

西安交通大学本科生课程：计算机系统结构

# 《计算机系统结构》

## 第四章 指令级并行执行与调度

郑庆华 教授

西安交大计算机系  
2013年4月

E-mail: qhzheng@mail.xjtu.edu.cn 郑庆华 教授

西安交通大学本科生课程：计算机系统结构

# 引言

- 标量处理机：只有数据表示和标量指令的处理机。
- 提高处理机指令执行速度的三条途径：
  - 1) Approach-1: CPU工作主频的提高
  - 2) Approach-2: 采用更好的算法和功能部件。例如：采用RISC，改进乘法、除法的算法。
  - 3) Approach-3: 指令并行技术：① 流水线技术；② 超标量超流水线技术，即：一个处理机中设置多个独立的部件。如：Pentium；③ 超长指令字，在一条指令中，设置有多个独立的操作字段，每个字段可以独立控制。

E-mail: qhzheng@mail.xjtu.edu.cn 郑庆华 教授

## 本章内容提纲

- 4.1 指令级并行的概念
- 4.2 指令的动态调度
- 4.3 动态分支预测技术
- 4.4 多指令流出技术
- 4.5 循环展开和指令调度
- 4.6 Pentium计算机原理

## 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)
- **本章研究**：如何通过各种可能的技术，获得更多的指令级并行性。  
**硬件+软件技术**  
必须要硬件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。

### 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**
  - 是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;
```

    - 每一次循环可以与其他循环重叠并行执行；
    - 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. **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。
  - 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：
  - 保持相关，但避免发生冲突。
  - 通过代码变换，消除相关。
7. **控制相关并不是一个必须严格保持的关键属性**

#### 8. 程序正确执行的两个基本属性：数据流和异常行为。

- **保持异常行为**：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。
  - 弱化为：指令执行顺序的改变不会引发新的异常。
- **数据流**：
  - 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。改为：

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

注意：假设已知R4在Skipnext之后不再用到，而且DSUBU不会产生异常，那么就可以挪到BEQZ之前。

## 本章内容提纲

- 4.1 指令级并行的概念
- 4.2 指令的动态调度
- 4.3 动态分支预测技术
- 4.4 多指令流出技术
- 4.5 循环展开和指令调度
- 4.6 Pentium计算机原理

## 4.2 指令的动态调度

- **静态调度**
  - 依靠编译器对代码进行静态调度，以减少相关和冲突
  - 基本思路：通过把相关的指令拉开距离以减少可能产生的停顿。
- **动态调度**
  - 在程序的执行过程中，依靠专门硬件对代码进行调度，减少数据相关导致的停顿。
    - 能够处理一些在编译时情况不明的相关（比如涉及到存储器访问的相关），并简化了编译器；
    - 能够使本来仅面向某一流水线优化编译的代码在其他的流水线（动态调度）上也能高效地执行。
  - 以增加硬件的复杂性为代价

### 4.2.1 动态调度的基本思想

到目前为止我们所使用流水线的最大的局限性：

- 指令必须按序流出和执行
- 考虑下面一段代码：

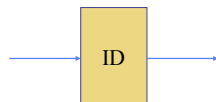
```
DIV.D F4, F0, F2
SUB.D F10, F4, F6
ADD.D F12, F6, F14
```

**SUB.D指令与DIV.D指令有F4相关，导致流水线停顿。**

**ADD.D指令与流水线中的任何指令都没有关系，但也因此受阻。**

#### 4.2.1 动态调度的基本思想

在前面的基本流水线中：



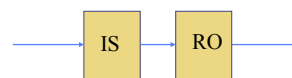
检测结构冲突  
检测数据冲突

一旦一条指令受阻，其后的指令都将停顿。

解决办法：允许乱序执行（Disorder Execution）

1. 为了允许乱序执行，我们将5段流水线的译码阶段再分为两个阶段：

- 流出（Issue, IS）：指令译码，检查是否存在结构冲突。（in-order issue）
- 读操作数（Read Operands, RO）：等待数据冲突消失，然后读操作数。（out of order execution）



检测结构冲突      检测数据冲突

2. 在前述5段流水线中，是不会发生WAR冲突和WAW冲突的。但乱序执行就使得它们可能发生了。

- 例如，考虑下面的代码

```
DIV.D    F10, F0, F2  
SUB.D    F10, F4, F6  
ADD.D    F6, F8, F14
```

存在反相关      } 存在输出相关

Tomasulo算法可以通过使用寄存器重命名来消除。

3. 动态调度的流水线支持多条指令同时处于执行状态

要求：具有多个功能部件、或者流水功能部件、或者兼而有之。

3. 指令乱序完成带来的最大问题：异常处理比较复杂，分精确异常处理和不精确异常处理；

- 不精确异常：当执行指令i导致发生异常时，处理机的现场（状态）与严格按程序顺序执行时指令i的现场不同。
  - 发生不精确异常的原因：
    - 流水线可能已经执行了按顺序位于指令i之后的指令
    - 流水线可能还没完成按顺序位于指令i之前的指令
  - 不精确异常使得在异常处理后难以接着继续执行程序
- 精确异常：如果发生异常时，处理机的现场跟严格按程序顺序执行时指令i的现场相同。

记分牌算法和Tomasulo算法是两种比较典型的动态调度算法。

#### 4.2.2 Tomasulo算法

1. 核心思想

- 记录和检测指令相关，操作数一旦就绪就立即执行，把发生WAR冲突的可能性减少到最小；
  - 通过寄存器换名来消除RAW冲突和WAW冲突。
2. IBM 360/91首先采用了Tomasulo算法。
    - IBM 360/91的设计目标是基于整个360系列的统一指令集和编译器来实现高性能，而不是设计和利用专用的编译器来提高性能。需要更多地依赖于硬件。
    - IBM 360体系结构只有4个双精度浮点寄存器，限制了编译器调度的有效性。
    - 360/91的访存时间和浮点计算时间都很长。（也是Tomasulo算法要解决的问题）

3. 寄存器换名可以消除WAR冲突和WAW冲突。

- 考虑以下代码：

```
DIV.D    F0, F2, F4  
ADD.D    F6, F0, F8  
S.D      F6, 0 (R1)  
SUB.D    F8, F10, F14  
MUL.D    F6, F10, F8
```

反相关 (F8) 导致WAR冲突      } 输出相关 (F6) 导致WAW冲突

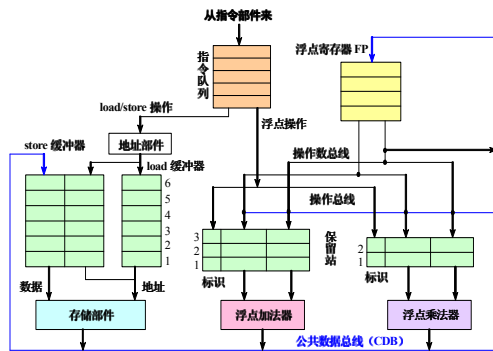
- 消除名相关

- 引入两个临时寄存器S和T
- 把这段代码改写为：

```
DIV.D    F0, F2, F4  
ADD.D    S, F0, F8  
S.D      S, 0 (R1)  
SUB.D    T, F10, F14  
MUL.D    F6, F10, T
```

两个F8都换名为T      } 两个F6都换名为S

#### 4. 基于Tomasulo算法的MIPS处理器浮点部件的基本结构



西安交通大学本科课程：计算机系统结构

### Three Stages of Tomasulo Algorithm

- Issue**—get instruction from FP Op Queue  
If reservation station free (no structural hazard), control issues instruction & sends operands (renames registers).
  - Execution**—operate on operands (EX)  
When both operands ready then execute;  
if not ready, watch Common Data Bus for result
  - Write result**—finish execution (WB)  
Write on Common Data Bus to all awaiting units;  
mark reservation station available
- Normal data bus: data + destination ("go to" bus)
  - Common data bus: data + source ("come from" bus)
    - 64 bits of data + 4 bits of Functional Unit source address
    - Write if matches expected Functional Unit (produces result)
    - Does the broadcast

E-mail: qhzheng@mail.xjtu.edu.cn

郑庆华 教授

➤ **保留站 (reservation station)**：保存一条已经流出并等待到本功能部件执行的指令。包括：操作码、操作数以及用于检测和解决冲突的信息。每个保留站都有一个唯一的标识字段。

- 在一条指令流出到保留站的时候，如果该指令的源操作数已经在寄存器中就绪，则取到该保留站。
- 如果操作数还没有计算出来，则记录将产生这个操作数的保留站的标识。

- 浮点加法器有3个保留站：ADD1, ADD2, ADD3
- 浮点乘法器有两个保留站：MULT1, MULT2

➤ **公共数据总线CDB**：一条重要的数据通路

- 所有功能部件的计算结果都送到CDB，由它直接广播到各个需要该结果的部件。
- 在具有多个执行部件且采用多流出（即每个时钟周期流出多条指令）的流水线中，需要采用多条CDB

➤ **Load缓冲器和Store缓冲器**

- 存放读/写存储器的数据或地址
- Load缓冲器的作用有3个：
  - 存放用于计算有效地址的分量
  - 记录正在进行的load访存，等待存储器的响应
  - 保存已从存储器取出的数据，等待CDB传输

□ store缓冲器的作用有3个：

- 存放用于计算有效地址的分量
- 保存正在进行的store访存的目标地址，该store正在等待存储数据的到达；
- 保存该store的地址和数据，直到存储部件接收

➤ **浮点寄存器FP**

- 共有16个浮点寄存器：F0, F2, F4, ..., F30。
- 它们通过一对总线连接到功能部件，通过CDB连接到store缓冲器。

➤ **指令队列**：按FIFO的顺序流出

➤ **运算部件**

- 浮点加法器：完成加法和减法操作
- 浮点乘法器：完成乘法和除法操作

5. 在Tomasulo算法中，寄存器换名是通过保留站和流出逻辑共同完成的

- 当指令流出时，如果其操作数还没有计算出来，则将该指令中相应的寄存器号换名为将产生这个操作数的保留站的标识。
- 指令流出到保留站后，其操作数寄存器号或者换成了数据本身（如果该数据已经就绪），或者换成了保留站的标识，不再与寄存器有关系。

6. Tomasulo算法的两个特点：

- 冲突检测和指令执行控制是分布的：保留站中的信息决定了什么时候指令可以在该功能部件开始执行
- 计算结果通过CDB直接从产生它的保留站传送到所有需要它的功能部件，而不用经过寄存器。

### 7. 每个保留站有以下6个字段：

- Op: 操作码
- Qj, Qk: 将产生源操作数的保留站号。
  - 等于0表示操作数已经就绪且在Vj或Vk中，或者不需要操作数。
- Vj, Vk: 源操作数的值。
  - 对于每一个操作数来说，V或Q字段只有一个有效。
  - 对于load来说，Vk字段用于保存偏移量。
- Busy: 为“yes”表示本保留站或缓冲单元“忙”。
- A: 仅load和store缓冲器有该字段。开始时存放指令中的立即数字段，地址计算后存放有效地址。
- Qi: 寄存器状态表。
  - 每个寄存器在该表中有对应的一项，用于存放将把结果写入该寄存器的保留站的站号。
  - 为0表示当前没有正在执行的指令要写入该寄存器，也即该寄存器中的内容就绪。

### Tomasulo算法举例

**例4.1** 对于下述指令序列，给出当第一条指令完成并写入结果时，Tomasulo算法所用的各信息表中的内容。

