计算机组织与体系结构

第十四讲

计算机科学与技术学院

舒燕君

Recap

- 流水线的性能（最大和实际）
  - ✓ 吞吐率、加速比、效率
  - ✓ 流水线性能的若干问题
- 流水线的冲突
  - ✓ 产生的原因（指令之间的关联）
  - ✓ 冲突的分类（结构冲突、数据冲突、控制冲突）
- 流水线的结构冲突
  - ✓产生的原因（流水线中的结构瓶颈）
  - ✓避免结构冲突的办法
  - ✓有些流水线的设计允许结构冲突的存在
- 流水线的数据冲突
  - ✓通过定向减少数据冲突带来的暂停

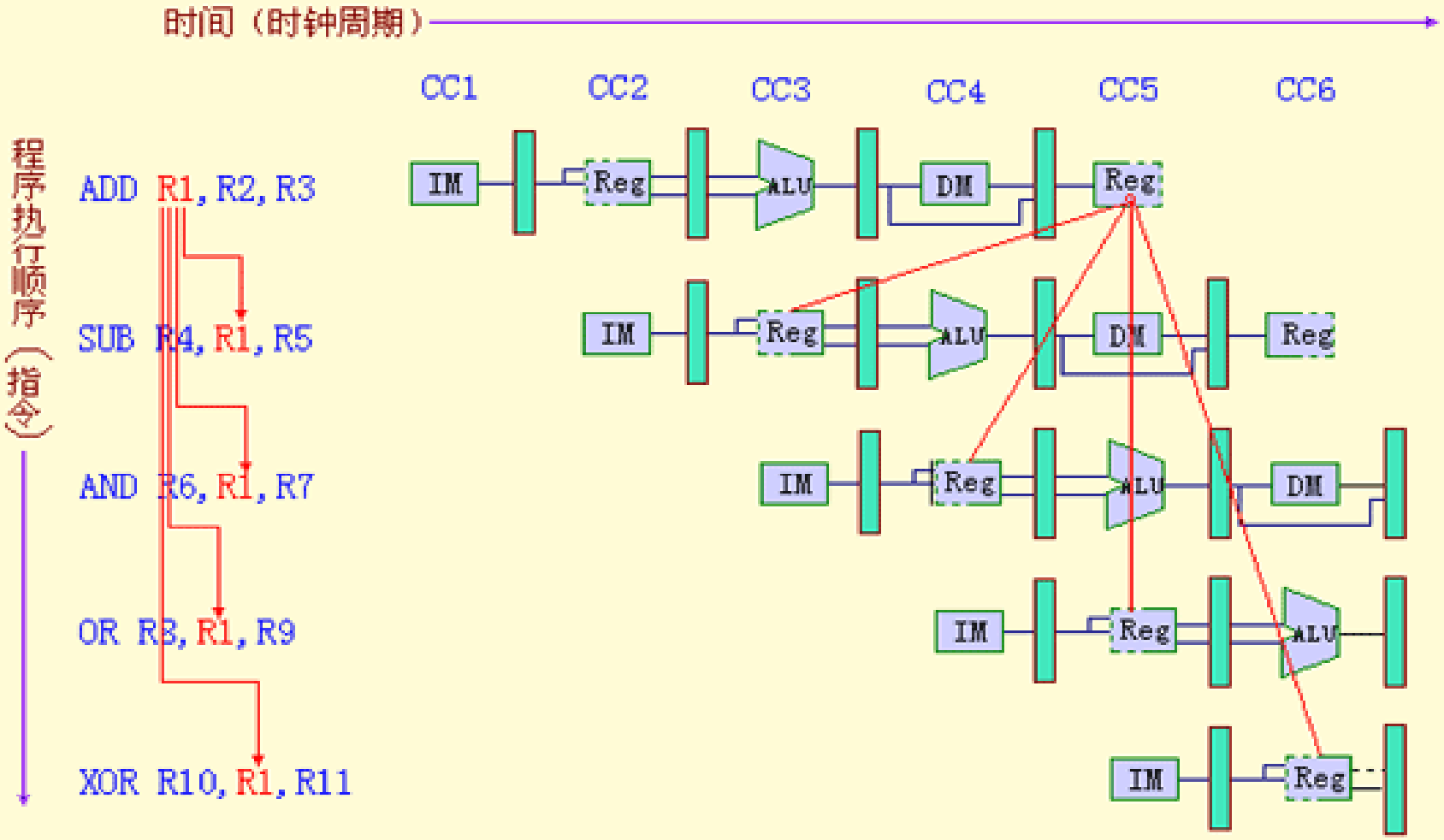
6.3　流水线中的冲突

6.3.1 流水线的结构冲突

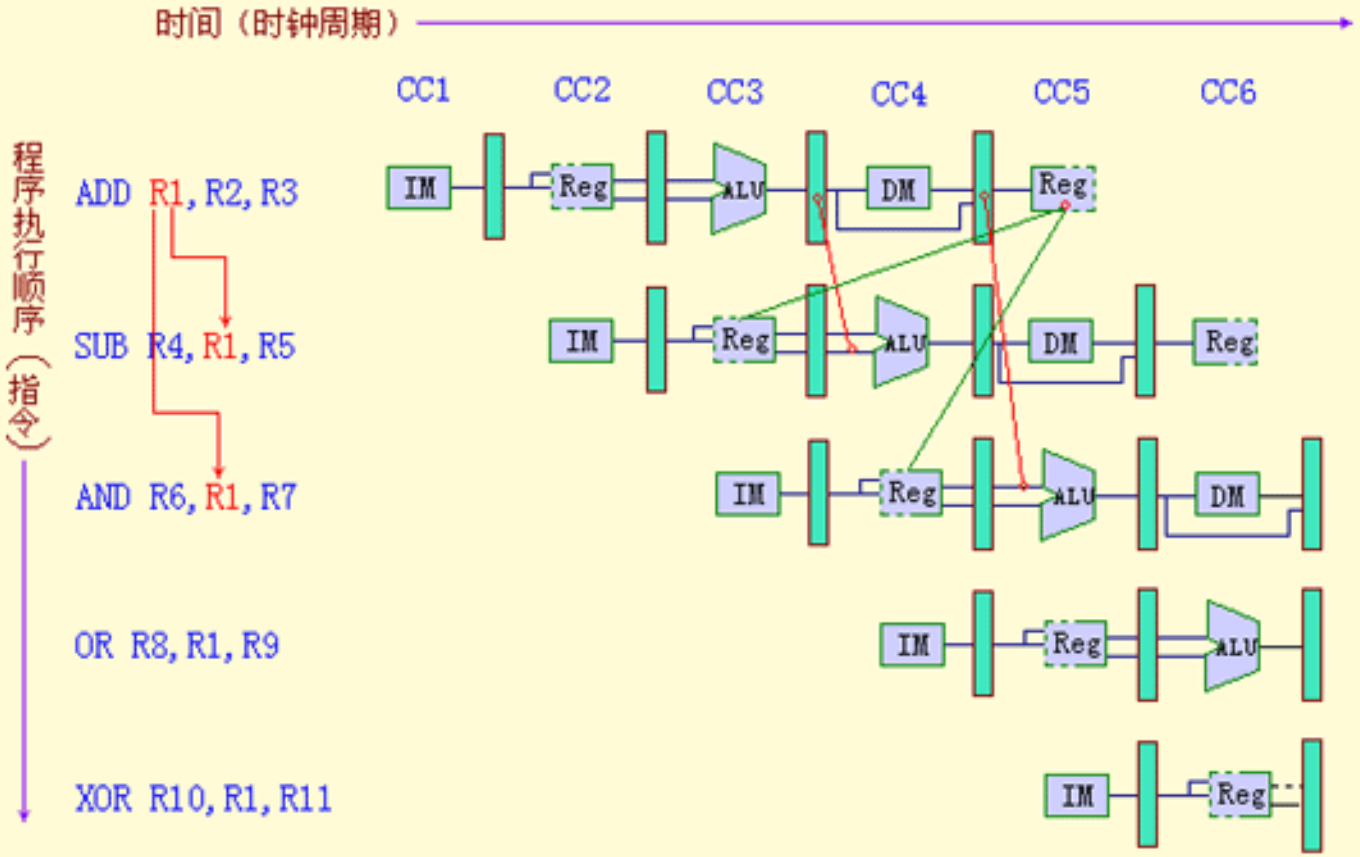
6.3.2 流水线的数据冲突

6.3.3 流水线的控制冲突

数据相关举例


时间（时钟周期）
CC1
CC2
CC3
CC4
CC5
CC6
程序执行顺序（指令）
ADD R1,R2,R3
SUB R4,R1,R5
AND R6,R1,R7
OR R8,R1,R9
XOR R10,R1,R11
IM
Reg
ALU
DM
Reg

