

计算机组织与体系结构

第十五讲

计算机科学与技术学院

舒燕君

Recap

- 流水线数据冲突控制的实现
 - ✓ 需要暂停的数据相关
 - ✓ 通过编译调度解决的数据相关
 - ✓ 流水线锁的检测与实现
 - ✓ 定向逻辑的实现
- 流水线的控制冲突
 - ✓ 减少分支开销的途径（改进、再改进）
 - ✓ 减少分支损失的办法
 - 冻结
 - 预测分支转移失败
 - 预测分支转移成功
 - 延迟分支
 - ✓ 各种分支处理方法的性能分析

6.4 实例分析：MIPS R4000

6.4.1 MIPS R4000的整型流水线

6.4.2 MIPS R4000的浮点流水线

6.4.3 MIPS R4000流水线性能分析

6.4.1 MIPS R4000的整型流水线

1. 指令集：64位MIPS-3指令集

2. MIPS R4000流水线结构

- ◆ 超流水结构 (Superpipelining)

- ◆ 访存操作流水化

3. 流水线各段的功能

6.4.1 MIPS R4000的整型流水线

4. 指令序列在流水线中的重叠执行过程

定向+插入暂停周期

5. 载入延迟为两个时钟周期

6. 指令序列在流水线中的执行时空图

7. R4000流水线的定向路径远多于MIPS流水线

ALU输入端的定向源有4个: EX/DF, DF/DS,
DS/TC, TC/WB

6.4.1 MIPS R4000的整型流水线

8. 分支处理

- ◆ 在EX段完成分支条件的计算，基本分支延迟为3个时钟周期
- ◆ 分支处理策略
 - 单周期延迟分支
 - 从失败处调度 时-空图

6.4.2 MIPS R4000的浮点流水线

- ◆ MIPS R4000 FPU包括浮点除法器、浮点乘法器和浮点加法器各1个
- ◆ 分为8段（R4000流水线的8个段）
- ◆ 多功能非线性流水线
- ◆ 双精度浮点操作指令延迟、初始化间隔和流水段的使用情况（使用情况）

6.4.3 MIPS R4000流水线性能分析

1. 引起流水线暂停的四个主要原因

- ◆ 载入暂停
- ◆ 分支暂停
- ◆ 浮点结果暂停
- ◆ 浮点结构性暂停

2. 暂停对MIPS R4000流水线CPI的影响

暂停对R4000流水线CPI的影响

整数平均:	流水线CPI	1.54
	载入暂停时钟周期数	0.16
	分支暂停时钟周期数	0.38
	浮点结果暂停时钟周期数	0.00
	浮点结构性暂停时钟周期数	0.00
浮点平均:	流水线CPI	2.48
	载入暂停时钟周期数	0.10
	分支暂停时钟周期数	0.33
	浮点结果暂停时钟周期数	0.95
	浮点结构性暂停时钟周期数	0.18

6.5 向量处理机实例：Cray-I

■ 什么是向量机？

具有向量数据表示和相应向量指令的流水线处理机称为**向量流水线处理机**，也称**向量处理机**。

与之对应的是标量处理机，不支持向量数据表示，没有提供向量指令。

■ 一个简单的FORTRAN循环程序

```
DO 10 i=1,N  
10    d[i] = a[i]*(b[i]+c[i])
```

A、B、C、D四个向量

水平（横向）处理方式

- ◆ 依次计算向量k和d的每个元素

... ..

$$k_i = b_i + c_i$$

$$d_i = k_i * a_i$$

分支 $i \leq N$?

... ..

- ◆ 循环的每个迭代(iteration)中有1次数据相关，1次控制相关，需要2次功能切换

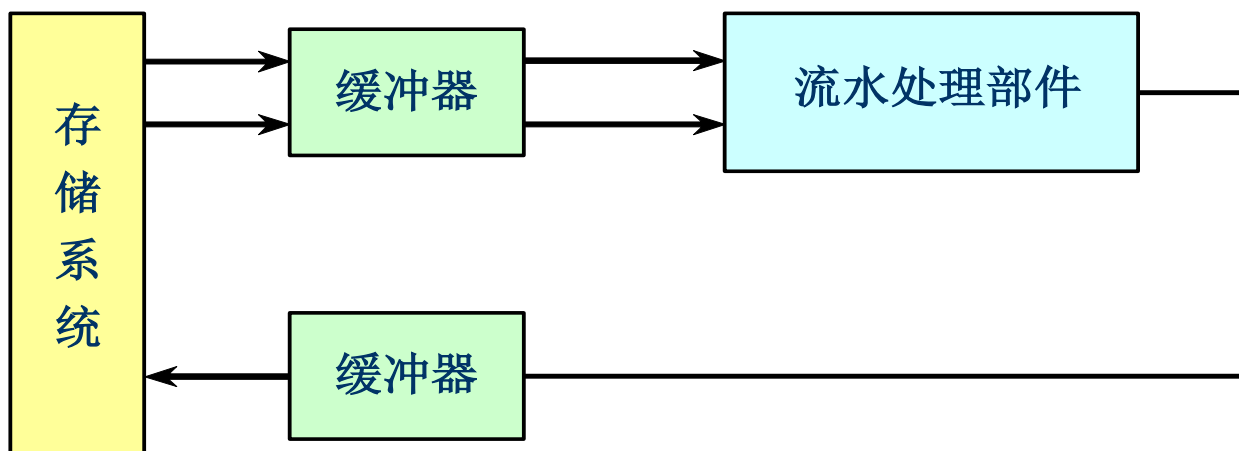
垂直（纵向）处理方式

- ◆ 计算向量k的每个元素之后，再计算向量d的每个元素，需要向量数据类型和向量指令的支持

$$K = B + C$$

$$D = K * A$$

- ◆ 没有分支；仅有1次数据相关；仅需要1次功能切换
- ◆ 需要存储器-存储器型操作的运算流水线



分组（纵横）处理方式

- ◆ 将长度为 N 的向量分为 m 组，每组有 n 个元素，组内按纵向方式处理，依次处理各组。
- ◆ 需要 m 次迭代；每次迭代执行两条向量指令，有1次数据相关，需要2次功能切换
- ◆ 需要寄存器-寄存器型操作的运算流水线
- ◆ 这种技术称为向量循环或分段开采

6.5 向量处理机实例：Cray-I

1. 性能指标

- 1GFLOPS、主频80M、向量长度64
- 无Data Cache，不支持Virtual Memory

2. [基本结构](#)

- 功能部件：12条并行工作的功能单元流水线，分别进行地址、向量、标量、浮点运算。
- 向量运算：整数加（3拍）、逻辑运算（2拍）、移位（4拍）、浮点加（6拍）、浮点乘（7拍）、浮点求倒数（14拍）
- 向量寄存器 V_0 - V_7
- 向量长度寄存器、向量屏蔽寄存器



向量处理机实例：Cray-I

3. 向量指令类型

- $V_k \leftarrow V_i \text{ op } V_j$
 - $V_k \leftarrow S_i \text{ op } V_j$
 - $V_k \leftarrow \text{Mem}$
 - $\text{Mem} \leftarrow V_k$
- 功能部件冲突：同一功能部件被一条以上的并行工作向量指令所使用。
- V_i 冲突：并行工作的各向量指令具有相同的源向量或结果向量。

向量处理机实例：Cray-I

4. CRAY-I体系结构特点

- ◆ 向量寄存器与功能单元的连接通路

每个Vi块都有单独总线可连到6个向量功能部件，而每个向量功能部件也各自都有把运算结果送回向量寄存器组的总线。

- ◆ 向量链接技术

一个向量功能部件得到的结果**直接送入**另一个向量功能部件的**操作数寄存器**时所发生的连接过程称为**链接（Chaining）**。

当两条指令出现“先写后读 (RAW)”相关时，若它们不存在功能部件冲突和向量寄存器(源或目的)冲突，就有可能把它们所用的功能部件头尾相接，形成一个链接流水线，进行**流水处理**。

