计 算 机 体 系 结 构

第七讲

计算机科学与技术学院

舒燕君

Recap

- 指令系统的设计与优化
  - ✓控制指令（调用与返回）
  - ✓操作码优化
- 指令系统的发展和改进
  - ✓CISC指令集功能设计（目标程序、高级语言和编译器、操作系统）
  - ✓ RISC指令集功能设计
- MIPS指令集简介
  - ✓指令和操作码定长，三类寻址方式
  - ✓指令格式I类、R类、J类
- 报告挑一个章节的主题写，电子版即可，考试之前提交

第 0 章　前言

第 1 章　计算机体系结构基本概念

第 2 章　指令系统

第 3 章　流水线技术

第 4 章　指令级并行

第 5 章　存储层次

第 6 章　输入输出系统

第3章 流水线技术

3.1 流水线概述

3.2 MIPS的基本流水线

3.3 流水线中的冲突

3.4 实例分析：MIPS R4000

3.1 流水线概述

3.1.1 流水线基本概念

3.1.2 流水线分类

3.1.1 流水线的基本概念

1. 产品生产流水线

(1) 一个问题

假设某产品的生产需要4道工序，该产品生产车间以前只有1个工人，1套生产该产品的机器。该工人工作8小时，可以生产120件（即每4分钟生产1件）。

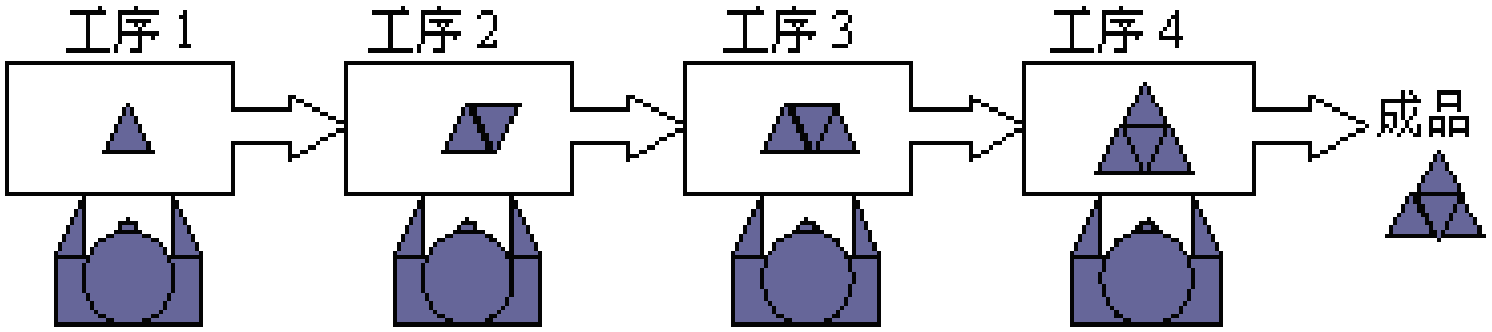
要将该产品日产量提高到480件，如何能实现目标？

3.1.1 流水线基本概念

(2) 两种解决方案

方案一：增加3名工人、3套设备。

方案二：产品生产采用流水线方式，分为4道工序；
增加3名工人，每人负责一道工序。



3.1.1 流水线基本概念

(3) 两种方案的工作过程对比

两种方案中，单件产品的生产时间均不变。

但在稳定情况下，

方案一：每4分钟，4件产品同时进入流水线，4件成品同时离开流水线，需要增加3套设备。

方案二：每分钟，1件产品进入流水线，1件成品离开流水线，不需要增加任何设备。

3.1.1 流水线基本概念

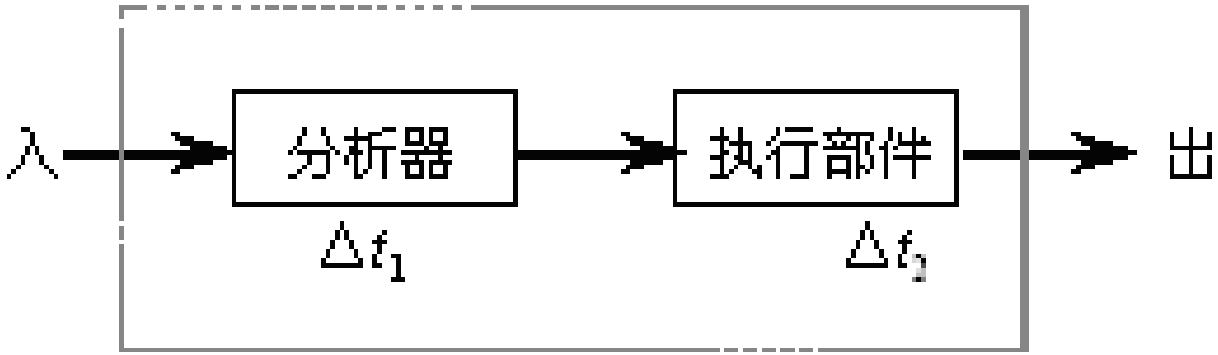
(4) 方案二的主要特点

每件产品还是要经过4道工序处理，单件产品的加工时间并没有改变，但它将各个工人的操作时间重叠在一起，使得每件产品的产出时间从表面上看是从原来的4分钟缩减到1分钟，提高了产品的产出率。

3.1.1 流水线基本概念

2.计算机中的流水线

◆ 指令流水线



◆ 功能部件流水线

3.1.1 流水线基本概念

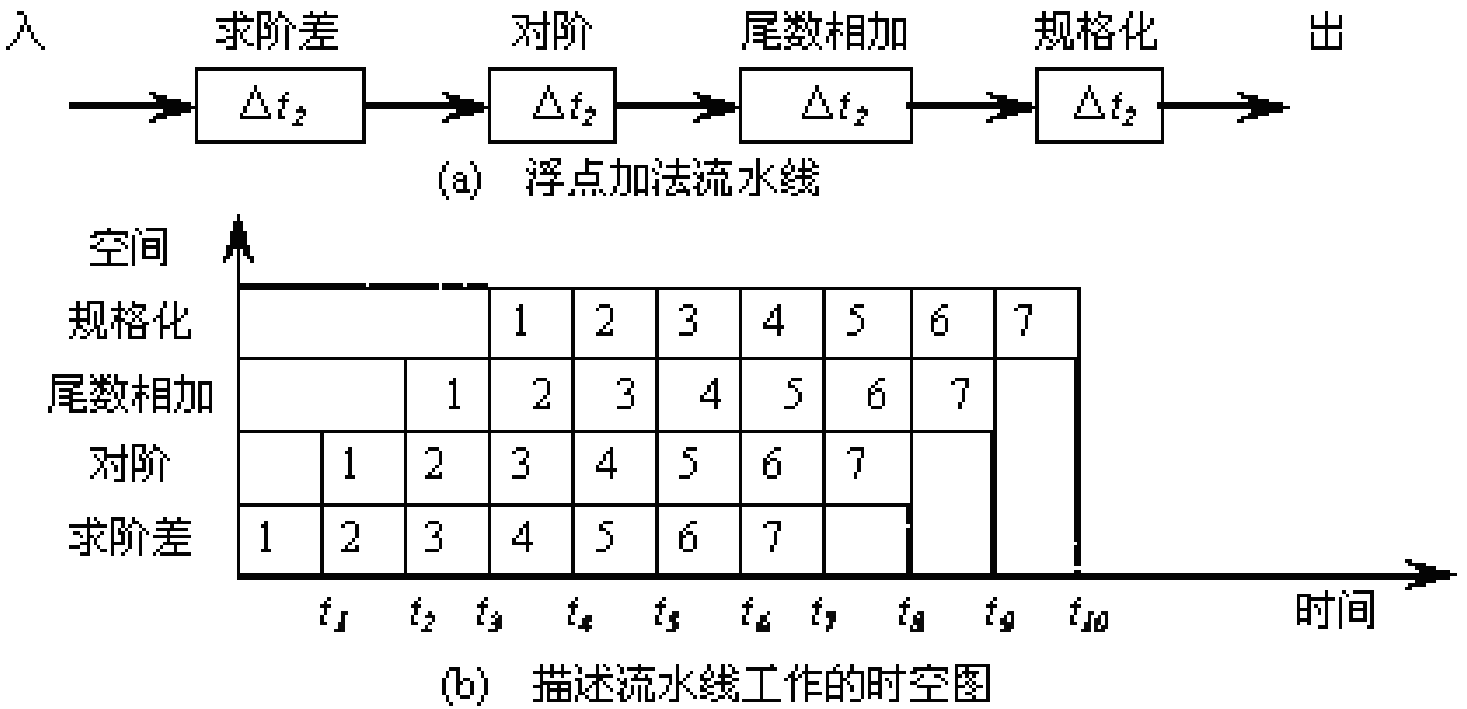
3. 流水技术的定义

将一重复的时序过程分解为若干子过程，每个子过程都可有效地在其专用功能段上与其它子过程同时执行，这种技术称为流水技术。

4. 时空图

从时间和空间两个方面描述流水线的工作过程，横坐标表示时间，纵坐标表示各流水段。

流水线技术原理


入
求阶差
对阶
尾数相加
规格化
出
Δt_2
Δt_2
Δt_2
Δt_2
(a) 浮点加法流水线
空间
规格化
尾数相加
对阶
求阶差
时间
t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8 t_9 t_{10}
(b) 描述流水线工作的时空图