采用定向技术消除数据相关



采用定向技术后的工作过程

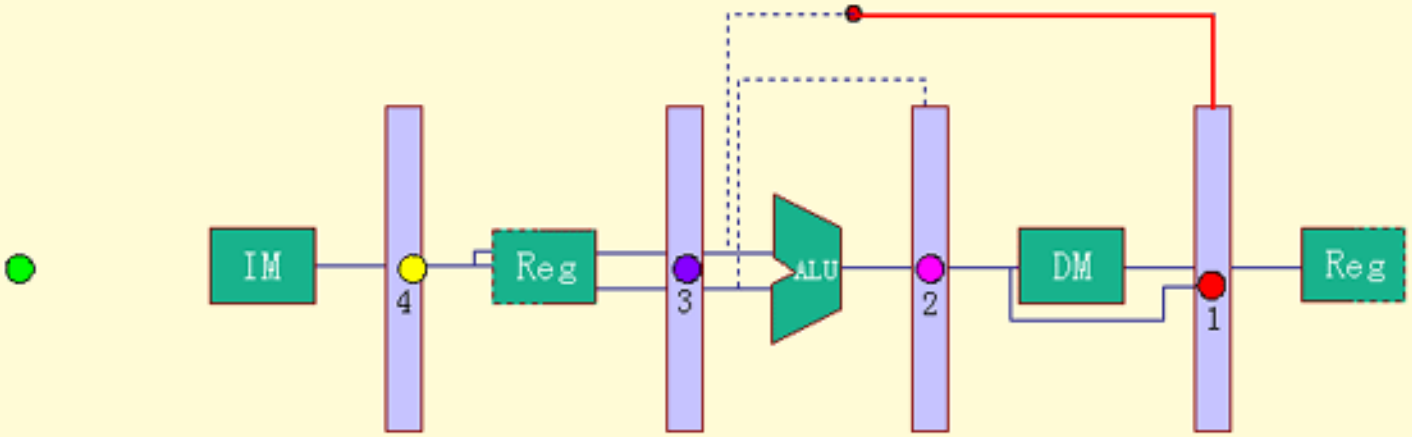
指令1　ADD R1,R2,R3

指令2　SUB R4,R1,R5

指令3　AND R6,R1,R7

指令4　OR R8,R1,R9

指令5　XOR R10,R1,R11



6.3.2 流水线的数据冲突

2. 通过定向技术减少数据冲突带来的暂停

- 进一步推广：一个结果不仅可以从某一功能单元的输出定向到其自身的输入，而且还可以定向到其它功能单元的输入。 (举例)

- 在MIPS中，任何流水寄存器到任何功能单元的输入都可能需要定向路径，将形成复杂的旁路网络。

- 两条指令访问同一存储单元，也可能引起数据冲突，例如访问数据Cache失效时。

- 本章只讨论寄存器数据冲突!

6.3.2 流水线的数据冲突

3. 数据冲突的分类

两条指令 *i* 和 *j*，都会访问同一寄存器*R*，假设*i* 先进入流水线，则它们对*R*有四种不同的访问顺序：

(1) 先写后读冲突(*RAW*) —— *i* 写*j* 读

- 如果*j* 在*i* 完成写之前从*R* 中读出数据，将得到错误的结果！
- 最常见的数据冲突，严重制约了CPU的性能，是程序最重要的特征之一！

6.3.2 流水线的数据冲突

3. 数据冲突的分类

(2) 写后写冲突(*WAW*) —— *i* 写 *j* 写

- 如果 *j* 在 *i* 之前完成写操作，*R*中将保存错误的结果!

  MIPS流水线不会出现这种冲突!

- 当流水线中有多个段可以写回，或者当流水线暂停某条指令的执行时，允许其后的指令可以继续前进时，可能引起这种类型的冲突。

  举例

6.3.2 流水线的数据冲突

3. 数据冲突的分类

(3) 先读后写冲突(*WAR*) —— *i* 读*j* 写

- 如果*j* 先将数据写入*R*，*i* 将读出错误的结果！

  MIPS流水线不会出现这种类型的冲突！
- 当有些指令在流水段后半部分读源操作数，另一些指令在流水段前半部分写结果，可能引起这种类型的相关。

(4) 读后读(*RAR*) —— *i* 读*j* 读

- 不引起数据相关！

6.3.2 流水线的数据冲突

4. 需要暂停的数据冲突

- 并非所有数据冲突都可以通过定向技术解决。

  例：
  ```
  LW   R1，0（R2）
  SUB  R4，R1，R5
  AND  R6，R1，R7
  OR   R8，R1，R9
  ```

- 增加流水线“流水线互锁”部件，当互锁硬件发现这种冲突后，就暂停流水线，直到相关消除。
- 这种情况下，暂停的时钟周期数称为“载入延迟”。

  例1　　时空图

6.3.2 流水线的数据冲突

5．对数据冲突的编译调度方法

- 流水线中常常会遇到多种类型的暂停

  例如，计算表达式A=B+C时会出现暂停

- 编译器可以通过重新排列代码的顺序来消除这种暂停，这种技术就是“流水线调度”或“指令调度”

例: 请为下列表达式生成没有暂停的MIPS指令序列

```
a = b - c;
d = e - f;
```

假设载入延迟为1个时钟周期。

- 举例：

请为下列表达式生成没有暂停的指令序列：

```
a = b - c ;
d = e - f ;
```

假设载入延迟为1个时钟周期。

调度前的代码		调度后的代码	
LW	Rb，b	LW	Rb，b
LW	Rc，c	LW	Rc，c
ADD	Ra，Rb，Rc	LW	Re，e
SW	Ra，a	ADD	Ra，Rb，Rc
LW	Re，e	LW	Rf，f
LW	Rf，f	SW	Ra，a
SUB	Rd，Re，Rf	SUB	Rd，Re，Rf
SW	Rd，d	SW	Rd，d

6.3.2 流水线的数据冲突

6. 对MIPS流水线控制的实现

- 指令发射(Issue)：指令从流水线的译码段进入执行段的过程称为指令发射。
- 检测数据冲突
  - ID段可以检测所有数据冲突
  - 也可以在使用一个操作数的时钟周期的开始(EX和MEM段的开始)检测相关，并确定必需的定向
  - 流水线相关硬件可以检测到的各种冲突情况

6.3.2 流水线的数据冲突

6. 对MIPS流水线控制的实现

(1) Load互锁的检测与实现

- 在ID段检测是否需要启动Load互锁，必须进行三种比较
- 一旦检测到冲突，控制部件必须在流水线中插入暂停周期，并使IF和ID段中的指令停止前进
  - 将ID/EX中控制部分清0
  - 保持IF/ID的内容不变

6.3.2 流水线的数据冲突

6. 对MIPS流水线控制的实现

(2) 定向逻辑的实现——例如

- 所有的定向都是从ALU/DM的输出到ALU、DM或0检测单元的输入

- 形成了一个旁路网络(图示)
  - ✓ 需要比较哪些信息?
  - ✓ ALU输入端应采用多少个输入的MUX?

6.3.3 流水线的控制冲突

1. 分支指令的实现

- 一旦分支转移成功，正确的地址要在Mem段才会被写入PC。
- 一旦ID段检测到分支指令，就暂停执行其后的指令，直到分支指令达到Mem段，确定新的PC为止。
- 分支转移成功将导致MIPS流水线暂停3个周期。

6.3.3 流水线的控制冲突

2. 减少分支开销的途径

- 两个基本途径：同时采用，缺一不可！
  - ✓ 在流水线中尽早判断分支转移是否成功
  - ✓ 转移成功时，尽早计算出转移目标地址

- 经改进，MIPS流水线可以将分支开销减少1拍
  - ✓ 将“=0?”测试提前到ID段
  - ✓ 在ID段增加一个加法器，计算分支目标地址

  下表列出了改进后流水线的分支操作

- 再改进，MIPS流水线可以将分支开销再减少1拍
  - ✓ 将分支判断结果和目标地址提前到ID/EX站前

6.3.3 流水线的控制冲突

3. 程序中分支指令的行为特点

(1) 各种能改变PC值的指令的执行频度

- 条件分支：
  - 整数程序：14-15%
  - 浮点程序：3-12%

  其中，向前分支与向后分支的比：3:1
- 无条件分支：≤4%（绝大多数）

(2) 条件分支转移成功的概率

- 向前转移成功率：60%；向后转移成功率：85%

6.3.3 流水线的控制冲突

4. 减少流水线分支损失的方法

(1) 冻结或排空流水线

- 思路：在流水线中停住或删除分支后的指令，直到知道转移目标地址
- 优点：简单

(2) 预测分支转移失败

- 思路：流水线继续照常流动，如果分支转移成功，将分支指令后的指令转换为空操作，并从分支目标处开始取指令执行；否则照常执行
- MIPS流水线的处理过程

6.3.3 流水线的控制冲突

4. 减少流水线分支损失的方法

(3) 预测分支转移成功

- 思路：始终假设分支成功，直接从分支目标处取指令执行
- 对MIPS流水线没有任何好处！

(4) 延迟分支（delayed branch）

- 思路：分支开销为n的分支指令后紧跟有n个延迟槽，流水线遇到分支指令时，按正常方式处理，顺带执行延迟槽中的指令，从而减少分支开销。

  延迟分支及指令的执行顺序

6.3.3 流水线的控制冲突

4. 减少流水线分支损失的方法

（1）具有一个分支延迟槽的MIPS流水线的执行过程

（2）什么样的指令能否放入分支延迟槽？

- 三种调度方法：从前调度；从目标处调度；从失败处调度
- 三种方法的要求与效果，存在限制因素
  - ✓ 编译器预测分支是否成功的能力
  - ✓ 放入延迟槽中的指令
- 取消分支　　（举例）

