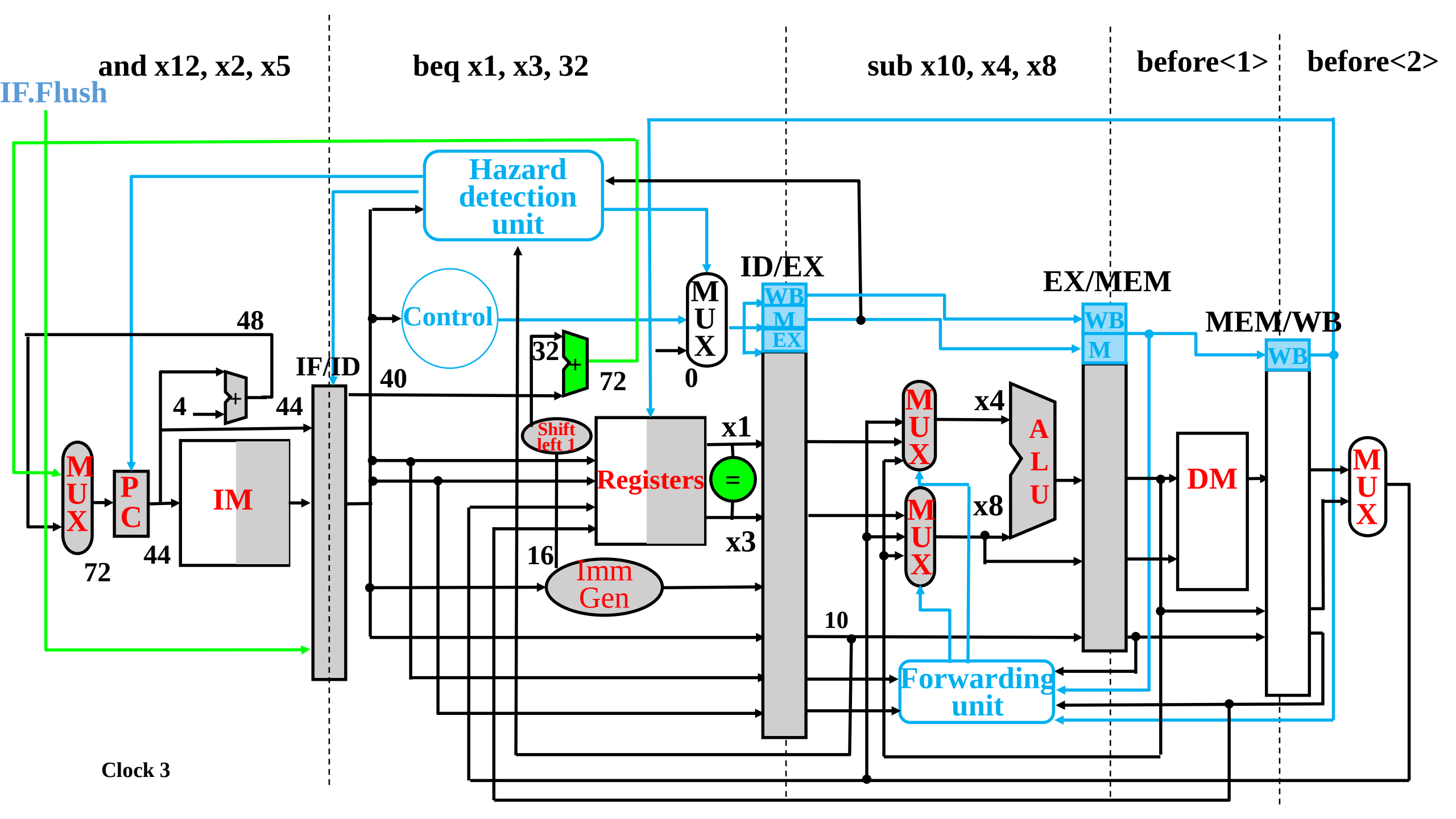
- 默认不发生分支跳转
- **如果发生跳转，则清除掉后续的三条指令**

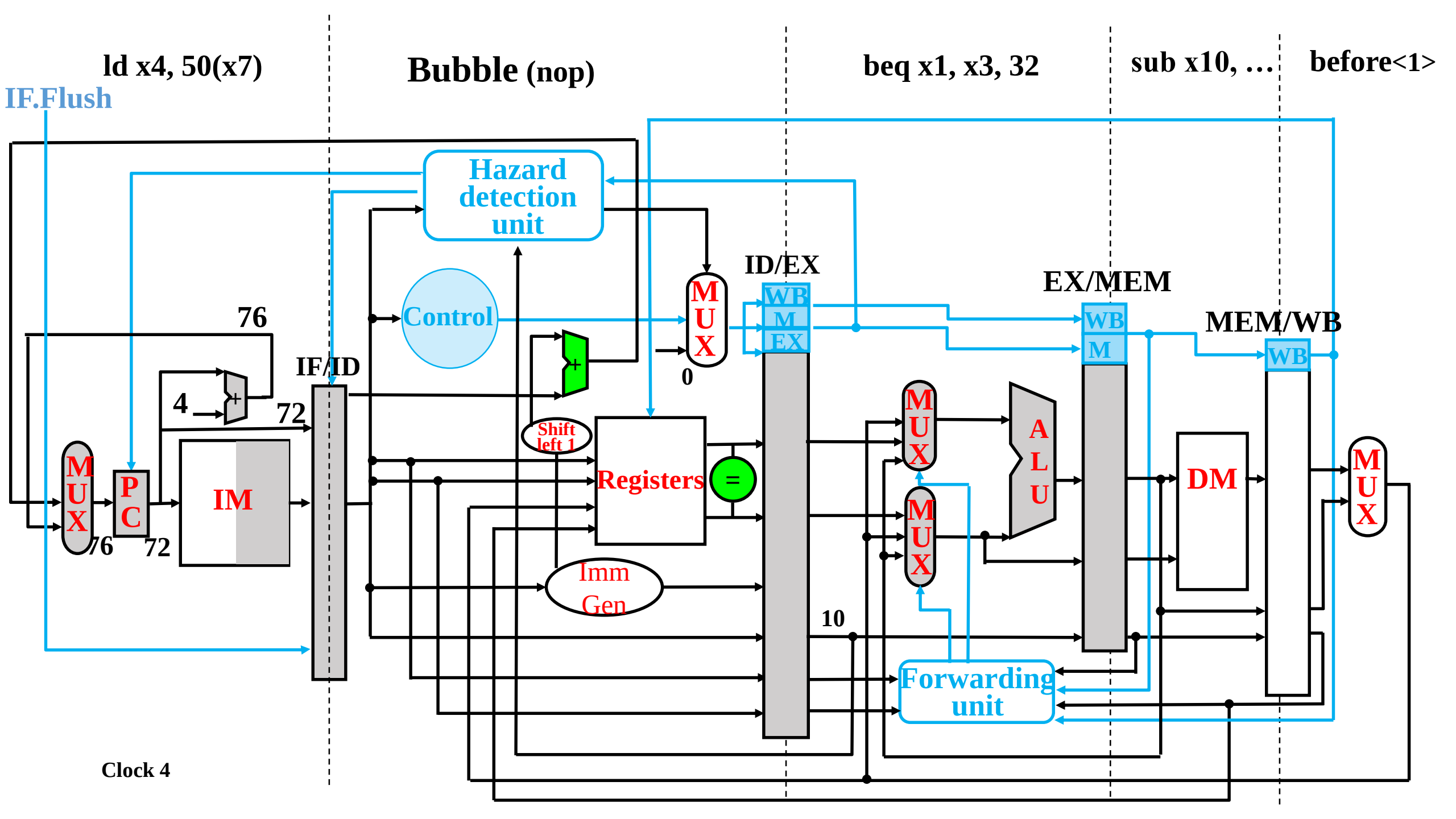


缩短分支延迟

- 移动硬件，**使得分支决定提前到ID阶段**
  - 需要**提早计算**分支目标地址（IF/ID寄存器中有PC和Immediate）
  - 需要**提早判断**分支条件
- 例：分支跳转发生

```
36:  sub  x10, x4, x8
40:  beq  x1,  x3, 32
44:  and  x12, x2, x5
48:  or   x13, x2, x6
52:  add  x14, x4, x2
56:  sub  x15, x6, x7
     ...
72:  ld   x4, 50(x7)
```