3.1.1 流水线基本概念

5. 流水线的特点

① 流水过程由多个相关的子过程组成，这些子过程称为流水线的“级”或“段”。段的数目称为流水线的“深度”。

② 每个子过程由专用的功能段实现。

③ 各功能段的时间应基本相等，通常为1个时钟周期（1拍）。

④ 流水线需要经过一定的通过时间才能稳定。

⑤ 流水技术适合于大量重复的时序过程。

3.1 流水线概述

3.1.1 流水线基本概念

3.1.2 流水线分类

3.1.2 流水线的分类

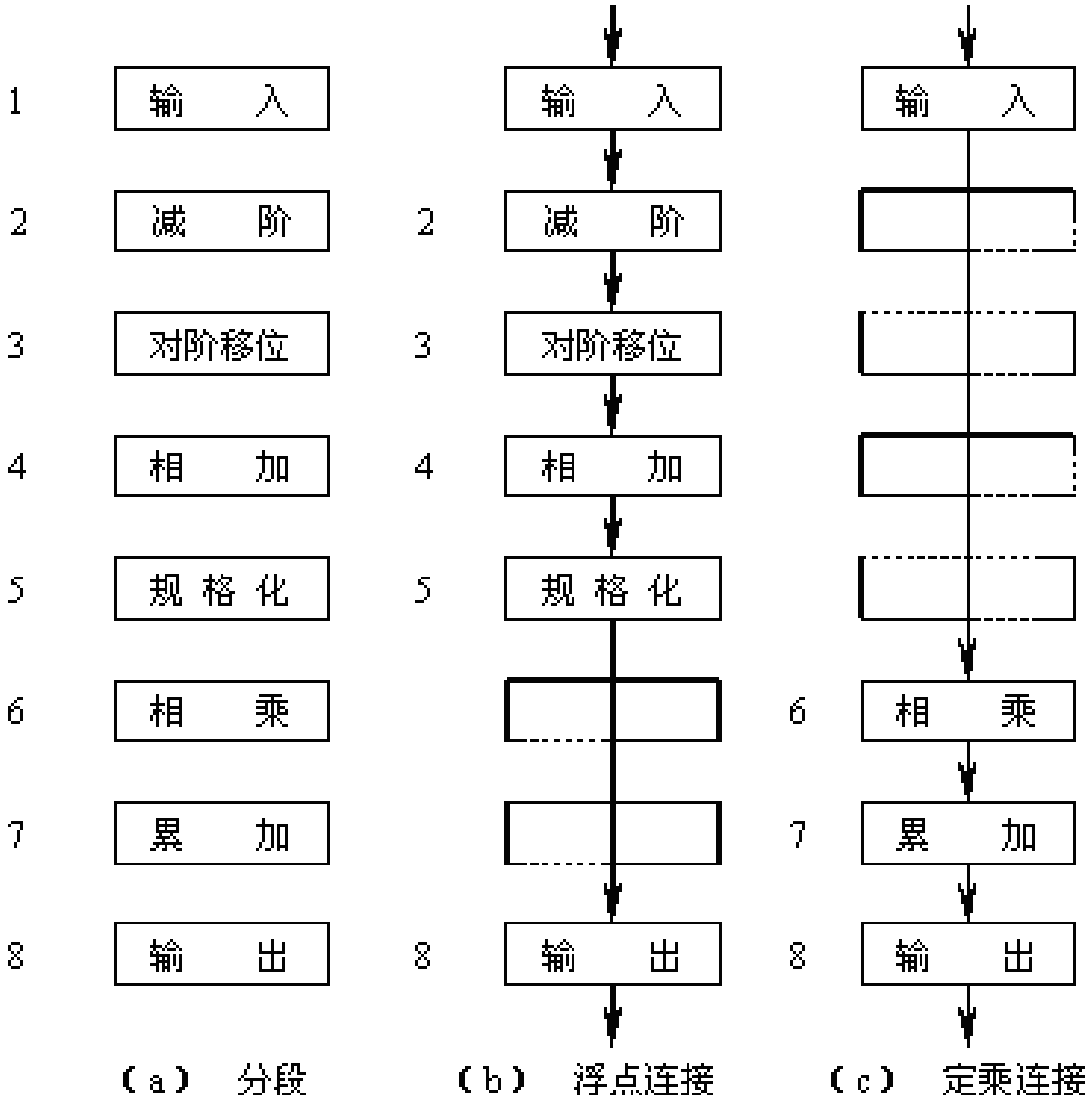
1.单功能流水线和多功能流水线

- 按流水线所完成的功能分类
- **单功能流水线，**是指只能完成一种固定功能的流水线。

  例如：功能单元流水线
- **多功能流水线，**是指各段可以进行不同的连接，从而完成不同的功能。

  例如：TI ASC多功能流水线

TI ASC的多功能流水线



3.1.2 流水线的分类

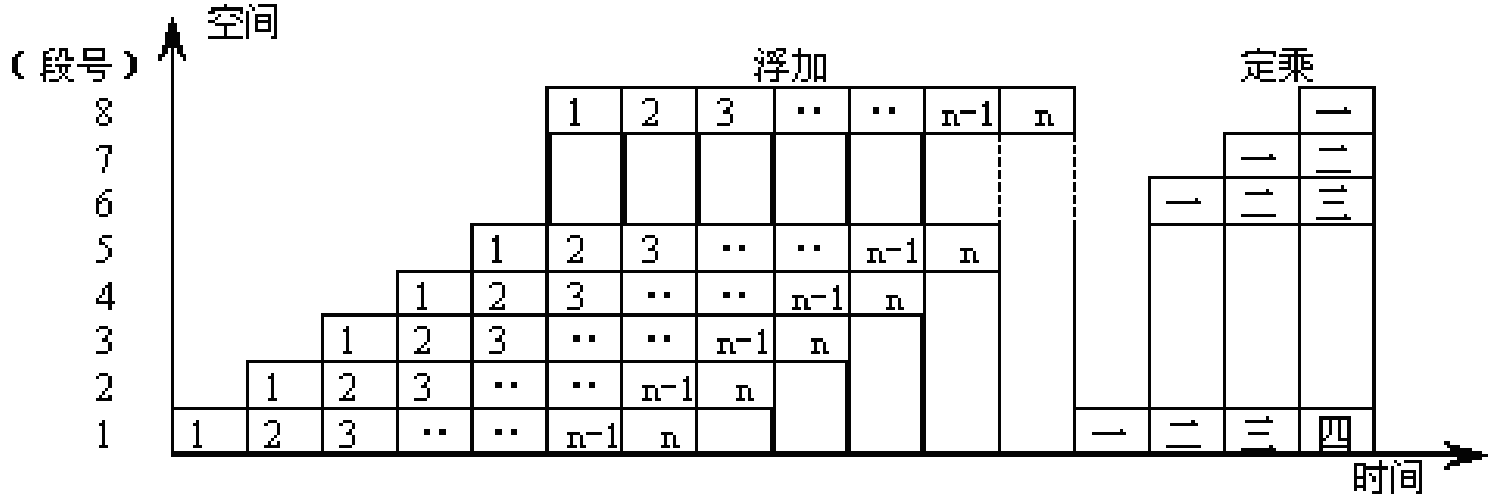
2.静态流水线和动态流水线

- 按同一时间内流水段的连接方式划分
- **静态流水线，**是指在同一时间内，流水线的各段只能按同一种功能的连接方式工作。

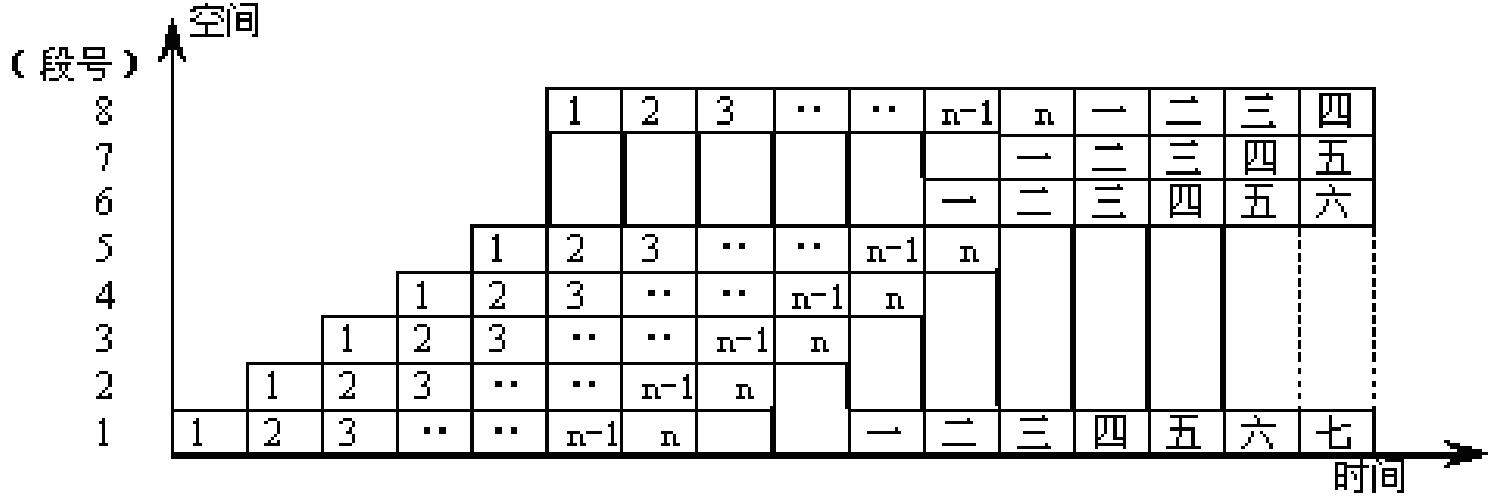
  例如：TI ASC的流水线

  适合于处理一串相同的运算操作
- **动态流水线，**是指在同一时间内，当某些段正在实现某种运算时，另一些段却在实现另一种运算，会使流水线的控制变得很复杂

动、静态流水线时空图



（a） 静态流水线



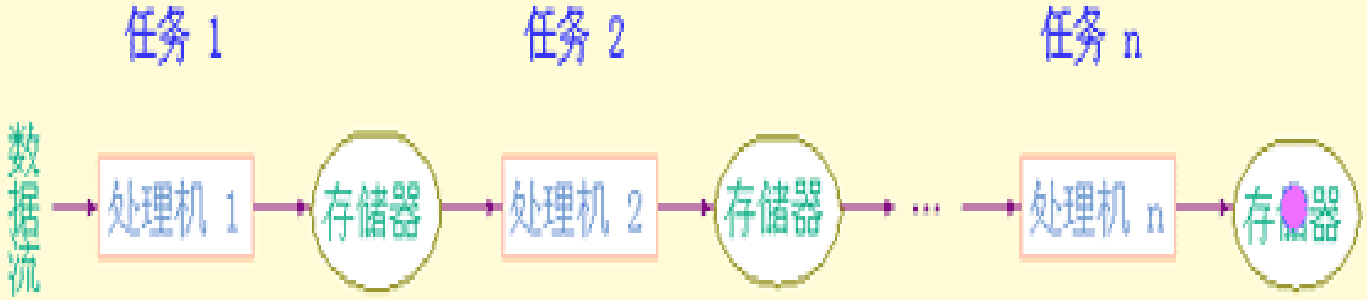
（b） 动态流水线

3.1.2 流水线的分类

3.部件级、处理机级及处理机间流水线

- 按流水的级别划分
- 部件级流水线，又叫运算操作流水线，是把处理机的算术逻辑部件分段，使得各种数据类型的操作能够进行流水。
- 处理机级流水线，又叫指令流水线，是把解释指令的过程按照流水方式处理。
- 处理机间流水线，又叫宏流水线，是由两个以上的处理机串行地对同一数据流进行处理，每个处理机完成一项任务。

宏流水线



3.1.2 流水线的分类

4. 标量流水处理机和向量流水处理机

- 按照数据表示来进行分类
- 标量流水处理机，是指处理机不具有向量数据表示，仅对标量数据进行流水处理。

  例如：IBM 360/91，Amdahl 470V/6等
- 向量流水处理机，是指处理机具有向量数据表示，并通过向量指令对向量的各元素进行处理。

  例如：TI ASC、STAR-100、CYBER-205、CRAY-1、YH-1等

3.1.2 流水线的分类

5.线性流水线和非线性流水线

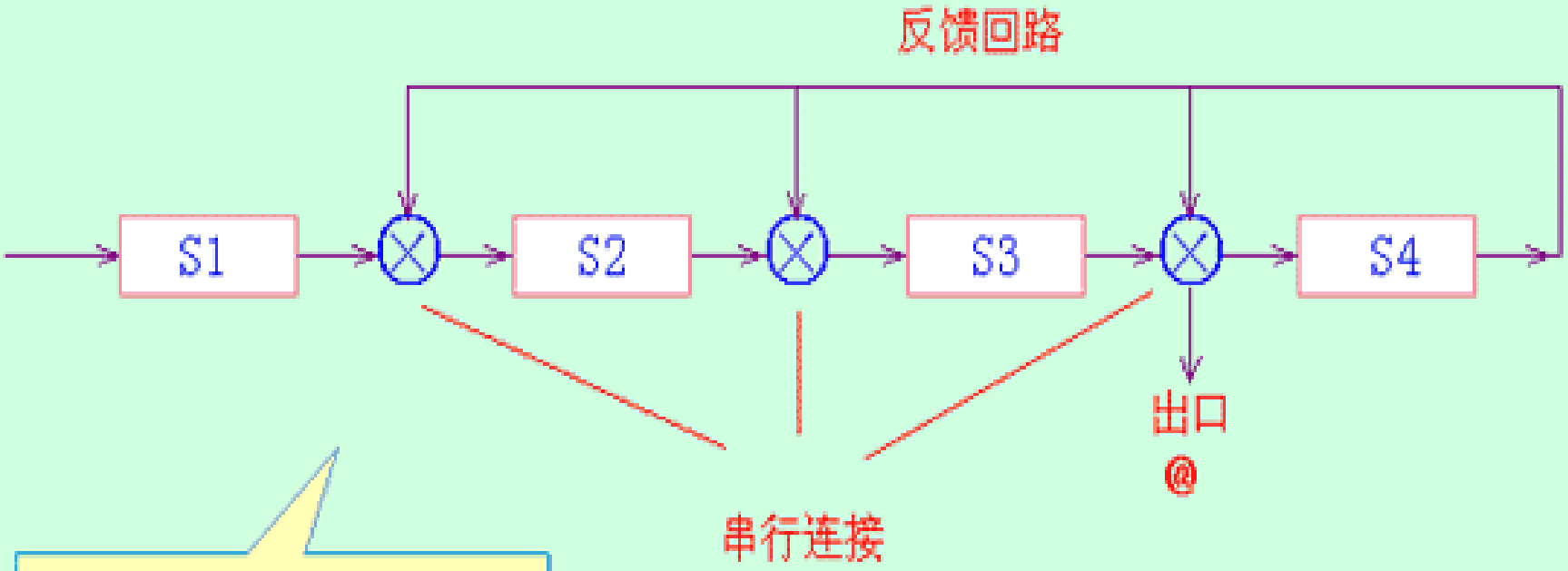
- 按照是否有反馈回路来进行分类
- 线性流水线是指流水线的各段串行连接，没有反馈回路。
- 非线性流水线是指流水线中除有串行连接的通路外，还有反馈回路。

  存在流水线调度问题。

确定什么时候向流水线引进新的输入，从而使新输入的数据和先前操作的反馈数据在流水线中不产生冲突，此即所谓流水线调度问题。

非线性流水线

（举 例）

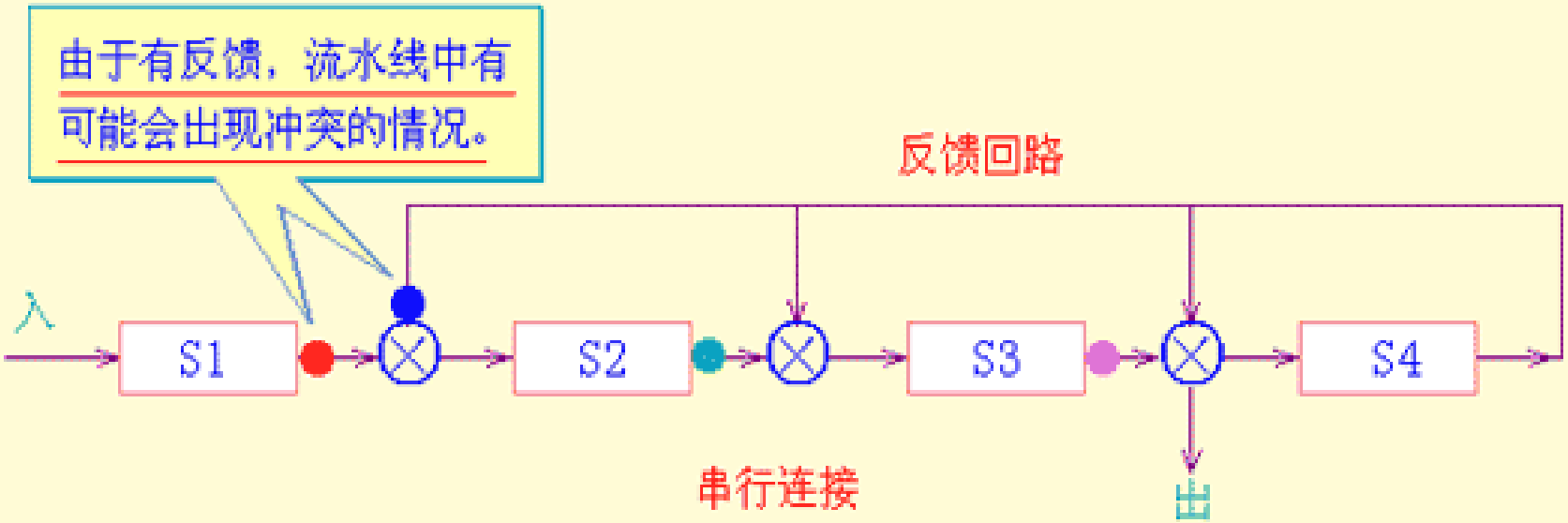


虽然流水线仅由四段构成，但有些段可能要重复通过。

例如任务 @ ：

→ S1 → S2 → S3 → S4 → S2 → S3 → S4 → S3 →

流水线的调度问题



所以：

在非线性流水线中，一个重要的问题是确定什么时候向流水线引进新的输入，从而使新输入的数据和先前操作的反馈数据在流水线中不产生冲突。这就是所谓的流水线调度问题。

3.2 MIPS基本流水线

3.2.1 MIPS的一种简单实现

3.2.2 基本MIPS流水线

3.2.3 流水线性能分析

3.2.1 MIPS的一种简单实现

- 处理器的指令字长为32位，包含
  - 32个32位通用寄存器R0~R31（R0值为0）；
  - 1个32位的指令寄存器IR；
  - 1个32位的程序计数器PC

- 取指令时，可以直接从指令存储器中提取32位的指令信息。取数据时，与数据存储器进行32位的数据交换。处理器的地址总线宽度是32位，数据总线宽度也是32位，无论是取指还是数据访问，都假设能在一个时钟周期内完成。

- 只讨论整数指令的实现（包括：Load和Store，等于0转移，整数ALU指令等。）

处理器功能及指令系统定义

- 存数指令 SW 30(R2), R1
- 取数指令 LW R1, 30(R2)

- 逻辑与指令 AND R1, R2, R3
- 逻辑或指令 OR R1, R2, R3

- 立即数加法指令 ADDI R4, R5, #6
- 立即数减法指令 SUBI R4, R5, #6

- 条件转移（零则转）指令 BEQZ R3, x
- 无条件转移指令 J x

给定的指令系统

J x	PC ← PC+x
BEQZ R3, x	If(R3=0) then PC ← PC+x
AND R1, R2, R3	Regs[R1] ←Regs[R2] and Regs[R3]
OR R1, R2, R3	Regs[R1] ←Regs[R2] or Regs[R3]
ADDI R4, R5, #6	Regs[R4] ←Regs[R5] + 6
SUBI R4, R5, #6	Regs[R4] ←Regs[R5] - 6
LW R1, 30(R2)	Regs[R1] ← Mem[30+Regs[R2]]
SW 30(R2), R1	Mem[30+Regs[R2]]← Regs[R1]

操作码

操作码占据了6位，最多可支持64种指令的设计。目前的指令系统仅包含了8种操作，下表定义这8种操作的操作码

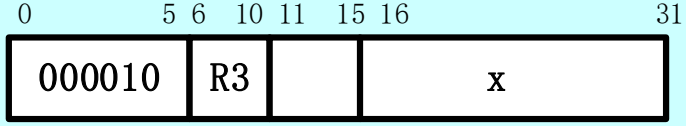
指令名称	助记符	二进制操作码
无条件跳转	J	000000
条件跳转	BEQZ	000010
逻辑与操作	AND	000100
逻辑或操作	OR	000100
立即数减法	SUBI	001000
立即数加法	ADDI	001010
存数操作	SW	001100
取数操作	LW	001110

