第4章 指令级并行

硬件

软件

4.1 指令级并行

4.1.1 指令级并行的概念

- 几乎所有的处理机都利用流水线来使指令重叠并行执行，以达到提高性能的目的。这种指令之间存在的潜在并行性称为指令级并行。

（ILP：Instruction-Level Parallelism）

1. 流水线处理机的实际CPI

- 理想流水线的CPI加上各类停顿的时钟周期数：

$$CPI_{流水线} = CPI_{理想} + 停顿_{结构冲突} + 停顿_{数据冲突} + 停顿_{控制冲突}$$

- 理想CPI是衡量流水线最高性能的一个指标。
- IPC：Instructions Per Cycle

（每个时钟周期完成的指令条数）

2. 基本程序块

- 基本程序块：一段除了入口和出口以外不包含其他分支的线性代码段。
- 程序平均每5～7条指令就会有一个分支。

3. 循环级并行：使一个循环中的不同循环体并行执行。

- 开发循环体中存在的并行性
  - 最常见、最基本
- 指令级并行研究的重点之一

例如，考虑下述语句：

```
for （i=1； i<=500； i=i＋1）
a[i]=a[i]＋s；
```

- 每一次循环都可以与其他的循环重叠并行执行；
- 在每一次循环的内部，却没有任何的并行性。

4. 最基本的开发循环级并行的技术

- 循环展开（loop unrolling）技术
- 采用向量指令和向量数据表示

5. 相关与流水线冲突

- 相关有三种类型：

  数据相关、名相关、控制相关

- 流水线冲突是指对于具体的流水线来说，由于相关的存在，使得指令流中的下一条指令不能在指定的时钟周期执行。

  流水线冲突有三种类型：结构冲突、数据冲突、控制冲突

- 相关是程序固有的一种属性，它反映了程序中指令之间的相互依赖关系。
- 具体的一次相关是否会导致实际冲突的发生以及该冲突会带来多长的停顿，则是流水线的属性。

6. 可以从两个方面来解决相关问题：

- 指令调度： 保持相关，但避免发生冲突。
- 通过代码变换(寄存器重命名)，消除相关。

7. 对于正确地执行程序来说，必须保持的最关键的两个属性是：数据流和异常行为。

- 保持异常行为是指：无论怎么改变指令的执行顺序，都不能改变程序中异常的发生情况。
  - 即原来程序中是怎么发生的，改变执行顺序后还是怎么发生。
  - 弱化为：指令执行顺序的改变不能导致程序中发生新的异常。
- 如果我们能做到保持程序的数据相关和控制相关，就能保持程序的数据流和异常行为。

- 举例说明

```
        DADDU    R2，R3，R4
        BEQZ     R2，L1
        LW       R1，0（R2）
L1 ：
```

➢ 数据流：指数据值从其产生者指令到其消费者指令的实际流动。

- 分支指令使得数据流具有动态性，因为它使得给定指令的数据可以有多个来源。
- 仅仅保持数据相关性是不够的，只有再加上保持控制顺序，才能够保持程序顺序。

- 举例：

```
        DADDU    R1，R2，R3
        BEQZ     R4，L1
        DSUBU    R1，R5，R6
L1 ：…
        OR       R7，R1，R8
```

- 有时，不遵守控制相关既不影响异常行为，也不改变数据流。
  - 可以大胆地进行指令调度，把失败分支中的指令调度到分支指令之前。

- 举例：

```
            DADDU     R1, R2, R3
            BEQZ      R12, Skipnext
            DSUBU     R4, R5, R6
            DADDU     R5, R4, R9
Skipnext:   OR        R7, R8, R9
```

4.5 循环展开和指令调度

4.5.1 循环展开和指令调度的基本方法

1. 充分开发指令之间存在的并行性，找出不相关的指令序列，让它们在流水线上重叠并行执行。
2. 增加指令间并行性最简单和最常用的方法
   - 开发循环级并行性——循环的不同迭代之间存在的并行性。
   - 在把循环展开后，通过重命名和指令调度来开发更多的并行性。

3. 编译器完成这种指令调度的能力受限于两个特性：
   - 程序固有的指令级并行性；
   - 流水线功能部件的执行延迟。
4. 本节中，假定我们使用的浮点流水线延迟为：

产生结果的指令	使用结果的指令	延迟（时钟周期数）
浮点计算	另一个浮点计算	3
浮点计算	浮点store（S.D）	2
浮点load（L.D）	浮点计算	1
浮点load（L.D）	浮点store（S.D）	0