  思路：分支指令中包含预测方向，若预测正确，正常执行延迟槽中的指令，否则将其转换为空操作

6.3.3 流水线的控制冲突

5. 各种分支处理方法的性能

(1) 假设理想CPI=1，则加速比

$$S=D/(1+C)=D/(1+f\times p_{分支})$$

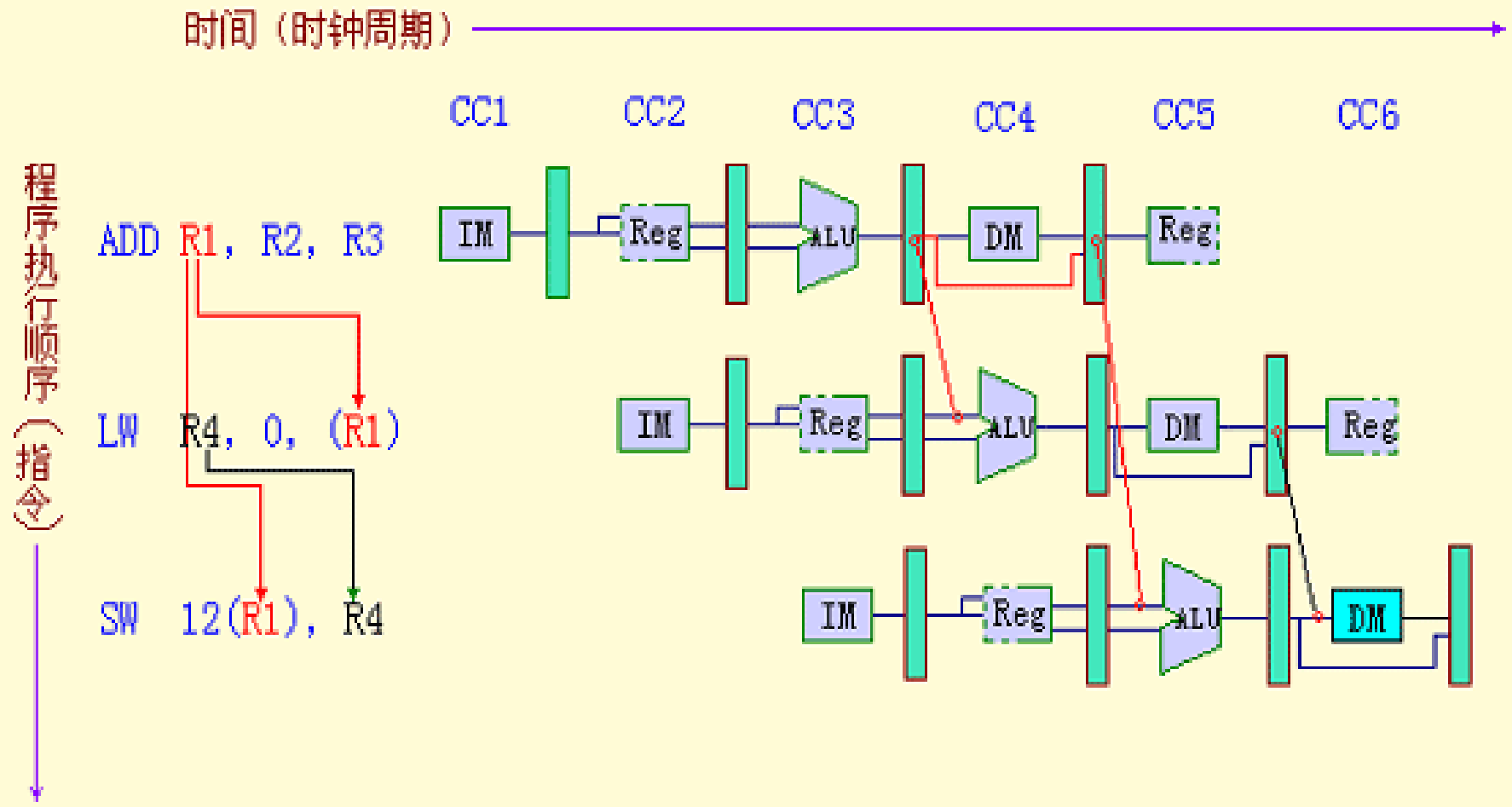
这里，D为流水线的深度，$p_{分支}$为分支开销，C为分支引起的流水线暂停时钟周期数(每条指令的平均值)，f为分支的出现频度。

(2) 下表列出了流水线中各种处理方法的开销。

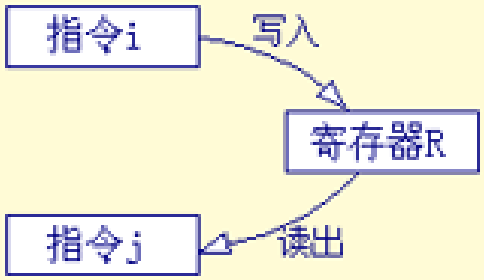
各种减少分支损失方法的效果

调度方法	每条条件分支指令的分支损失		每条无条件分支指令的损失	每条分支指令的平均分支损失		具有分支暂停的有效 CPI	
	整 型平 均	浮 点平 均		整 型平 均	浮 点平 均	整 型平 均	浮 点平 均
暂停流水线	1.00	1.00	1.00	1.00	1.00	1.17	1.15
预测分支成功	1.00	1.00	1.00	1.00	1.00	1.17	1.15
预测分支失败	0.62	0.70	1.00	0.69	0.74	1.12	1.11
延迟分支	0.25	0.35	0.00	0.21	0.30	1.04	1.04