每条指令的格式描述

J name

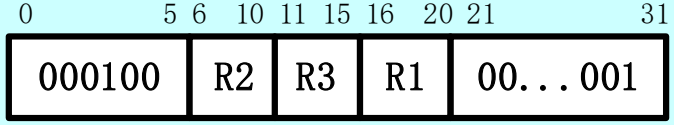
PC←PC+x

BEQZ R3,X



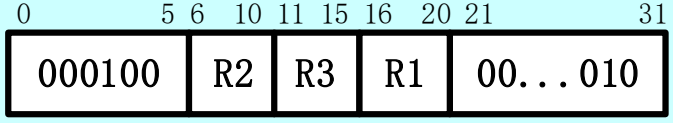
if R3=0 then PC←PC+x

AND R1,R2,R3



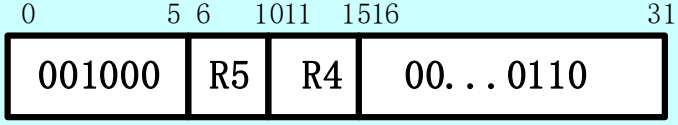
R1←R2 and R3

OR R1,R2,R3



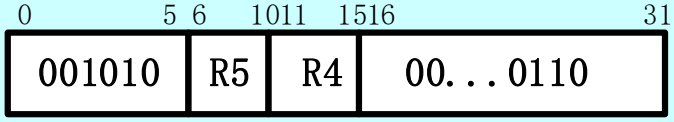
R1←R2 or R3

SUBI R4,R5,#6



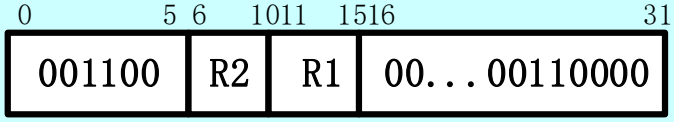
R4←R5-6

ADDI R4,R5,#6



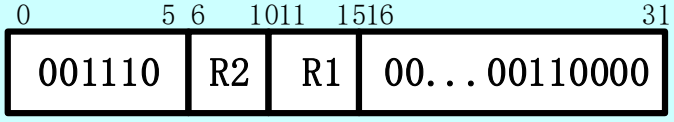
R4←R5+6

SW 30(R2),R1



Mem[R2+30]←R1

LW R1,30(R2)



R1←Mem[R2+30]

指令格式举例

- 38410030

  00111000010000010000000000110000

  LW R1,30(R2)

- 10430801

  00010000010000110000100000000001

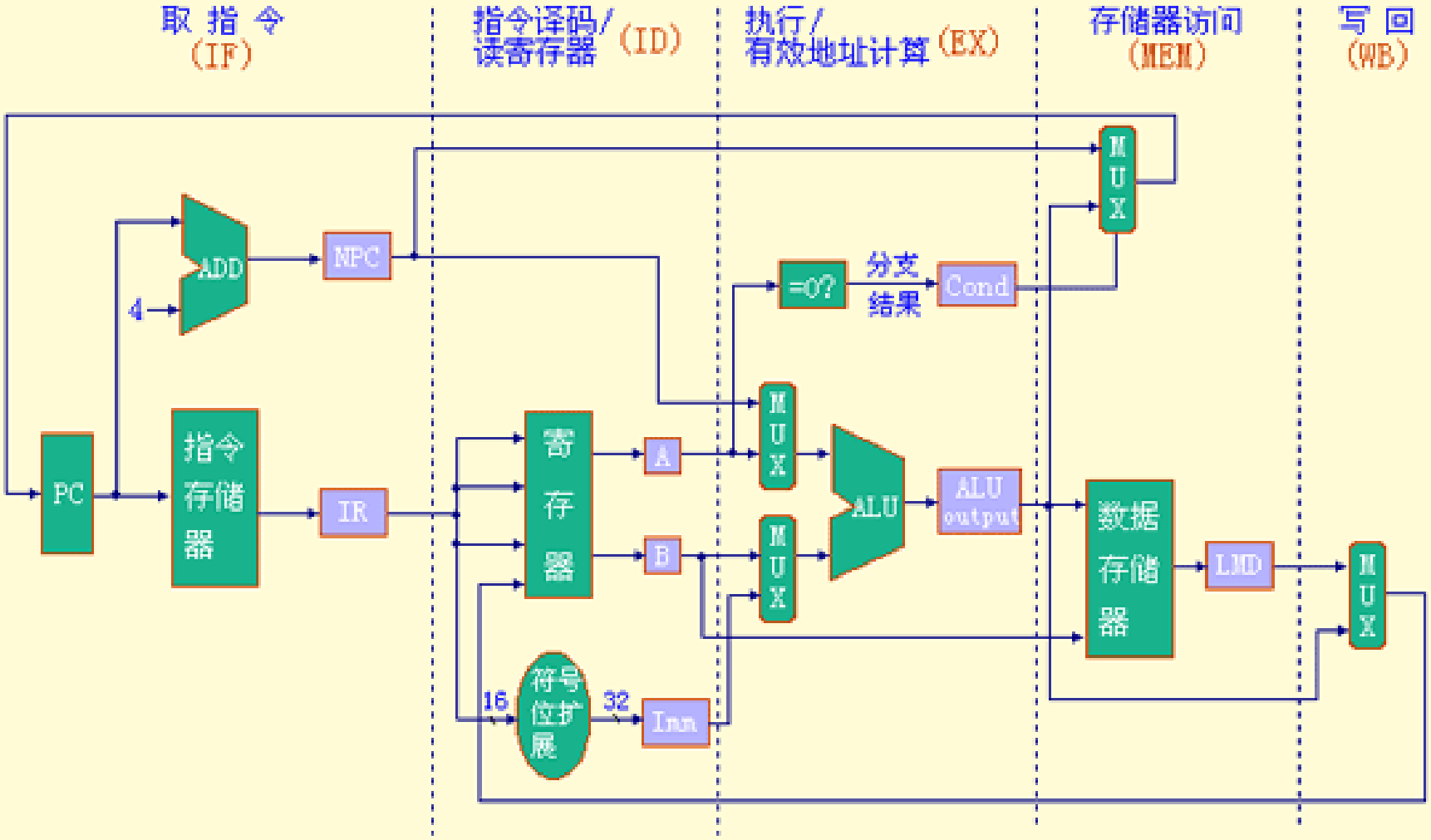
  AND R1,R2,R3

3.2.1 MIPS的一种简单实现

实现MIPS指令的一种简单数据通路

- 将指令执行划分为5个阶段
  - 取指令周期
  - 指令译码/读寄存器周期
  - 执行/有效地址计算周期
  - 存储器访问/分支完成周期
  - 写回周期

实现DLX指令的一种简单数据通路



3.2.1 MIPS的一种简单实现

1.取指令周期(IF)

操作为：

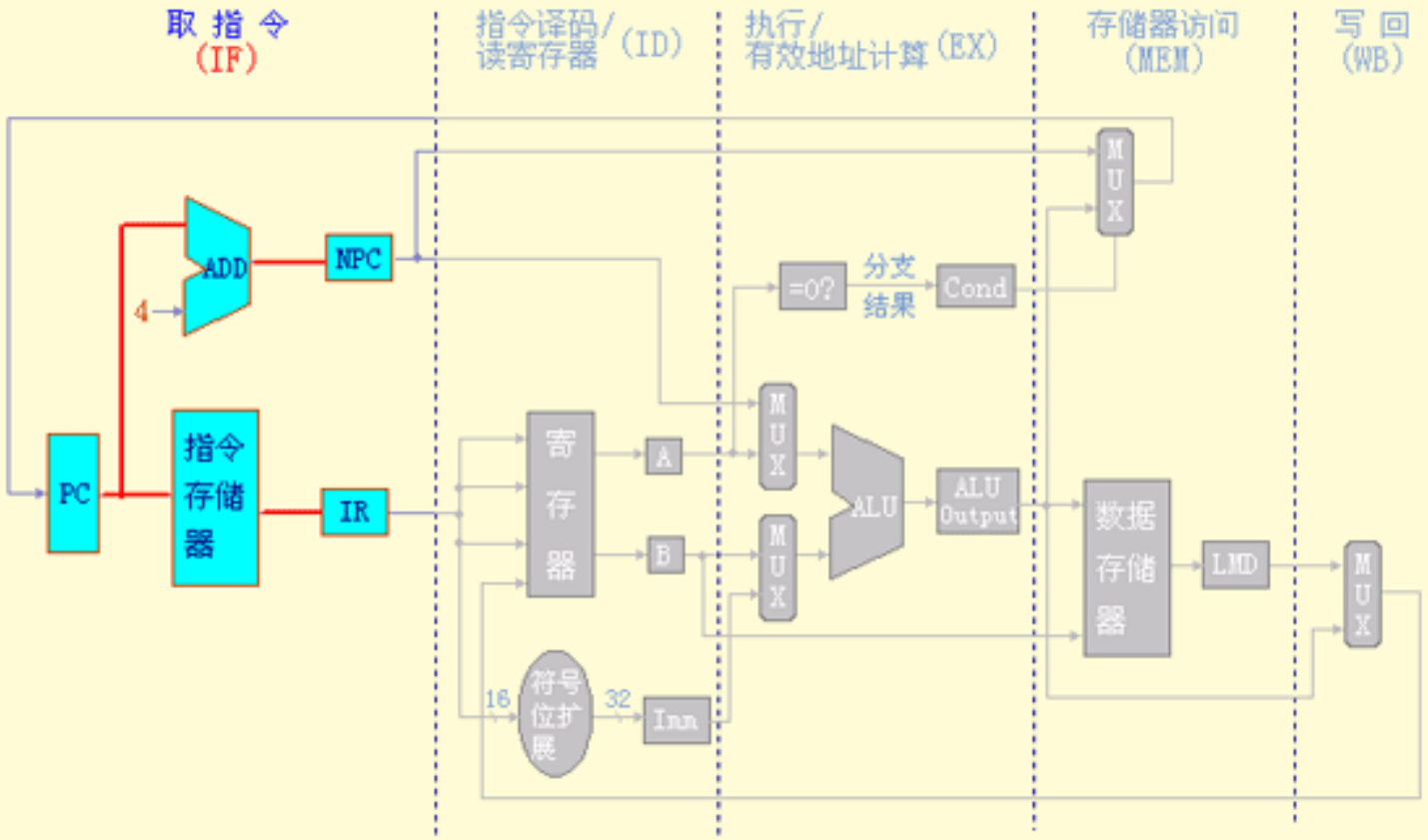
根据PC值从存储器中取出指令，并将指令送入指令寄存器IR；PC值增加4，指向顺序的下一条指令，并将下一条指令的地址放入临时寄存器NPC中。

IR←IMem[PC]

NPC←PC+4

图示

取指令周期的操作


取 指 令
(IF)
指令译码/读寄存器 (ID)
执行/有效地址计算 (EX)
存储器访问
(MEM)
写 回
(WB)
ADD
4
NPC
PC
指令存储器
IR
寄存器
A
B
16
符号位扩展
32
Imm
=0?
分支结果
Cond
MUX
MUX
MUX
ALU
ALU Output
数据存储器
LMD
MUX

3.2.1 MIPS的一种简单实现

2.指令译码/读寄存器周期(ID)

操作为：

进行指令译码，读IR寄存器（指令寄存器），按照相应寄存器号读寄存器文件，并将读出结果放入两个临时寄存器A和B中。同时对IR寄存器中内容的低16位进行符号扩展，然后将符号扩展之后的32位立即值保存在临时寄存器Imm中。

