

第三节 高档微处理器

一、Pentium基本型

主要特征

- 超级标量结构 (双指令流水线)
- 双Cache (指令Cache和数据Cache分离)
- 分支预测技术
- 64位数据总线
- RISC和CISC的结合



(一) 内部结构

1、功能部件

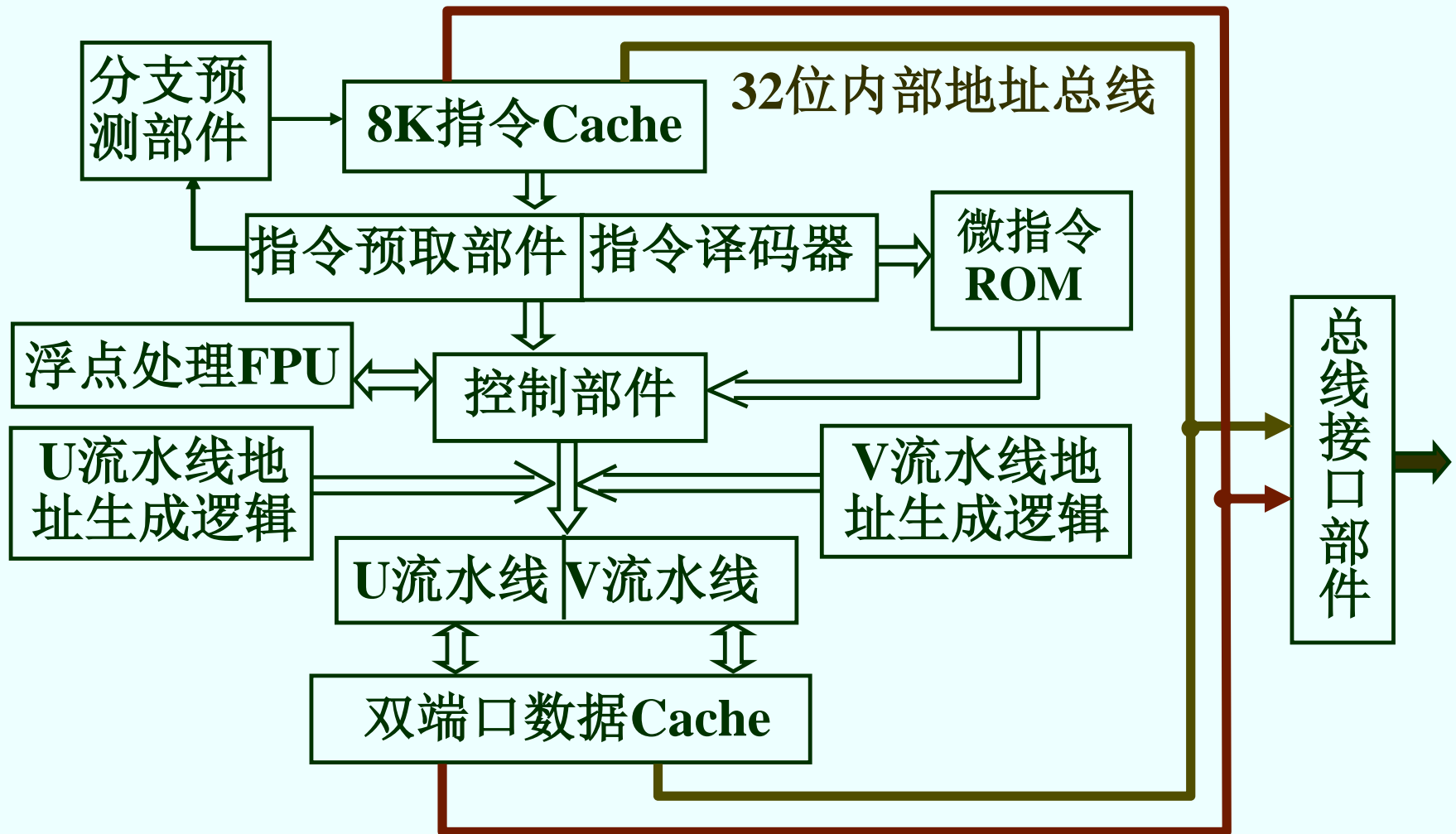
从功能模块的角度, 在80486的基础上增加了:

- 一条指令流水线(包括相应的地址生成部件)
- 一个Cache

形成了双执行部件和双Cache结构。分支预测部件则是由于流水线的引入, 为提高流水线效率而设置的性能增强部件。

如下图所示:

64位内部数据总线



(1) 指令部件

U和V两条独立的指令流水线，各自有独立的地址生成逻辑，并与各自独立的内部数据总线与双端口Cache连接。从而实现了从地址生成到数据存取的独立操作。

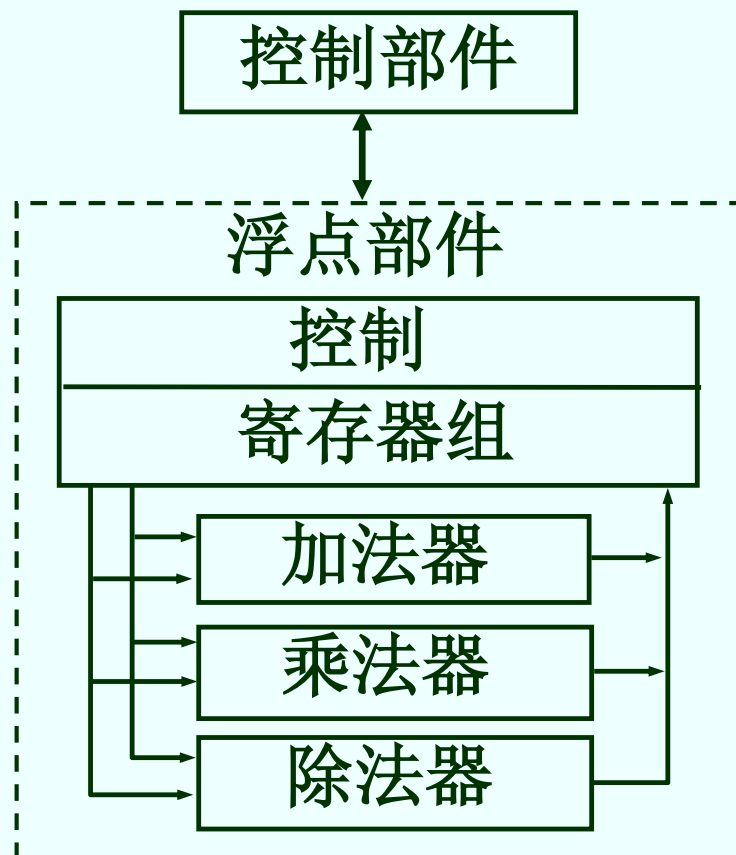
- U流水线: 可执行任何整数及浮点两类指令;
- V流水线: 只能执行简单整数指令

(2) 双Cache

数据Cache与指令Cache分离, 提高操作的并行性

(3) 浮点部件

速度是80486内部浮点部件的3~5倍。



可独立寻址的8个数值寄存器

	79	78	64	63	0
R0	符号	指数	有效位		
R1					
R2					
R3					
R4					
R5					
R6					
R7					

浮点加法器、乘法器和除法器可同时进行三种不同运算。

- 浮点指令流水线

分为8级流水线:

预取指令

指令译码1

指令译码2

地址生成

取操作数

执行1

执行2

写回结果和错误报告

} 与整数流水线中的
五个步骤同步执行

(4) 微指令ROM

采用CISC技术, 结合了RISC的设计思想, 一些简单指令(如加、减、移位等)直接用硬件逻辑电路实现, 复杂指令仍采用微程序的设计思想(微指令ROM实现复杂指令)。

简单指令例:

MOV reg, reg/mem/imm

MOV mem, reg/imm

ALU reg, reg/imm

INC reg/mem

DEC reg/mem

PUSH reg/mem

POP reg 等

复杂指令, 如:

ADD reg, mem

ADD mem, mem

无需任何微码控制, 在一个时钟周期内执行的指令

2、寄存器

通用寄存器、变址寄存器，段寄存器以及指针与80486基本相同。增加了大量的测试寄存器和检查寄存器、性能监测的寄存器。

如测试寄存器1~12, 用于奇偶校验测试、TLB线性地址测试, Cache测试、分支预测BTB测试等。

增加控制寄存器CR₄(6个有效位), 其中的控制位PSE:

$$\text{PSE} = \begin{cases} 0 & \text{页面大小为4K} \\ 1 & \text{页面大小为4M} \end{cases}$$

页面太大: 失去分页的优点;

页面太小: 命中率下降 (所访问数据在页面的概率)

(二) Pentium的主要技术改进

1. 超级标量结构(Super Scalar Architecture)

针对微处理器可并发的指令数, 两种典型结构:

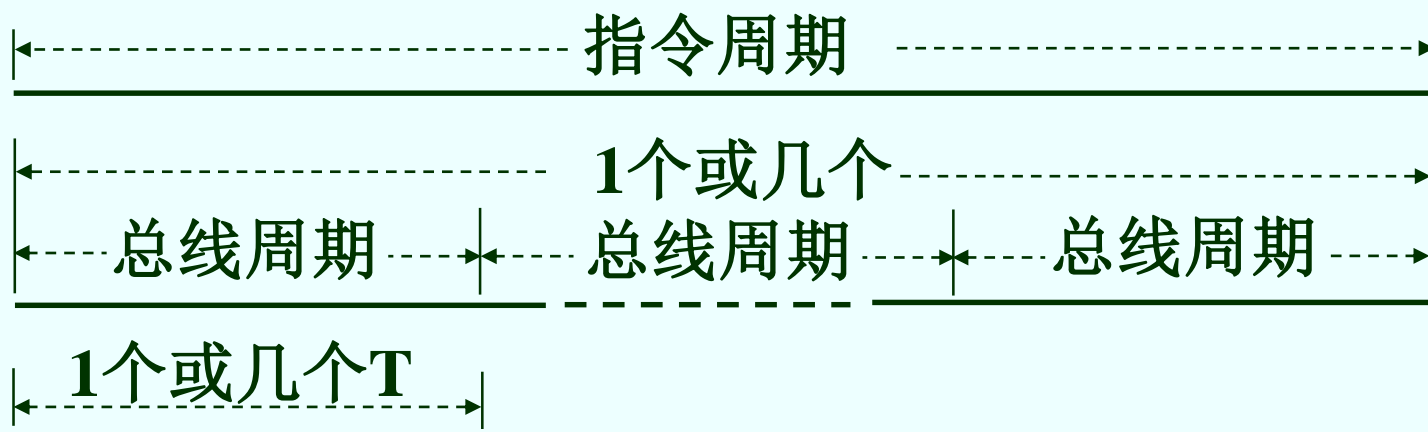
标量结构 超级标量结构

什么是标量处理器(Scalar Processor):

- 处理器内部只有一个指令执行部件
- 一个周期执行一条指令 (?)

计算机系统三种基本的周期信号:

- 指令周期
- 机器周期(总线周期)
- 时钟周期(节拍周期)



一个周期是否能执行完成一条指令需要从是否具有流水线功能来分析:

➤ 非流水线执行方式

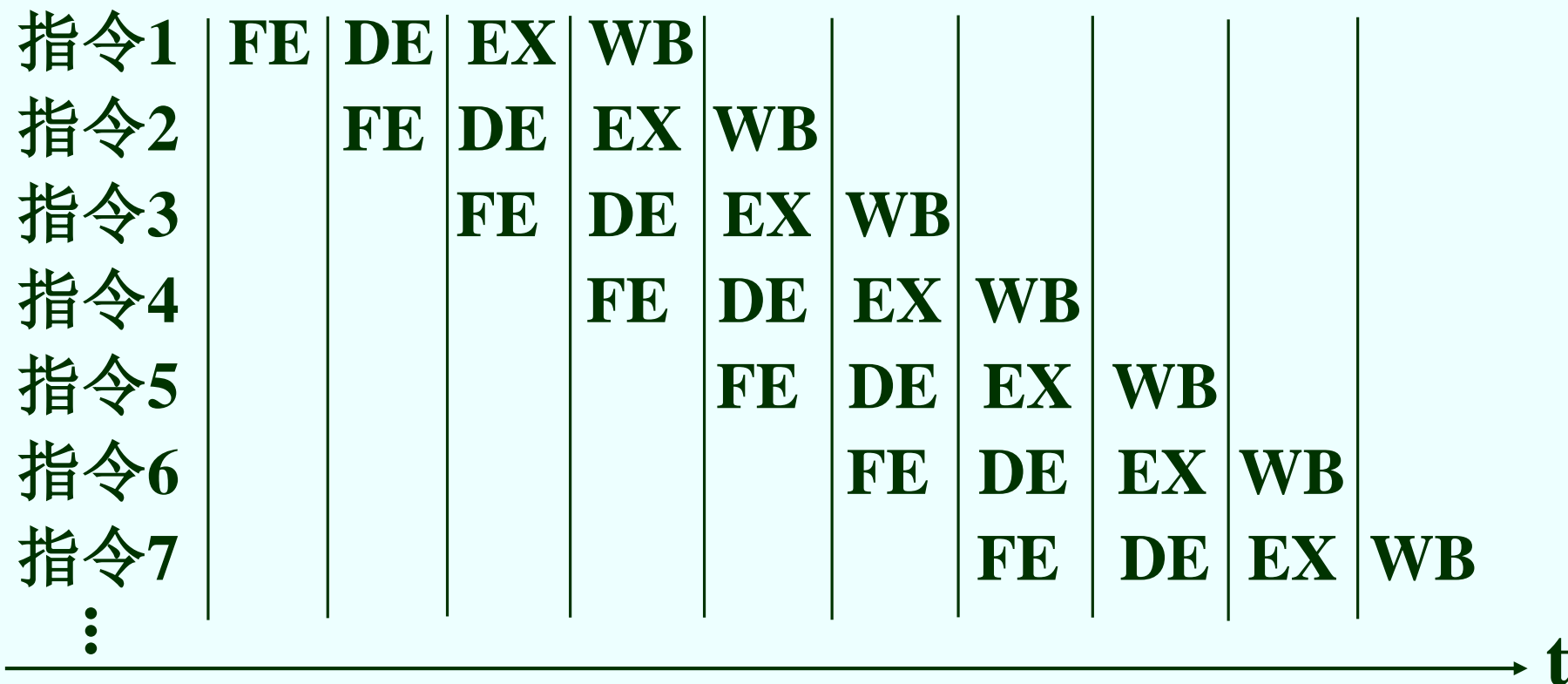
指令串行执行, 一个指令周期完成一条指令。

➤ 流水线执行方式

指令分解为多个独立的执行阶段。

每一执行阶段的时间长度: 一个总线周期(或者更短)

此时, CPU执行指令的过程如下图所示:



在流水线方式下：一个总线周期可以执行一条指令；如果将一个总线周期长度定义为一个时钟周期，则一个时钟周期可以完成一条指令。

因此，可以用时钟周期数来作为衡量CPU性能的指标：

➤ IPC (Instructions/per cycle)

即每一个总线周期(时钟周期)完成的指令数。

非流水线指令执行方式: $IPC < 1$

流水线指令执行方式: $IPC = 1$ (理想状况)

怎样使 $IPC > 1$?