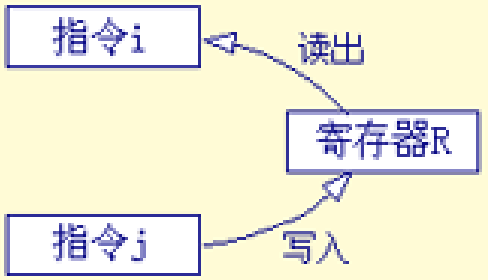
到数据存储器和ALU的定向路径



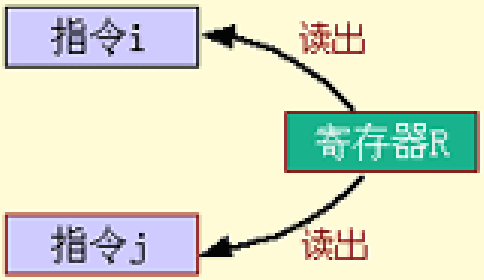
两条指令对同一寄存器的读/写顺序



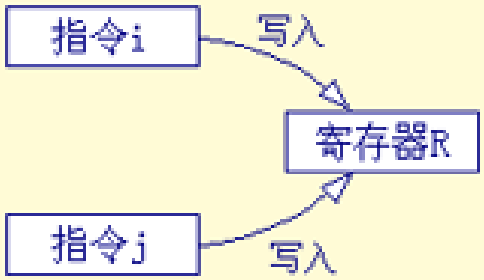
先写后读



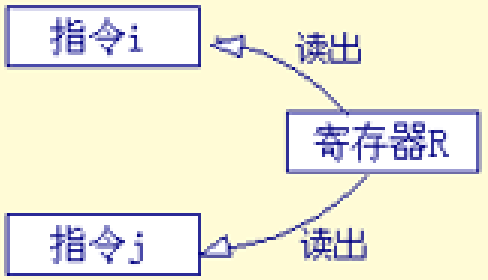
先读后写



读后读



写后写



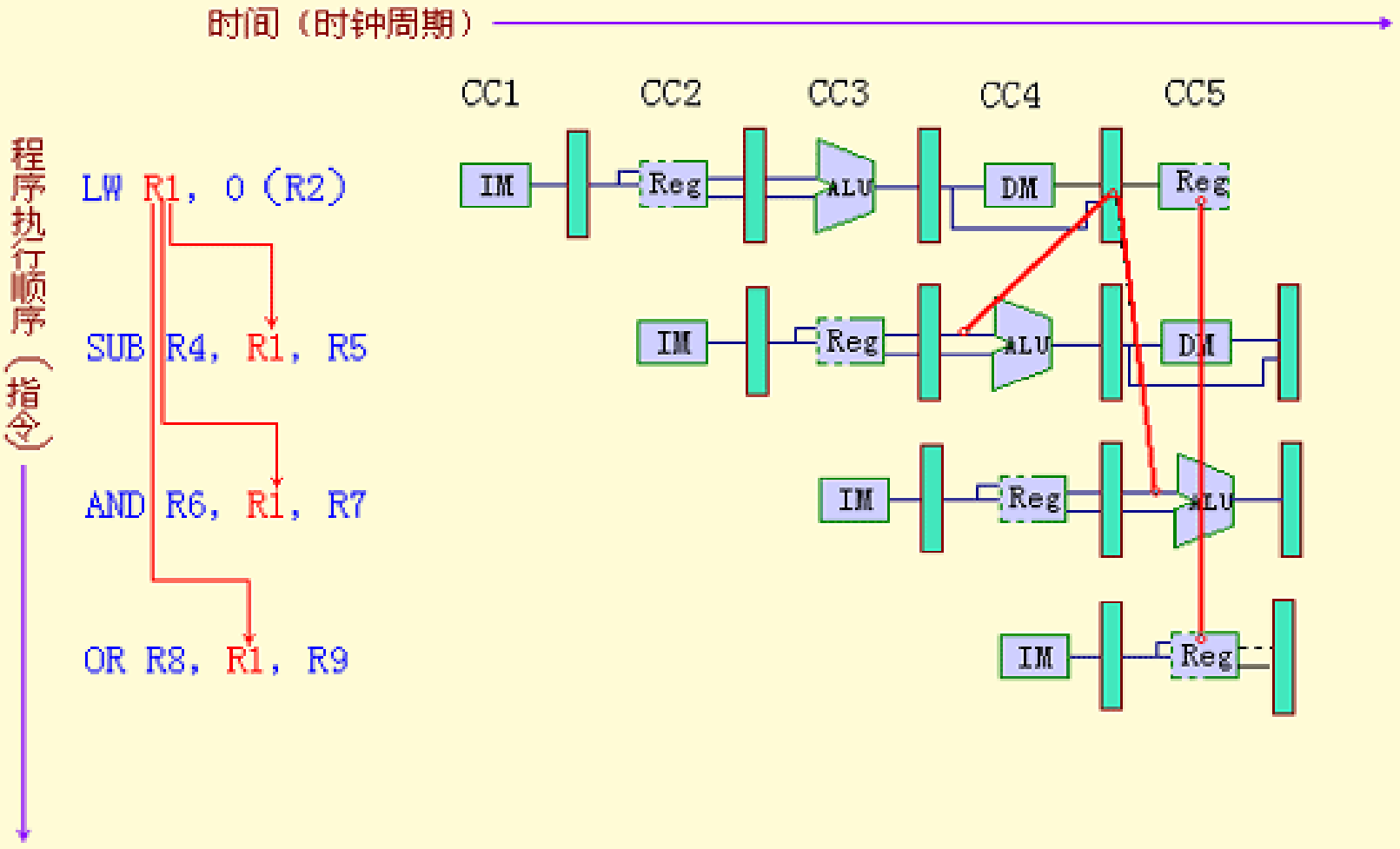
读后读

不可能出现相关

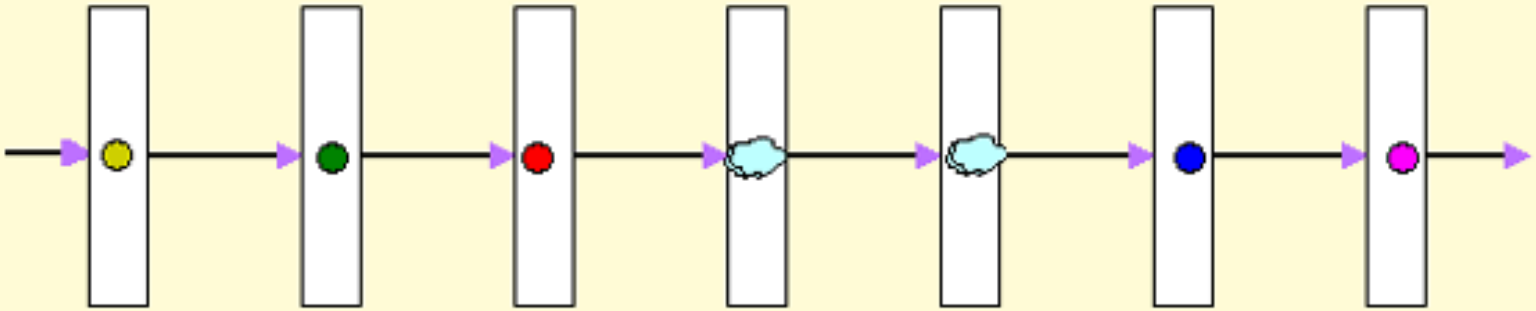
修改后的DLX流水线会发生WAW相关

LW R1, 0, (R2)	IF	ID	EX	MEM1	MEM2	WB
ADD R1, R2, R3		IF	ID	EX	WB	

定向技术不能解决的数据相关

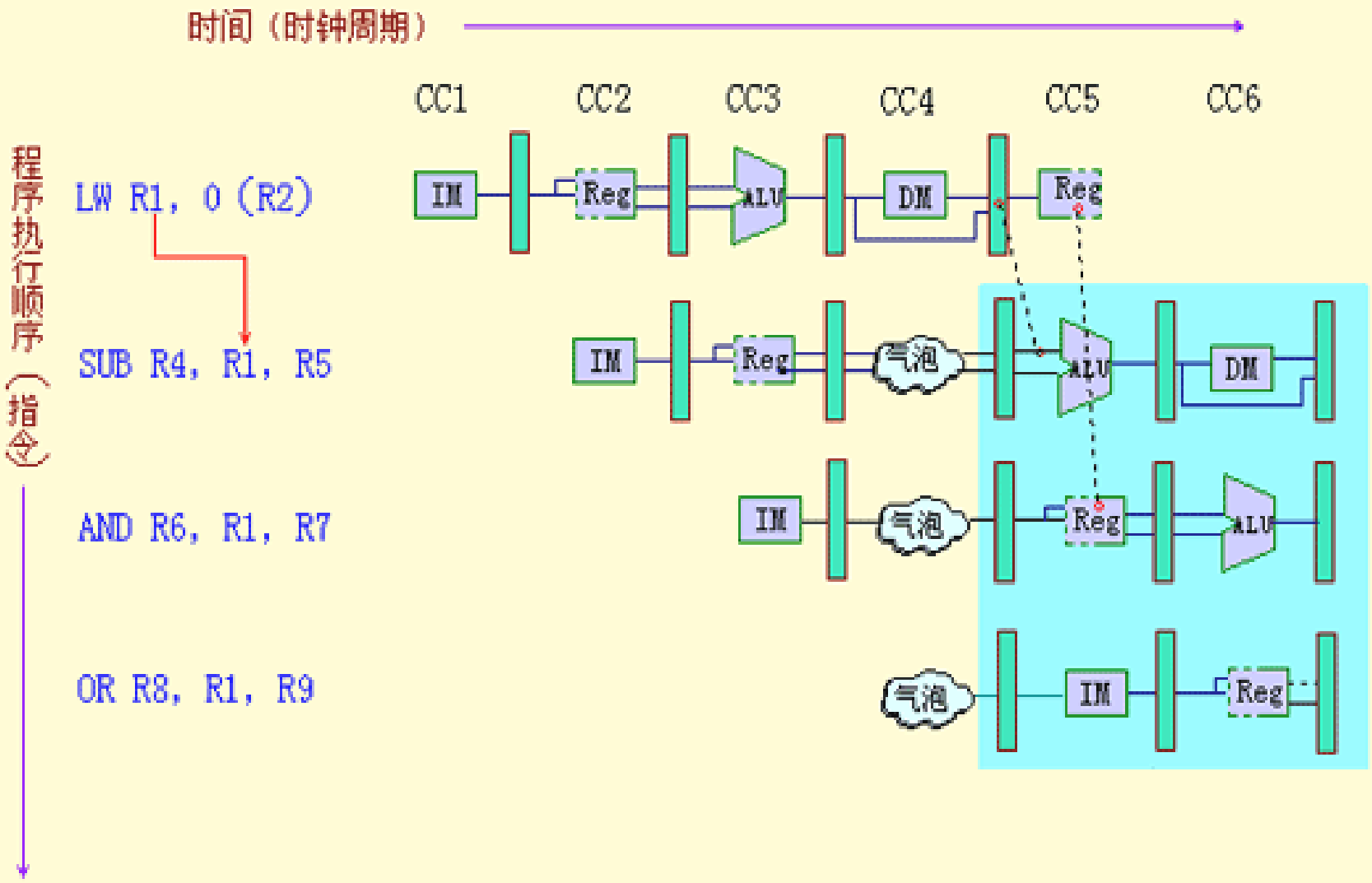


相关时暂停流水线



流 水 线

在插入“暂停”的情况下流水线的工作过程



在插入“暂停”情况下，流水线的时空图

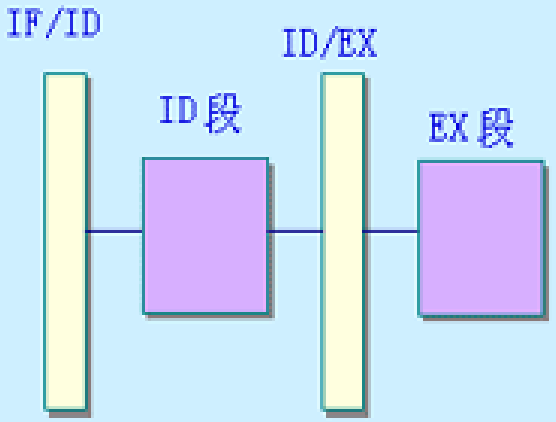
LW R1,0(R2)	IF ID EX MEM WB
SUB R4,R1,R5	IF ID stall EX MEM WB
AND R6,R1,R7	IF stall ID EX MEM WB
OR R8,R1,R9	stall IF ID EX MEM WB