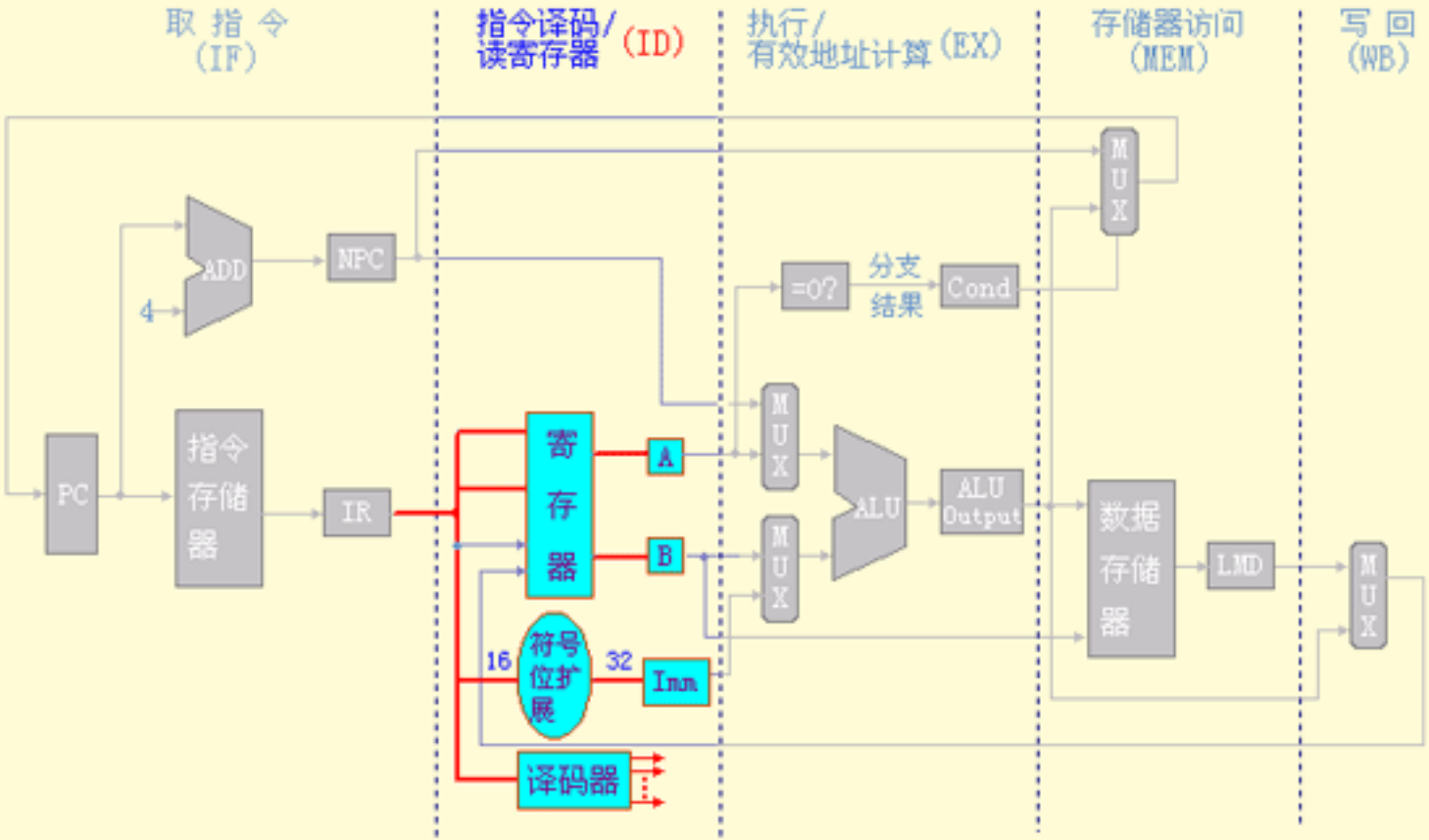
$$A \leftarrow Regs[IR_{6..10}]$$

$$B \leftarrow Regs[IR_{11..15}]$$

$$Imm \leftarrow ((IR_{16})^{16} \#\# IR_{16..31})$$

MIPS的固定字段译码技术:图示

指令译码/读寄存器周期的操作



3.2.1 MIPS的一种简单实现

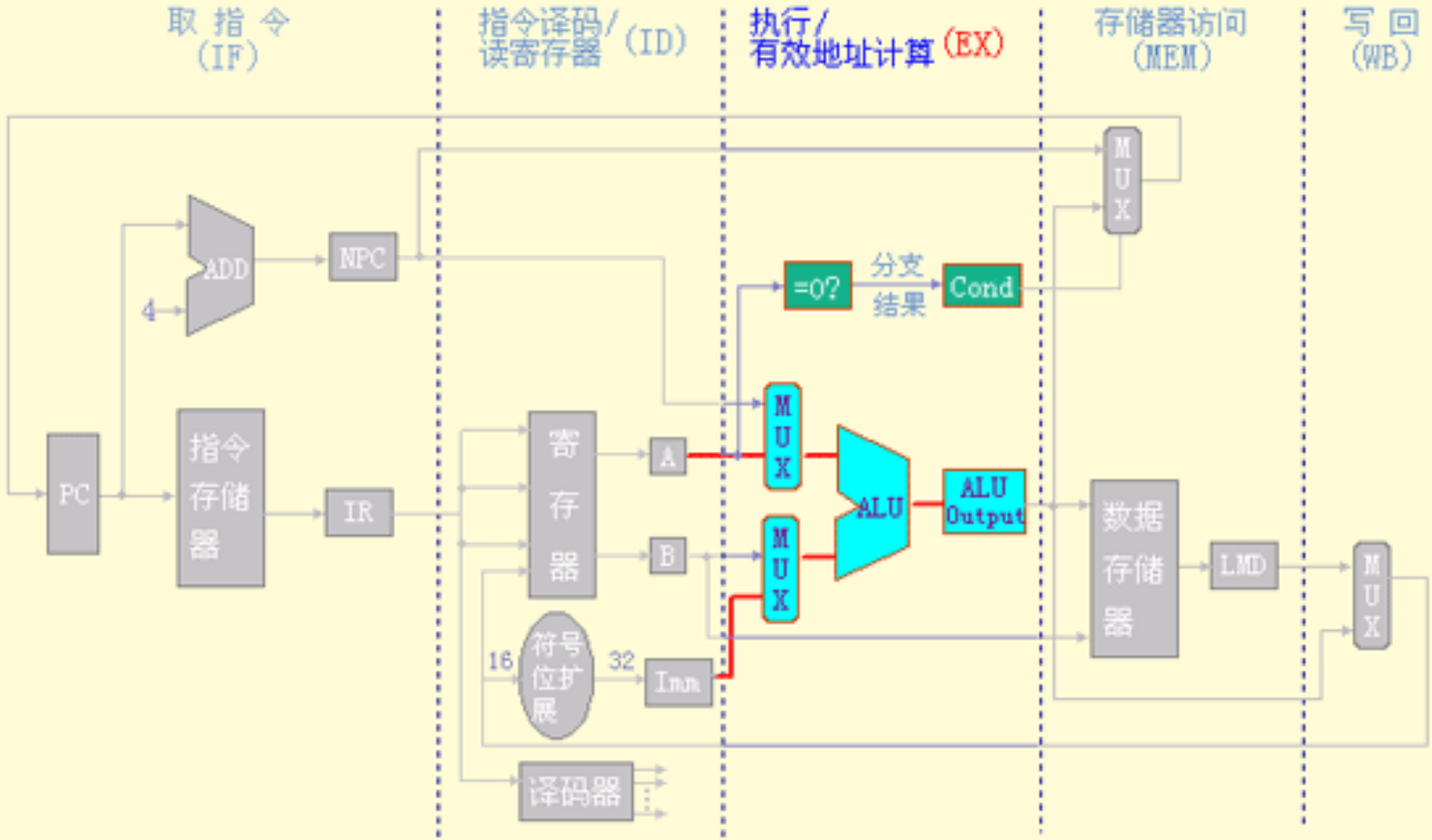
3.执行/有效地址计算周期(EX)

操作为：

存储器访问：	**ALUoutput←A+Imm**
寄存器-寄存器ALU：	**ALUoutput←A func B**
寄存器-立即值ALU:	**ALUoutput←A op Imm**
分支操作：	**ALUoutput←NPC+Imm**
	Cond←(A op 0)

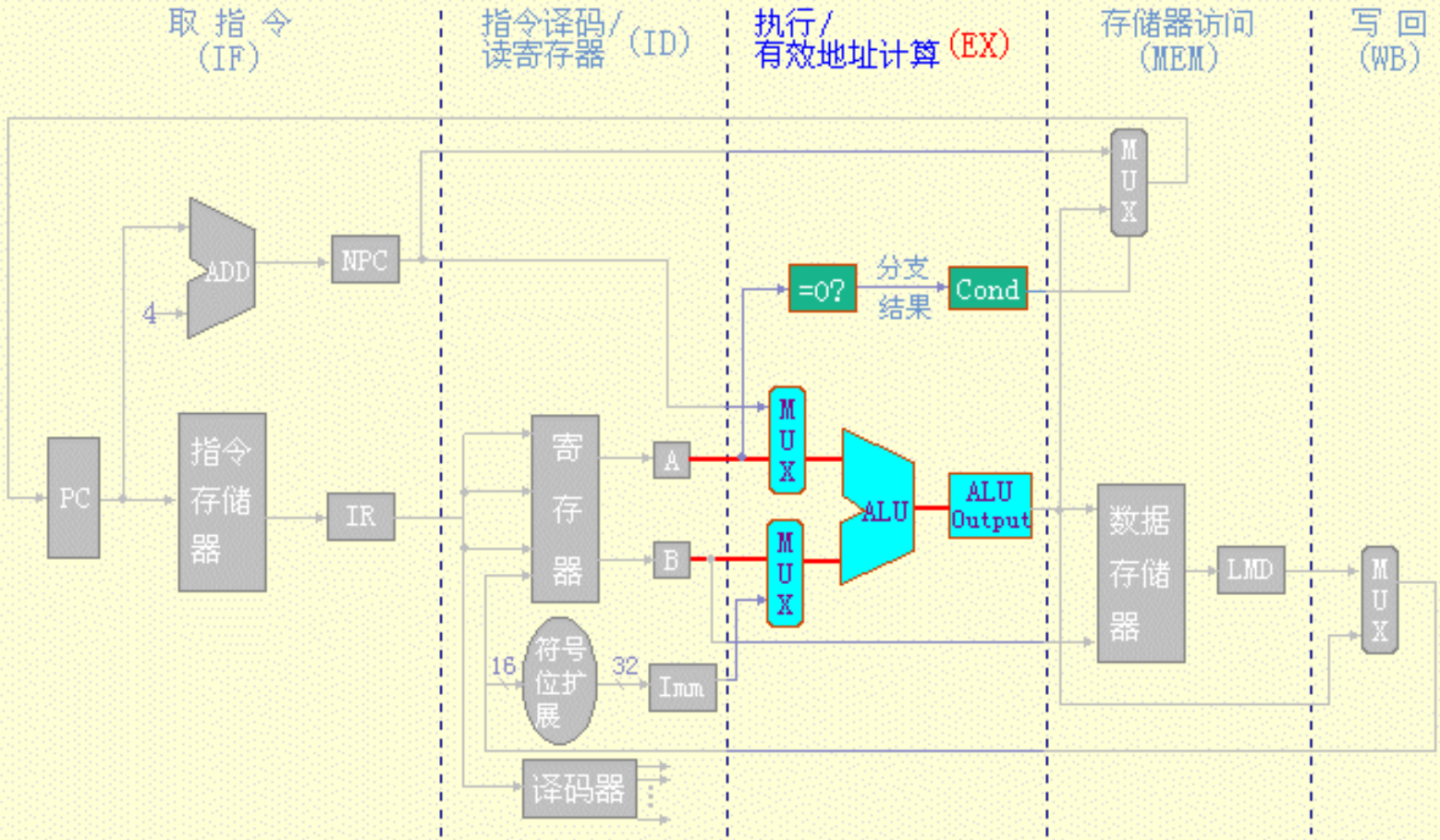
问题：为什么执行和有效地址计算可以合并？

执行／有效地址计算周期的操作
（存储器访问指令）

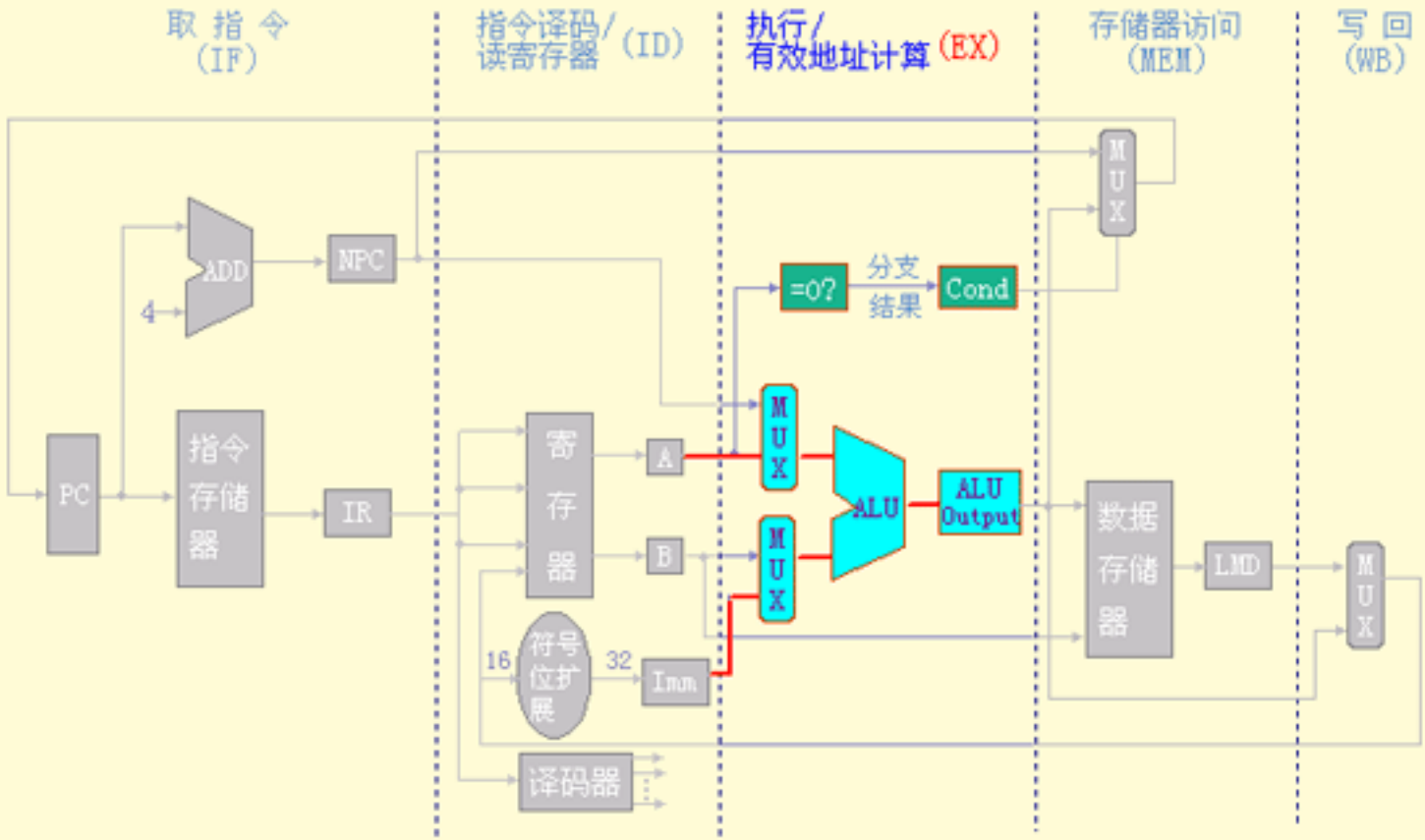


执行／有效地址计算周期的操作

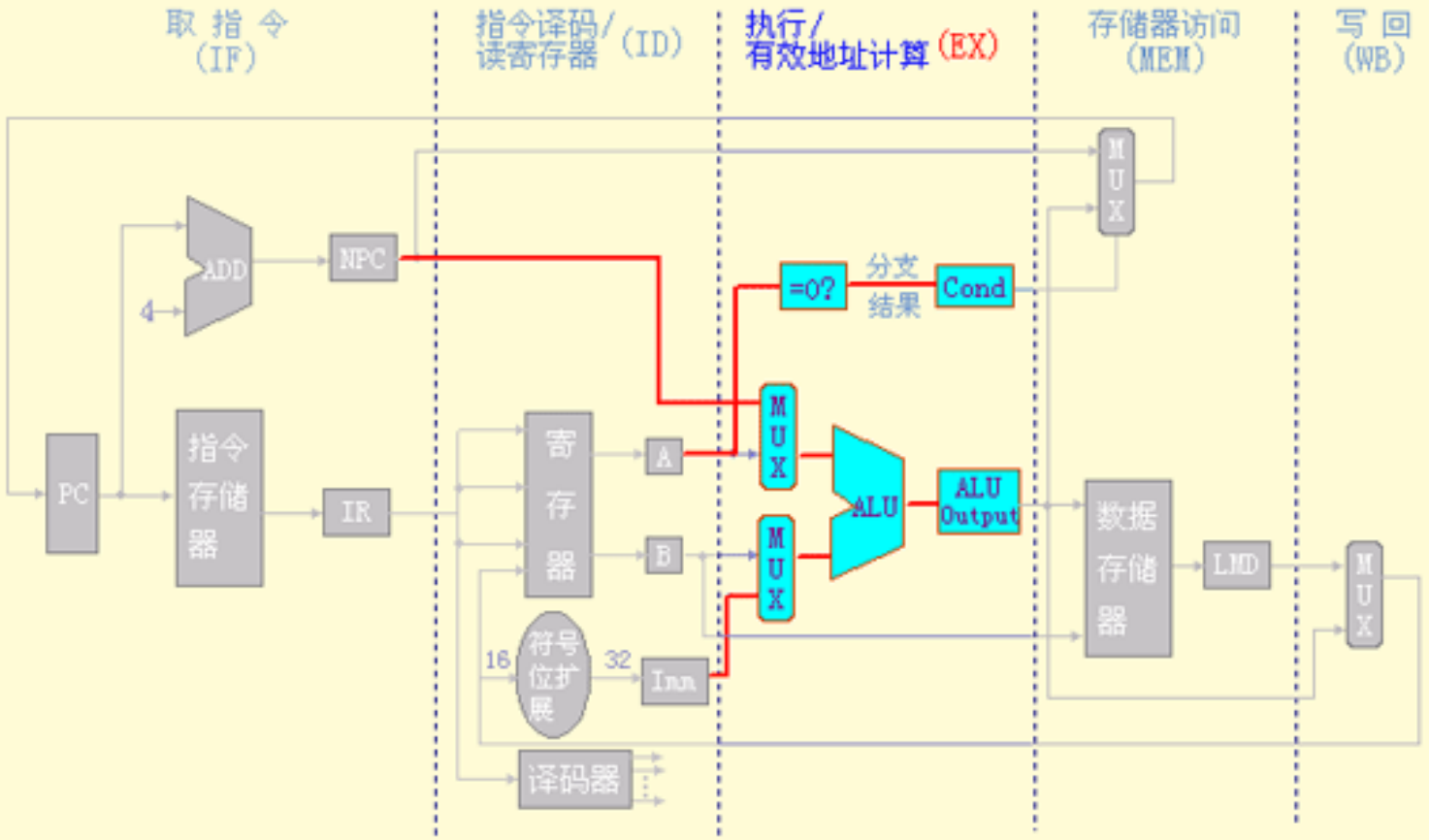
（寄存器-寄存器ALU操作指令）



执行／有效地址计算周期的操作
（寄存器-立即值ALU操作指令）



执行／有效地址计算周期的操作
（分支操作指令）



3.2.1 MIPS的一种简单实现

4.访存/分支操作完成周期(MEM)