and x12, x2, x5
beq x1, x3, 32
sub x10, x4, x8
before<1>
before<2>
IF.Flush
Hazard detection unit
Control
MUX
0
ID/EX
WB
M
EX
EX/MEM
WB
M
MEM/WB
WB
48
4
+
44
IF/ID
40
32
+
72
Shift left 1
x1
x3
=
Registers
16
Imm Gen
MUX
PC
72
IM
44
MUX
x4
x8
ALU
DM
MUX
10
Forwarding unit
Clock 3

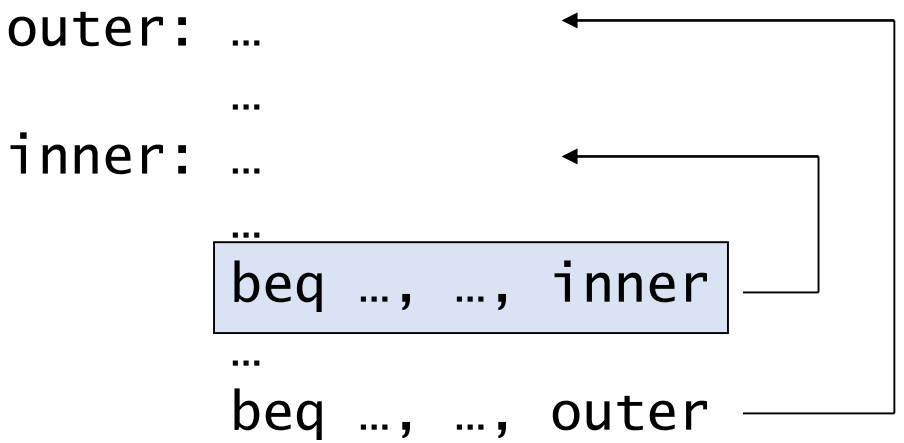

ld x4, 50(x7)
Bubble (nop)
beq x1, x3, 32
sub x10, …
before<1>
IF.Flush
Hazard detection unit
Control
ID/EX
EX/MEM
MEM/WB
WB
M
EX
MUX
0
76
4
72
IF/ID
PC
IM
Shift left 1
Registers
=
Imm Gen
ALU
DM
10
Forwarding unit
Clock 4

动态分支预测

- 对于更深的流水线，从时钟周期数的角度来说，分支预测错误的代价会增大。
- 使用动态分支预测
  - 一种实现方式：采用**分支预测缓存**或**分支历史表**
  - 是一块按照分支指令的低位地址索引定位的小容量存储器
  - 包含一个或多个位（bit）以表明一个分支最近是否发生了跳转
- 1-Bit预测机制：用1位表示最近是否发生了跳转
  - 查表，使用表中记录结果作为预测结果
  - 开始取指令
  - 如果预测错误，则清掉错误指令，并更改分支预测缓存中的记录