链接特性实质上是把流水线“定向”的思想引入到向量执行过程的结果。

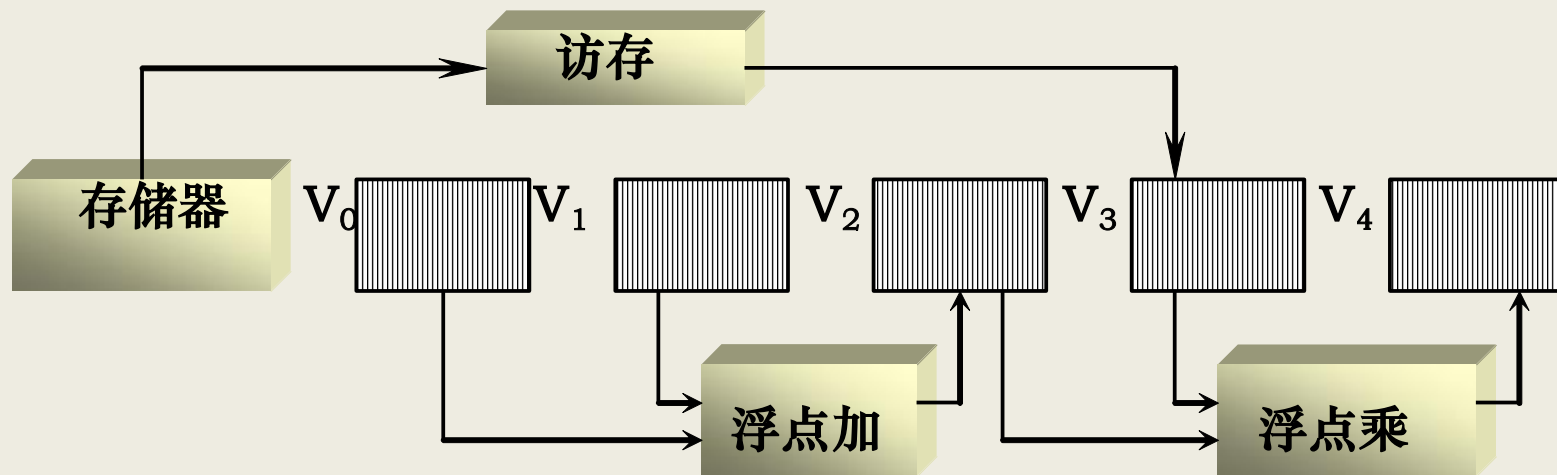
5. 向量链接技术实例分析

例：对向量运算 $D=A*(B+C)$ ，若向量长度 $N \leq 64$ ，
向量元素为浮点数，则在B、C取到V0、V1后，
就可用以下三条向量指令求解：

$V3 \leftarrow \text{存储器}$ (访存，载入A)

$V2 \leftarrow V0 + V1$ (浮点加)

$V4 \leftarrow V2 * V3$ (浮点乘，将D存入V4)



假设：向量处理机将元素从 V_i 送往功能部件及把结果存入 V_i 都需要1拍；浮点加法和访存操作都需要6拍；浮点乘操作需要7拍。

➤ 3向量条指令全部串行：

$$[(1+6+1)+N-1]+[(1+6+1)+N-1]+[(1+7+1)+N-1]=3N+22 \text{ 拍}$$

➤ 第1、2条向量指令并行，再串行执行第3条向量指令：

$$[(1+6+1)+N-1]+[(1+7+1)+N-1]=2N+15 \text{ 拍}$$

➤ 第1、2条向量指令并行，再第3条向量指令链接执行：

第一个结果被存入 V_4 需要经过（链接流水线建立时间）：

$$1(\text{送})+6(\text{浮加})+1(\text{入})+1(\text{送})+7(\text{浮乘})+1(\text{入})=17(\text{拍})$$

此后，每拍将得到一个结果送入 V_4 。

总的完成时间为： $17+(N-1)$ 拍

6. 向量链接技术应考虑的问题

- ◆ 设定合适的向量功能部件和操作数寄存器

- ◆ 链接时机问题

- 只有在前一条向量指令的第一个结果元素送入结果向量寄存器的那一个时钟周期才可以进行链接
- 当一条向量指令的两个源操作数分别是两条先行指令的结果寄存器时，要求先行的两条指令产生运算结果的时间必须相等，即要求有关功能部件的通过时间相等。
- 所有可以链接执行的向量指令的向量长度应相等

向量机总结

- 向量机适合挖掘规整的数据级并行
 - 同样的操作作用在许多数据元素上
 - 提高性能、设计简单（向量内的操作相互独立）
- 性能的提升受限于代码的向量化
 - 标量操作限制着向量机的性能
- 很多已有的ISA扩展了一些SIMD操作
 - Intel MMX/SSEn/AVX, PowerPC AltiVec, ARM Advanced SIMD
- 另一种SIMD变体，图形处理单元GPU

本章小结

1. 流水线的基本概念和分类。
2. 衡量流水线性能的主要指标有吞吐率、加速比和效率。
3. 流水线的冲突有结构冲突、数据冲突、控制冲突，分别可以通过硬件技术或软件技术解决。
4. 向量计算机针对大量不相关的数据进行同一种运算，实现了具有向量数据表示和相应向量指令的向量流水线。

第六章作业

王志英教材

第111页： T2、T3、T4、T5。

第 1 章 计算机系统概论

第 2 章 计算机系统量化分析基础

第 3 章 总线

第 4 章 指令系统

第 5 章 CPU设计与实现

第 6 章 基本流水线技术

第 7 章 指令级并行

第 8 章 存储系统的结构与优化

第 9 章 IO系统

第七章 指令级并行

- 7.1 指令级并行的概念
- 7.2 指令的动态调度
- 7.3 控制相关的动态解决技术
- 7.4 多指令流出技术

7.1 指令级并行的概念

7.1.1 循环展开调度的基本方法

7.1.2 相关性

7.1 指令级并行的概念

- 当指令之间不存在相关时，它们在流水线中是可以重叠起来并行执行的。这种指令序列中存在的潜在并行性称为指令级并行
 - Instruction-Level Parallelism
 - 简记为ILP
- 如何知道指令之间可以并行？硬、软件如何支持指令级并行？如何研究这些问题？
 - 硬件技术或者软件技术都可以提高指令级并行性
 - 必须要硬件技术和软件技术互相配合，才能够最大限度地挖掘出程序中存在的指令级并行

性能评价：CPI计算

- 流水线处理器的实际CPI（平均每条指令使用的周期数）等于理想流水线的CPI加上各类停顿引起的周期数的总和

$$\begin{aligned} \text{CPI}_{\text{流水线}} &= \text{CPI}_{\text{理想}} \\ &+ \text{停顿}_{\text{结构冲突}} \\ &+ \text{停顿}_{\text{先写后读}} \\ &+ \text{停顿}_{\text{先读后写}} \\ &+ \text{停顿}_{\text{写后写}} \\ &+ \text{停顿}_{\text{控制冲突}} \end{aligned}$$

- 减少其中的任何一种停顿，都可以有效地减少CPI，从而提高流水线的性能

本章研究的技术及克服的停顿

技术	主要克服的停顿	相关章节
基本流水线调度	数据 先写后读 冲突停顿	7.1
循环展开	控制 冲突停顿	7.1
寄存器换名	数据 写后写 和 先读后写 冲突停顿	7.1
指令动态调度（记分牌和Tomasulo算法）	各种 数据 冲突停顿	7.2
动态分支预测	控制 冲突停顿	7.3
前瞻（Speculation）	所有 数据/控制 冲突停顿	7.3
多指令流出（超标量和超长指令字）	提高 理想CPI	7.4

软件和硬件的支持

- 上述技术中有些技术主要是硬件支持
 - 循环展开
 - 寄存器换名的动态调度（基本的Tomasulo's）
 - 动态分支指令预测
 - 每个周期多发射
 - 前瞻技术
- 所有的技术都必须和软件，特别是编译器合作完成

几个基本概念

- **基本（程序）块**：一段除了入口和出口以外不包含其它分支的线性代码段
 - 程序平均每6~7条指令就会有一个分支
 - 必须在多个基本块之间开发指令级的并行性
- **循环级并行**：开发循环体的不同迭代之间存在的并行性
- 开发循环级并行的基本技术方法
 - 指令调度（scheduling）
 - 循环展开（loop unrolling）
 - 换名（renaming）