```

L.D  F6,34 (R2)
L.D  F2,45 (R3)
MUL.D  F0,F2,F4
SUB.D F8,F2,F6
DIV.D F10,F0,F6
ADD.D  F6,F8,F2
    
```

当采用Tomasulo算法时，在上述给定的时刻，保留站、load缓冲器以及寄存器状态表中的内容。

指令	指令状态表		
	流出	执行	写结果
L.D F6, 34(R2)	✓	✓	✓
L.D F2, 45(R3)	✓	✓	
MUL.D F0, F2, F4	✓		
SUB.D F8, F6, F2	✓		
DIV.D F10, F0, F6	✓		
ADD.D F6, F8, F2	✓		

名称	保留站						
	Busy	Op	Vj	Vk	Qj	Qk	A
Load1	no						
Load2	yes	LD					45+Regs[R3]
Add1	yes	SUB	Mem[34+Regs[R2]]		Load2		
Add2	yes	ADD			Add1	Load2	
Add3	no						
Mult1	yes	MUL	Reg[F4]		Load2		
Mult2	yes	DIV	Mem[34+Regs[R2]]		Mult1		

	寄存器状态表							
	F0	F2	F4	F6	F8	F10	...	F30
Qi	Mult1	Load2		Add2	Add1	Mult2	...	

Tomasulo算法具有两个主要的优点：

- 冲突检测逻辑是分布的（通过保留站和CDB实现）
  - 如果有多条指令已经获得了一个操作数，并同时在等待同一运算结果，那么这个结果一产生，就可以通过CDB同时播送给所有这些指令，使它们可以同时执行。
- 消除了WAW冲突和WAR冲突导致的停顿
  - 使用保留站进行寄存器换名，并且操作数一旦就绪就将之放入保留站。

**例4.2** 对于例4.1中的代码，假设各种操作的延迟为：

load: 1个时钟周期

加法: 2个时钟周期

乘法: 10个时钟周期

除法: 40个时钟周期

给出MUL.D指令准备写结果时各状态表的内容。

**【解】**

MUL.D指令准备写结果时各状态表的内容如下图所示。

指令		指令状态表		
		流出	执行	写结果
L.D	F6, 34(R2)	✓	✓	✓
L.D	F2, 45(R3)	✓	✓	✓
MUL.D	F0, F2, F4	✓	✓	
SUB.D	F8, F6, F2	✓	✓	✓
DIV.D	F10, F0, F6	✓		
ADD.D	F6, F8, F2	✓	✓	✓

名称	保留站						
	Busy	Op	Vj	Vk	Qj	Qk	A
Load1	no						
Load2	yes						
Add1	yes						
Add2	yes						
Add3	no						
Mult1	yes	Mul	Mem[45+Regs[R3]]	Reg[F4]			
Mult2	yes	DIV		Mem[34+Regs[R2]]	Mult1		

	寄存器状态表								
	F0	F2	F4	F6	F8	F10	...	F30	
Qi	Mult1			Mult2	...				

### (三) Tomasulo算法的具体实现

#### 各符号的意义

- **r**: 分配给当前指令的保留站或者缓冲器单元编号;
- **rd**: 目标寄存器编号;
- **rs, rt**: 操作数寄存器编号;
- **imm**: 符号扩展后的立即数;
- **RS**: 保留站;
- **result**: 浮点部件或load缓冲器返回的结果;
- **Qi**: 寄存器状态表;
- **Regs[]**: 寄存器组;

- 与rs对应的保留站字段: **Vj, Qj**
- 与rt对应的保留站字段: **Vk, Qk**
- **Qi, Qj, Qk**的内容或者为0, 或者是一个大于0的整数。
  - ▣ **Qi为0**表示相应寄存器中的数据就绪。
  - ▣ **Qj, Qk为0**表示保留站或缓冲器单元中的**Vj**或**Vk**字段中的数据就绪。
  - ▣ 当它们为**正整数**时, 表示相应的寄存器、保留站或缓冲器单元正在等待结果。

符号说明: (举例)

MUL.D F4, F0, F2    L.D F2, 45 (R3)

↑    ↑    ↑       ↑    ↑    ↑

**rd rs rt**    **rt im rs**

**i j k**    **k m j**

S.D F3, 40 (R4)

↑    ↑    ↑

**rt im rs**

**k m j**

#### 1. 指令流出

- 浮点运算指令

进入条件: 有空闲保留站 (设为r)

操作和状态表内容修改:

```
if (Qi[rs] = 0) // 检测第一操作数是否就绪
{ RS[r].Vj ← Regs[rs]; // 第一操作数就绪。把寄存器rs
                        // 中的操作数取到当前保留站的Vj。
  RS[r].Qj ← 0; // 置Qj为0, 表示当前保留站的Vj
                // 中的操作数就绪。
}
else // 第一操作数没有就绪
{ RS[r].Qj ← Qi[rs] } // 进行寄存器换名, 即把将产生该
                      // 操作数的保留站的编号放入当前保留站的Qj。
```

MUL.D F4, F0, F2

↑    ↑    ↑

**rd rs rt**

**j k**

```

if (Qi[rt] = 0)           // 检测第二操作数是否就绪
{ RS[r].Vk ← Regs[rt];    // 第二操作数就绪。把寄存器rt中的
                          // 操作数取到当前保留站的Vk。
  RS[r].Qk ← 0 }          // 置Qk为0，表示当前保留站的Vk中的
                          // 操作数就绪。
else                       // 第二操作数没有就绪
{ RS[r].Qk ← Qi[rt] }      // 进行寄存器换名，即把将产生该操作
                          // 数的保留站编号放入当前保留站Qk。
RS[r].Busy ← yes;         // 置当前保留站为“忙”
RS[r].Op ← Op;             // 设置操作码
Qi[rd] ← r;               // 把当前保留站的编号r放入rd所对应
                          // 的寄存器状态表项，以便rd将来接收结果。

```

L.D F2, 45 (R3)

$\begin{matrix} \uparrow & \uparrow & \uparrow \\ \text{rt} & \text{im} & \text{rs} \\ \text{k} & \text{m} & \text{j} \end{matrix}$

> **load和store指令**

进入条件: 缓冲器有空闲单元 (设为r)

操作和状态表内容修改:

```

if (Qi[rs] = 0)           // 检测第一操作数是否就绪
{ RS[r].Vj ← Regs[rs];    // 第一操作数就绪，把寄存器rs中的
                          // 操作数取到当前缓冲器单元的Vj
  RS[r].Qj ← 0 }          // 置Qj为0，表示当前缓冲器单元的Vj
                          // 中的操作数就绪。
else // 第一操作数没有就绪
{ RS[r].Qj ← Qi[rs] }      // 进行寄存器换名，即把将产生该
                          // 操作数的保留站的编号存入当前缓冲器单元的Qj。

```

L.D F2, 45 (R3)

$\begin{matrix} \uparrow & \uparrow & \uparrow \\ \text{rt} & \text{im} & \text{rs} \\ \text{k} & \text{m} & \text{j} \end{matrix}$

```

RS[r].Busy ← yes;         // 置当前缓冲器单元为“忙”
RS[r].A ← Imm;            // 把符号位扩展后的偏移量放入
                          // 当前缓冲器单元的A
对于load指令:
Qi[rt] ← r;               // 把当前缓冲器单元的编号r放入
                          // load指令的目标寄存器rt所对应的寄存器
                          // 状态表项，以便rt将来接收所取的数据。

```

S.D F3, 40 (R4)

$\begin{matrix} \uparrow & \uparrow & \uparrow \\ \text{rt} & \text{im} & \text{rs} \\ \text{k} & \text{m} & \text{j} \end{matrix}$

对于Store指令:

```

if (Qi[rt] = 0)           // 检测要存储的数据是否就绪
{ RS[r].Vk ← Regs[rt];    // 该数据就绪，把它从寄存器rt取到
                          // store缓冲器单元的Vk
  RS[r].Qk ← 0 }          // 置Qk为0，表示当前缓冲器单元的Vk
                          // 中的数据就绪。
else                       // 该数据没有就绪
{ RS[r].Qk ← Qi[rt] }      // 进行寄存器换名，即把将产生该数
                          // 据的保留站的编号放入当前缓冲器单元的Qk。

```

## 2. 执行

- > 浮点操作指令
  - 进入条件: (RS[r].Qj = 0) 且 (RS[r].Qk = 0);  
// 两个源操作数就绪
  - 操作和状态表内容修改: 进行计算，产生结果。
- > load/store指令，关键是计算有效地址
  - 进入条件: (RS[r].Qj = 0) 且 r 成为 load/store  
缓冲队列的头部 // r 为保留站编号
  - 操作和状态表内容修改:
 

```
RS[r].A ← RS[r].Vj + RS[r].A; // 计算有效地址
```

对于load指令，在完成有效地址计算后，还要进行:  
从Mem[RS[r].A]读取数据; // 从存储器中读取数据

## 3. 写结果

- > 浮点运算指令和load指令
- 进入条件: 保留站r执行结束，且CDB就绪。
- 操作和状态表内容修改:
- ```