流水线相关检测硬件可以检测到的各种相关情况

相关情况	指令序列范围	动　　作
没有相关	LW　R1, 45(R2) ADD　R5, R6, R7 SUB　R8, R6, R7 OR　R9, R6, R7	这三条指令与R1无关，故不可能出现相关。
需要暂停的相关	LW　R1, 45(R2) ADD　R5, R1, R7 SUB　R8, R6, R7 OR　R9, R6, R7	比较器检测到ADD指令中使用R1作为源寄存器，并在ADD指令进入EX 段之前将之暂停。(同时也将SUB和OR指令暂停)
通过定向消除的相关	LW　R1, 45(R2) ADD　R5, R6, R7 SUB　R8, R1, R7 OR　R9, R6, R7	比较器检测到SUB指令中使用R1作为源寄存器，并及时的在SUB指令进入EX段之前将Load指令的结果定向到ALU。
按顺序访问的相关	LW　R1, 45(R2) ADD　R5, R6, R7 SUB　R8, R6, R7 OR　R9, R1, R7	LW指令在前半周期将结果写入R1 OR指令在后半周期读R1

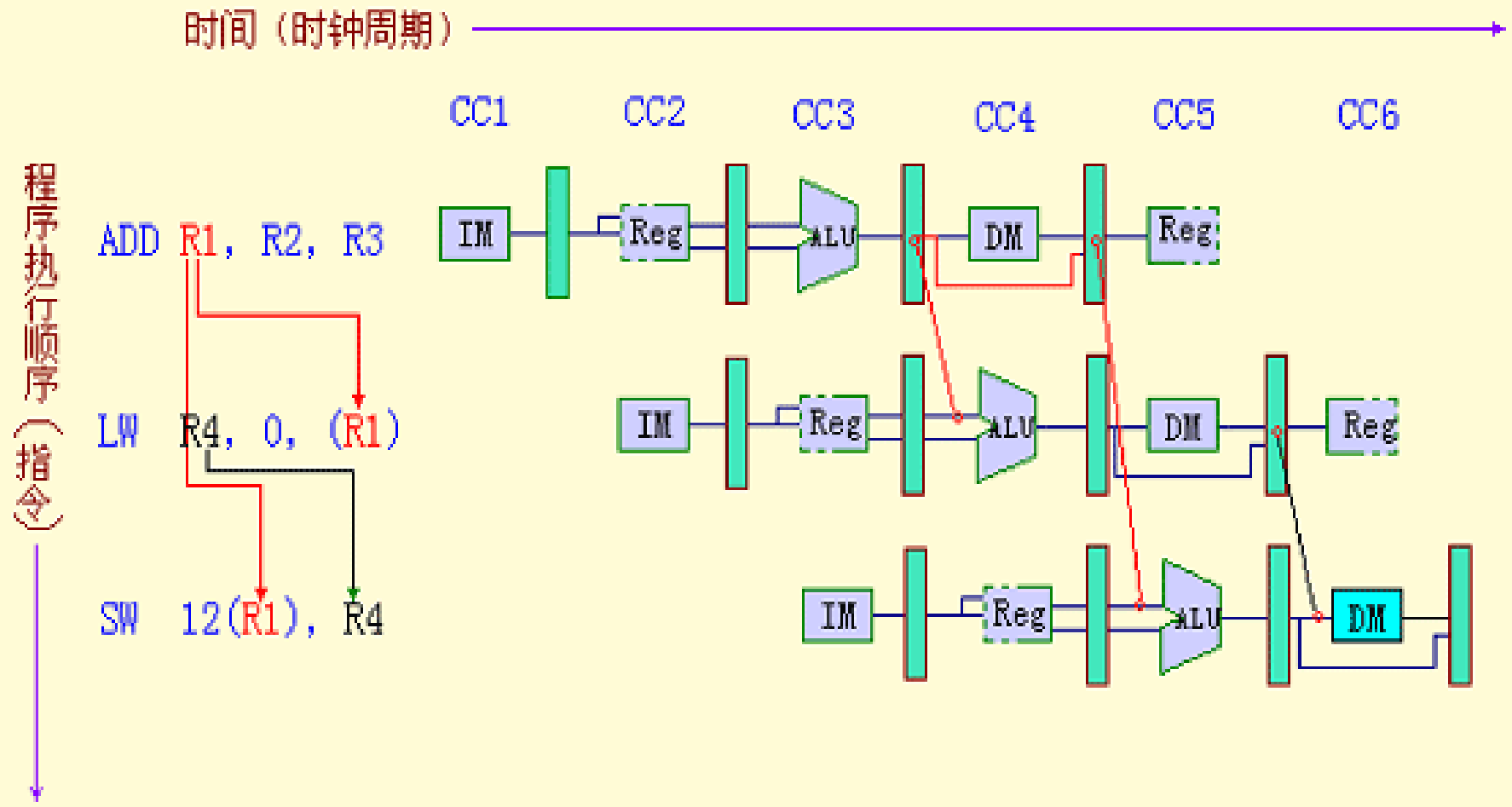
在ID段检测Load互锁需进行三种比较

- Load、Store、ALU 立即数、分支等指令：
  - 源寄存器**rs1**：$IR_{6...10}$
  - 结果寄存器**rd**：$IR_{11...15}$
- R-R ALU指令：
  - 源寄存器**rs1**：$IR_{6...10}$
  - 源寄存器**rs2**：$IR_{11...15}$
  - 结果寄存器**rd**：$IR_{16...20}$

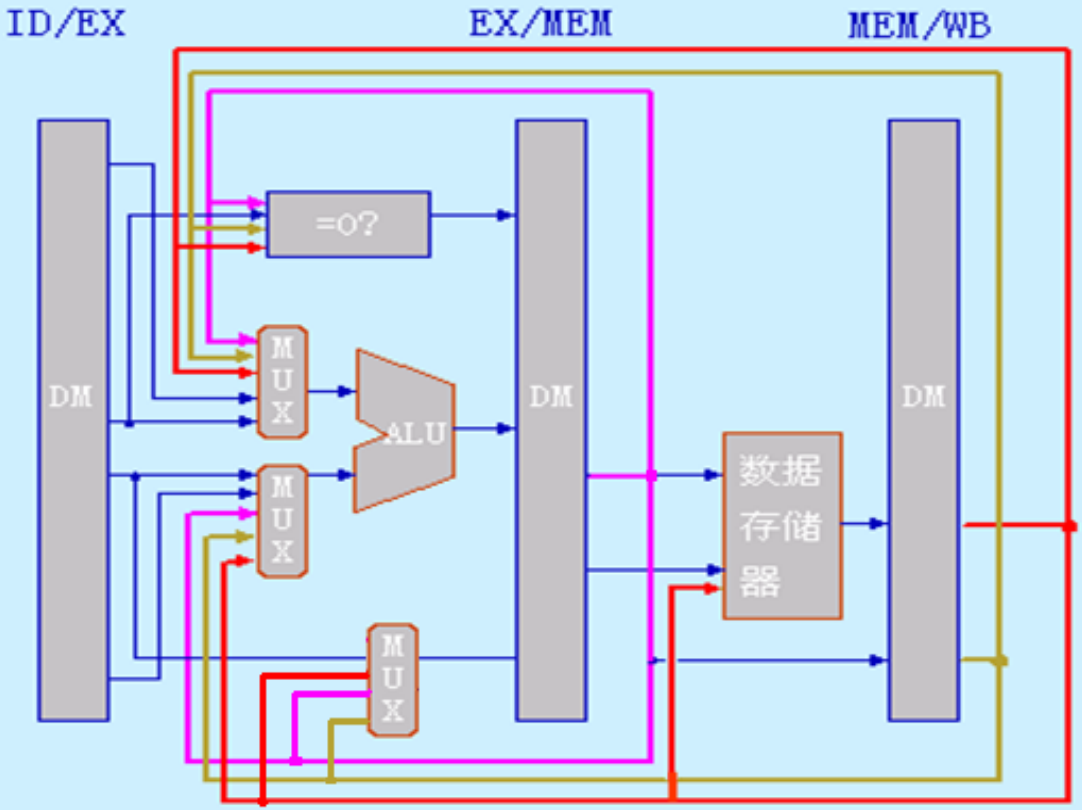


ID/EX 的操作码域 ($ID/EX.IR_{0...5}$)	IF/ID的操作码域 ($IF/ID.IR_{0...5}$)	匹配操作数域
Load	R-R ALU	$ID/EX.IR_{11...15} = IF/ID.IR_{6...10}$
Load	R-R ALU	$ID/EX.IR_{11...15} = IF/ID.IR_{11...15}$
Load	Load、Store、ALU立即数或分支	$ID/EX.IR_{11...15} = IF/ID.IR_{6...10}$