1-Bit预测机制的缺点

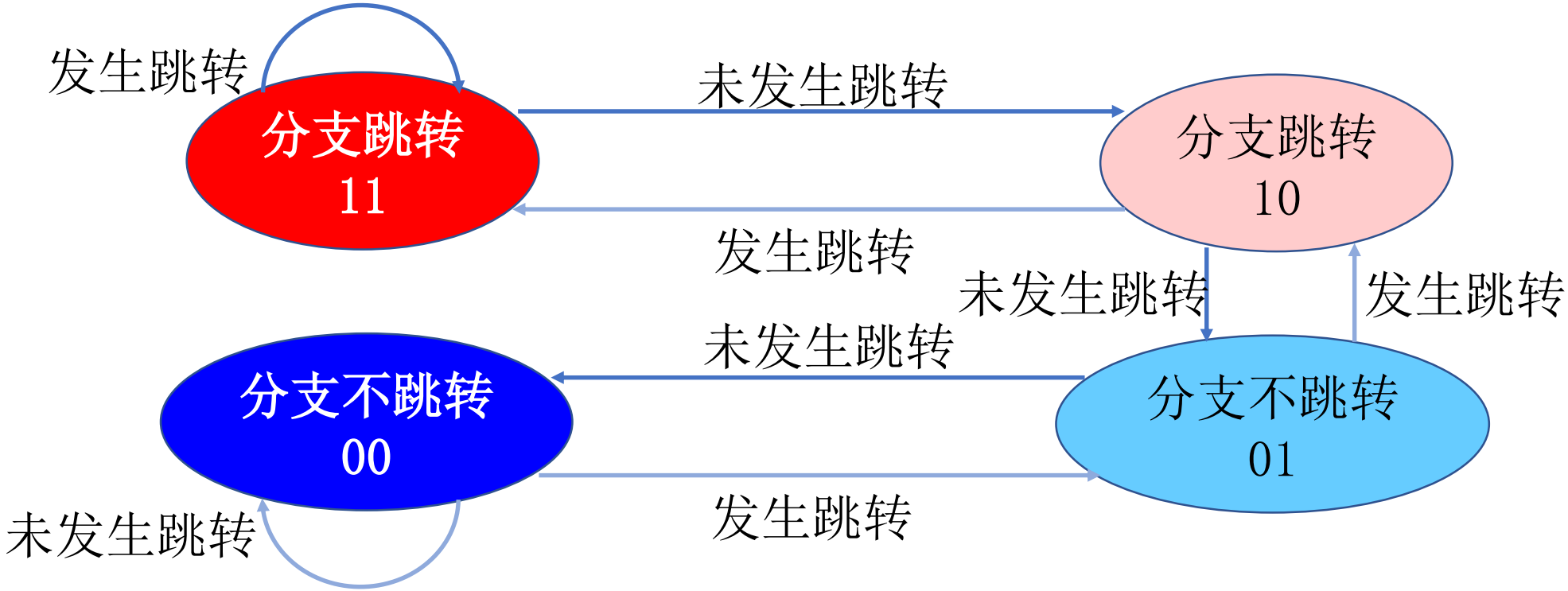
- 一个条件分支总是发生跳转，但一旦其不发生跳转时，会导致两次预测错误，而不是只造成一次错误



  - 内层循环的最后一次迭代预测发生跳转，但实际不发生跳转，预测错误，更改预测值为不跳转。
  - 下一次外层循环开始后，内层循环第一次迭代发生跳转，预测不发生跳转，预测错误，更改预测值为跳转

2-Bit预测机制

- 对于左侧两个状态，只有在发生了连续两次错误时预测结果才会被改变



在一个分支经常跳转或经常不跳转的情况下（大多数分支都是这样的），只会发生一次预测失效

分支目标计算

- 除了判断分支是否发生，还要计算分支目标地址
  - 发生跳转时需要一个时钟周期的代价来计算分支目标地址

解决办法：使用分支目标缓存

- 分支地址的Cache
- 按取指令时的PC索引
  - 如果cache命中且指令被预测为分支发生，
    则可以查表获得跳转地址，不必计算。

第五章

- 流水线概述
- 流水线数据通路和控制
- 数据冒险：前递与停顿
- 控制冒险
- **例外（异常）**

例外（异常）和中断

- 例外（异常）和中断是控制逻辑需要实现的任务之一
  - 除分支指令外，它是另一种改变指令执行控制流的方式
- 不同的ISA对于例外和中断的定义不同，比如： **Intel x86使用中断同时指代两者**
- 例外(exception，也叫异常)
  - 对于RISC-V来说，指代**意外的控制流变化**，而这些变化无须区分产生原因是来自于处理器内部以及外部
- 中断(interrupt)
  - 仅指代由处理器外部事件引发的控制流变化
- 就目前而言，我们只涉及例外的控制逻辑
- 对例外进行时序优化是困难的