操作为：

访存操作：

Load： **LMD←DMem[ALUoutput]**

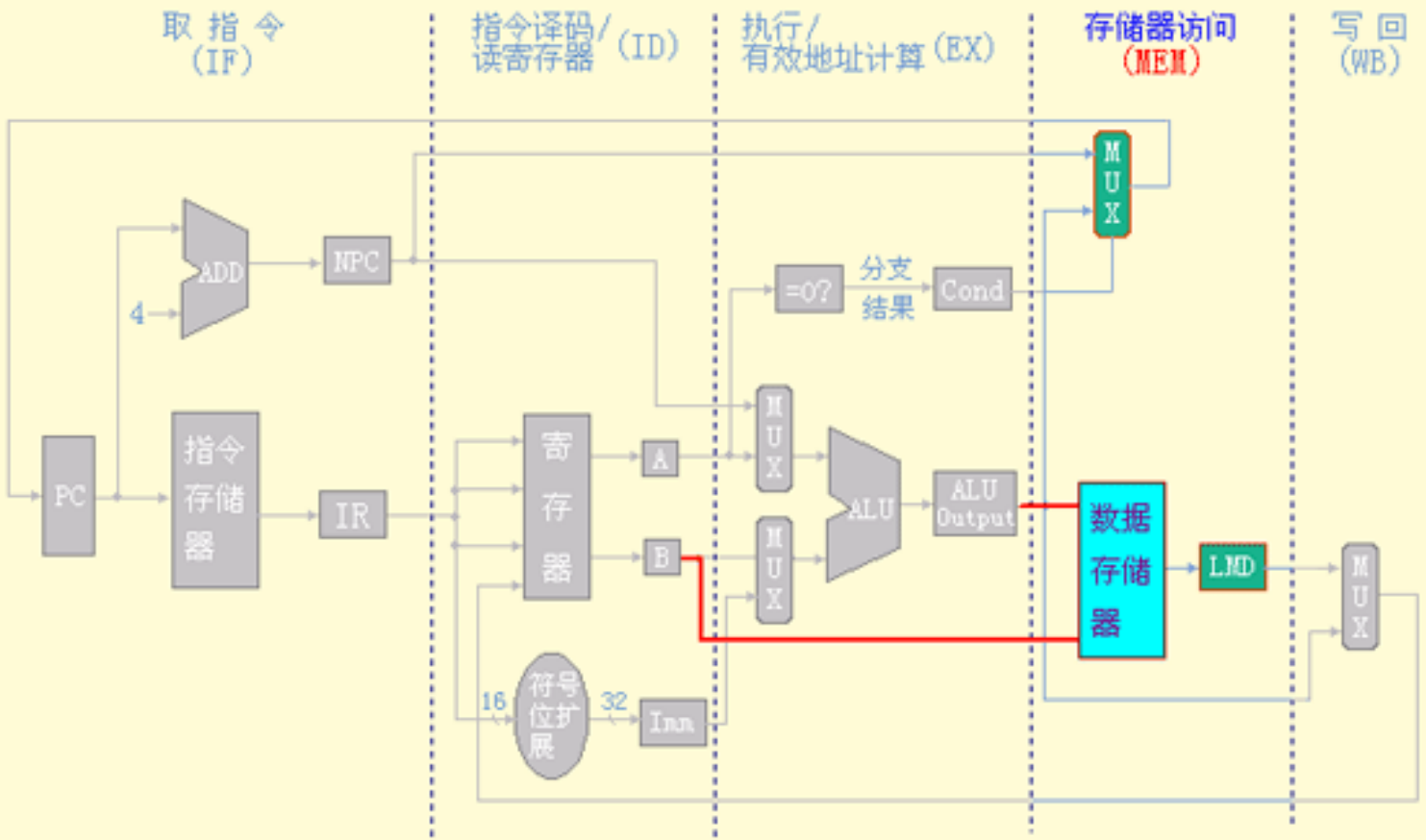
Store： **DMem[ALUoutput]←B**

分支操作： **if (Cond) PC←ALUoutput**

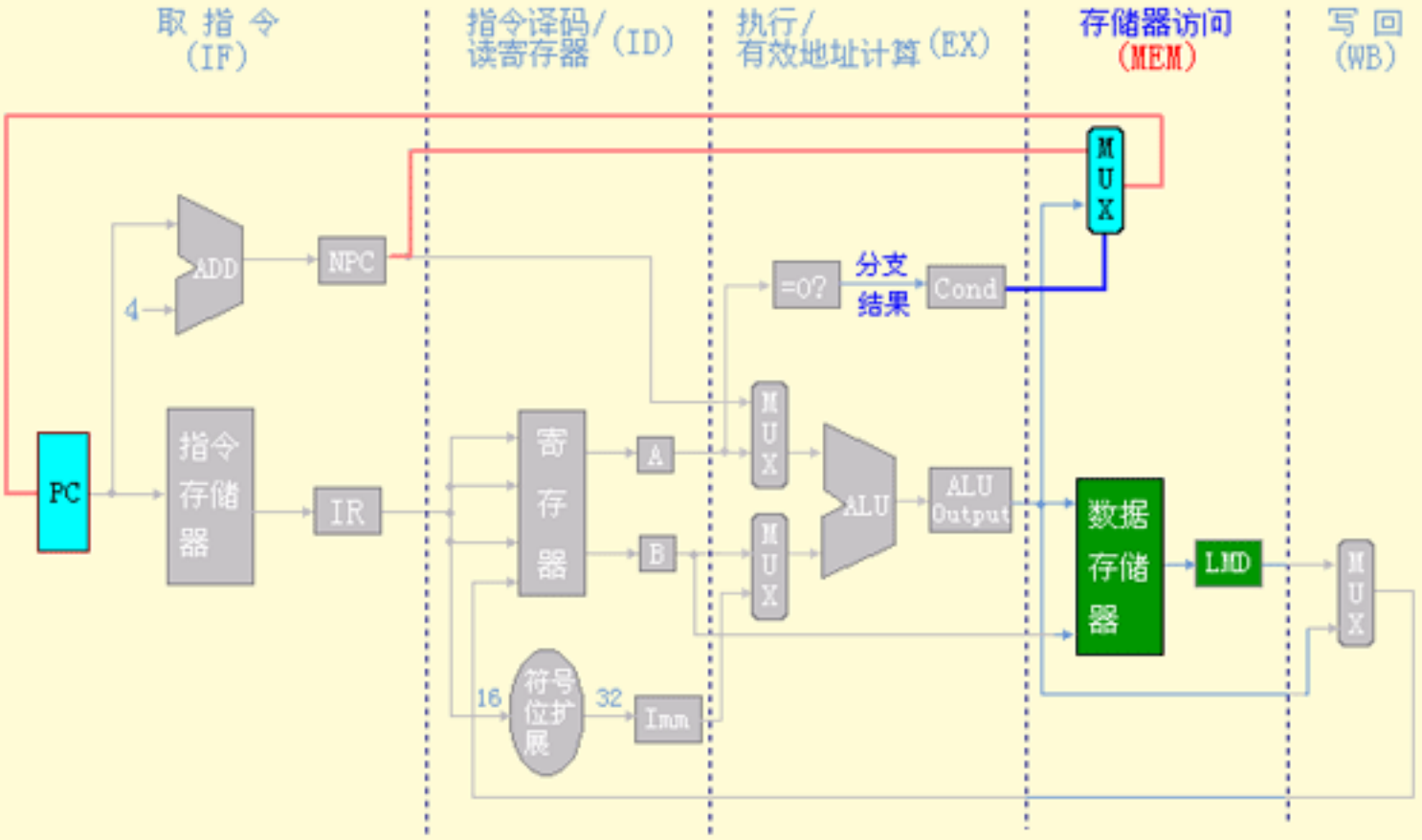
else PC←NPC

存储器访问/分支完成周期的操作

（存储器访问操作指令）



存储器访问/分支完成周期的操作
（分支操作指令）



3.2.1 MIPS的一种简单实现

5.写回周期(WB)