7.1 指令级并行的概念

7.1.1 循环展开调度的基本方法

7.1.2 相关性

7.1.1 循环展开调度的基本方法

- 循环展开是展开循环体若干次，将**循环级并行**转化为**指令级并行**的技术
- 这个过程既可以通过编译器**静态**完成，也可以通过硬件**动态**进行
 - 开发循环级并行性的另外一个重要技术是向量处理技术
 - 具有向量处理指令的典型机器是向量计算机，有关向量处理和向量计算机的内容本章不作讨论
- 本章中的**分支指令**就是指**条件转移指令**

本章通用浮点流水线延迟表

- 编译器在完成这种指令调度时，受限于以下两个特性：
 - 一是程序固有的指令级并行性
 - 二是流水线功能部件的执行延迟
- 本章中使用的浮点流水线的延迟如下表

产生结果指令	使用结果指令	延迟时钟数
浮点计算	另外的浮点计算	3
浮点计算	浮点数据存操作（SD）	2
浮点数据取操作（LD）	浮点计算	1
浮点数据取操作（LD）	浮点数据存操作（SD）	0

整数流水线其他特性说明

- 整数流水线采用改进的MIPS整数流水线
 - 载入延迟为1个节拍
 - 由于数据的取操作的结果可以毫无停顿的通过相关通路机制传送到数据存部件，所以延迟为0
 - 有定向通道或旁路机制
- 分支指令，由整数流水线执行
 - 分支条件检测调整到ID段
 - 如果分支指令使用上一条指令的结果作为分支条件，将要延迟1节拍
 - 分支指令有1个节拍的延迟槽
- 浮点运算一般为64位

循环展开实例

- 对于下面的源代码，在不进行指令调度和进行指令调度两种情况下，分析代码一次循环的执行时间。

```
for (i=1; i<=1000; i++)  
    x[i] = x[i] + s;
```

MIPS汇编语言程序

- 程序转换成MIPS汇编语言程序

```
Loop:  LD          F0,0(R1)    ;F0为向量元素
        ADDDD      F4,F0,F2    ;加常数F2
        SD          0(R1),F4    ;保存结果
        SUBI        R1,R1,#8    ;修改指针
        BNEZ        R1,Loop     ;循环控制
```

循环无调度执行

时钟	无调度	
1	LD	F0, 0 (R1)
2	? ADDD	F4, F0, F2



产生结果指令	使用结果指令	延迟时钟数
浮点计算	另外的浮点计算	3
浮点计算	浮点数据存操作 (SD)	2
浮点数据取操作 (LD)	浮点计算	1
浮点数据取操作 (LD)	浮点数据存操作 (SD)	0

循环无调度执行

时钟	无调度	
1	LD	F0, 0 (R1)
2	<i>stall</i>	
3	ADDD	F4, F0, F2
4	? SD	0 (R1), F4

产生结果指令	使用结果指令	延迟时钟数
浮点计算	另外的浮点计算	3
浮点计算	浮点数据存操作 (SD)	2
浮点数据取操作 (LD)	浮点计算	1
浮点数据取操作 (LD)	浮点数据存操作 (SD)	0

循环无调度执行

时钟	无调度
1	LD F0 , 0 (R1)
2	<i>stall</i>
3	ADDD F4 , F0 , F2
4	<i>stall</i>
5	<i>stall</i>
6	SD 0 (R1) , F4
7	? SUBI R1 , R1 , #8

循环无调度执行

时钟	无调度	
1	LD	F0, 0(R1)
2	<i>stall</i>	
3	ADDD	F4, F0, F2
4	<i>stall</i>	
5	<i>stall</i>	
6	SD	0(R1), F4
7	SUBI	R1, R1, #8
8	? BNEZ	R1, LOOP

如果分支指令使用上一条指令的结果作为分支条件，将要延迟1节拍

循环无调度执行

时钟	无调度
1	LD F0 , 0 (R1)
2	<i>stall</i>
3	ADDD F4 , F0 , F2
4	<i>stall</i>
5	<i>stall</i>
6	SD 0 (R1) , F4
7	SUBI R1 , R1 , #8
8	<i>stall</i>
9	BNEZ R1 , LOOP

循环无调度执行

时钟	无调度
1	LD F0, 0 (R1)
2	<i>stall</i>
3	ADDD F4, F0, F2
4	<i>stall</i>
5	<i>stall</i>
6	SD 0 (R1), F4
7	SUBI R1, R1, #8
8	<i>stall</i>
9	BNEZ R1, LOOP
10	?

分支指令有1个节拍的延迟槽

循环无调度执行

时钟	无调度
1	LD F0 , 0 (R1)
2	<i>stall</i>
3	ADDD F4 , F0 , F2
4	<i>stall</i>
5	<i>stall</i>
6	SD 0 (R1) , F4
7	SUBI R1 , R1 , #8
8	<i>stall</i>
9	BNEZ R1 , LOOP
10	<i>stall</i>

循环无调度执行结果分析

- 每遍循环需要10个时钟节拍
- 只有3个时钟节拍(LD, ADDD, SD)
 - 有效比率为30%
 - 空转5个时钟节拍 (50%)
 - 循环控制2个时钟节拍 (20%)
- 调度代码，减少空转

调度代码

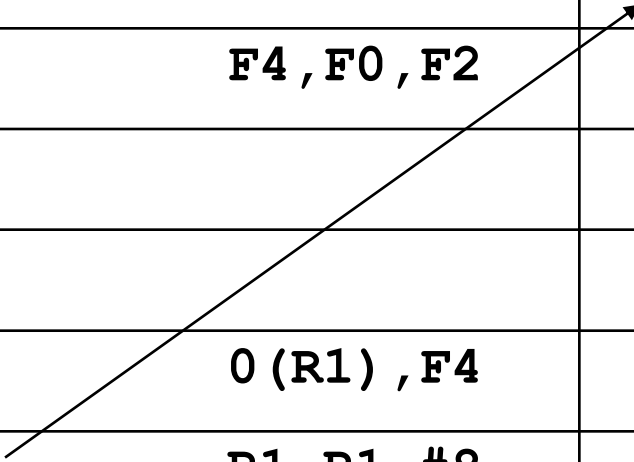
时钟	未调度	调度后
1	LD F0 , 0 (R1)	LD F0 , 0 (R1)
2	<i>Stall</i>	
3	ADDD F4 , F0 , F2	
4	<i>Stall</i>	
5	<i>Stall</i>	
6	SD 0 (R1) , F4	
7	SUBI R1 , R1 , #8	
8	<i>Stall</i>	
9	BNEZ R1 , Loop	
10	<i>Stall</i>	

调度代码

时钟	未调度	调度后
1	LD F0 , 0 (R1)	LD F0 , 0 (R1)
2	<i>Stall</i>	?
3	ADDD F4 , F0 , F2	
4	<i>Stall</i>	
5	<i>Stall</i>	
6	SD 0 (R1) , F4	
7	SUBI R1 , R1 , #8	
8	<i>Stall</i>	
9	BNEZ R1 , Loop	
10	<i>Stall</i>	

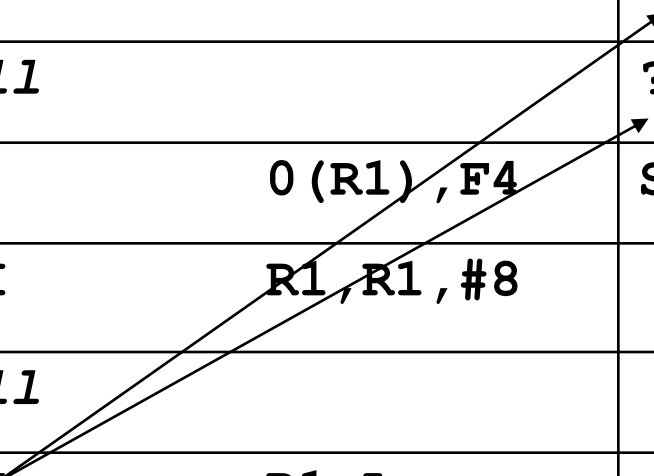
调度代码

时钟	未调度	调度后
1	LD F0 , 0 (R1)	LD F0 , 0 (R1)
2	<i>Stall</i>	SUBI R1 , R1 , #8
3	ADDD F4 , F0 , F2	
4	<i>Stall</i>	
5	<i>Stall</i>	
6	SD 0 (R1) , F4	
7	SUBI R1 , R1 , #8	
8	<i>Stall</i>	
9	BNEZ R1 , Loop	
10	<i>Stall</i>	