RISC-V中的例外和中断示例

事件类型	来源	RISC-V中的表示
系统重启	外部	例外（异常）
I/O设备请求	外部	中断
用户程序进行操作系统调用	内部	例外（异常）
未定义指令	内部	例外（异常）
硬件故障	皆可	皆可

RISC-V体系结构中如何处理例外

- **保存发生例外的指令地址**，将控制权转交给操作系统
  - 使用**系统例外程序计数器**(Supervisor Exception Program Counter, SEPC)
    保存发生例外的指令地址，64位
- **保存例外发生的原因**
  - **系统例外原因寄存器**(Supervisor Exception Cause Register, SCAUSE)
  - SCAUSE：64位寄存器，大多数位未被使用。

    例如，2被编码为未定义指令，12被编码为硬件故障
- **跳转到统一的入口地址**，进行例外处理
  - $0000\ 0000\ 1C09\ 0000_{hex}$

另一种例外处理方式(x86等)

- 采用向量式中断（x86中统称为中断）
  - 例外原因决定后续控制流的起始地址
- 基址寄存器 + 例外原因（作为偏移量）
  - 未定义指令的偏移量：　　$00\ 0100\ 0000_2$
  - 硬件故障的偏移量：　　$01\ 1000\ 0000_2$
- 转到例外处理程序

例外处理程序的工作

- 如果可以重启程序的执行
  - 完成例外处理的所有操作
  - 使用SEPC寄存器中的内容重启程序的正常执行
- 否则（未定义指令或硬件故障等）
  - 停止当前程序的执行
  - 使用SEPC, SCAUSE等来报告错误