操作为：

- 寄存器-寄存器型ALU指令：

  $Reg[IR_{16..20}] \leftarrow ALUoutput$

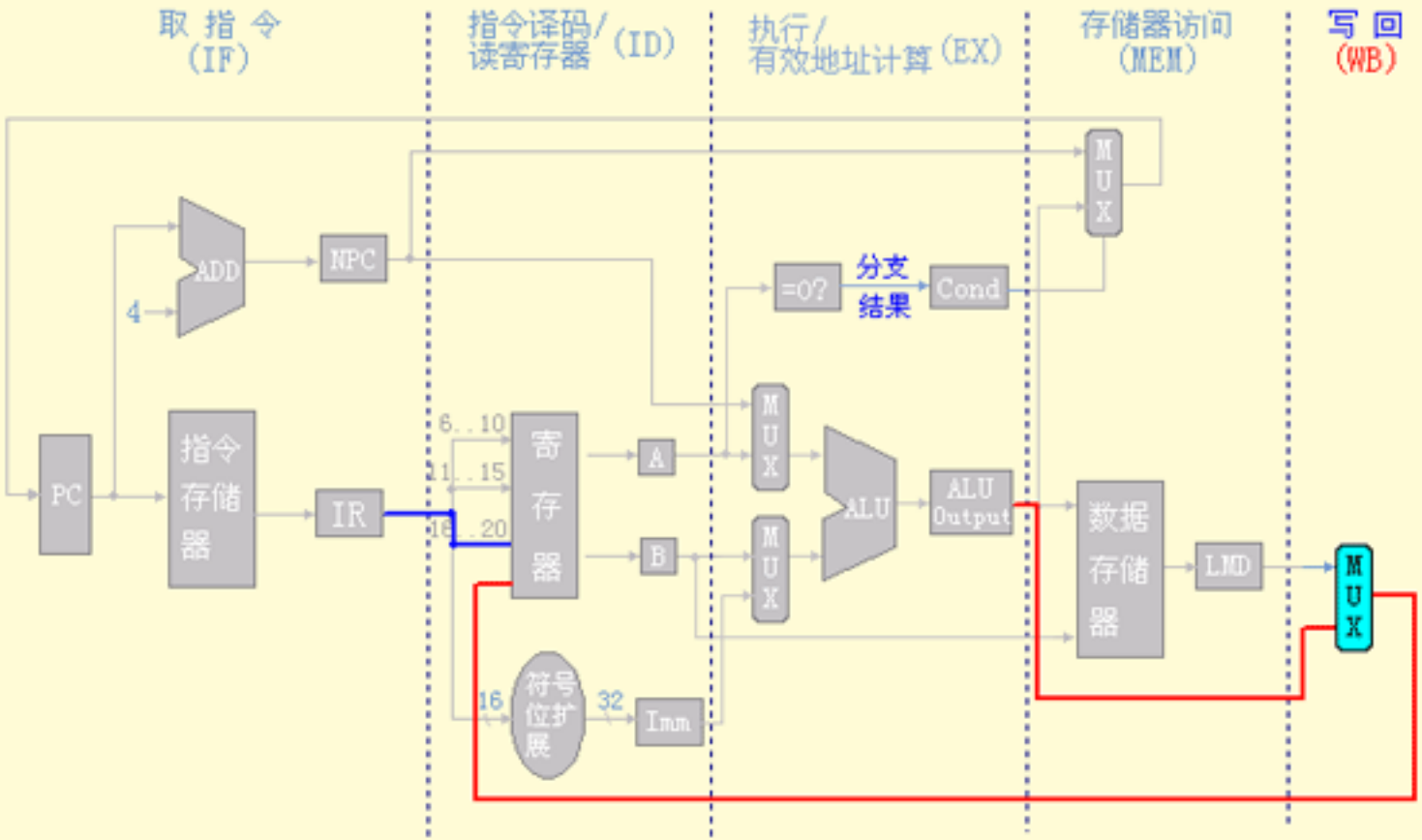
- 寄存器-立即值型ALU指令：

  $Reg[IR_{11..15}] \leftarrow ALUoutput$

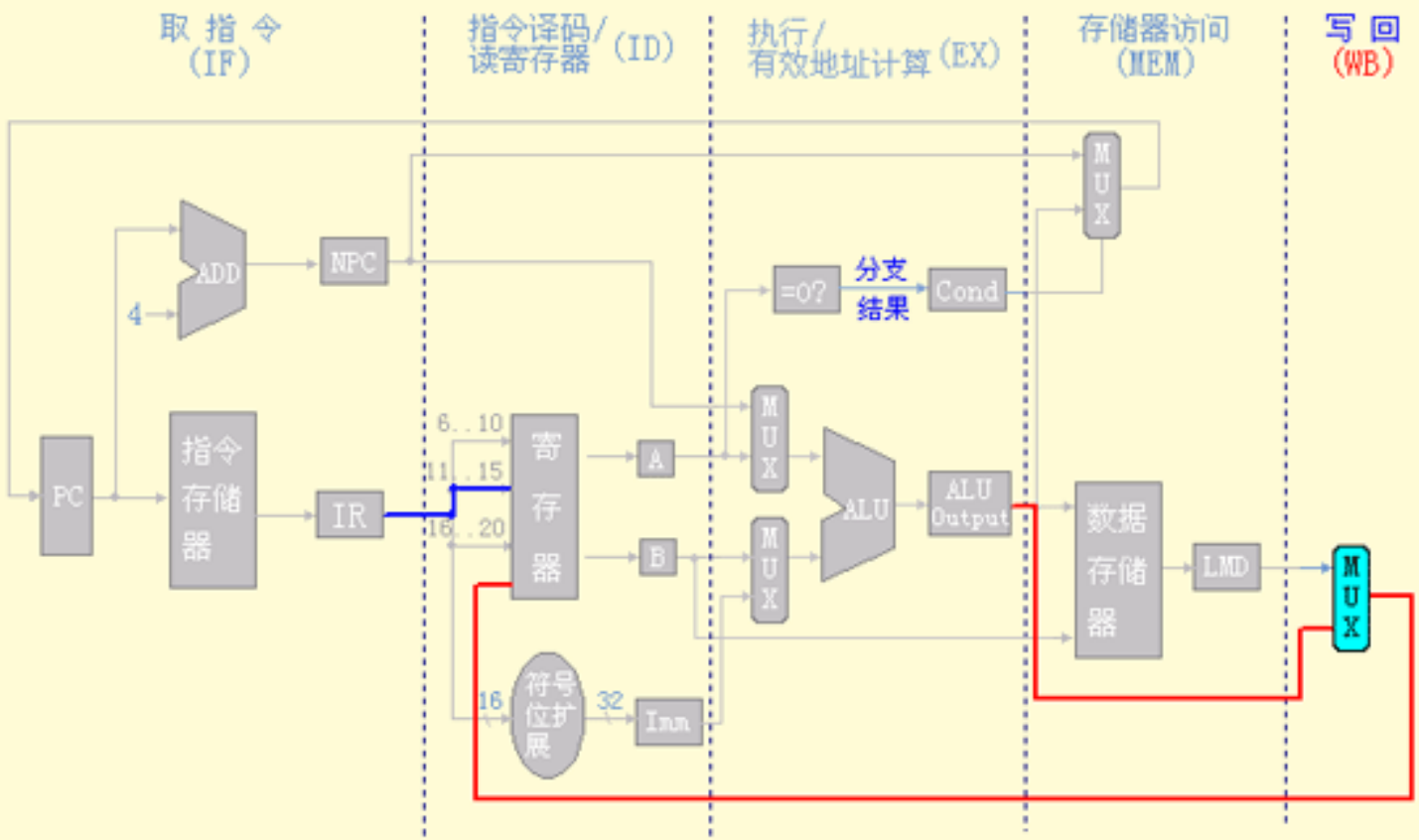
- Load指令：

  $Reg[IR_{11..15}] \leftarrow LMD$

写回周期的操作
（寄存器-寄存器型ALU指令）

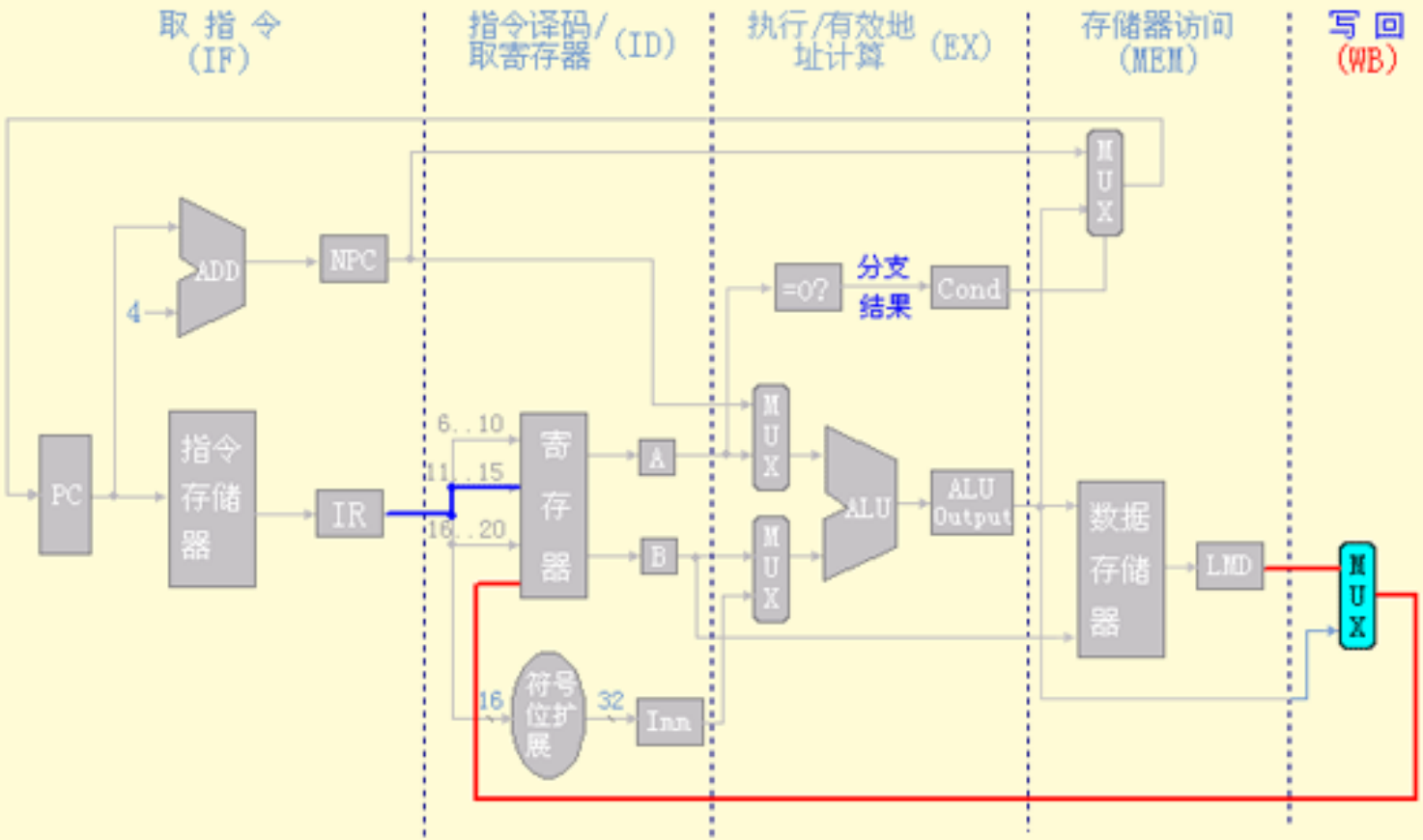


写回周期的操作
（寄存器-立即值型ALU指令）



写回周期的操作

（Load指令）



3.2.1 MIPS的一种简单实现

6.性能分析

在该数据通路上，

分支指令需要4个时钟周期

其它指令需要5个时钟周期

假设分支指令占总指令数的12%，问CPI=?

$$CPI=4\times12\%+5\times(1-12\%)=4.88$$

结论：就性能和硬件开销而言，上述实现不是一种优化实现！

3.2.1 MIPS的一种简单实现

7.改进方法

(1)在Mem周期完成ALU指令

假设ALU指令数占指令总数的44%，则在时钟周期时间不变的同时，CPI可以降低至4.44

(2)如要进一步降低CPI，可能需要延长时钟周期时间，使每个时钟周期中能够完成更多的工作

(3)采用单周期实现，可以将CPI降低为1，但时钟周期时间却会增加为原来的5倍

一般不采用这种方法，为什么？

- 流水技术

3.2.2 基本的MIPS流水线

1.一种简单的MIPS流水线

将数据通路流水化，

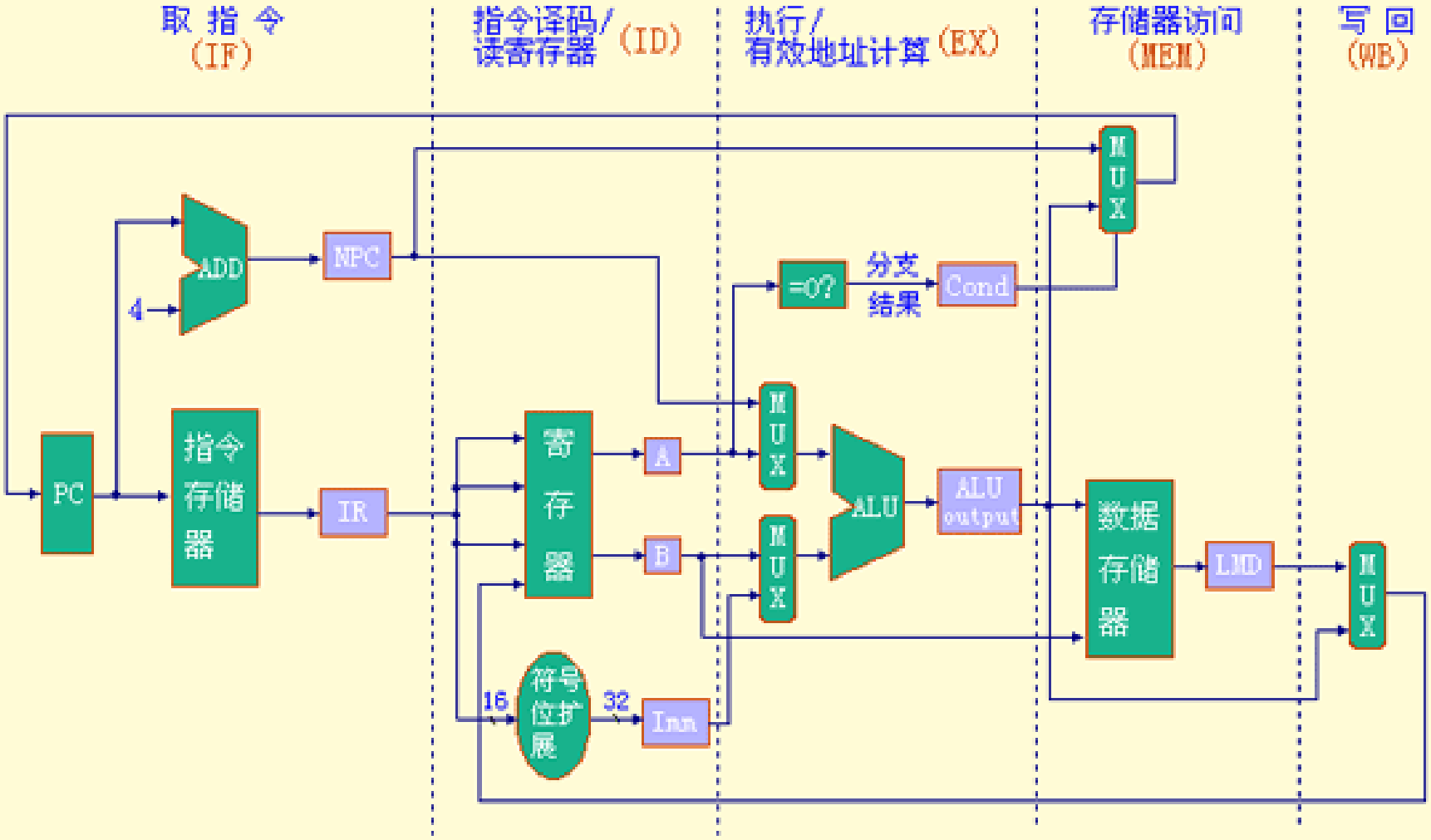
- 数据通路中的每一个周期就成为流水线的一段
- 每个时钟周期启动一条指令

——得到了一条简单的MIPS流水线。

简单MIPS流水线的流水过程：

- 时-空图
- 按时间错开的数据通路

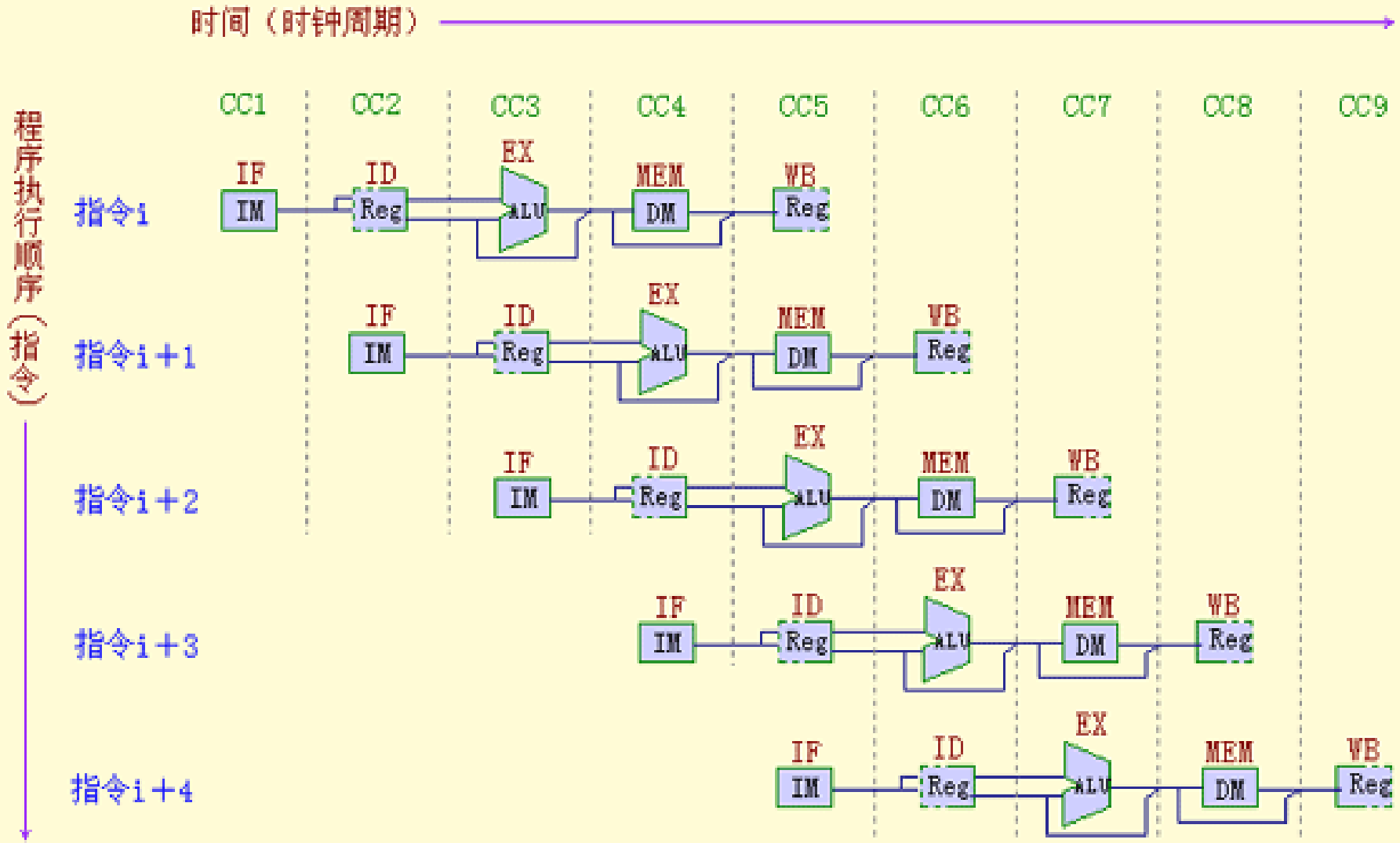
实现DLX指令的一种简单数据通路



一种简单的MIPS流水线

指令编号	时钟周期								
	1	2	3	4	5	6	7	8	9
指令i	IF	ID	EX	MEM	WB				
指令i+1		IF	ID	EX	MEM	WB			
指令i+2			IF	ID	EX	MEM	WB		
指令i+3				IF	ID	EX	MEM	WB	
指令i+4					IF	ID	EX	MEM	WB

流水线可以看成是按时间错开的数据通路序列



3.2.2 基本的MIPS流水线

2.实现流水技术应解决的一些问题

（1）应保证流水线各段不会在同一时钟周期内使用相同的通路资源。

- 例如，不能要求一个ALU既做有效地址计算，又做减法操作
- IF与Mem两个阶段都要访问存储器，怎样避免访存冲突?
- ID和WB两个阶段都要访问寄存器，是否存在冲突？怎样避免?