调度代码

时钟	未调度	调度后
1	LD F0, 0(R1)	LD F0, 0(R1)
2	<i>Stall</i>	SUBI R1, R1, #8
3	ADDD F4, F0, F2	ADDD F4, F0, F2
4	<i>Stall</i>	??
5	<i>Stall</i>	??
6	SD 0(R1), F4	SD 0(R1), F4
7	SUBI R1, R1, #8	
8	<i>Stall</i>	
9	BNEZ R1, Loop	
10	<i>Stall</i>	



调度代码

时钟	未调度	调度后
1	LD F0 , 0 (R1)	LD F0 , 0 (R1)
2	<i>Stall</i>	SUBI R1 , R1 , #8
3	ADDD F4 , F0 , F2	ADDD F4 , F0 , F2
4	<i>Stall</i>	<i>Stall</i>
5	<i>Stall</i>	BNEZ R1 , Loop
6	SD 0 (R1) , F4	SD 8 (R1) , F4
7	SUBI R1 , R1 , #8	
8	<i>Stall</i>	
9	BNEZ R1 , Loop	
10	<i>Stall</i>	

如何进行调度

- 通过一个好的编译器来调度可以
 - 调换SUBI和SD的位置
 - 将SD指令移到BNEZ的延迟槽内
 - 改变SD存取指令访问内存地址的偏移量

调度代码结果分析

- 每遍循环6个时钟节拍
- 和未调度代码比较，加速比 $10/6=1.7$
- 3个有效时钟节拍 (LD, ADDD, SD)
 - 节拍有效比率50%
 - 1拍空转 (占17%)
 - 2拍循环控制 (占33%)
- 如何进一步减少空转和循环控制占用的比率？

循环展开

- 如果将循环展开3次得到4个循环体(假设数组是4的倍数的元素)

```
Loop:    LD      F0,0(R1)
         ADDDD   F4,F0,F2
         SD      0(R1),F4
         LD      F6,-8(R1)
         ADDDD   F8,F6,F2
         SD      -8(R1), F8
         LD      F10,-16(R1)
         ADDDD   F12,F10,F2
         SD      -16(R1), F12
         LD      F14,-24(R1)
         ADDDD   F16,F14,F2
         SD      -24(R1), F16
         SUBI    R1,R1,#-32
         BNEZ    R1,Loop
```

循环无调度执行

时钟	无调度
1	LD F0 , 0 (R1)
2	<i>stall</i>
3	ADDD F4 , F0 , F2
4	<i>stall</i>
5	<i>stall</i>
6	SD 0 (R1) , F4
7	SUBI R1 , R1 , #8
8	<i>stall</i>
9	BNEZ R1 , LOOP
10	<i>stall</i>

执行时间分析

Loop:	LD	F0,0(R1)	1
	ADDD	F4,F0,F2	2,3
	SD	0(R1),F4	4,5,6
	LD	F6,-8(R1)	7
	ADDD	F8,F6,F2	8,9
	SD	-8(R1), F8	10,11,12
	LD	F10,-16(R1)	13
	ADDD	F12,F10,F2	14,15
	SD	-16(R1), F12	16,17,18
	LD	F14,-24(R1)	19
	ADDD	F16,F14,F2	20,21
	SD	-24(R1), F16	22,23,24
	SUBI	R1,R1,#-32	25
	BNEZ	R1,Loop	26,27
	stall		28

结果分析

- 循环使用28个时钟节拍
 - 14个空转节拍
 - 每个LD有1个空转个节拍 – 共4拍
 - 每个ADDD有2个空转节拍 - 共8拍
 - SUBI有1个空转节拍 - 共1拍
 - BRANCH有1个空转节拍 - 共1拍
 - 有14个指令流出节拍
- 平均每遍循环7个时钟节拍
- 共计使用9个寄存器

循环展开+指令调度

1. Loop:	LD	F0,0(R1)
2.	LD	F6,-8(R1)
3.	LD	F10,-16(R1)
4.	LD	F14,-24(R1)
5.	ADDD	F4,F0,F2
6.	ADDD	F8,F6,F2
7.	ADDD	F12,F10,F2
8.	ADDD	F16,F14,F2
9.	SD	0(R1),F4
10.	SD	-8(R1),F8
11.	SUBI	R1,R1,#-32
12.	SD	16(R1),F12
13.	BNEZ	R1,R2,Loop
14.	SD	8(R1),F16