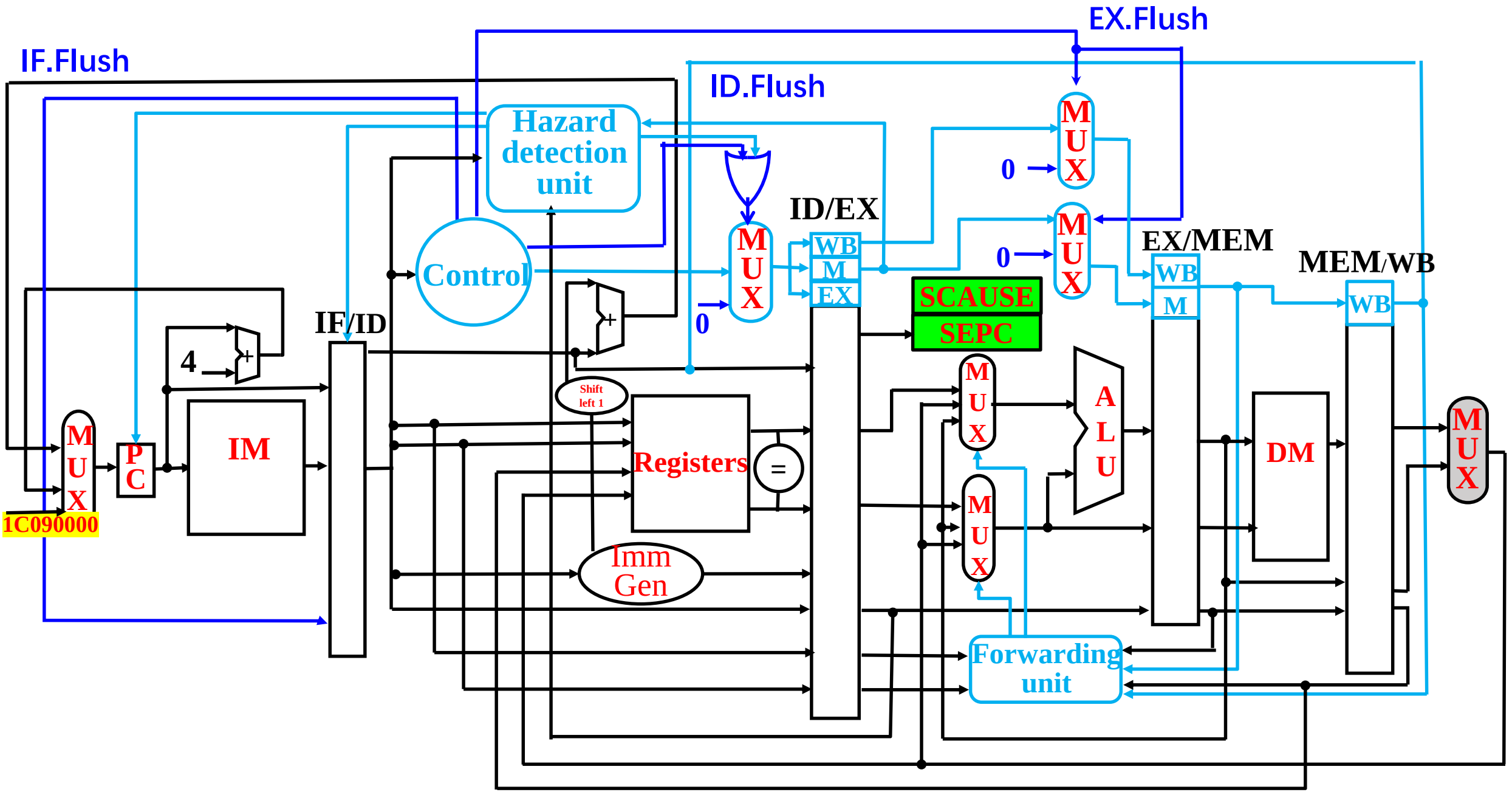
流水线实现中的例外

- 流水线实现中，将例外处理看作另一种控制冒险。
- 假设add指令EX阶段发生了硬件故障。

```
40          sub  x11, x2, x4
44          and  x12, x2, x5
48          or   x13, x2, x6
4C          add  x1,  x2, x1
50          sub  x15, x6, x7
54          ld   x16, 100(x7)
…
```