∀x (if (Qi[x] = r)        // Case1: 对于任何一个正在等待该结果
                          // 的浮点寄存器x
  { Regs[x] ← result;     // 向该寄存器写入结果
    Qi[x] ← 0 }           // 把该寄存器的状态置为数据就绪
  ∀x (if (RS[x].Qj = r)   // Case2: 对于任何一个正在等待该结果
                          // 作为第一操作数的保留站x
  { RS[x].Vj ← result;    // 向该保留站的Vj写入结果
    RS[x].Qj ← 0 }        // 置Qj为0，表示该保留站的
                          // Vj中的操作数就绪

```



$\forall x$  (if (RS[x].Qk = r) // Case3: 对于任何一个正在等待该结果作为  
 // 第二操作数的保留站x  
 {RS[x].Vk  $\leftarrow$  result; // 向该保留站的Vk写入结果  
 RS[x].Qk  $\leftarrow$  0}; // 置Qk为0, 表示该保留站的Vk中的  
 // 操作数就绪。  
 RS[r].Busy  $\leftarrow$  no; // 释放当前保留站, 将之置为空闲状态。  
 > store指令  
 进入条件: 保留站r执行结束, 且RS[r].Qk = 0  
 // 要存储的数据已经就绪  
 操作和状态表内容修改:  
 Mem[RS[r].A]  $\leftarrow$  RS[r].Vk // 数据写入存储器, 地址由store  
 // 缓冲器单元的A字段给出。  
 RS[r].Busy  $\leftarrow$  no; // 释放当前缓冲器单元, 将之置为空闲状态。

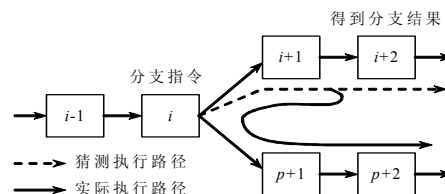
## 本章内容提纲

- 4.1 指令级并行的概念
- 4.2 指令的动态调度
- 4.3 动态分支预测技术
- 4.4 多指令流出技术
- 4.5 循环展开和指令调度
- 4.6 Pentium计算机原理

1. 采用动态分支预测技术的目的
  - > 预测分支是否成功
  - > 尽快找到分支目标地址（或指令），以避免控制相关造成流水线停顿
2. 需要解决的关键问题
  - > 如何记录分支的历史信息；
  - > 如何根据这些信息来预测分支的去向（甚至取到指令）。
3. 动态分支预测：在程序运行时，根据分支指令过去的表现预测其将来的行为。

### 4. 分支预测的有效性取决于：

- > 预测的准确性
- > 在预测错误时，作废已预取和分析的指令，恢复现场，并从另一条分支路径重新取指令。决定分支开销的3个因素：
  - 流水线的结构
  - 预测的方法
  - 预测错误时的恢复策略等

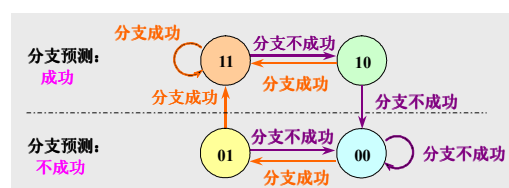


### 4.3.1 采用分支历史表 BHT

1. 分支历史表BHT（Branch History Table）或分支预测缓冲器（Branch Prediction Buffer）
  - > 最简单的动态分支预测方法
  - > 用BHT来记录分支指令最近一次或几次的执行情况（成功或不成功），并据此进行预测。
2. 只有1个预测位的分支预测缓冲：只记录分支指令最近一次的历史，BHT中只需要1位二进制位。

### 3. 采用两位二进制位来记录历史

- 提高预测的准确度
- 研究表明：两位分支预测的性能与n位（n>2）分支预测的性能差不多。
- > 两位分支预测的状态转换如下所示：





➤ 两位分支预测中的操作有两个步骤：

□ Step1 分支预测

- 当分支指令到达译码段（ID）时，根据从BHT读出的信息进行分支预测。
- 若预测正确，就继续处理后续的指令，流水线没有断流。否则，就要作废已经预取和分析的指令，恢复现场，并从另一条分支路径重新取指令。

□ Step2 状态修改

4. BHT方法只在以下情况下才有意义：判定分支是否成功所需的时间大于确定分支目标地址所需的时间。
5. 研究结果表明：对于SPEC89测试程序来说，具有大小为4K的BHT的预测准确率为82%~99%。一般来说，采用4K的BHT就可以了。

### 4.3.2 采用分支目标缓冲器BTB

目标：将分支的开销降为 0

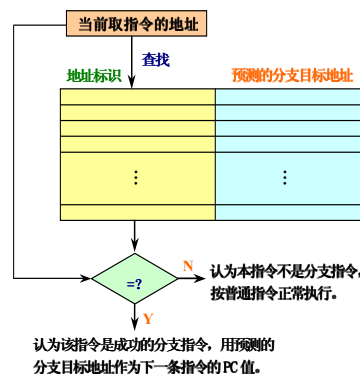
方法：分支目标缓冲

- 将分支成功的分支指令的地址和它的分支目标地址都放到一个缓冲区中保存起来，缓冲区以分支指令的地址作为标识。
- 这个缓冲区就是分支目标缓冲器（Branch-Target Buffer，简记为BTB，或者Branch-Target Cache）。

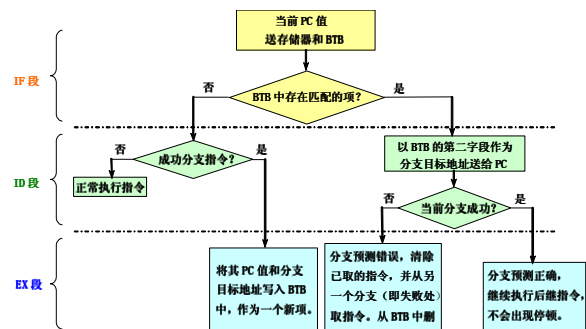
#### 1. BTB的结构

- 看成是用专门的硬件实现的一张表。
- 表格中的每一项至少有两个字段：

- 执行过的成功分支指令的地址：该表的匹配标识
- 预测的分支目标地址。



#### 2. 采用BTB后，在流水线各个阶段所进行的相关操作：



### 3. 采用BTB后，各种可能情况下的延迟：

| 指令在BTB中? | 预测 | 实际情况 | 延迟周期 |
|----------|----|------|------|
| 是        | 成功 | 成功   | 0    |
| 是        | 成功 | 不成功  | 2    |
| 不是       |    | 成功   | 2    |
| 不是       |    | 不成功  | 0    |

#### 4. BTB的另一种形式：在分支目标缓冲器中存放一条或者多条分支目标处的指令。有以下潜在的好处：

- 更快地获得分支目标处的指令
- 可以一次提供分支目标处的多条指令，这对于多流出处理器是很有必要的：（如Case语句）

### 4.3.3 基于硬件的前瞻执行（Speculation）

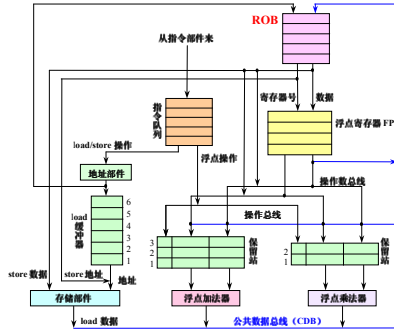
**基本思想：**对分支指令的结果进行猜测，并假设这个猜测总是对的，然后按这个猜测结果继续取、流出和执行后续的指令。只是执行指令的结果不是写回到寄存器或存储器，而是放到一个称为ROB（ReOrder Buffer）的缓冲器中。等到相应的指令得到“确认”（commit，即确实是应该执行的）之后，才将结果写入寄存器或存储器。

#### 1. 基于硬件的前瞻执行结合了三种思想：

- 动态分支预测：用来选择后续执行的指令
- 乱序执行：前瞻地执行后续指令
- 动态调度：对基本块的各种组合进行跨基本块的调度。

## 2. 对Tomasulo算法加以扩充，就可以支持前瞻执行。

把Tomasulo算法的写结果和指令完成加以区分，分成两个不同的段：  
写结果，指令确认。



### ➤ 写结果段

- ❑ 把前瞻执行的结果写到**ROB**中；
- ❑ 通过**CDB**在指令之间传送结果，供需要用到这些结果的指令使用。
- **指令确认段**：在分支指令的结果出来后，对相应指令的前瞻执行给予确认。
  - ❑ 如果前面所做的猜测是对的，把在**ROB**中的结果写到寄存器或存储器。
  - ❑ 如果发现前面对分支结果的猜测是错误的，那就不予以确认，并从那条分支指令的另一条路径开始重新执行。

## 3. 前瞻的关键思想：允许指令乱序执行，但必须顺序确认。

### 4. 支持前瞻执行的浮点部件的结构

#### ➤ **ROB**中的每一项由以下4个字段组成：

- ❑ **指令类型**：指出该指令是分支指令、**store**指令或寄存器操作指令
- ❑ **目标地址**：给出指令执行结果应写入的目标寄存器号（如果是**load**和**ALU**指令）或存储器单元的地址（如果是**store**指令）。
- ❑ **数值字段**：用来保存指令前瞻执行的结果，直到指令得到确认
- ❑ **就绪字段**：指出指令是否已经完成执行并且数据已就绪

#### ➤ Tomasulo算法中保留站的换名功能是由**ROB**来完成的。

## 5. 采用前瞻执行机制后，在原Tomasulo算法的基础上改造为以下指令执行步骤：

### ➤ Step1：流出

- ❑ 从浮点指令队列的头部取一条指令。
- ❑ 如果有空闲的保留站（设为**r**）且有空闲的**ROB**项（设为**b**），就流出该指令，并把相应的信息放入保留站**r**和**ROB**项**b**。
- ❑ 如果保留站或**ROB**全满，便停止流出指令，直到它们都有空闲的项。

### ➤ Step2：执行

- ❑ 如果有操作数尚未就绪，就等待，并不断地监测**CDB**，检测**RAW**冲突
- ❑ 当两个操作数都已在保留站中就绪后，就可以执行该指令的操作。

### ➤ Step3：写结果

- ❑ 当结果产生后，将该结果连同本指令在流出段所分配到的**ROB**项的编号放到**CDB**上，经**CDB**写到**ROB**以及所有等待该结果的保留站
- ❑ 释放产生该结果的保留站。
- ❑ **store**指令在本阶段完成，其操作为：
  - 如果要写入存储器的数据已经就绪，就把该数据写入分配给该**store**指令的**ROB**项。
  - 否则，就监测**CDB**，直到那个数据在**CDB**上播送出来，这时才将之写入分配给该**store**指令的**ROB**项。

### ➤ Step4：确认，对分支指令、**store**指令以及其他指令的处理不同：

- ❑ **其他指令**：当该指令到达**ROB**队列的头部而且其结果已经就绪时，就把该结果写入该指令的目标寄存器，并从**ROB**中删除该指令。