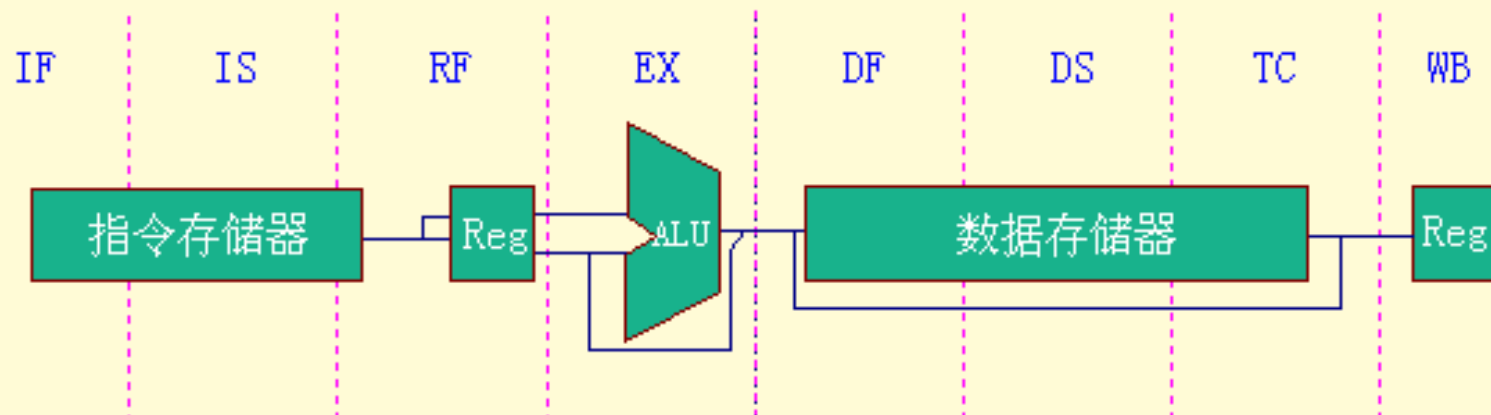
“循环展开+指令调度”结果分析

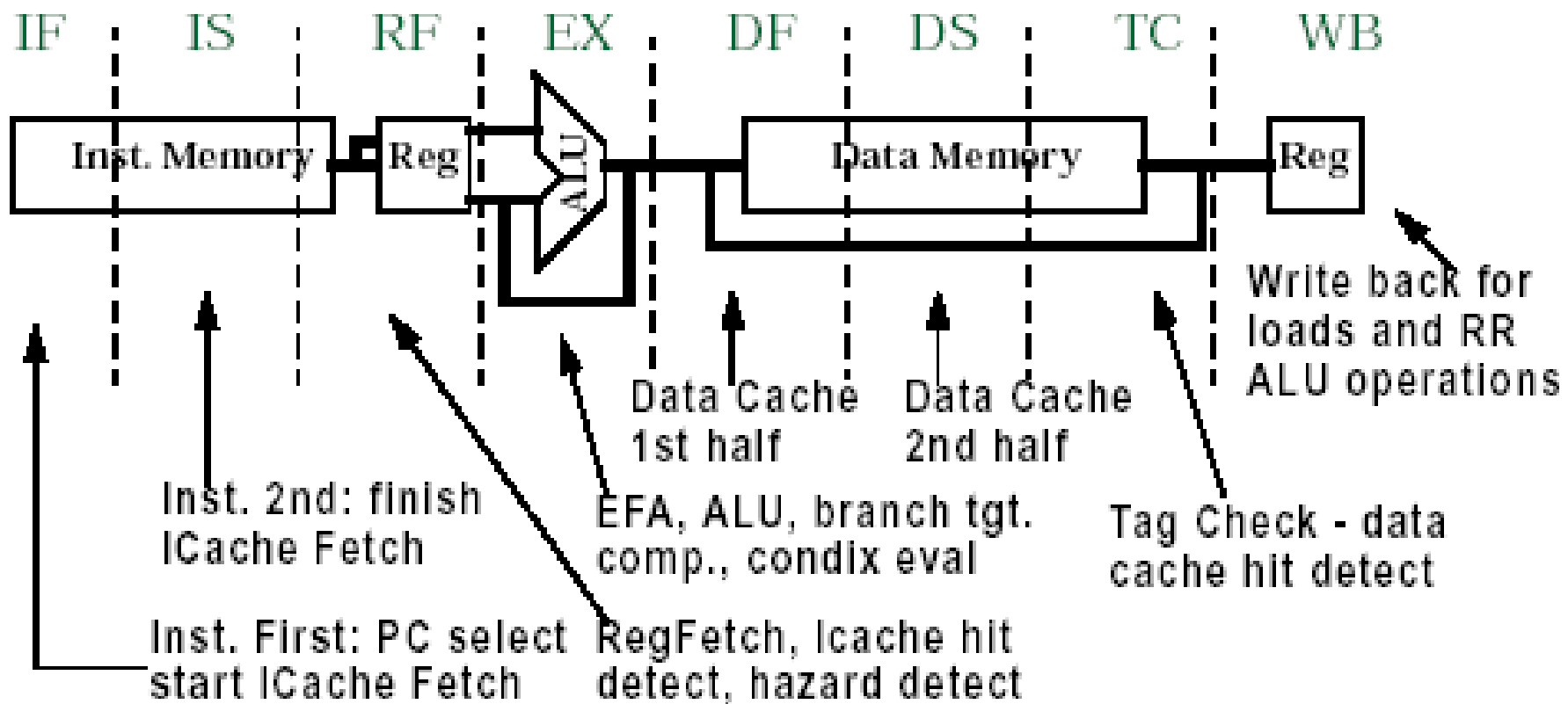
- 每遍循环时间下降为14个时钟节拍
 - 每个元素平均使用3.5个时钟节拍
- 比较
 - 没有循环展开，有指令调度
 - 每个元素6拍
 - 有循环展开，没有指令调度
 - 每个元素7拍

循环展开和指令调度的总结

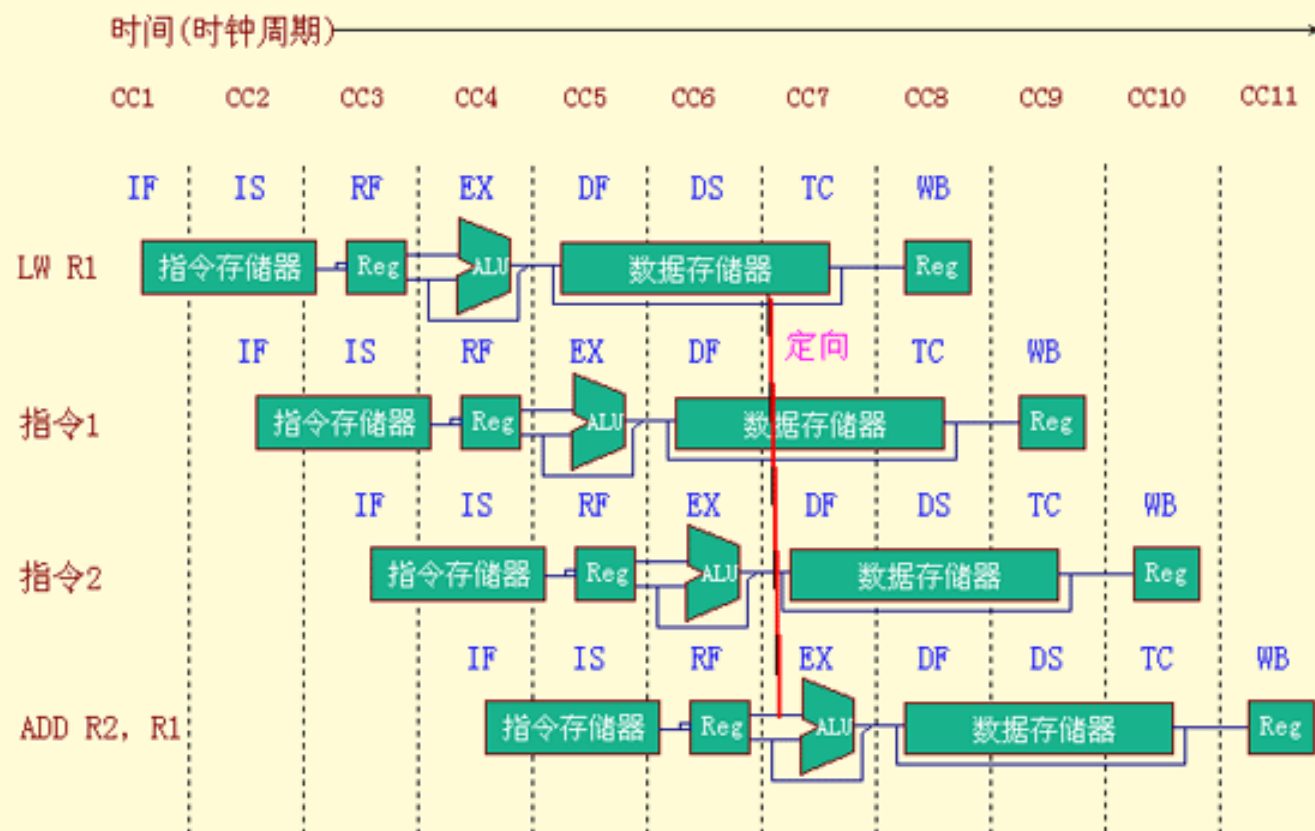
- 保证正确性（循环控制和操作数偏移量的修改）
- 注意有效性（找到不同循环体之间的无关性）
- 使用不同的寄存器
- 减少循环控制中的测试指令和分支指令
- 注意分析Load/Store指令的内存地址
- 注意新的相关性