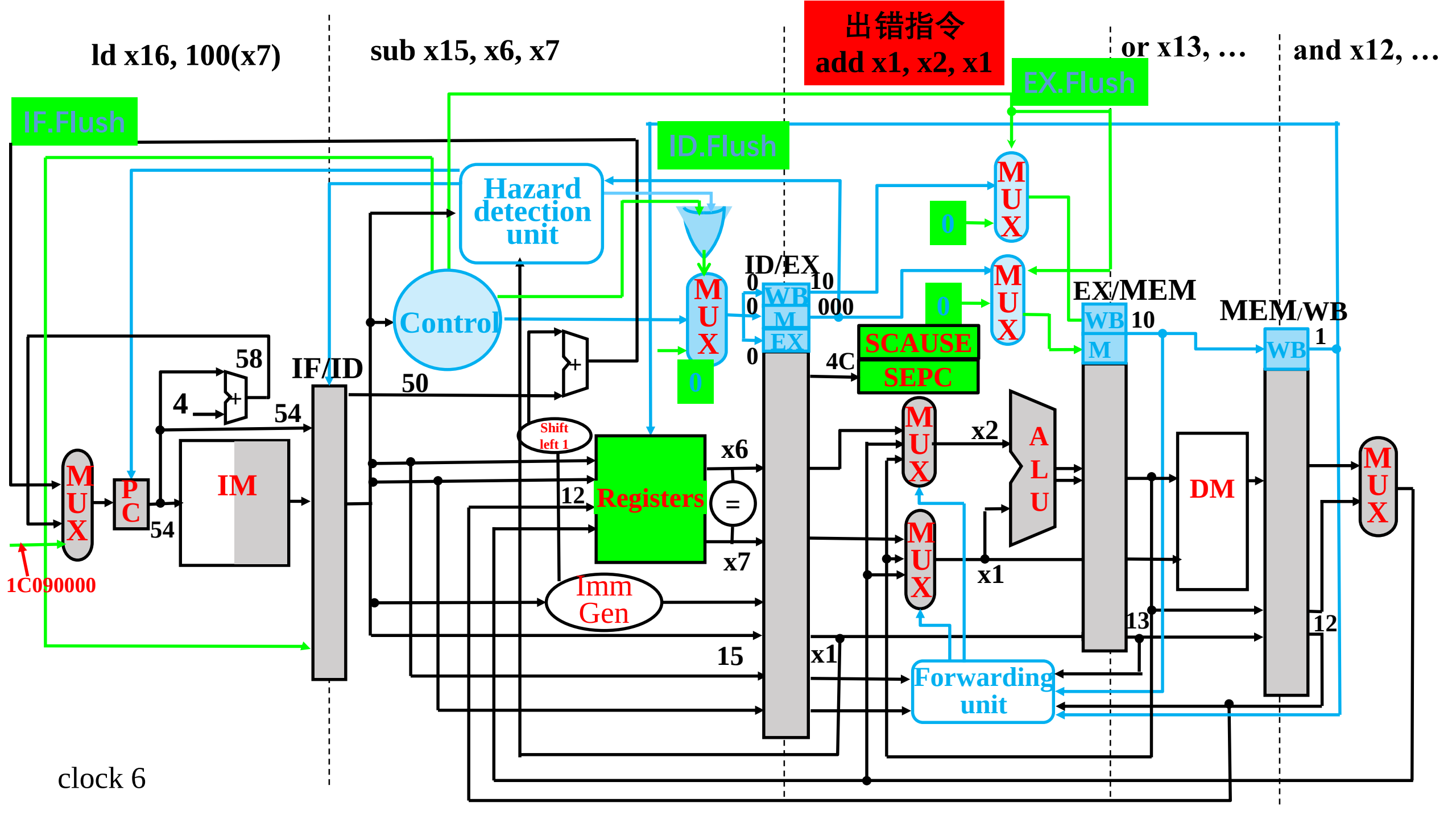
- Handler

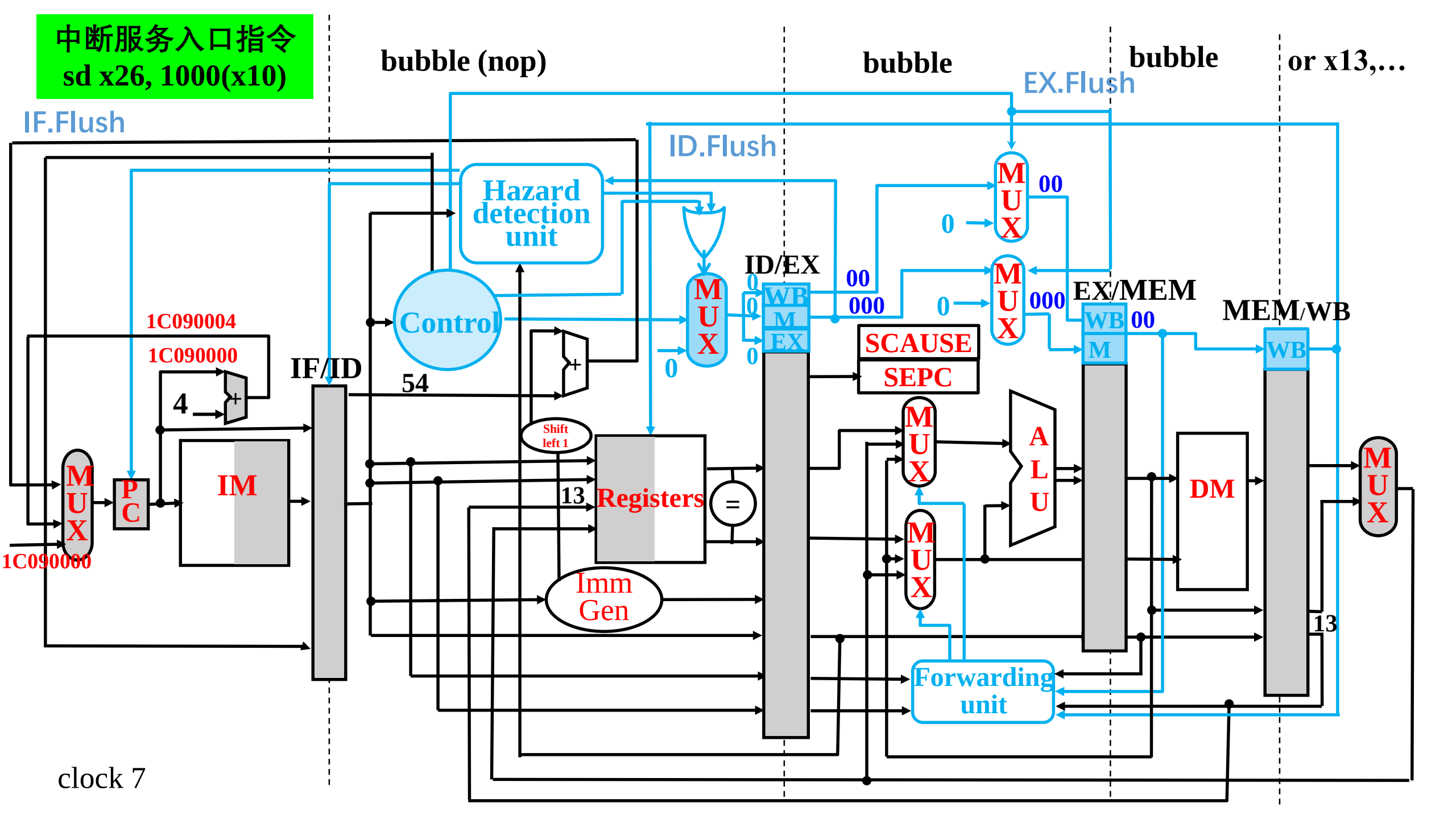
```
1C090000        sd  x26, 1000(x10)
1C090004        sd  x27, 1008(x10)
…
```


IF.Flush
ID.Flush
EX.Flush
Hazard detection unit
Control
IF/ID
ID/EX
EX/MEM
MEM/WB
WB
M
EX
SCAUSE
SEPC
MUX
0
4
PC
IM
1C090000
Shift left 1
Registers
=
Imm Gen
ALU
DM
Forwarding unit

流水线实现中的例外

- 考虑add指令 add x1, x2, x1 的EX阶段发生了硬件故障
  - IF阶段的指令：变为nop操作
  - ID阶段的指令：增加新的逻辑部件，使得译码阶段的输出为0
  - EX阶段的指令：清空控制信号，即：让控制信号相关的多路选择器输出为0
  - add之前的指令，正常完成
  - 设置SEPC和SCAUSE的寄存器值
  - 转到例外处理程序


ld x16, 100(x7)
sub x15, x6, x7
出错指令
add x1, x2, x1
or x13, …
and x12, …
IF.Flush
ID.Flush
EX.Flush
Hazard detection unit
Control
MUX
0
ID/EX
WB
M
EX
000
SCAUSE
SEPC
4C
EX/MEM
MEM/WB
IF/ID
58
4
54
50
PC
IM
1C090000
Shift left 1
Registers
x6
x7
12
15
Imm Gen
=
ALU
x2
x1
DM
13
Forwarding unit
clock 6


中断服务入口指令
sd x26, 1000(x10)
bubble (nop)
bubble
bubble
or x13,…
IF.Flush
ID.Flush
EX.Flush
Hazard detection unit
Control
ID/EX
EX/MEM
MEM/WB
WB
M
EX
SCAUSE
SEPC
1C090004
1C090000
1C090000
IF/ID
54
4
Shift left 1
13
Registers
Imm Gen
IM
PC
MUX
ALU
DM
Forwarding unit
00
000
0
13
clock 7

总结

- 流水线概述
- 流水线数据通路和控制
- 数据冒险：前递与停顿
- 控制冒险
- 例外