- ❑ **store**指令  
处理与上面类似，只是它把结果写入存储器。
- ❑ **分支指令**
  - 当预测错误的分支指令到达**ROB**队列的头部时，清空**ROB**，并从分支指令的另一个分支重新开始执行。（**错误的前瞻执行**）
  - 当预测正确的分支指令到达**ROB**队列的头部时，该指令执行完毕。

例4.3 假设浮点功能部件的延迟时间为：加法2个时钟周期，乘法10个时钟周期，除法40个时钟周期。对于下面的代码段，给出当指令MUL.D即将确认时的状态表内容。

L.D F6, 34 (R2)  
L.D F2, 45 (R3)  
MUL.D F0, F2, F4  
SUB.D F8, F6, F2  
DIV.D F10, F0, F6  
ADD.D F6, F8, F2

前瞻执行中MUL.D确认前，保留站和ROB的状态

| 名称    | 保留站  |     |                  |                  |    |    |      |   |
|-------|------|-----|------------------|------------------|----|----|------|---|
|       | Busy | Op  | Vj               | Vk               | Qj | Qk | Dest | A |
| Add1  | no   |     |                  |                  |    |    |      |   |
| Add2  | no   |     |                  |                  |    |    |      |   |
| Add3  | no   |     |                  |                  |    |    |      |   |
| Mult1 | no   | MUL | Mem[45+Regs[R2]] | Regs[F4]         |    |    | #3   |   |
| Mult2 | yes  | DIV |                  | Mem[34+Regs[R2]] | #3 |    | #5   |   |

| 项号 | ROB  |                   |     |     |                  |  |
|----|------|-------------------|-----|-----|------------------|--|
|    | Busy | 指令                | 状态  | 目的  | Value            |  |
| 1  | no   | L.D F6, 34 (R2)   | 确认  | F6  | Mem[34+Regs[R2]] |  |
| 2  | no   | L.D F2, 45 (R3)   | 确认  | F2  | Mem[45+Regs[R3]] |  |
| 3  | yes  | MUL.D F0, F2, F4  | 写结果 | F0  | #2×Regs[F4]      |  |
| 4  | yes  | SUB.D F8, F6, F2  | 写结果 | F8  | #1—#2            |  |
| 5  | yes  | DIV.D F10, F0, F6 | 执行  | F10 |                  |  |
| 6  | yes  | ADD.D F6, F8, F2  | 写结果 | F6  | #4+#2            |  |

| 字段     | 浮点寄存器状态 |    |    |     |     |     |     |     |
|--------|---------|----|----|-----|-----|-----|-----|-----|
|        | F0      | F2 | F4 | F6  | F8  | F10 | ... | F30 |
| ROB项编号 | 3       |    |    | 6   | 4   | 5   |     |     |
| Busy   | yes     | no | no | yes | yes | yes | ... | no  |

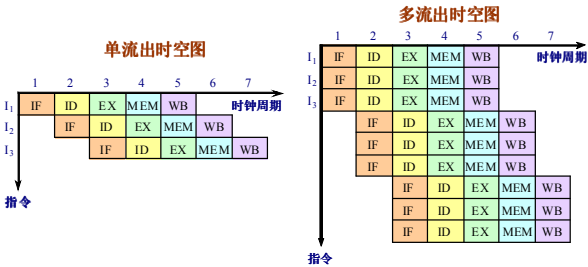
6. 前瞻执行的特点

- 通过ROB实现了指令的乱序执行，顺序完成。
- 能够实现精确异常。
- 很容易地推广到整数寄存器和整数功能单元上。
- 主要缺点：所需的硬件太复杂。

本章内容提纲

- 4.1 指令级并行的概念
- 4.2 指令的动态调度
- 4.3 动态分支预测技术
- 4.4 多指令流出技术
- 4.5 循环展开和指令调度
- 4.6 Pentium计算机原理

单流出和多流出处理机执行指令的时空图



西安交通大学本科生课程：计算机系统结构

## 超标量处理机与超流水线处理机

三种高性能的指令级并行处理机：

- 1、超标量处理机(Super Scalar Processor)
- 2、超流水线处理机 (Super Pipelining Processor)
- 3、超标量流水线处理机 (Super Pipelining Super Scalar Processor)

E-mail: qhzheng@mail.xjtu.edu.cn 郑庆华 教授

西安交通大学本科生课程：计算机系统结构

## 四种不同处理机性能比较

| 机器类型      | K段流水线<br>基准标量处理机 | m度<br>超标量处理机 | n度<br>超流水线处理机 | (m,n)度<br>超标量超流水线处理机 |
|-----------|------------------|--------------|---------------|----------------------|
| 流水线周期     | 1个               | 1            | 1/n           | 1/n                  |
| 同时发射指令条数  | 1条               | m            | 1             | m                    |
| 指令发射等待时间  | 1个               | 1            | 1/n           | 1/n                  |
| 指令并行度 ILP | 1                | m            | n             | m * n                |

E-mail: qhzheng@mail.xjtu.edu.cn 郑庆华 教授

西安交通大学本科生课程：计算机系统结构

## 4.4.1 超标量处理机

典型特征：有多个操作部件，一个或几个比较大的通用寄存器堆。

典型处理机：Intel 的Pentium，Motolara的MC88110，IBM 的Power PC，SUN的Super Sparc，以及Opteron 的AMD等。

E-mail: qhzheng@mail.xjtu.edu.cn 郑庆华 教授

西安交通大学本科生课程：计算机系统结构

## 4.4.2 指令多发射技术

### 1、指令多发射技术的基本概念

- 迄今为止介绍的各类提高性能的技术都是围绕使CPI=1这一目标展开的。
  - 如：流水线中消除数据相关、控制相关、静态调度、动态调度等
- 根据公式 $CPUtime = IC \times CPI \times cycle\ time$ ，进一步提高性能的启发是使CPI < 1

E-mail: qhzheng@mail.xjtu.edu.cn 郑庆华 教授

西安交通大学本科生课程：计算机系统结构

## 多发射技术基本概念

- 在传统每一周期发射一条指令的系统中，是无法实现CPI<1的。也就是说，要达到CPI<1，必须要求实现在一个时钟周期里发射多条指令，即指令的多发射技术。
- 多发射技术的两种方法：
  - superscalar(超标量)方法
  - VLIW(超长指令字)方法
- 实现指令多发射技术的前提：
  - 有足够硬件，即功能单元、寄存器、及存储器带宽的基础上。也就是说不存在结构竞争。

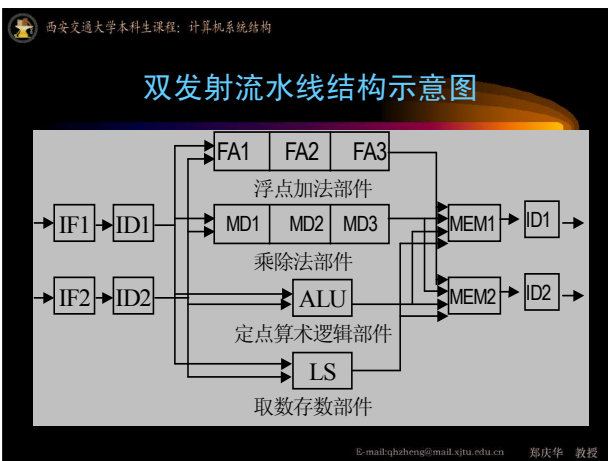
E-mail: qhzheng@mail.xjtu.edu.cn 郑庆华 教授

西安交通大学本科生课程：计算机系统结构

## 4.4.3 Superscalar的基本概念

- 在一个周期里能发射可变数量的指令，通常为1~8条指令/cycle，设这个上限为n，就称该处理机为n流出；
- 同时发射的指令按一定规律搭配，即有一定限制，不能自由搭配；
- 用静态调度（compiler完成）和/或动态调度（硬件完成）方法确定可同时发射的指令条数。

E-mail: qhzheng@mail.xjtu.edu.cn 郑庆华 教授



西安交通大学本科生课程：计算机系统结构

## 双发射处理器的流水时序

| 指令类型 | Pipe stages |    |    |     |     |     |     |    |
|------|-------------|----|----|-----|-----|-----|-----|----|
| 整数指令 | IF          | ID | EX | MEM | WB  |     |     |    |