（2）PC计算问题

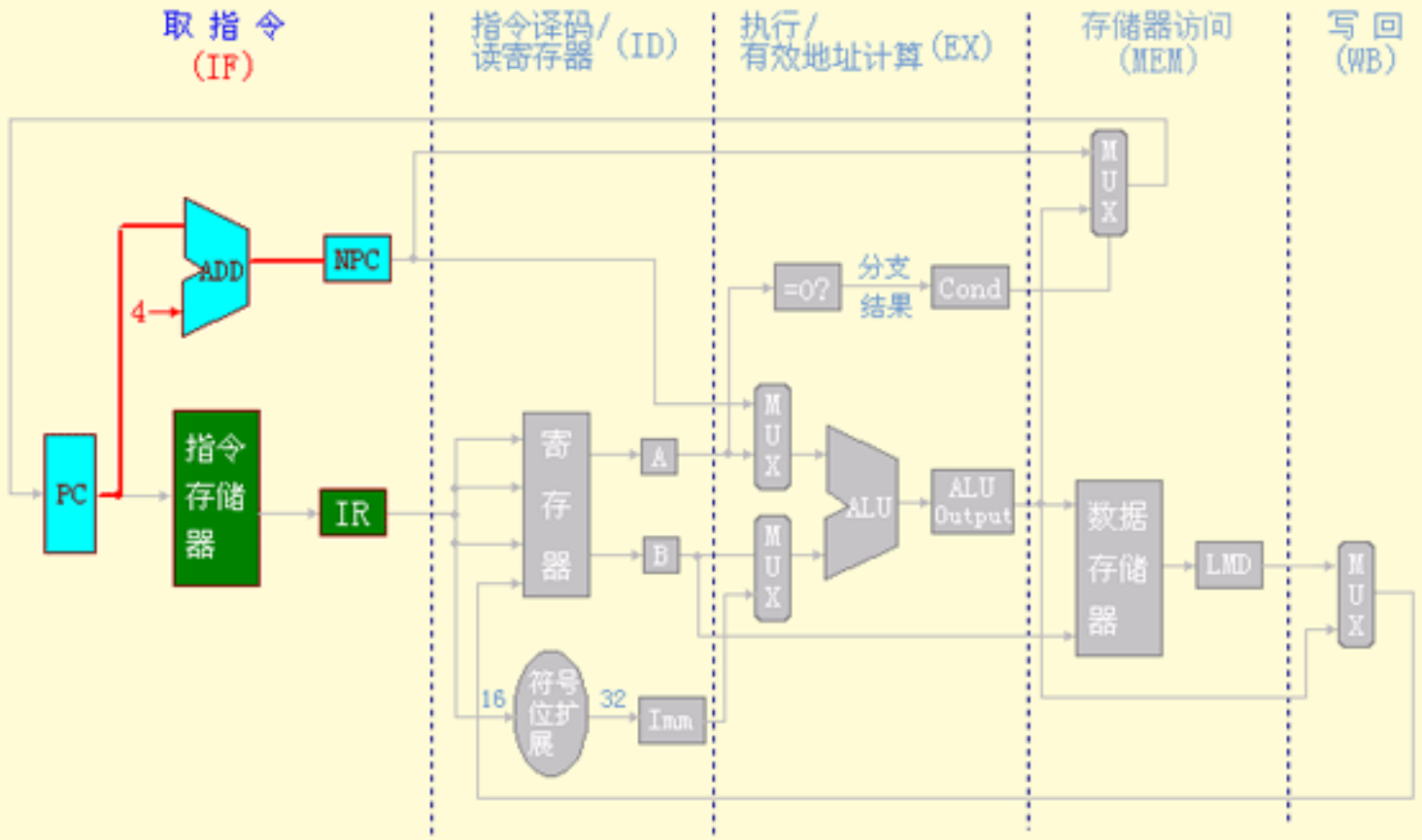
为了能够在每个时钟周期启动一条新的指令，流水线必须在IF段获得下一条指令的地址，并将其保存在PC中。

但是，分支指令会改变PC的值，而且只有在Mem段结束时，这个新值才会被写入PC，出现矛盾。

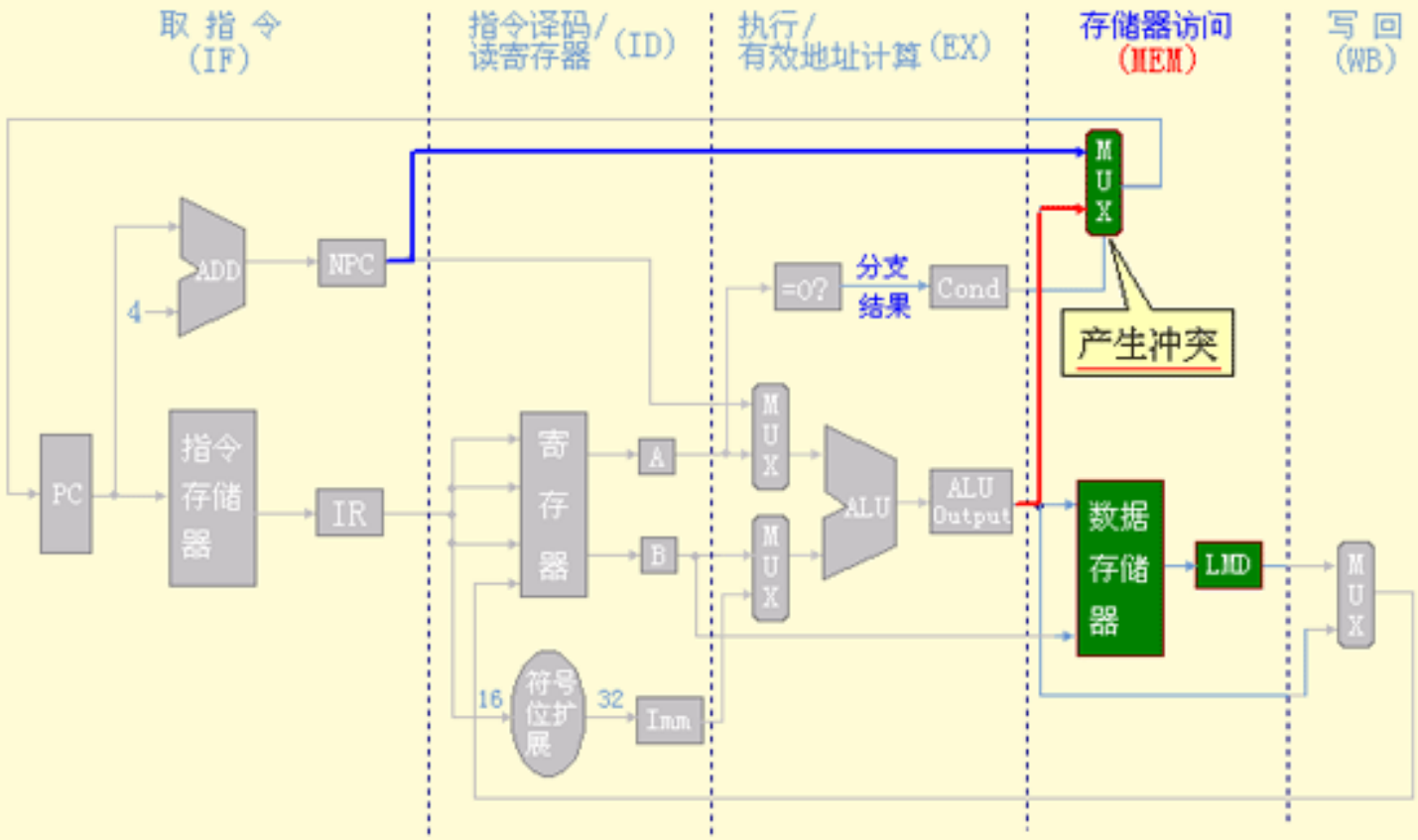
解决方法：

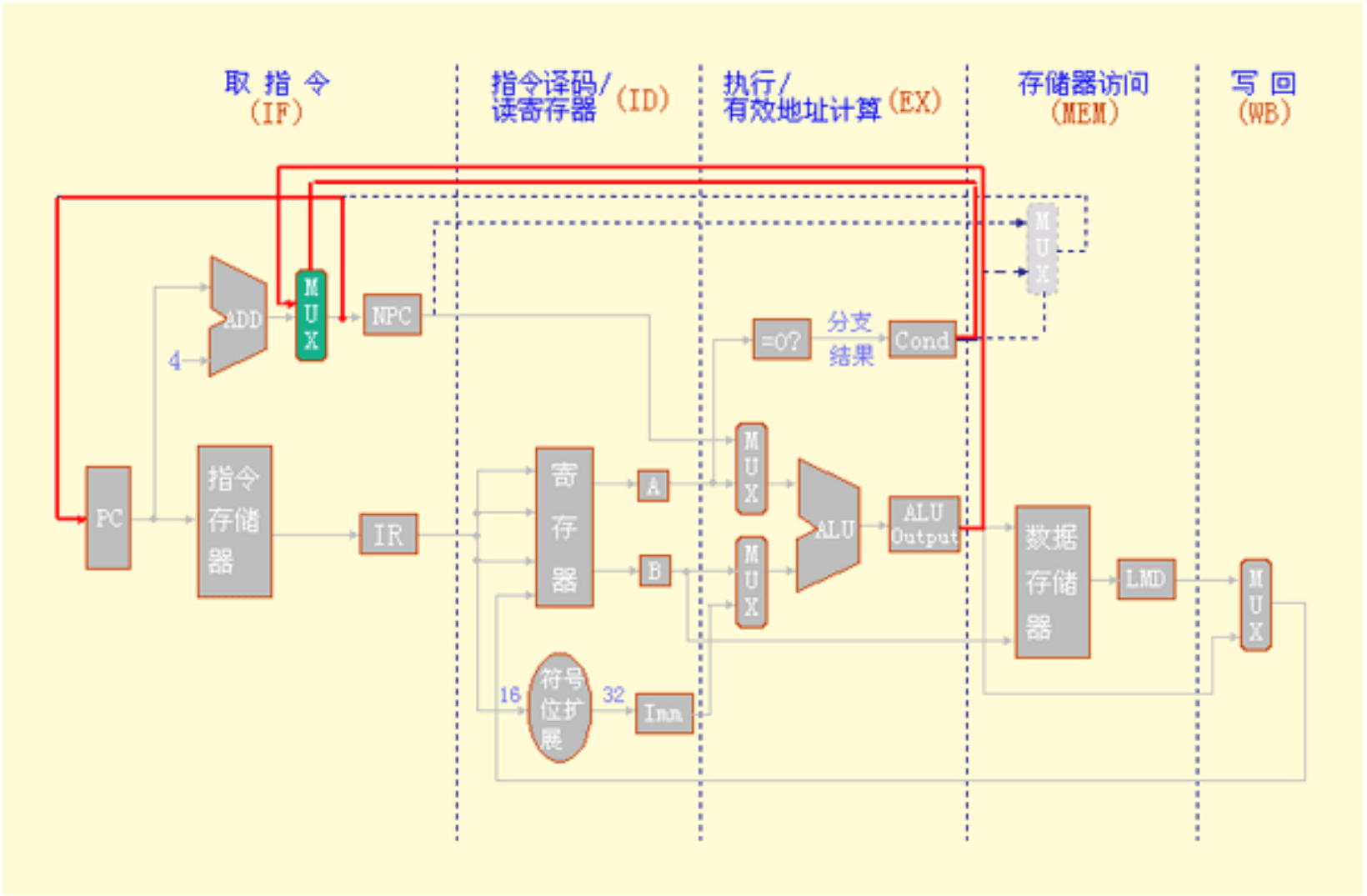
对于顺序执行，可以修改数据通路，在IF段完成PC计算。但分支指令如何处理?

PC值顺序加4


取 指 令
(IF)
指令译码/
读寄存器 (ID)
执行/
有效地址计算 (EX)
存储器访问
(MEM)
写 回
(WB)
ADD
4
NPC
PC
指令
存储
器
IR
寄
存
器
A
B
16
符号
位扩
展
32
Imm
=0?
分支
结果
Cond
M
U
X
ALU
ALU
Output
数据
存储
器
LMD

分支指令改变PC值


取 指 令
(IF)
指令译码/
读寄存器 (ID)
执行/
有效地址计算 (EX)
存储器访问
(MEM)
写 回
(WB)
M
U
X
ADD
4
NPC
=0?
分支
结果
Cond
产生冲突
PC
指令
存储
器
IR
寄
存
器
A
B
M
U
X
ALU
ALU
Output
数据
存储
器
LMD
M
U
X
16
符号
位扩
展
32
Imm

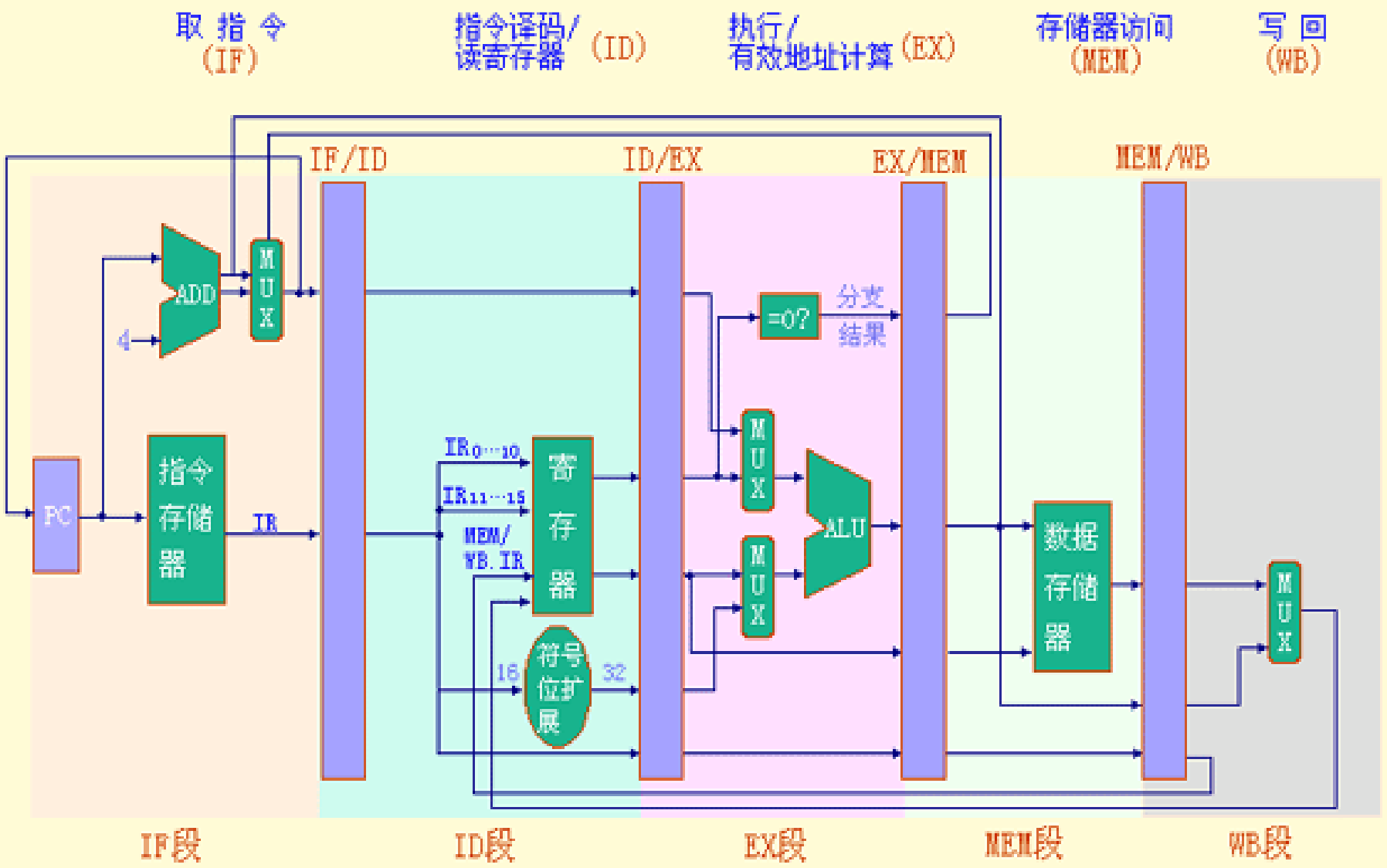

取 指 令
(IF)
指令译码/
读寄存器 (ID)
执行/
有效地址计算 (EX)
存储器访问
(MEM)
写 回
(WB)
ADD
4
M
U
X
NPC
PC
指令
存储
器
IR
寄
存
器
A
B
16
符号
位扩
展
32
Imm
=0?
分支
结果
Cond
M
U
X
M
U
X
ALU
ALU
Output
数据
存储
器
LMD
M
U
X
M
U
X