例4.6　对于下面的源代码，转换成MIPS汇编语言，在不进行指令调度和进行指令调度两种情况下，分析其代码一次循环所需的执行时间。

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

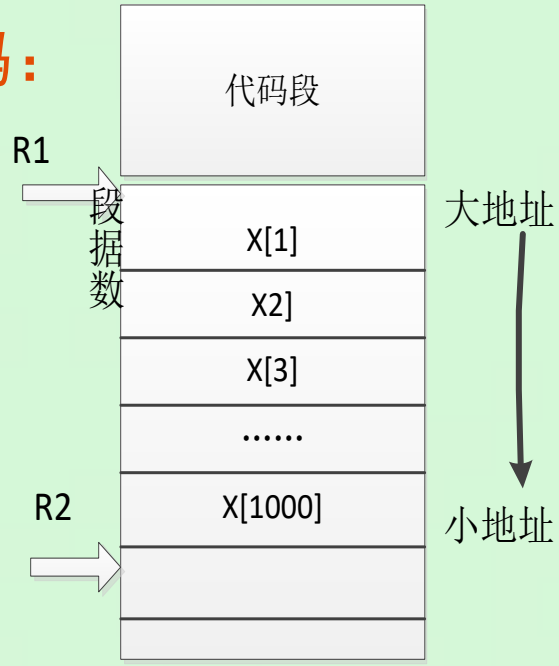
解：把该程序翻译成MIPS汇编语言代码：

假设：

①R1的初值是指向第一个元素

②8（R2）指向最后一个元素。

③浮点寄存器F2：用于保存常数s。



产生结果的指令	使用结果的指令	延迟（时钟周期数）
浮点计算	另一个浮点计算	3
浮点计算	浮点store（S.D）	2
浮点load（L.D）	浮点计算	1
浮点load（L.D）	浮点store（S.D）	0

4.5 循环展开和指令调度（课本4.5节）

```
Loop：L.D    F0, 0（R1）
1→
ADD.D        F4, F0, F2
2→
S.D          F4, 0（R1）
             DADDIU  R1, R1, #-8
1→
             BNE     R1, R2, Loop
```

① $F_0 \leftarrow R_1$

② $F_4 \leftarrow F_0 + F_2$

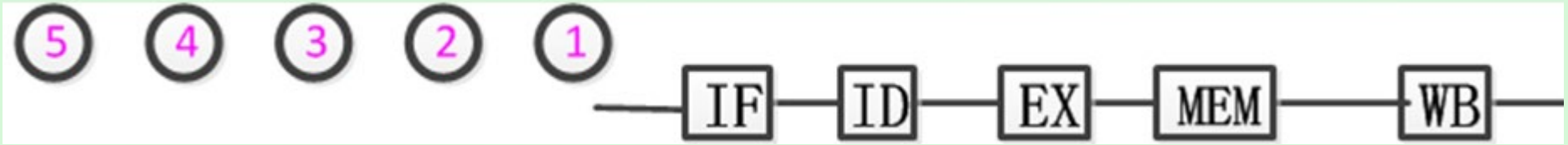
③ $R_1 \leftarrow F_4$

④ $R_1 \leftarrow R_1 - 8$

⑤ $R_1 != R_2$?

其中：

- 整数寄存器R1：指向向量中的当前元素。（初值为向量中最高端元素的地址）

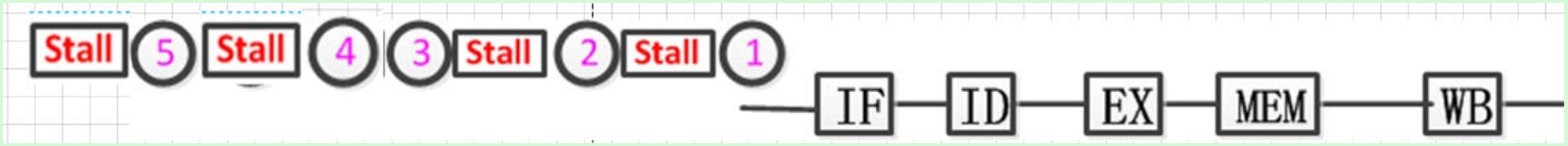


指令解释流水线

- 不进行指令调度的情况下，程序的实际执行情况：

		指令流出时钟
Loop: L.D F0,0（R1）	①	1
（空转）		2
ADD.D F4,F0,F2	②	3
（空转）		4
（空转）		5
S.D F4, 0（R1）	③	6
DADDIU R1,R1, # -8	④	7
（空转）		8
BNE R1,R2,Loop	⑤	9
（空转）（等待分支指令ID阶段产生Cond）		10

每个元素的操作需要10个时钟周期，其中5个是空转周期。



1. 如果进行如下的指令调度：

- 把DADDIU指令调度到L.D指令和ADD.D指令之间的“空转”拍；
- 把S.D指令放到了分支指令的延迟槽中；
- 对存储器地址偏移量进行调整；

指令调度前

```
Loop: L.D      F0,0（R1）
      （空转）
      ADD.D    F4,F0,F2
      （空转）
      （空转）
      S.D      F4, 0（R1）
      DADDIU   R1,R1,#-8
      （空转）
      BNE      R1,R2,Loop
      （空转）
```