| 浮点指令 | IF          | ID | EX | MEM | WB  |     |     |    |
| 整数指令 |             | IF | ID | EX  | MEM | WB  |     |    |
| 浮点指令 |             | IF | ID | EX  | MEM | WB  |     |    |
| 整数指令 |             |    | IF | ID  | EX  | MEM | WB  |    |
| 浮点指令 |             |    | IF | ID  | EX  | MEM | WB  |    |
| 整数指令 |             |    |    | IF  | ID  | EX  | MEM | WB |
| 浮点指令 |             |    |    | IF  | ID  | EX  | MEM | WB |

E-mail: qhzheng@mail.xjtu.edu.cn

郑庆华 教授

西安交通大学本科生课程：计算机系统结构

### 多发射处理器的局限性

既然可以在一个时钟周期内发射5条指令，那么为什么不同时发射50条指令呢？

- 多发射方法存在困难：
  - 程序中原有ILP（Instruction Level Parallel）有限
  - 多发射处理器硬件复杂性高，成本高

E-mail: qhzheng@mail.xjtu.edu.cn 郑庆华 教授

西安交通大学本科生课程：计算机系统结构

### 4.4.4 VLIW的基本概念

- 在一个时钟周期里发射固定数量的指令，实际为一条长指令，或固定的指令包；
- VLIW也是按固定格式组织的；
- VLIW是由Compiler组织的，即不能由硬件动态组织。

E-mail: qhzheng@mail.xjtu.edu.cn 郑庆华 教授

西安交通大学本科生课程：计算机系统结构

### 超长指令字计算机特点

以一条长指令实现多个操作的并行执行，减少存储器访问。主要特点：

- 单一的控制流：只有一个控制器，每个周期启动一条指令。
- 超长指令字被分成多个控制字段，每个字段直接独立地控制每个功能部件。
- 在编译阶段完成超长指令中多个可并行执行操作的调度。

E-mail: qhzheng@mail.xjtu.edu.cn 郑庆华 教授

西安交通大学本科生课程：计算机系统结构

### VLIW究竟该有多长？

- 以一个拥有7个功能单元的VLIW处理器为例，指令长度有多少位？
- 设7个功能单元可支持2个整数操作，2个FP操作，2个memory访问操作和1个Br，则实际上一条VLIW含7条指令。
- 为支持每一功能单元正常工作，需分配每一个功能单元相应的数据域。一般每一数据域为16-24位。
- VLIW长度为： $16 \times 7 = 112 \text{ bits}$   
或： $24 \times 7 = 168 \text{ bits}$

E-mail: qhzheng@mail.xjtu.edu.cn 郑庆华 教授

### ➤ VLIW存在的一些问题

- 程序代码长度增加了
  - ❑ 提高并行性而进行的大量的循环展开。
  - ❑ 指令字中的操作槽并非总能填满。
- 采用了锁步机制
 

任何一个操作部件出现停顿时，整个处理机都要停顿。
- 机器代码的不兼容性

### 4.4.5 理想情况下的超标量处理机的性能

■ 单流水线性能： $T(1,1) = (k + N - 1)\Delta t$

■ m度超标量处理机性能：

$$T(m,1) = \left(k + \frac{N-m}{m}\right)\Delta t$$

KΔt为开始m条指令的执行时间

■ m度超标量处理机对单流水线的加速比：

$$S(m,1) = \frac{T(1,1)}{T(m,1)} = \frac{m(k + N - 1)}{N + m(k - 1)}$$

## 4.5 超流水线处理机

- 指令执行时序
- 典型处理机结构
- 超流水线处理机性能

### • 两种定义：

定义1：一个周期内能够分时发射多条指令的处理机称为超流水线处理机。

定义2：指令流水线有8个或更多功能段的流水线处理机称为超流水线处理机。

### • 两种不同并行性：

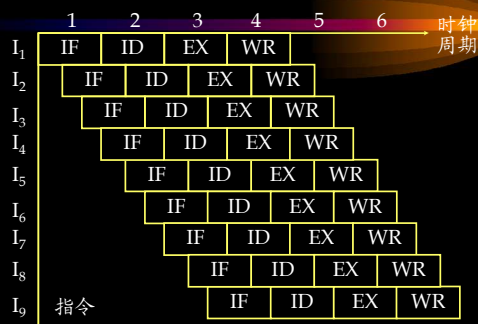
超标量处理机采用的是空间并行性

超流水线处理机采用的是时间并行性

### ➤ 指令执行时序

- 每隔1/n个时钟周期发射一条指令，流水线周期为1/n个时钟周期
- 在超标量处理机中，流水线的有些功能段还可以进一步细分，例如：ID功能段可以再细分为译码、读第一操作数和读第二操作数三个流水段。也有些功能段不能再细分，如WR功能段一般不再细分。

### 每个时钟周期分时发送3条指令的超流水线

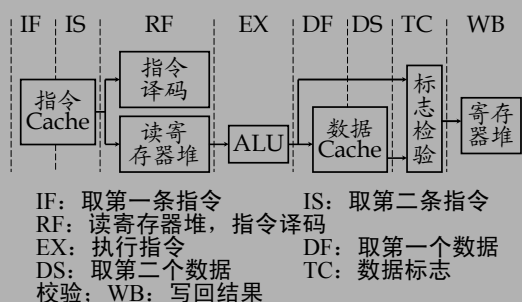


### ➤ 典型处理机结构

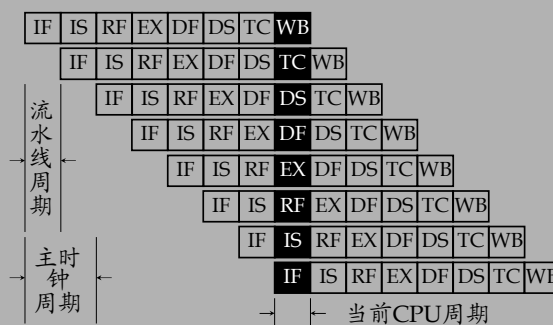
- MIPS R4000处理机每个时钟周期包含两个流水段，是一种很标准的超流水线处理机结构。指令流水线有8个流水段
- 有两个Cache，指令Cache和数据Cache，容量各8KB，每个时钟周期可以访问Cache两次。
- 在一个时钟周期内可以从指令Cache中读出两条指令，从数据Cache中读出或写入两个数据。
- 主要运算部件有整数部件和浮点部件



## MIPS R4000处理机的流水线操作



## MIPS R4000正常指令流水线工作时序



### 超流水线处理机性能

- 指令级并行度为(1, n)的超流水线处理机, 执行N条指令所需要的时间为:

$$T(1, n) = (k + \frac{N-1}{n})\Delta t$$

- 超流水线处理机相对于单流水线普通标量处理机的加速比为:

$$S(1, n) = \frac{T(1, 1)}{T(1, n)} = \frac{(k + N - 1)\Delta t}{(k + \frac{N-1}{n})\Delta t}$$

- 超流水线处理机的加速比的最大值为:  $S(1, n)_{MAX} = n$

## 超标量超流水线处理机

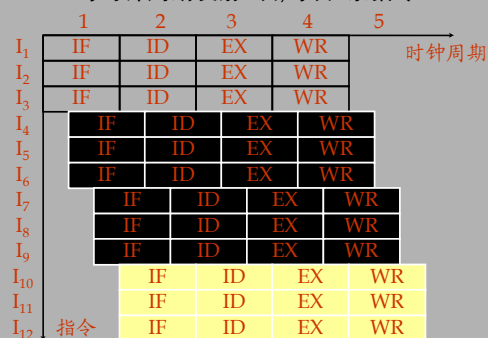
- 指令执行时序
- 典型处理机结构
- 超标量流水线处理机性能

把超标量与超流水线技术结合在一起, 就成为超标量超流水线处理机

### 指令执行时序

超标量超流水线处理机在一个时钟周期内分时发射指令n次, 每次同时发射指令m条, 每个时钟周期总共发射指令  $m \times n$  条。

### 每时钟周期发射3次, 每次3条指令





西安交通大学本科生课程：计算机系统结构

### 超标量超流水线处理机性能

- 指令级并行度为(m,n)的超标量超流水线处理机，连续执行N条指令所需要的时间为：

$$T(m,n) = (k + \frac{N-m}{m \cdot n}) \Delta t$$

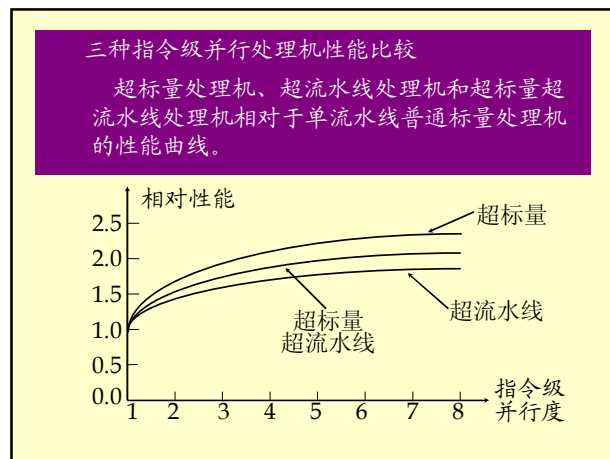
- 超标量超流水线处理机相对于单流水线标量处理机的加速比为：

$$S(m,n) = \frac{S(1,1)}{S(m,n)} = \frac{(k+N-1)\Delta t}{(k + \frac{N-m}{m \cdot n})\Delta t}$$

$$S(m,n) = \frac{m \cdot n \cdot (k+N-1)}{m \cdot n \cdot k + N - m}$$

- 在理想情况下，超标量超流水线处理机加速比的最大值为： $S(m,n)_{MAX} = m \cdot n$

E-mail: qhzheng@mail.xjtu.edu.cn 郑庆华 教授



西安交通大学本科生课程：计算机系统结构

### 得出结论：

#### 1、三种处理机的性能关系

- ✓ 超流水线处理机的启动延迟比超标量处理机大
- ✓ 条件转移造成的损失，超流水线处理机要比超标量处理机大；
- ✓ 超标量处理机指令执行部件的冲突要比超流水线处理机小。

#### 2、最大指令级并行度

一个特定程序由于受到本身的数据相关和控制相关的限制，它的指令级并行度的最大值是一个确定的值，主要由程序自身的语义决定，与这个程序运行在哪一种处理机上无关。

E-mail: qhzheng@mail.xjtu.edu.cn 郑庆华 教授

### 本章内容提纲

- 4.1 指令级并行的概念
- 4.2 指令的动态调度
- 4.3 动态分支预测技术
- 4.4 多指令流出技术
- 4.5 循环展开和指令调度
- 4.6 Pentium计算机原理

#### 4.5.1 循环展开和指令调度的基本方法

- 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。
- 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。

- 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。
- 本节中，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         |

假设采用第3章的5段整数流水线(P70):

- 分支的延迟: 1个时钟周期。
- 整数load指令的延迟: 1个时钟周期。
- 整数运算部件是全流水或者重复设置了足够的份数。

**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R1的初值是指向第一个元素, 8 (R2) 指向最后一个元素。