R4000流水线的结构

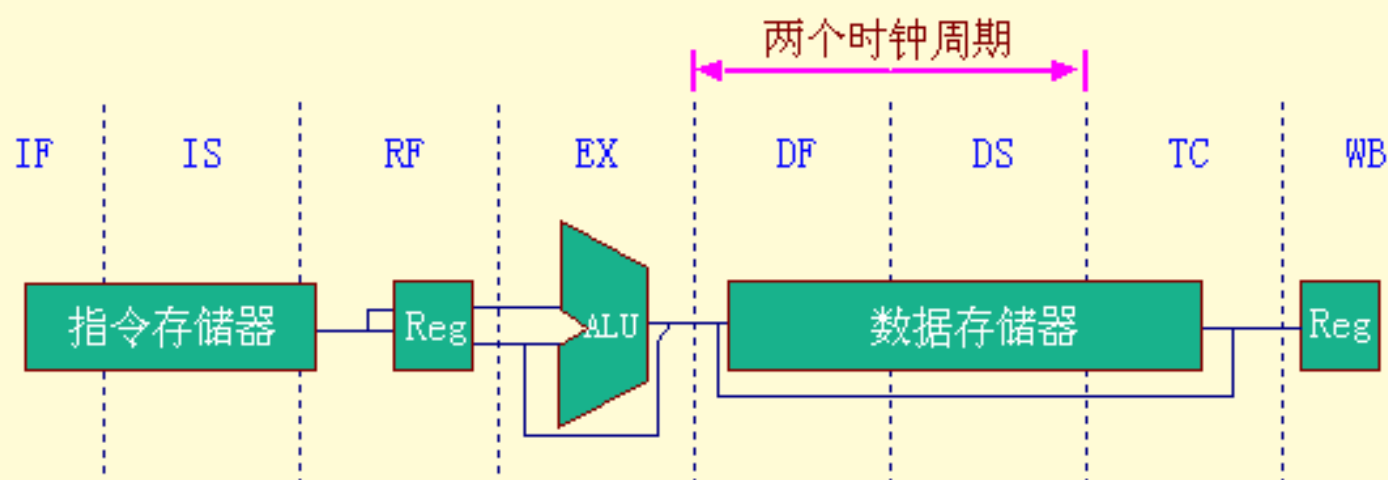




指令序列在R4000流水线中的重叠执行过程



载入延迟为两个时钟周期

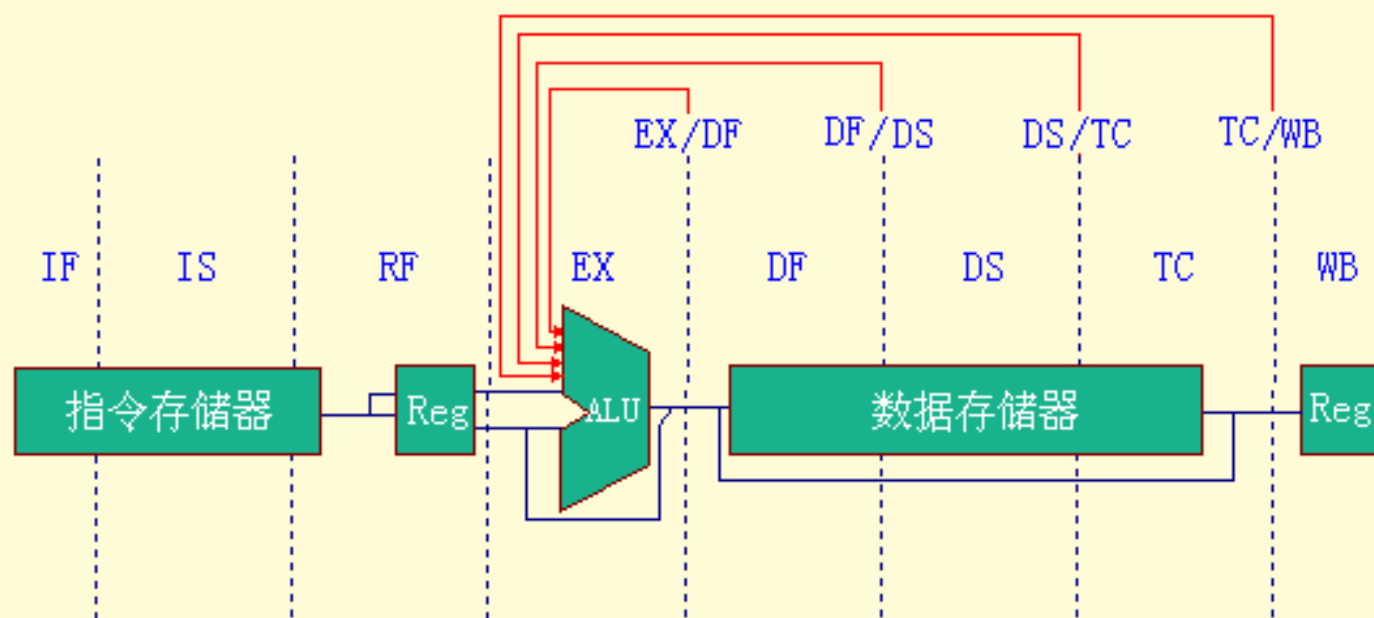


指令系列在R4000流水线中的执行时空图

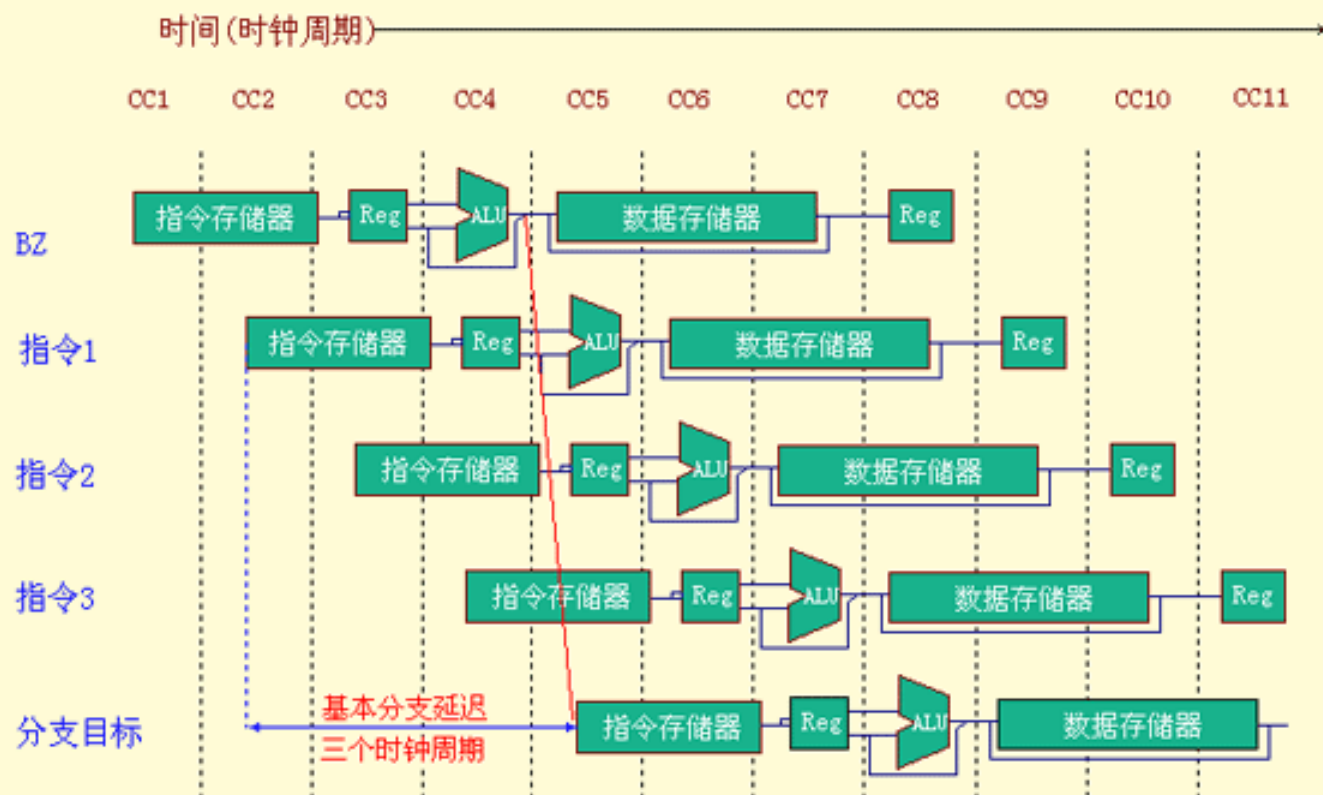
指令序列	时钟周期								
	1	2	3	4	5	6	7	8	9
LW R1		IS	RF	EX	DF	DS	TC	WB	
ADD R2, R1			IS	RF	stall	stall	EX	DF	DS
SUB R3, R1			IF	IS	stall	stall	RF	EX	DF
OR R4, R1				IF	stall	stall	IS	RF	EX

定向

R4000的流水线中ALU输入有四个定向源



R4000流水线的分支延迟



R4000流水线处理分支指令的时空图

指令序列	时钟周期								
	1	2	3	4	5	6	7	8	9
分支指令	IF	IS	RF	EX	DF	DS	TC	WB	
延迟槽		IF	IS	RF	EX	DF	DS	TC	WB
暂停			stall	stall	stall	stall	stall	stall	stall
暂停				stall	stall	stall	stall	stall	stall
分支目标					IF	IS	RF	EX	DF