指令调度后

```
Loop: L.D      F0, 0(R1)
      DADDIU   R1,R1,#-8
      ADD.D    F4, F0, F2
      （空转）
      BNE      R1,R2,Loop
      S.D      F4, 8(R1)
```

④ ↑ 间隔 >1指 ⑤

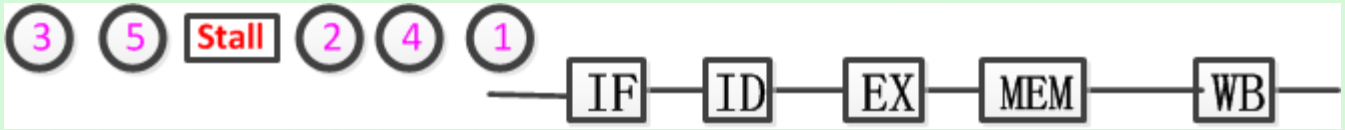
DADDIU: IF ID EX MEM WB

BNE: IF ~~ID~~ → ID

stall

		指令流出时钟
Loop:	L.D　　F0, 0(R1)	1
	DADDIU R1, R1, #-8	2
	ADD.D　F4, F0, F2	3
	（空转）	4
	BNE　　R1, Loop	5
	S.D　　F4, 8(R1)	6

一轮循环的操作时间从10个时钟周期减少到6个，其中5个周期是有指令执行的，1个为空转周期。



指令送出的顺序已经变为：1, 4, 2, 5, 3

1. 例子中的问题及解决方案

- 只有L.D、ADD.D和S.D这3条指令是有效操作。
  - 占用3个时钟周期。
  - 而DADDIU、空转和BEN这3个时钟周期都是附加的循环控制开销。
- 循环展开技术
  - 把循环体的代码复制多次并按顺序排列，然后相应调整循环的结束条件。
  - 这给编译器进行指令调度带来了更大的空间。

例4.7：　将上述例子中的循环展开4次得到4个循环体，然后对展开后的指令序列在不调度和调度两种情况下，分析代码的性能。假定R1的初值为32的倍数，即循环次数为4的倍数。消除冗余的指令，并且不要重复使用寄存器。

```
Loop: L.D    F0, 0(R1)
      ADD.D  F4, F0, F2
      S.D    F4, 0(R1)
      DADDUI R1, R1, #8
      BNE    R1, R2, Loop
```

解：无需在循环体后面增加补偿代码

1. 分配寄存器（不重复使用寄存器 ）：

- F0、F4：用于展开后的第1个循环体
- F2：保存常数s
- F6、F8：展开后的第2个循环体
- F10、F12：第3个循环体
- F14、F16：第4个循环体

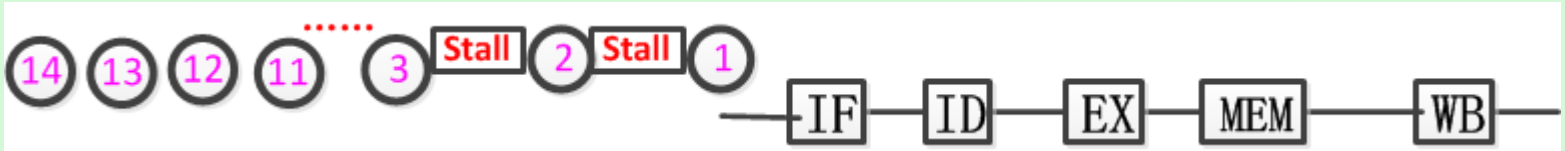
物用来存放 X[i]
i=1,2,3,4.

2. 相应的修改偏移地址（ 以R1为参考）：

0（R1），-8（R1），-16（R1），-24（R1）

4个循环体展开后没有调度的代码如下：

				指令流出时钟
Loop:	L.D	F0, 0(R1)	①	1
	（空转）			2
	ADD.D	F4, F0, F2	②	3
	（空转）			4
	（空转）			5
	S.D	F4, 0（R1）	③	6
	L.D	F6, -8(R1)	④	7
	（空转）			8
	ADD.D	F8, F6, F2	⑤	9
	（空转）			10
	（空转）			11
	S.D	F8, -8（R1）	⑥	12
	L.D	F10, -16(R1)	⑦	13
	（空转）			14
	ADD.D	F12, F10, F2	⑧	15
	（空转）			16
	（空转）			17
	S.D	F12, -16(R1)	⑨	18
	L.D	F14, -24(R1)	⑩	19
	（空转）			20
	ADD.D	F16, F14, F2	⑪	21
	（空转）			22
	（空转）			23
	S.D	F16, -24(R1)	⑫	24
	DADDIU	R1, R1, #-32	⑬	25
	（空转）			26
	BNE	R1, R2, Loop	⑭	27
	（空转）			28