```
Loop: L.D      F0,0 (R1) //取一个向量元数放入F0
      ADD.D    F4,F0,F2 //加上在F2中的标量
      S.D      F4, 0 (R1) //存结果
      DADDIU   R1,R1, #-8 //指针减去8
      BNE     R1,R2,Loop // 继续
```

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)
- 浮点寄存器F2: 用于保存常数s。

- 不进行指令调度的情况下, 程序的执行情况:

|                     | 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      |
| DADDIU R1,R1, #-8   | 7      |
| (空转)                | 8      |
| BNE R1,R2,Loop      | 9      |
| (空转)                | 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。

- 指令调度以后, 程序的执行情况如下:

- 把DADDIU指令调度到了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

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

|                    | 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      |

一个元素的操作时间从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。

- 例子中的问题及解决方案

只有L.D、ADD.D和S.D这三条指令是有效操作, 占用3个时钟周期。而DADDIU、空转和BNE这三个时钟周期都是附加的循环控制开销。

- **循环展开技术:** 把循环体代码复制多次并按顺序排列, 调整循环结束条件, 为指令调度带来更大的空间

**例4.7 (体现循环展开技术的特点)**

将上述例子中的循环展开3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。

**【解】:** 无需在循环体后面增加补偿代码

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
- 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

- 展开后没有调度的代码如下：

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

- 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  |
|        | L.D    | F6,-8(R1)   | 2  |
|        | L.D    | F10,-16(R1) | 3  |
|        | L.D    | F14,-24(R1) | 4  |
|        | ADD.D  | F4,F0,F2    | 5  |
|        | ADD.D  | F8,F6,F2    | 6  |
|        | ADD.D  | F12,F10,F2  | 7  |
|        | ADD.D  | F16,F14,F2  | 8  |
|        | S.D    | F4,0(R1)    | 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问题：

- **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
- **删除多余的测试指令和分支指令**，并对循环结束代码和新的循环体代码进行相应的修正。

## 本章内容提纲

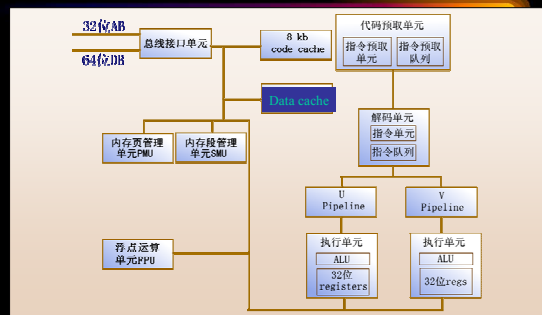
- 4.1 指令级并行的概念
- 4.2 指令的动态调度
- 4.3 动态分支预测技术
- 4.4 多指令流出技术
- 4.5 循环展开和指令调度
- 4.6 Pentium计算机原理



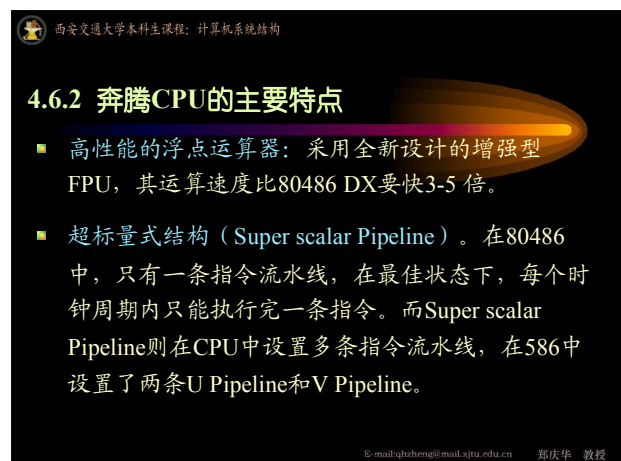
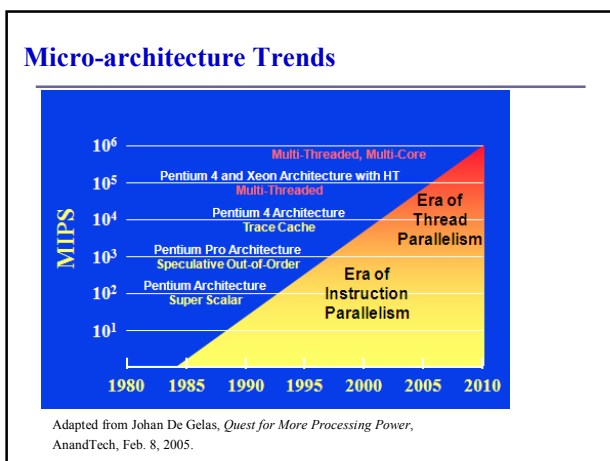
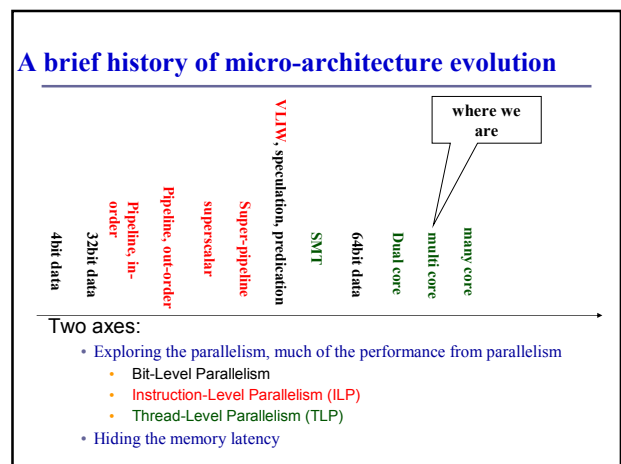
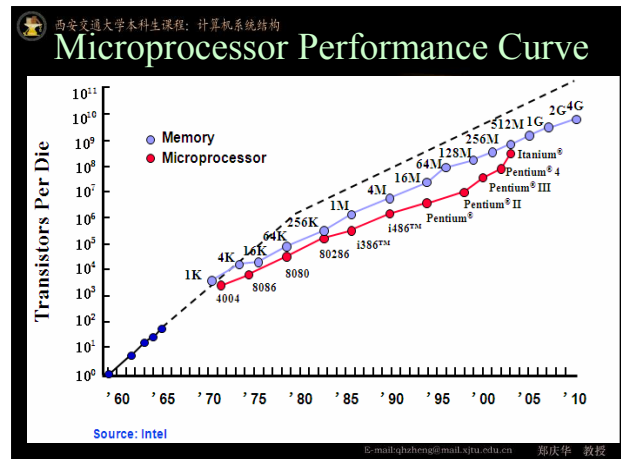
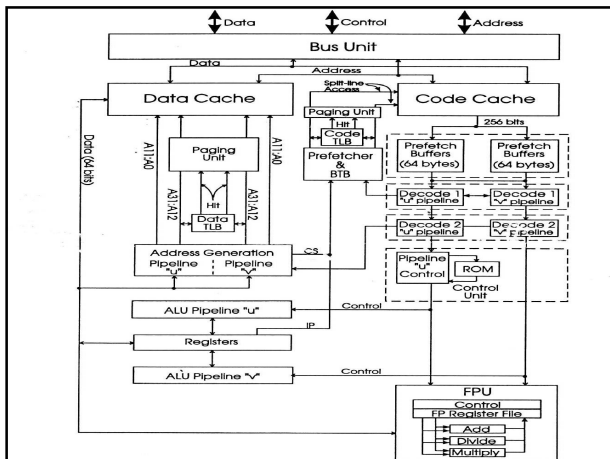
西安交通大学本科生课程：计算机系统结构

### 4.6.1 Pentium 微处理器的系统结构

- 1993年由Intel公司推出，是一种64位的微处理器。



E-mail: qhzheng@mail.xjtu.edu.cn 郑庆华 教授



西安交通大学本科生课程：计算机系统结构

- 双重分离式高速缓存 (Dual on-Board Caches): 将高速缓存分为指令Cache和Data Cache, 各自为8KB, 从而使指令能全速执行。
- 64位的数据总线: 数据传输率为528MB/S, 比80486的32位的105MB/S快了5倍, 再和PCI局部总线相接, 解决了数据堵塞的情况。
- 分支指令预测.

E-mail: qhzheng@mail.xjtu.edu.cn 郑庆华 教授

西安交通大学本科生课程：计算机系统结构

### 4.6.3 Pentium Pro/II 高能奔腾处理器

这是由Intel公司于1995年推出的, 其地址线为36根, 数据线宽度为64位。性能为Pentium的2倍, 但采用的工艺却和普通Pentium一样。

E-mail: qhzheng@mail.xjtu.edu.cn 郑庆华 教授

西安交通大学本科生课程：计算机系统结构

#### 1、设计目标

在利用与Pentium相同的半导体生产工艺条件下, 使其性能提高一倍。

#### 2、方法和途径

改造和改进Pentium的体系结构。将大、中、小型计算机上采用并得到验证的技术, 学术界最新提出的技术以及对原Pentium采用的系统结构技术相互交融, 并用到 Pentium Pro的体系结构中。这种融合的产物, 就是被Intel公司称之为动态执行 (Dynamic Execution) 的体系结构技术, 是继Super Scalar之后的又一重大发展。

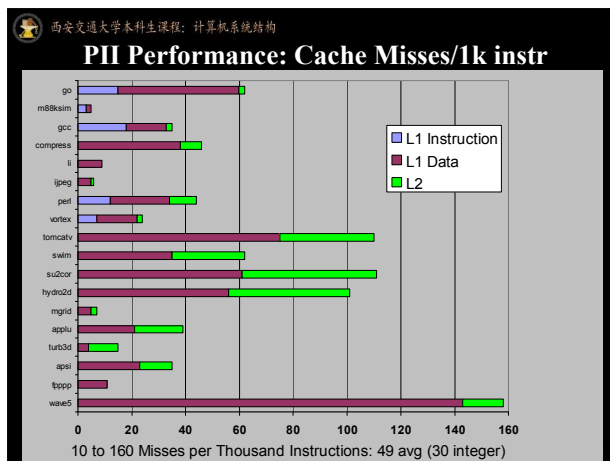
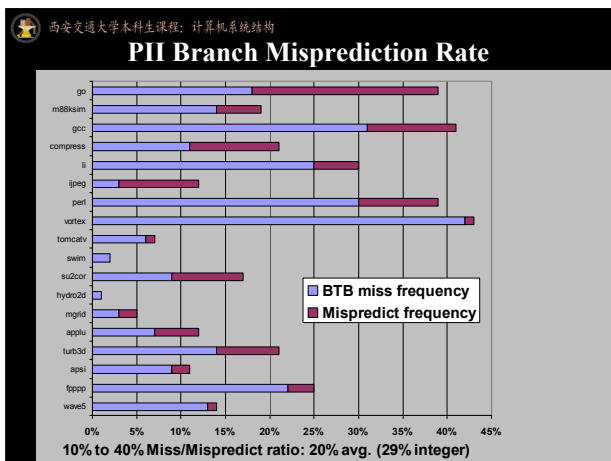
E-mail: qhzheng@mail.xjtu.edu.cn 郑庆华 教授

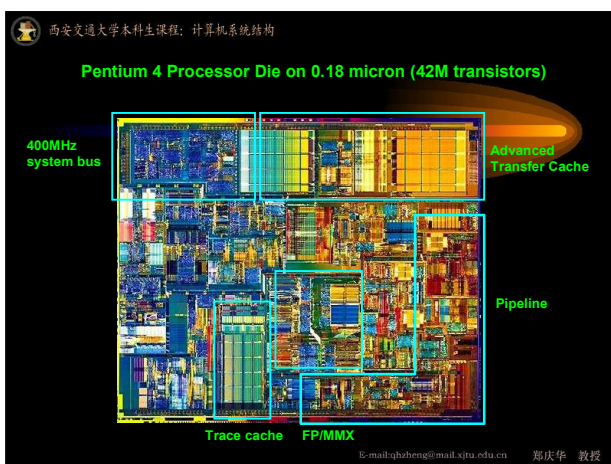
西安交通大学本科生课程：计算机系统结构

### The PII Micro architecture

- Fetched/decoded every cycle.
- Instructions are translated to Risk instructions
- Register renaming and ROB is used.
- Pipeline is 14 stages:
  - 8 stages to fetch/decode/dispatch in-order.
  - 3 stages to execute out-of-order
  - 3 stages to commit

E-mail: qhzheng@mail.xjtu.edu.cn 郑庆华 教授



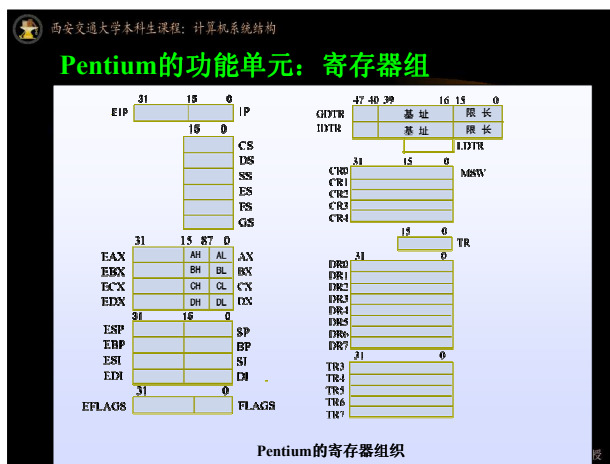
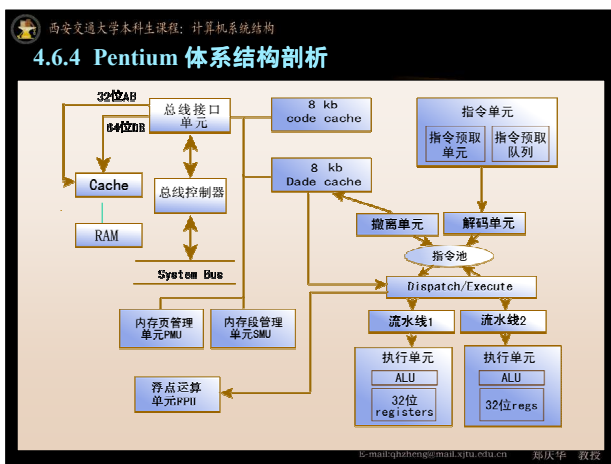


西安交通大学本科生课程：计算机系统结构

### Today's Microprocessor

- Intel Pentium IV Processor
  - Technology
    - 0.13μ process, 55M transistors, 82W
    - 3.2 GHz, 478pin Flip-Chip PGA2
  - Performance
    - 1221 Ispec, 1252 Fspec on SPEC 2000
      - Relative performance to SUN 300MHz Ultrasparc (100)
    - 40% higher clock rate, 10~20% lower IPC compared to P III
  - Pipeline
    - 20-stage out-of-order (OOO) pipeline, hyperthreading
    - 2 ALUs run at 6.4GHz
  - Cache hierarchy
    - 12K micro-op trace cache/8 KB on-chip D cache
    - On-chip 512KB L2 ATC (Advanced Transfer Cache)
    - Optional on-die 2MB L3 Cache
  - 800MHz system bus, 6.4GB/s on P III 133MHz bus
    - Compared with 1.06GB/s on P III 133MHz bus
    - Implemented by quad-pumping on 200MHz system bus

E-mail: qhzheng@mail.xjtu.edu.cn 郑庆华 教授



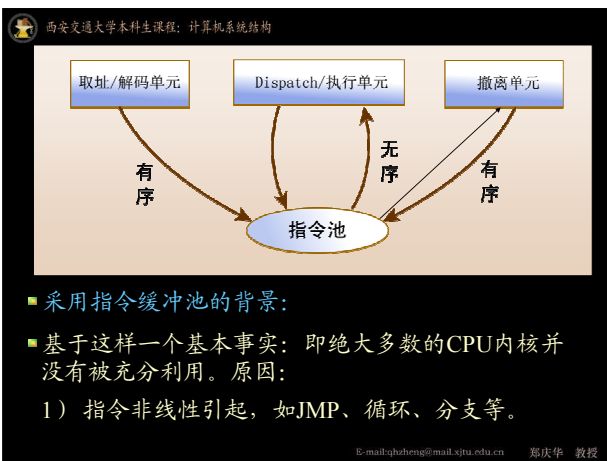
西安交通大学本科生课程：计算机系统结构

### 4-6-5 体系结构的改进问题及其性能分析

- Pentium的超标量流水线设置了两条指令处理流水线，在理想情况下最多可以执行两条指令，并将每条流水线分为5级（步），而 Pentium Pro则采用三条指令流水线，并将其分为12级（步），即所谓超流水线（Super Pipeline），使各指令间的并行性更好。
- 关键在于消除“取指”和“执行”周期之间存在的指令线性排序的限制，并采用指令池的技术打开一个较宽的指令窗口。