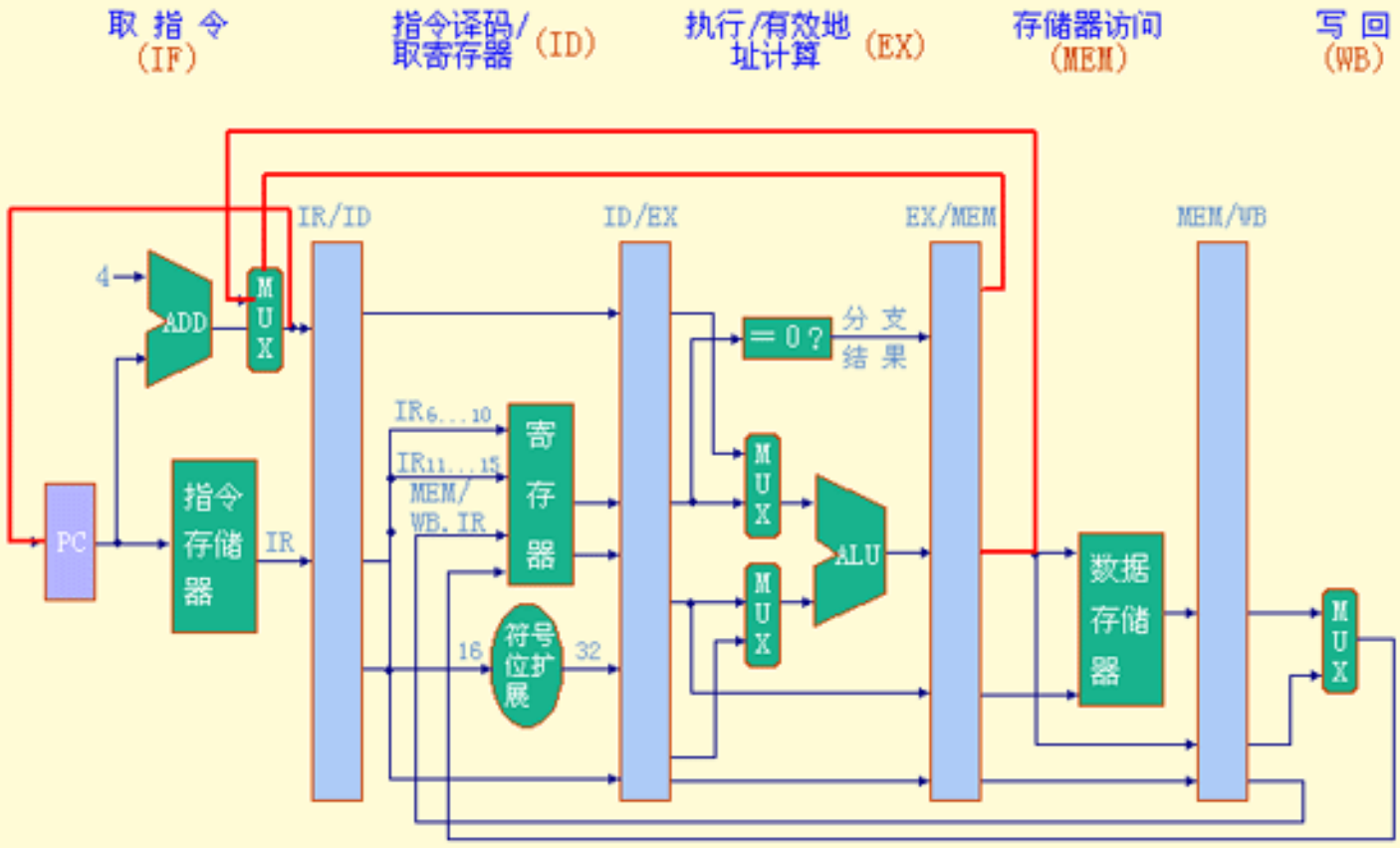
到数据存储器和ALU的定向路径



定 向 路 径



转移成功时，PC值的改变

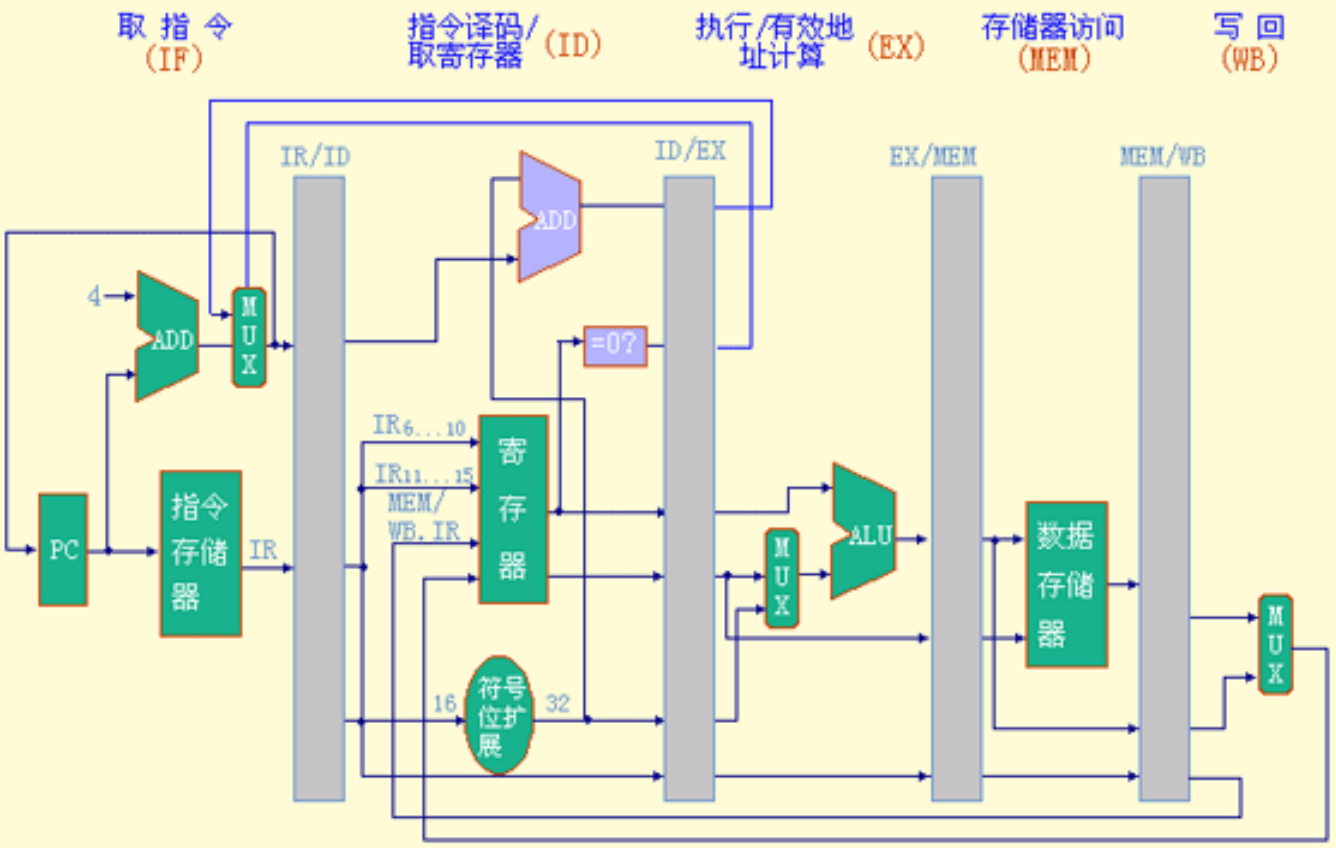


分支转移成功导致暂停 3 个时钟周期

(DLX流水线)

分支指令	IF	ID	EX	MEM	WB				
分支后继指令		IF	stall	stall	IF	ID	EX	MEM	WB
分支后继指令+1						IF	ID	EX	MEM	WB
分支后继指令+2							IF	ID	EX	MEM
分支后继指令+3								IF	ID	EX
分支后继指令+4									IF	ID
分支后继指令+5										IF