两个暂停周期

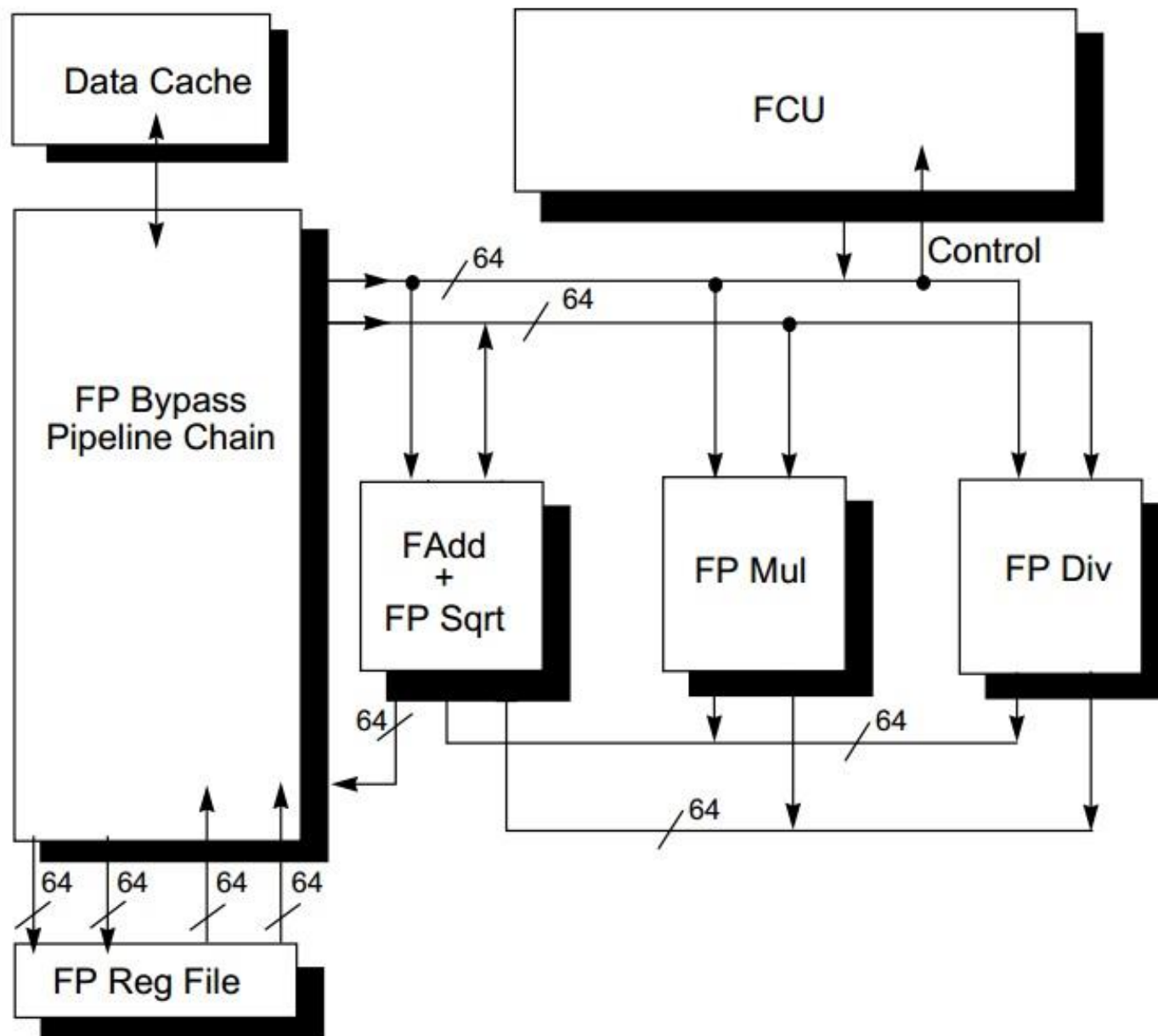
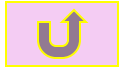


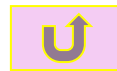
Figure 6-1 FPU Functional Block Diagram

R4000浮点流水线中8个流水段



流水段	功能部件	描述
A	浮点加法器	尾数加
D	浮点除法器	除法
E	浮点乘法器	例外测试
M	浮点乘法器	乘法第一阶段
N	浮点乘法器	乘法第二阶段
R	浮点加法器	舍入
S	浮点加法器	操作数移位
U		展开浮点数

双精度浮点操作指令延迟、初始化间隔和流水段的使用情况



浮点指令	延迟	初始化间隔	使用的流水段
加、减	4	3	U,S+A,A+R,R+S
乘	8	4	U,E+M,M,M,M,N,N+A,R
除	36	35	U,A,R,D ²⁸ ,D+A,D+R,D+A,D+R,A,R
求平方根	112	111	U,E,(A+R) ¹⁰⁸ ,A,R
取反	2	1	U,S
求绝对值	2	1	U,S
浮点比较	3	2	U,A,R

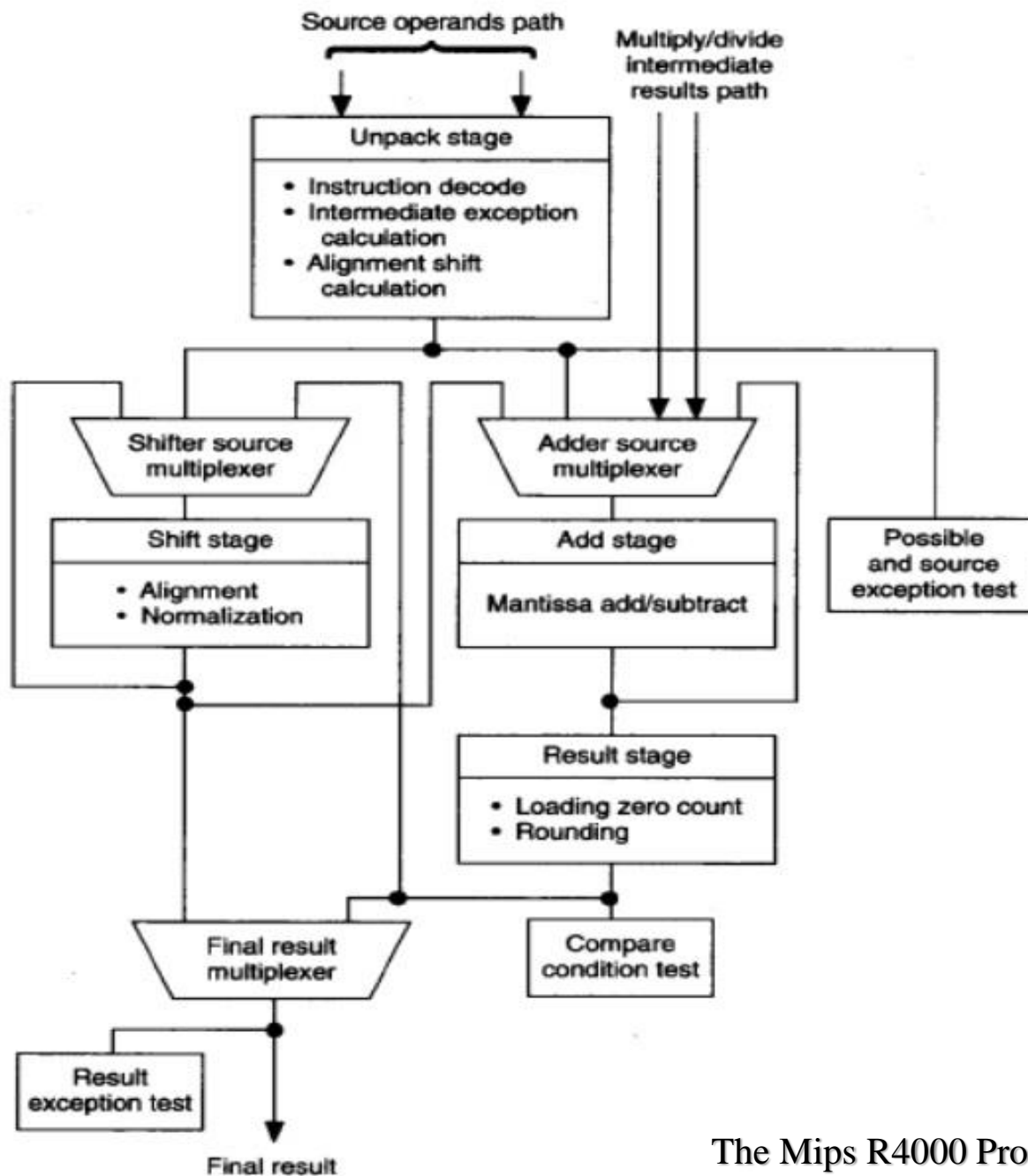


Figure 7. Adder logical block diagram.