E-mail: qhzheng@mail.xjtu.edu.cn 郑庆华 教授



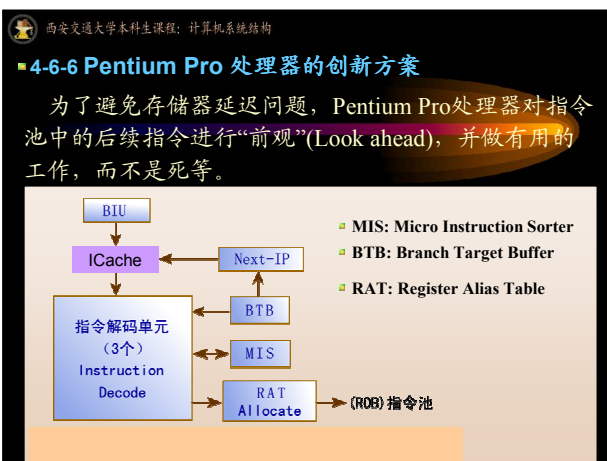


西安交通大学本科生课程：计算机系统结构

■ 要点：

- step1. 先进行推测执行（speculate execution），将结果放入指令池中。
- step2. 实现指令的有序撤离 in order retirement。
- step3. 在推测执行时，采用别名寄存器。

E-mail: qhzheng@mail.xjtu.edu.cn 郑庆华 教授



西安交通大学本科生课程：计算机系统结构

- 根据Next\_IP索引从cache中取出下一条要执行的指令（预测）。
- 而Next\_IP又是根据BTB(Branch Target Buffer) 的输入和中断状态，以及来自执行过程中的错误预测得到索引的。
- BTB: 是一块容量较小的相关存储器, 它监视 ICache 索引并根据转移的历史记录预测下一条要访问 ICache 的索引。

E-mail: qhzheng@mail.xjtu.edu.cn 郑庆华 教授

西安交通大学本科生课程：计算机系统结构

- Pentium中BTB只有512项，预测准确率达到90%以上。
- MIS: Micro Instruction Sorter: 微指令排序器，将3个解码器解码后的指令进行排序和标记操作。
- RAT (Register Alias Table): 寄存器别名表，赋以指令预测执行时的寄存器别名，此后被送入指令池中。指令池也是一段CAM(Content Addressable Memory)或AM(Associative Memory)存储器。

E-mail: qhzheng@mail.xjtu.edu.cn 郑庆华 教授

西安交通大学本科生课程：计算机系统结构

■ 推测执行的基本做法：

- 1) 当CPU在某一条指令上受阻等待时，以推测方式先执行后续指令，由于推测执行将破坏源程序中的正常执行顺序，故推测执行（speculative execution）的结果不能交付给固定的机器状态（如PSW和regs等），而应首先放在指令池中，等待按序撤离（in-order retirement）。
- 2) 另一方面，由于CPU内核执行指令是数据流驱动方式（Data-Driven），只要数据准备好，相应的指令便可驱动执行，而不是以程序原来的顺序执行。即CPU执行指令池中的指令是不按顺序的。

E-mail: qhzheng@mail.xjtu.edu.cn 郑庆华 教授



西安交通大学本科生课程：计算机系统结构

3) 一般情况下，IP将前观(look ahead)20~30条指令，根据概率，这20~30条指令中将平均可能出现5条转移指令，为了减少对寄存器错误依赖，Dispatch/执行单元将寄存器进行改名(用一组别名寄存器替代)。

4) 在指令执行完后进行撤离时，做到有序撤离(in order retirement)，并将指令的运行结果交付给程序员可见的Flag和各寄存器。

E-mail: qhzheng@mail.xjtu.edu.cn 郑庆华 教授

西安交通大学本科生课程：计算机系统结构

### 3) 撤离单元

功能：a. 要监测哪些微操作已经完成，并确定哪一个是符合原来程序中的下一个。b. 必须按程序原来的顺序重新排序。c. 当出现中断、陷阱、出错、断点和误预测的干扰下完成这些工作。

■ 每个时钟周期内最多可以撤离3条微指令。

E-mail: qhzheng@mail.xjtu.edu.cn 郑庆华 教授

西安交通大学本科生课程：计算机系统结构

### Pentium的指令系统特点

- 通过对图形、MPEG视频、音频合成，语音识别、图像处理、游戏及视频会议等的分析，得出了其中最密集的计算过程。
- 发现：
  - ① 短小而高度重复的循环；
  - ② 频繁的乘积和累加；
  - ③ 高度并行的操作。
 是其中的要害。
- 为此提出了MMX技术，设计为一套适用的基本整数指令，使该指令集可以广泛应用于大众多媒体和通信软件。该技术重点包括：

E-mail: qhzheng@mail.xjtu.edu.cn 郑庆华 教授

西安交通大学本科生课程：计算机系统结构

- SIMD
  - 57条新指令
  - 8个64位的MMX寄存器
  - 4种新的数据类型
    - 压缩型字节- 8个字节一位→64位数
    - 压缩型字 - 4个16位的字→ 64位数
    - 压缩型双字-
    - 压缩型四字- 1个64位数
- 1. PADP[B,W,D]
 

|                                |                                |                                |                                |
|--------------------------------|--------------------------------|--------------------------------|--------------------------------|
| a <sub>3</sub>                 | a <sub>2</sub>                 | a <sub>1</sub>                 | a <sub>0</sub>                 |
| +                              | +                              | +                              | +                              |
| b <sub>3</sub>                 | b <sub>2</sub>                 | b <sub>1</sub>                 | b <sub>0</sub>                 |
| -----                          |                                |                                |                                |
| a <sub>3</sub> +b <sub>3</sub> | a <sub>2</sub> +b <sub>2</sub> | a <sub>1</sub> +b <sub>1</sub> | a <sub>0</sub> +b <sub>0</sub> |

E-mail: qhzheng@mail.xjtu.edu.cn 郑庆华 教授

西安交通大学本科生课程：计算机系统结构

### 2. PMADD —— 压缩型字相乘并相加结果对

非常适合于实现点积运算。同量点积，如FFT, DCT, 矩阵乘等。

|                                                              |                                                              |                |                |
|--------------------------------------------------------------|--------------------------------------------------------------|----------------|----------------|
| a <sub>3</sub>                                               | a <sub>2</sub>                                               | a <sub>1</sub> | a <sub>0</sub> |
| *                                                            | *                                                            | *              | *              |
| b <sub>3</sub>                                               | b <sub>2</sub>                                               | b <sub>1</sub> | b <sub>0</sub> |
| -----                                                        |                                                              |                |                |
| a <sub>3</sub> b <sub>3</sub> +a <sub>2</sub> b <sub>2</sub> | a <sub>1</sub> b <sub>1</sub> +a <sub>0</sub> b <sub>0</sub> |                |                |

E-mail: qhzheng@mail.xjtu.edu.cn 郑庆华 教授

西安交通大学本科生课程：计算机系统结构

### 关于Pentium Pro的技术总结：

动态执行

1. 改进的转移预测：以给核心提供较多可执行的指令和数据流。
2. 数据流分析：选择最合适的指令进行分派/执行。
3. 推测执行(Speculative execution)：以优先顺序执行指令。

技术创新：正是由于上述三项技术的有机结合，使设在Pentium Pro处理器再采用与Pentium相同的半导体制造工艺条件下，使性能提高了一倍。

E-mail: qhzheng@mail.xjtu.edu.cn 郑庆华 教授



## 本章习题

- 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.9
- 补充作业：请运用先行控制技术，说明 Pentium 计算机的工作原理，说明指令预测执行技术的实现原理。