（3）合理划分流水段，每段内的操作都必须在一个时钟周期内完成。

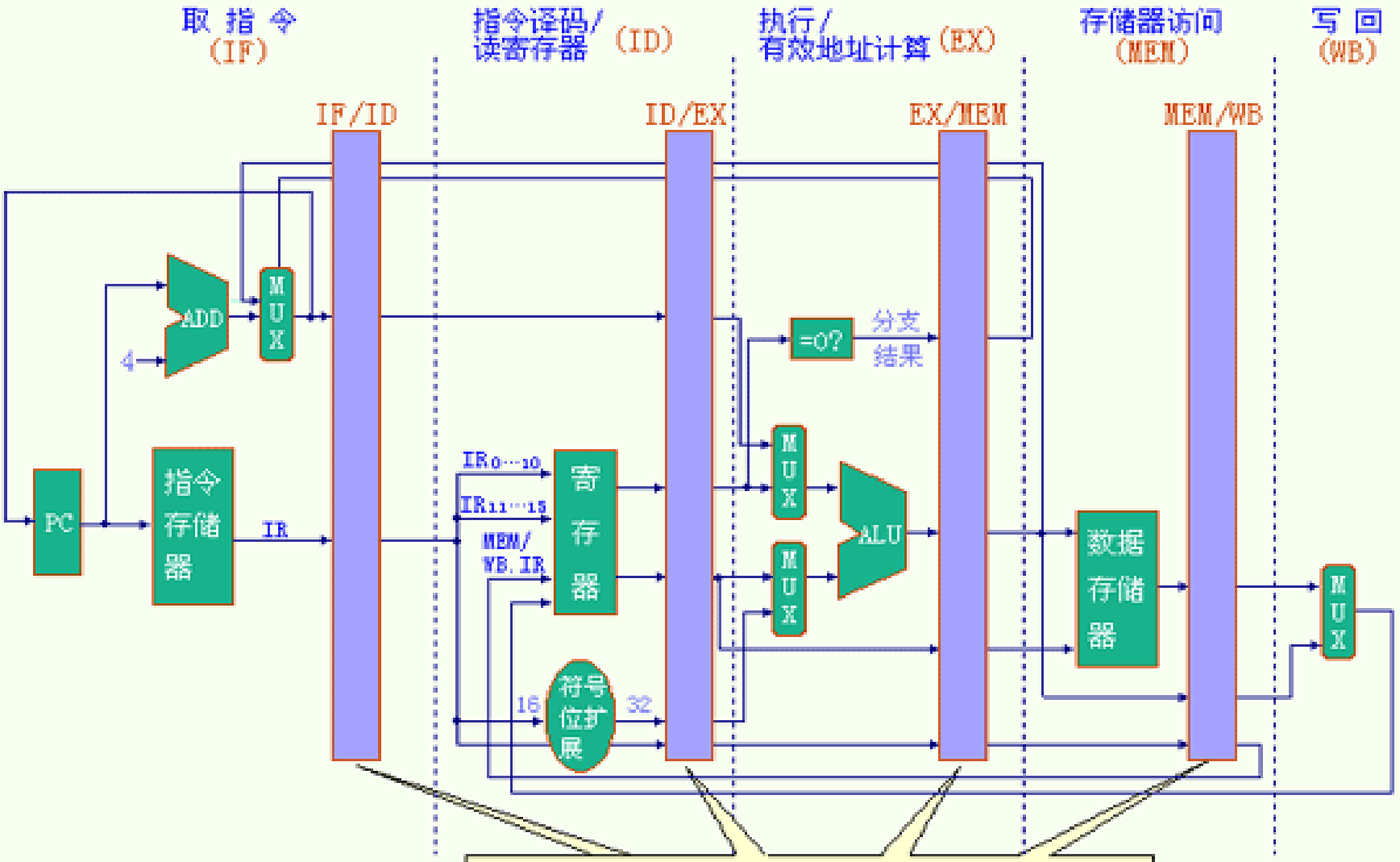
（4）流水线寄存器设计

- 为防止寄存器中的值在为流水线中某条指令所用时被流水线中其它的指令所重写，可在流水线各段之间设置流水线寄存器文件，也称锁存器。
- 流水线寄存器文件的命名
- 段A与B之间的流水线寄存器文件称为A/B
- 流水线寄存器的作用
- 流水线寄存器文件的构成

DLX流水线的数据通路

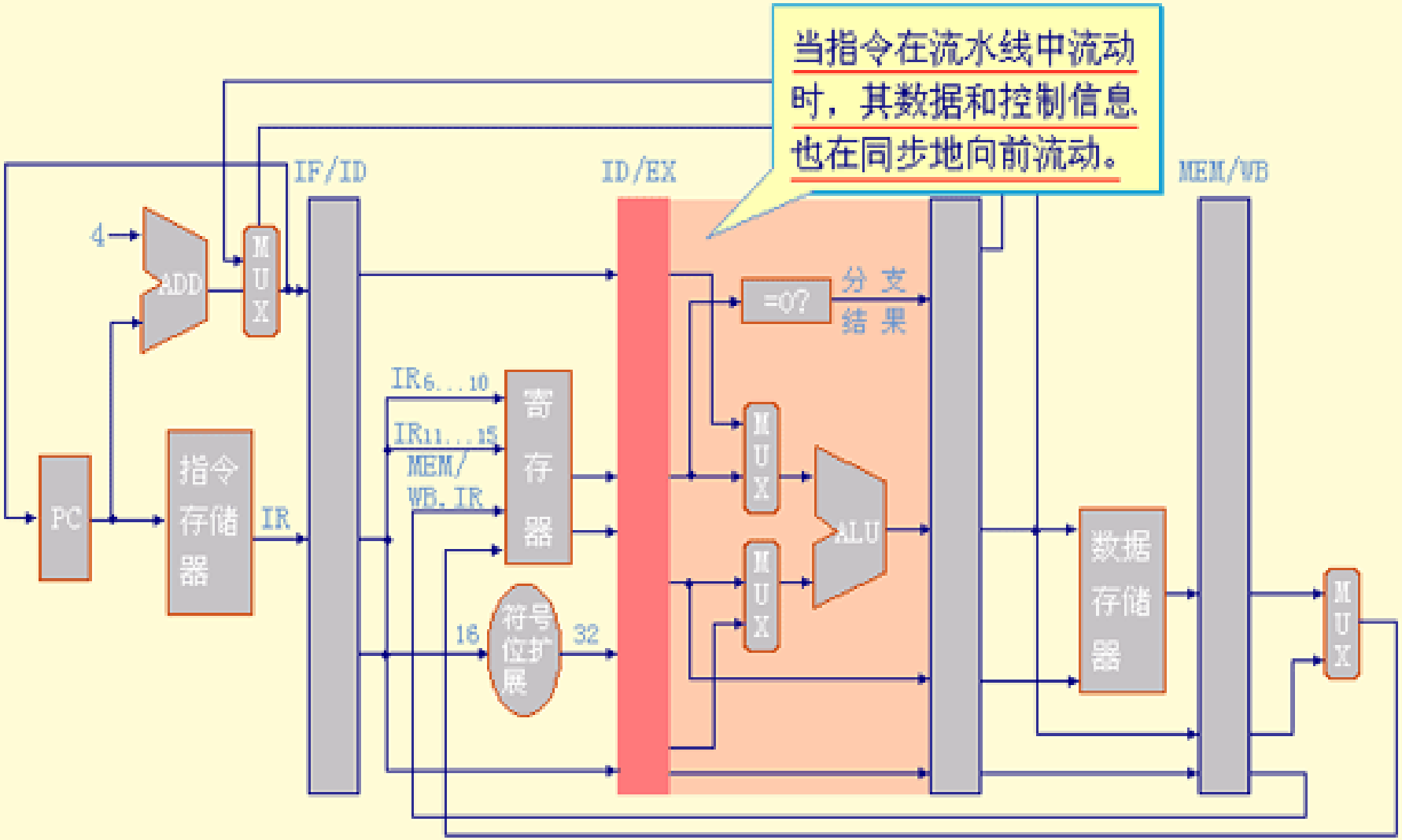


DLX流水线寄存器的命名

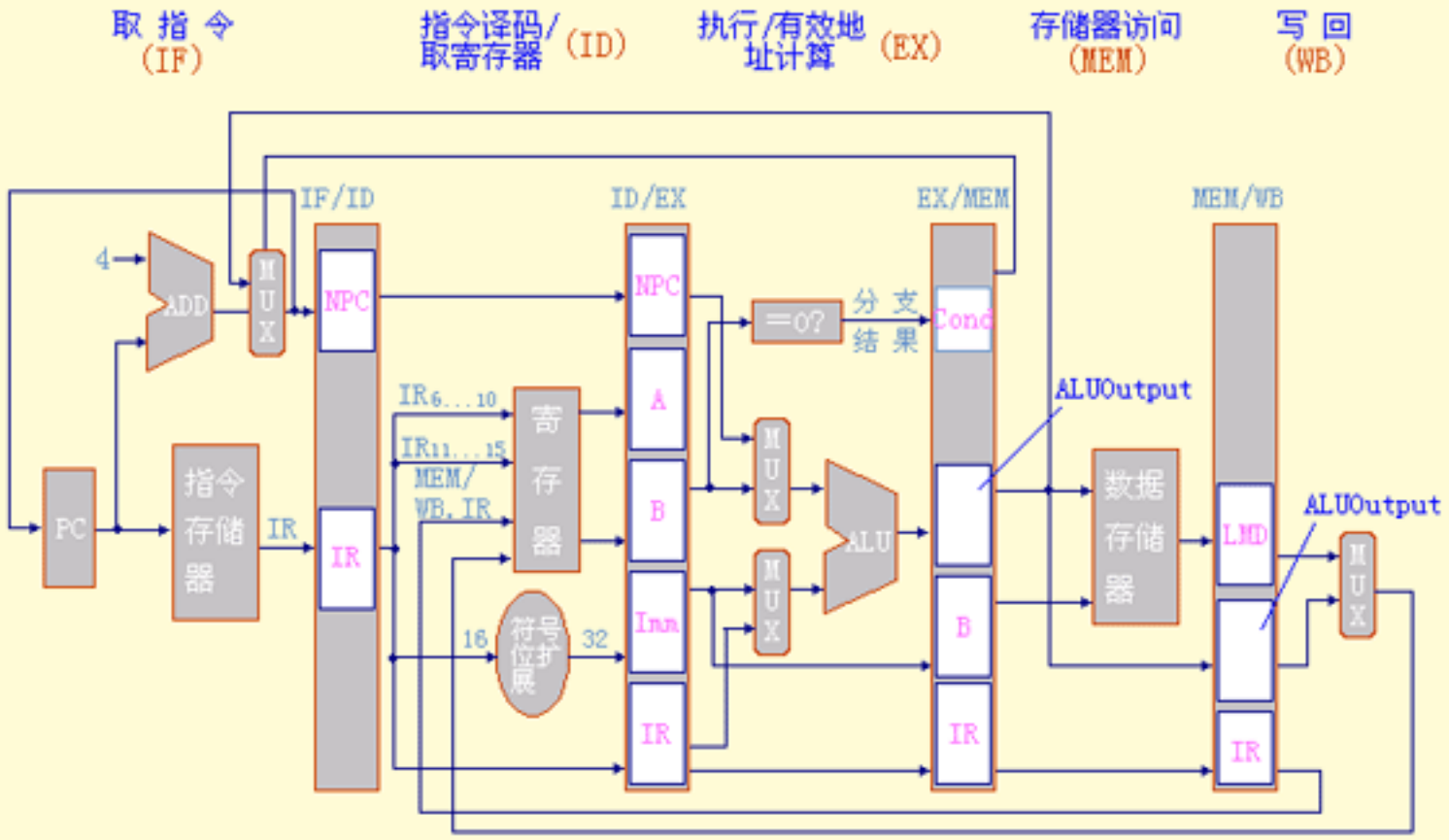


流水线寄存器组中所包含的寄存器的命名类似于域的命名。即采用：寄存器组．寄存器名

流水线寄存器的作用


当指令在流水线中流动时，其数据和控制信息也在同步地向前流动。
IF/ID
ID/EX
MEM/WB
4
ADD
MUX
PC
指令存储器
IR
IR6...10
IR11...15
MEM/WB.IR
寄存器
16
符号位扩展
32
=0?
分支结果
MUX
MUX
ALU
数据存储器
MUX

流水线寄存器的构成


取 指 令 (IF)
指令译码/取寄存器 (ID)
执行/有效地址计算 (EX)
存储器访问 (MEM)
写 回 (WB)
IF/ID
ID/EX
EX/MEM
MEM/WB
4
ADD
MUX
NPC
PC
指令存储器
IR
IR6...10
IR11...15
MEM/WB.IR
寄存器
16
符号位扩展
32
A
B
Imm
IR
=0?
分支结果
Cond
ALU
ALUOutput
数据存储器
LMD
ALUOutput

3.MIPS流水线的操作

在任一时刻，流水中的指令仅在流水线中的某一段内执行操作。

因此，只要知道每一流水段在各指令下进行何种操作，就知道了整个流水线的操作。

下表给出了MIPS流水线中每一段的操作

MIPS流水线的每个流水段的操作

流水段	任何指令类型		
IF	IF/ID.IR ← Mem[PC] IF/ID.NPC,PC ← (if EX/MEM.cond {EX/MEM.ALUOutput} else {PC+4});		
ID	ID/EX.A ← Regs[IF/ID.$IR_{6...10}$]; ID/EX.B ← Regs[IF/ID.$IR_{11...15}$]; ID/EX.NPC ← IF/ID.NPC; ID/EX.IR ← IF/ID.IR; ID/EX.Imm ← $(IR_{16})^{16}$##$IR_{16...31}$;		
	ALU 指令	Load/Store 指令	分支指令
EX	EX/MEM.IR ← ID/EX.IR; EX/MEM.ALUOutput ← ID/EX.A op ID/EX.B 或 EX/MEM.ALUOutput ← ID/EX.A op ID/EX.Imm; EX/MEM.cond ← 0;	EX/MEM.IR ← ID/EX.IR; EX/MEM.B ←ID/EX.B EX/MEM.ALUOutput ← ID/EX.A + ID/EX.Imm; EX/MEM.cond ← 0;	EX/MEM.ALUOutput← ID/EX.NPC+ID/EX.Imm; EX/MEM.cond ← (ID/EX.A op 0);

MIPS流水线的每个流水段的操作（续）

流水段	任何指令类型		
	ALU 指令	Load/Store 指令	分支指令
MEM	MEM/WB.IR ←EX/MEM.IR; MEM/WB.ALUOutput ← EX/MEM.ALUOutput;	MEM/WB.IR ← EX/MEM.IR; MEM/WB.LMD ← Mem[EX/MEM.ALUOutput]; 或 Mem[EX/MEM.ALUOutput] ← EX/MEM.B;	
WB	Regs[MEM/WB.$IR_{16...20}$] ← MEM/WB.ALUOutput; 或 Regs[MEM/WB.$IR_{11...15}$] ← MEM/WB.ALUOutput;	Regs[MEM/WB.$IR_{11...15}$] ← MEM/WB.LMD;	

4.MIPS流水线中多路选择器的控制

主要是确定如何控制那四个多路选择器：

- ALU输入端的两个MUX由ID/EX.IR所指出的指令类型控制
- IF段的MUX由EX/MEM.Cond域的值控制
- WB段的MUX由当前指令类型(Load/ALU)控制

DLX 流水线中
对多路寄存器MUX 的控制