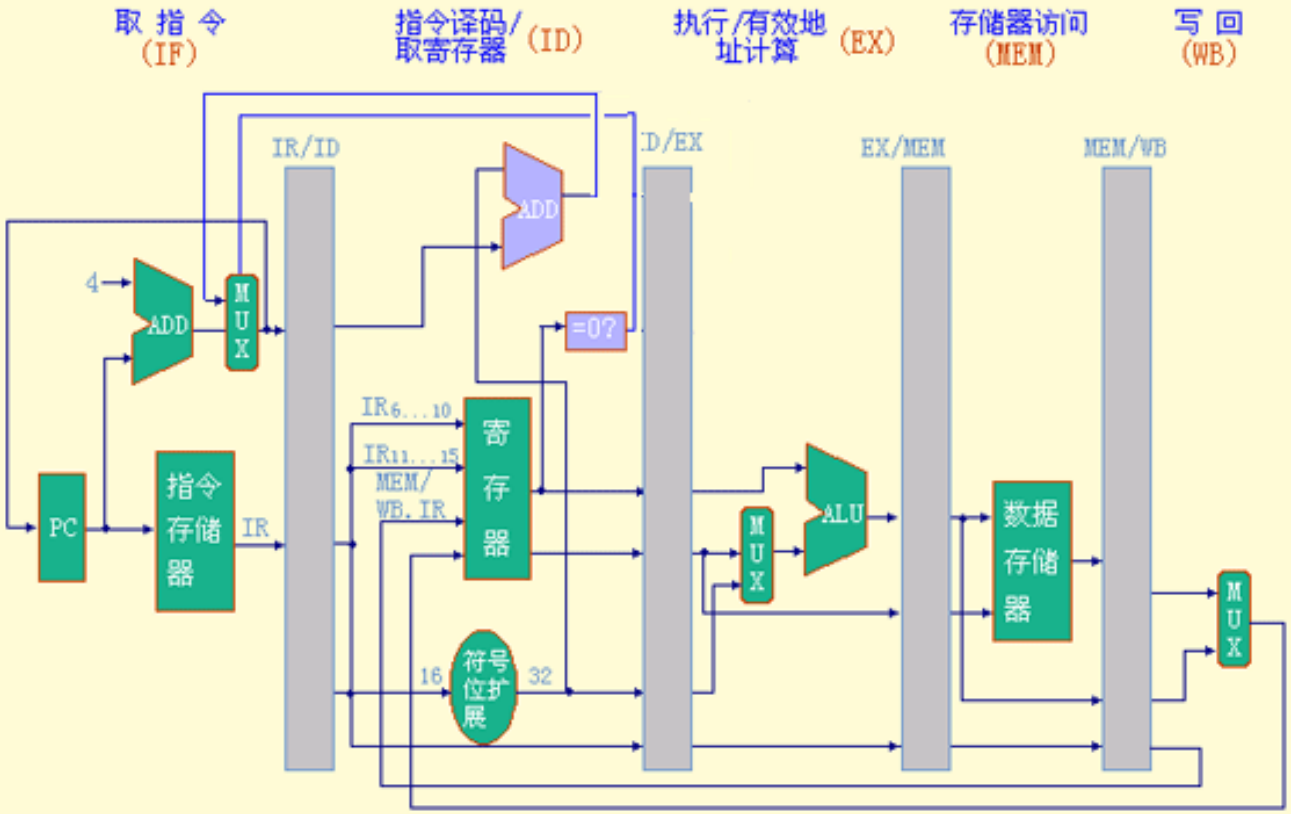
修改DLX以减少分支开销



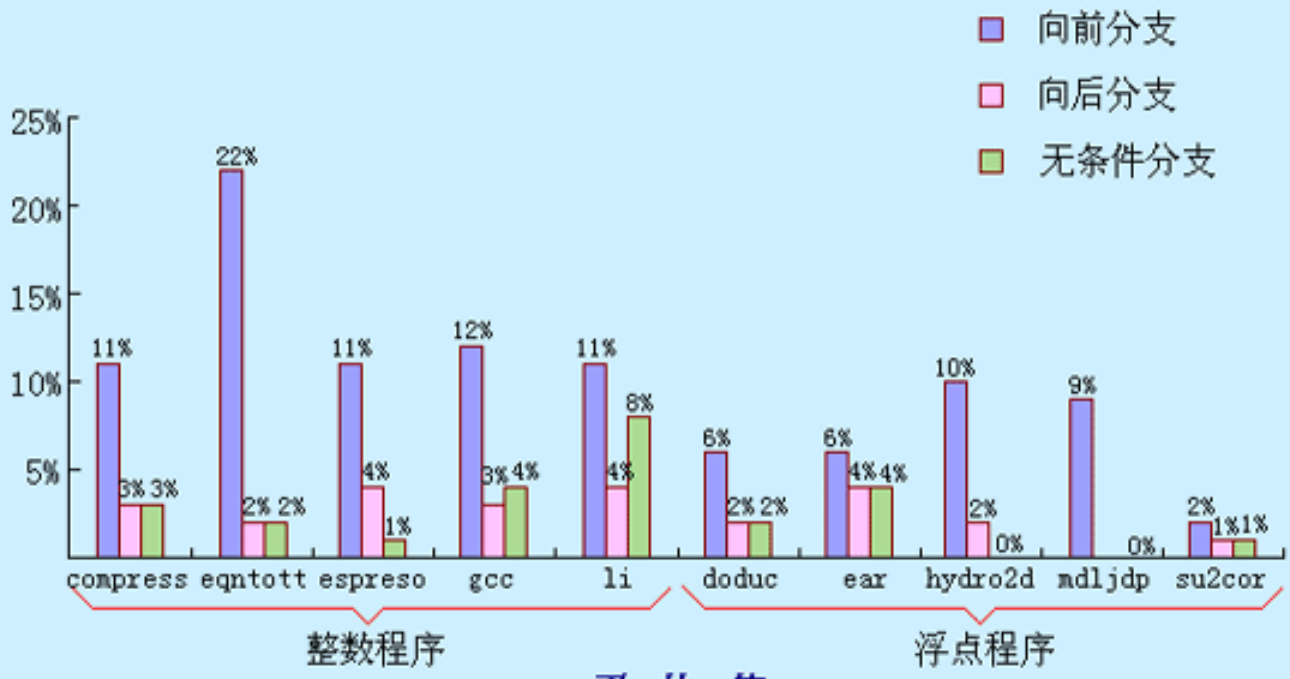
改进后流水线的分支操作

流 水 段	分 支 指 令 操 作
IF	IF/ID.IR ← Mem[PC]; IF/ID.NPC, PC ← (if ID/EX.Cond {ID/EX.NPC} else {PC+4});
ID	ID/EX.A ←Regs[IF/ID.IR $_{6..10}$]; ID/EX.B← Regs[IF/ID.IR $_{11..15}$]; ID/EX.IR ← IF/ID.IR; ID/EX.Imm ← (IF/ID.IR$_{16}$) ^{16} ## IF/ID.IR$_{16..31}$; ID/EX.cond ← (Regs[IF/ID.IR $_{6..10}$] op 0); ID/EX.NPC ← IF/ID.NPC +(IF/ID.IR$_{16}$)^{16} ## IF/ID.IR$_{16..31}$;
EX	
MEM	
WB	