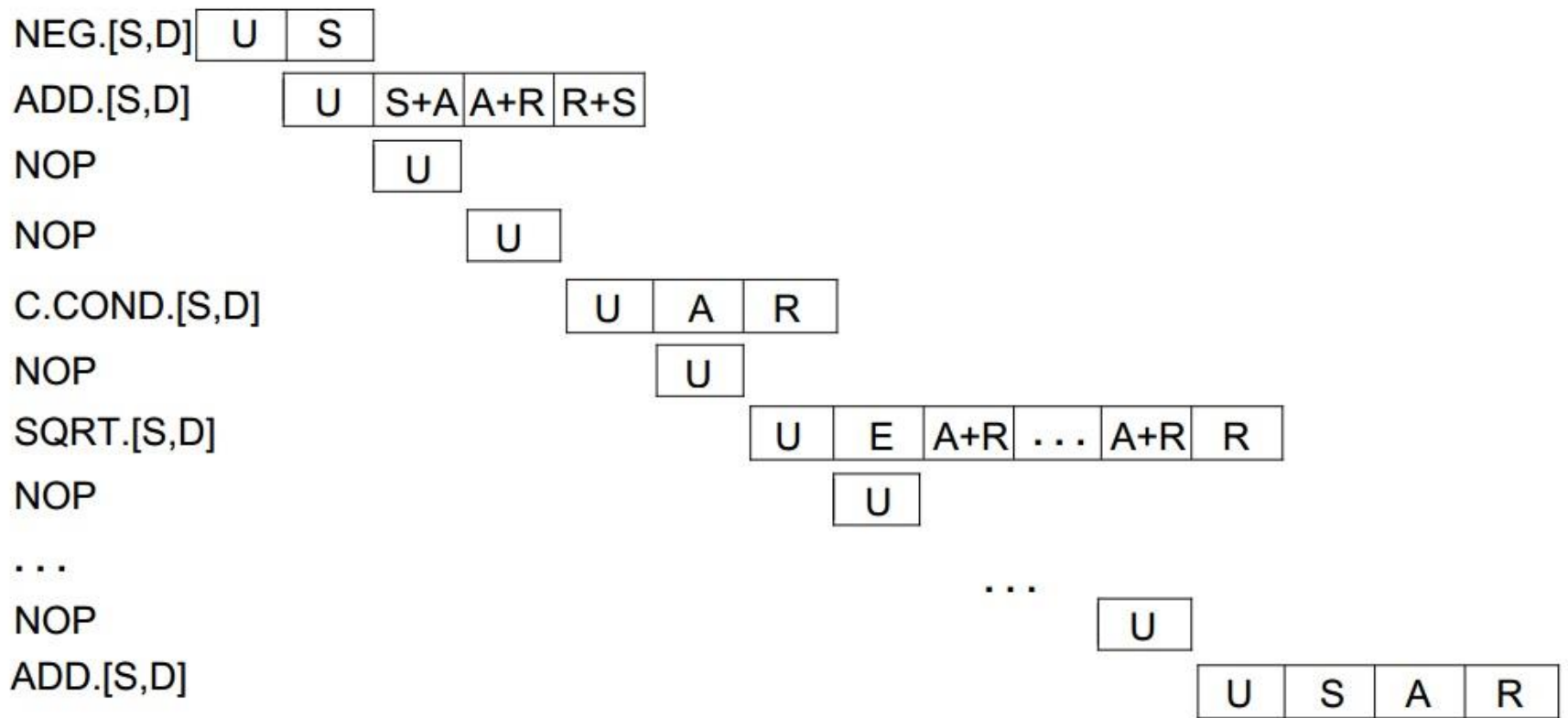
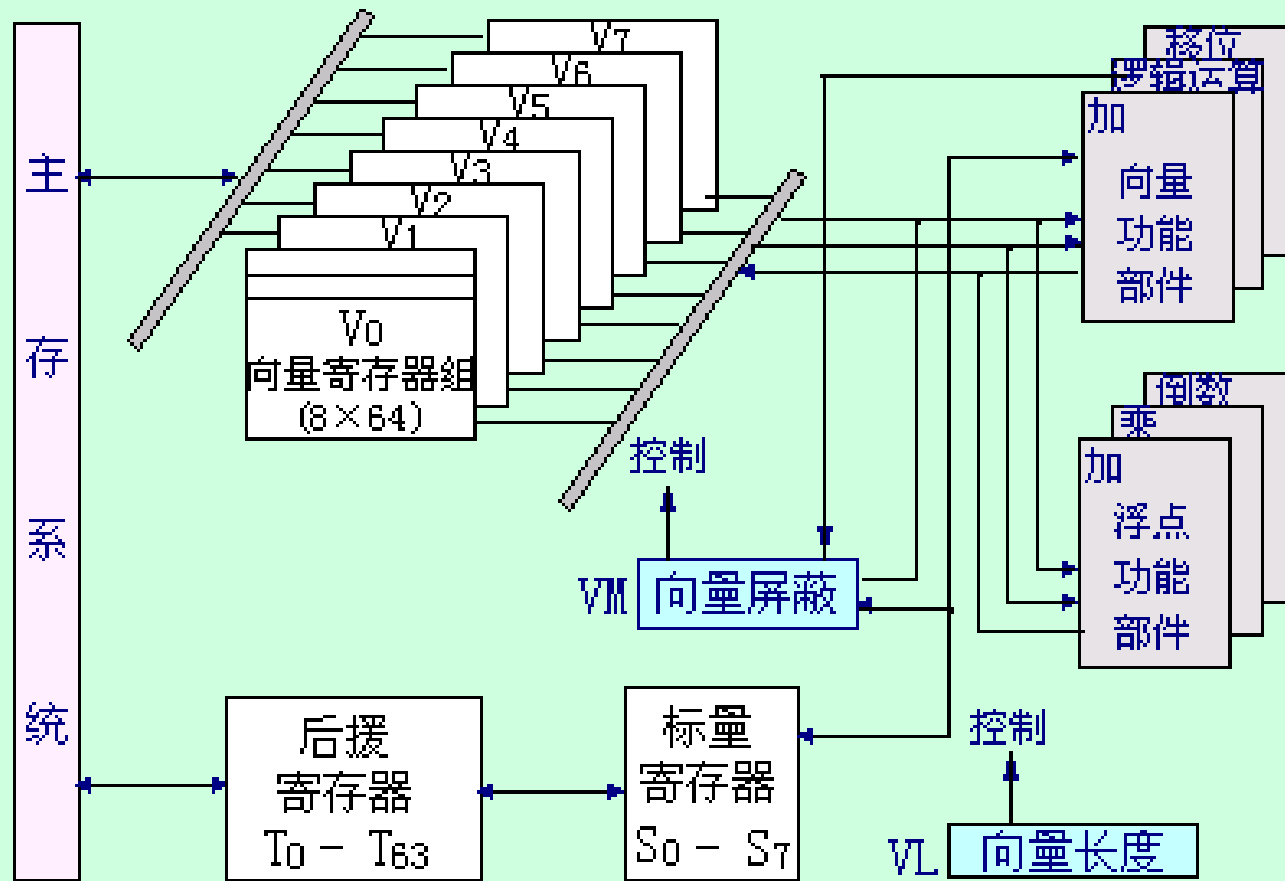
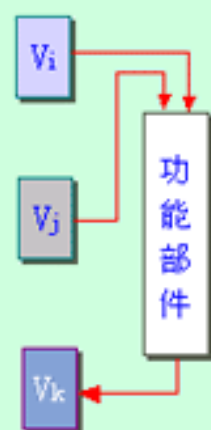


Figure 6-13 Instruction Cycle Overlap in FPU Adder

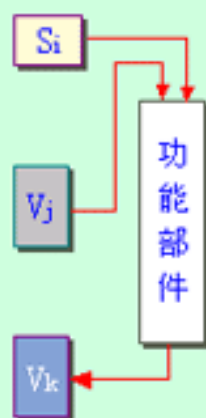
CRAY-1的基本结构



CRAY-1的向量指令类型



$V_k \leftarrow V_i \text{ op } V_j$



$V_k \leftarrow S_i \text{ op } V_j$



$V_k \leftarrow \text{主存}$



$\text{主存} \leftarrow V_i$

链接特征