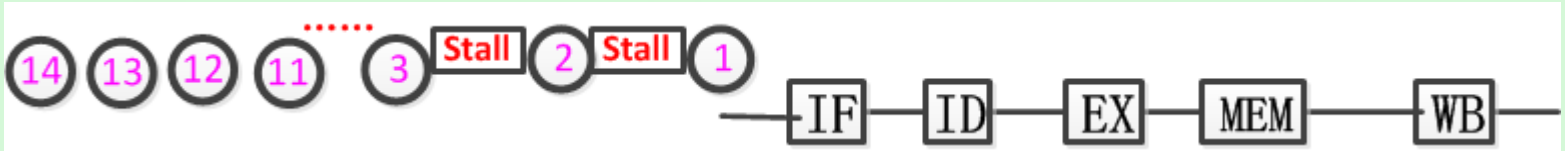
- 结果分析:
  - 这个循环每遍共使用了28个时钟周期。
  - 有4个循环体，完成4个元素的操作。平均每个元素使用28/4=7个时钟周期
  - 原始循环的每个元素需要10个时钟周期。节省的时间：从减少循环控制的开销中获得的。
  - 在整个展开后的循环中，实际指令只有14条，其他14个周期都是空转。

In conclusion, it's not efficient.

移动时只移动LD指令（往前）和SD指令（往后）：

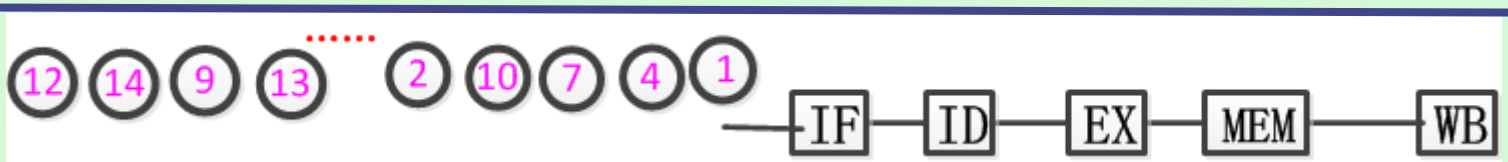
			指令流出时钟
Loop:	L.D	F0,0(R1)	1
	（空转）		2
	ADD.D	F4,F0,F2	3
	（空转）		4
	（空转）		5
	S.D	F4, 0 (R1)	6
	L.D	F6,-8(R1)	7
	（空转）		8
	ADD.D	F8,F6,F2	9
	（空转）		10
	（空转）		11
	S.D	F8, -8 (R1)	12
	L.D	F10,-16(R1)	13
	（空转）		14

			指令流出时钟
	ADD.D	F12,F10,F2	15
	（空转）		16
	（空转）		17
	S.D	F12,-16(R1)	18
	L.D	F14,-24(R1)	19
	（空转）		20
	ADD.D	F16,F14,F2	21
	（空转）		22
	（空转）		23
	S.D	F16,-24(R1)	24
	DADDIU	R1,R1,#-32	25
	（空转）		26
	BNE	R1,R2,Loop	27
	（空转）		28



对指令序列进行优化调度，以减少空转周期：

				指令流出时钟
Loop:	L.D	F0,0(R1)	①	1
	L.D	F6,-8(R1)		2
	L.D	F10,-16(R1)		3
	L.D	F14,-24(R1)		4
	ADD.D	F4,F0,F2	②	5
	ADD.D	F8,F6,F2		6
	ADD.D	F12,F10,F2		7
	ADD.D	F16,F14,F2		8
	S.D	F4,0(R1)	③	9
	S.D	F8,-8(R1)		10
	DADDIU	R1,R1,#-32		12
	S.D	F12,16(R1)		11
	BNE	R1,R2,Loop		13
	S.D	F16,8(R1)		14



- 结果分析：
  - 没有数据相关引起的空转等待。
  - 整个循环仅仅使用了14个时钟周期。
  - 平均每个元素的操作使用14/4=3.5个时钟周期。
  - 通过循环展开、寄存器重命名和指令调度，可以有效地开发出指令级并行。

- 循环展开和指令调度时要注意以下几个方面：
  - 保证正确性。
    - 在循环展开和调度过程中尤其要注意两个地方的正确性：循环控制，操作数偏移量的修改。
  - 注意有效性。
    - 只有能够找到不同循环体之间的无关性，才能有效地使用循环展开。
  - 使用不同的寄存器,否则可能导致新的冲突。
  - 注意对存储器数据的相关性分析
  - 注意新的相关性
    - 由于原循环不同次的迭代在展开后都到了同一次循环体中，因此可能带来新的相关性。