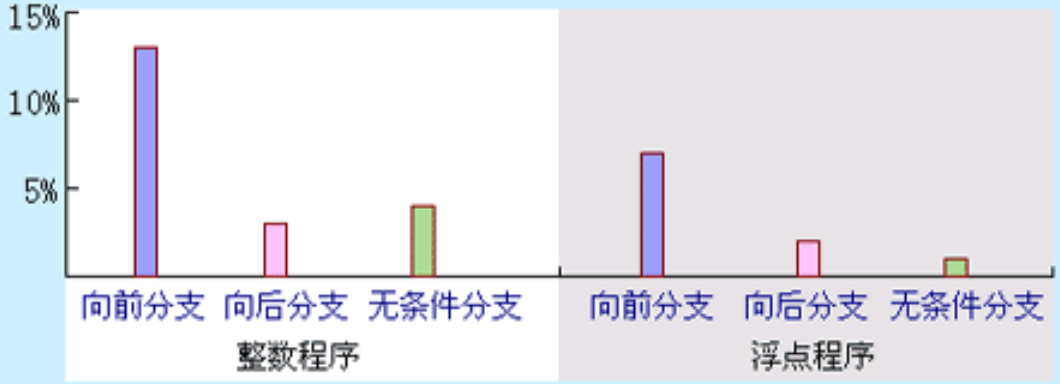
修改DLX以减少分支开销



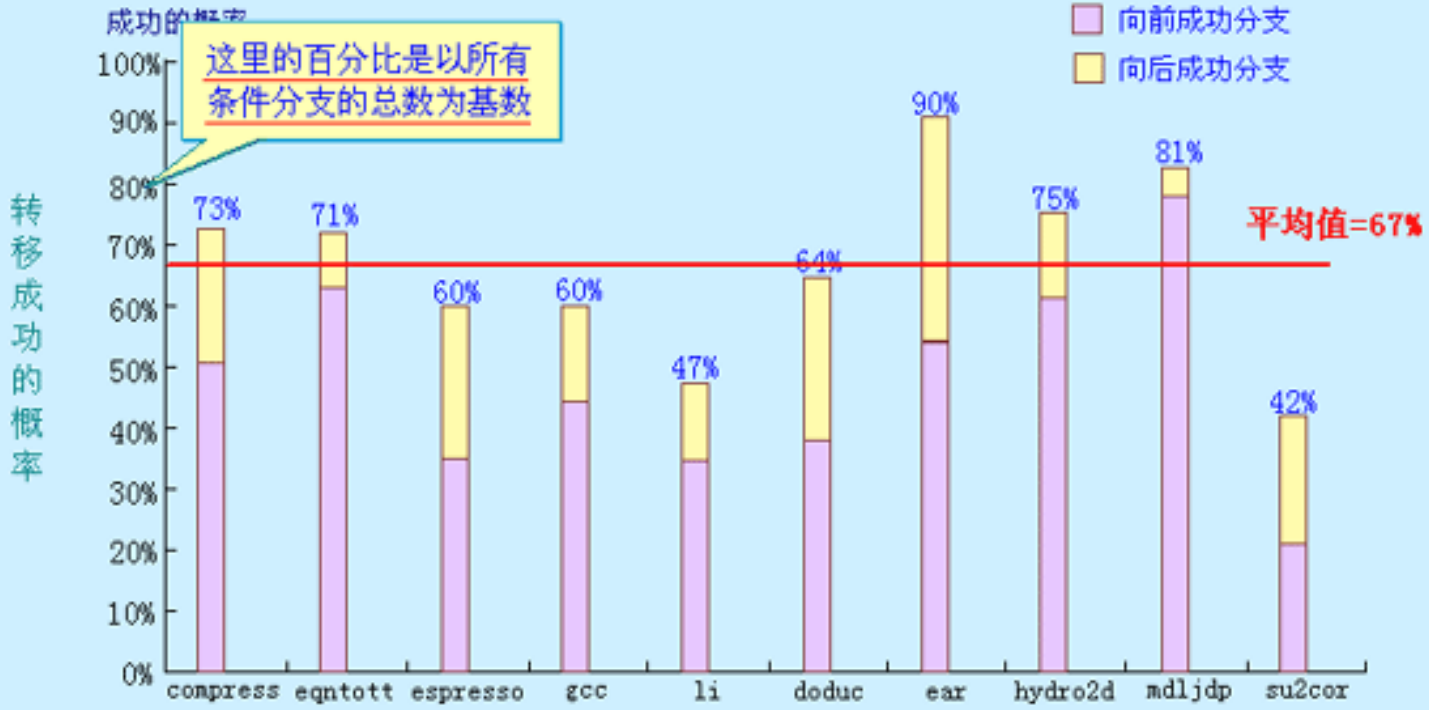
各种能改变PC值的指令的执行频率



平 均 值



条件分支转移成功的概率

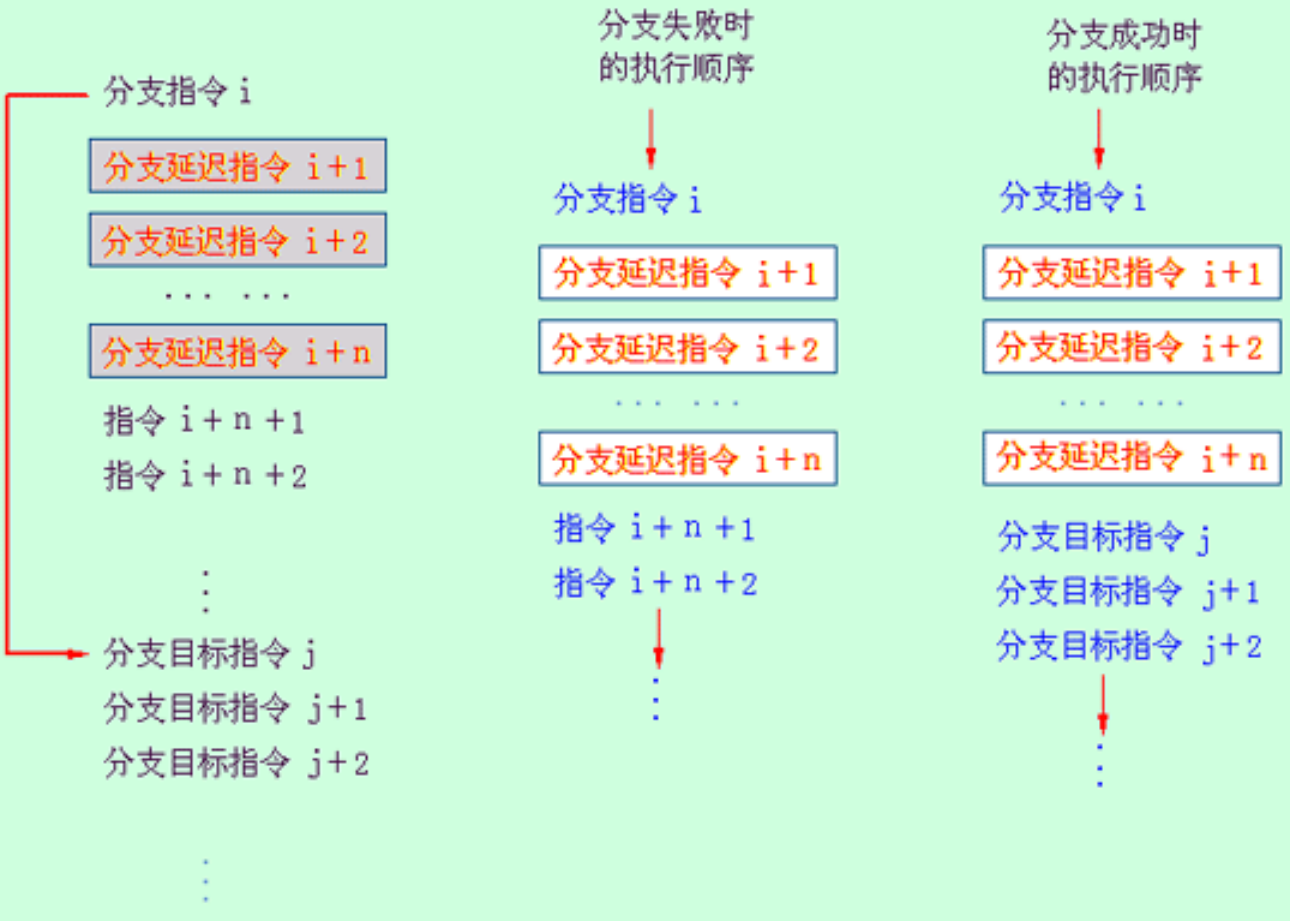


DLX流水线对分支的处理过程

分支指令 i(失败)	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

分支指令 i(成功)	IF	ID	EX	MEM	WB				
指令 i+1		IF	ID	idle	idle	idle			
分支目标 j			IF	ID	EX	MEM	WB		
分支目标 j+1				IF	ID	EX	MEM	WB	
分支目标 j+2					IF	ID	EX	MEM	WB

延迟分支以及指令的执行顺序



DLX流水线的执行过程

（具有一个分支延迟槽）

分支失败的情况

分支指令(失败)	IF	ID	EX	MEM	WB				
分支延迟指令(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

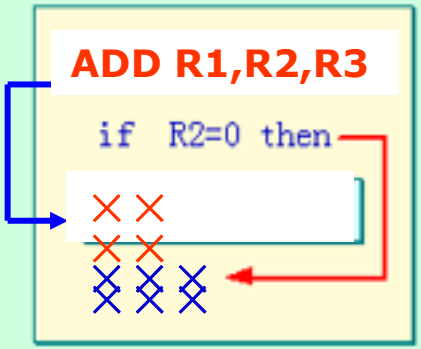
分支成功的情况

分支指令(成功)	IF	ID	EX	MEM	WB				
分支延迟指令(i+1)		IF	ID	EX	MEM	WB			
分支目标指令 j			IF	ID	EX	MEM	WB		
分支目标指令 j+1				IF	ID	EX	MEM	WB	
分支目标指令 j+2					IF	ID	EX	MEM	WB

分支延迟指令的三种调度方法

从前调度

调度后



Notake

If

ADD

ADD

If

take

If

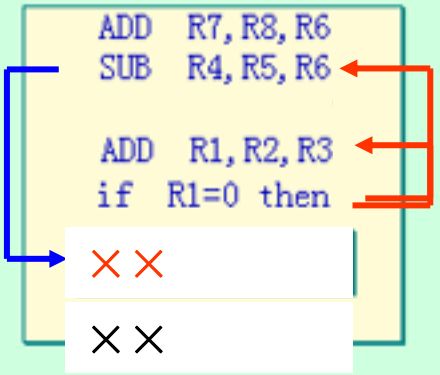
ADD

ADD

If

从目标处调度

调度后



Notake

ADD

SUB

ADD

if

SUB

Take

ADD

SUB

ADD

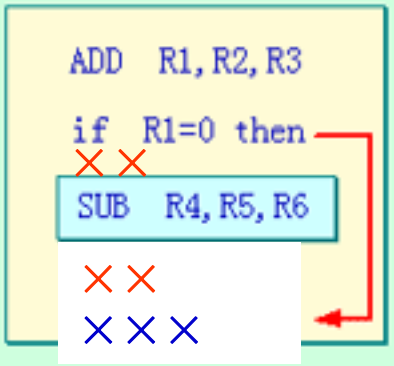
if

SUB

ADD

从失败处调度

调度后



Notake

ADD

If

SUB

SUB

Take

ADD

If

SUB

三种方法的要求及效果

调度策略	对调度的要求	其作用前提
从前调度	被调度的指令必须与分支结果无关	任何情况
从目标处调度	必须保证在分支失败时执行被调度的指令不会导致错误，可能需要复制指令	分支成功时
从失败处调度	必须保证在分支成功时执行被调度的指令不会导致错误	分支失败时

“预测成功--取消”分支的执行过程

分支指令 i(成功)	IF	ID	EX	MEM	WB				
分支延迟指令 i+1		IF	ID	EX	MEM	WB			
分支目标 j			IF	ID	EX	MEM	WB		
分支目标 j+1				IF	ID	EX	MEM	WB	
分支目标 j+2					IF	ID	EX	MEM	WB

分支指令 i(失败)	IF	ID	EX	MEM	WB				
分支延迟指令 i+1		IF	ID	idle	idle	idle			
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