处理器内部有多个指令执行部件, 并且多个部件独立并行工作, 则可以使 $IPC > 1$ 。

Pentium具有 $IPC > 1$ 处理能力的处理器。

超级标量结构:

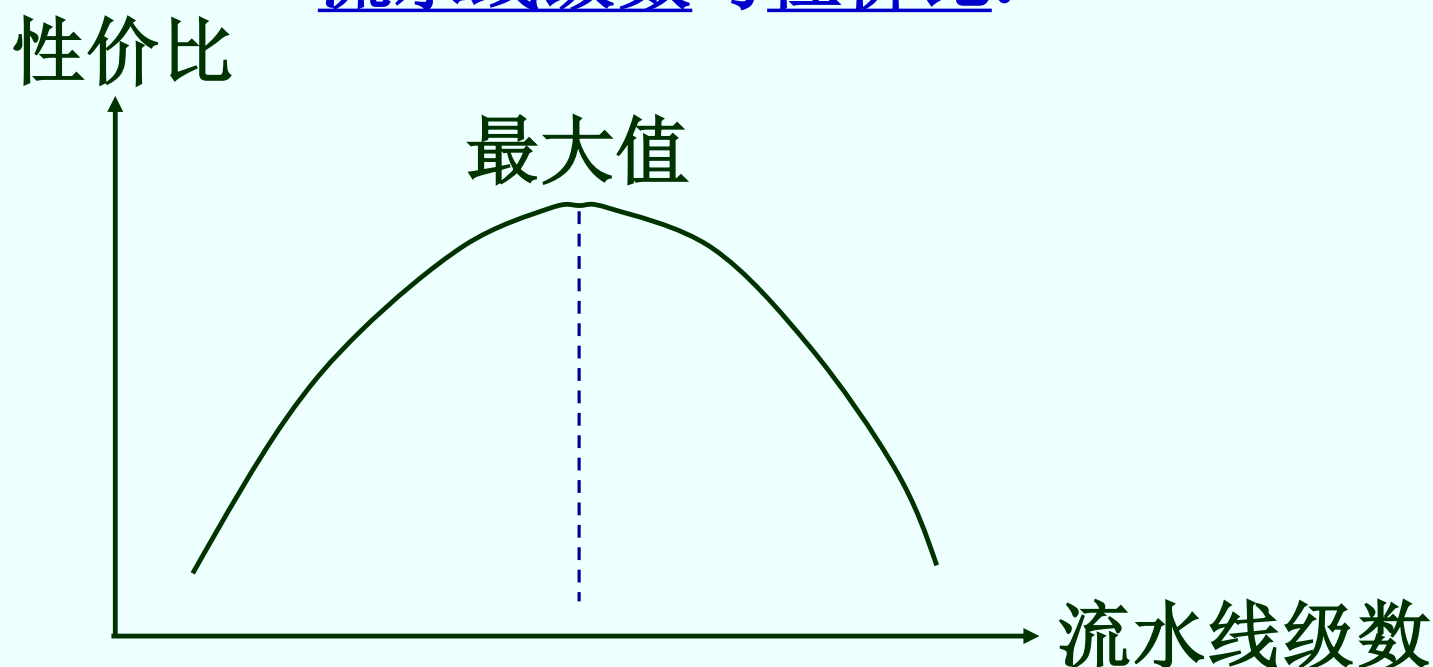
- ① 处理器内有多个独立并行的流水线指令部件
- ② 具有 $IPC > 1$ 的处理能力。

➤ $CPI(Cycle/per\ Instructions) = 1 / IPC$

➤ 流水线级数与性能和性价比:

理论上讲: 流水线级数越多(流水线深度越深), 并发指令数越多, 性能越高; 但随着流水线级数的增加, 各级之间的缓冲(锁存)延时也将逐步累加, 从而妨碍性能的提升, 同时硬件越复杂, 成本越高。

流水线级数与性价比:



2. Pentium的指令流水线结构

- 五级流水线:

指令预取(PF)

指令译码(D1)

地址生成(D2)

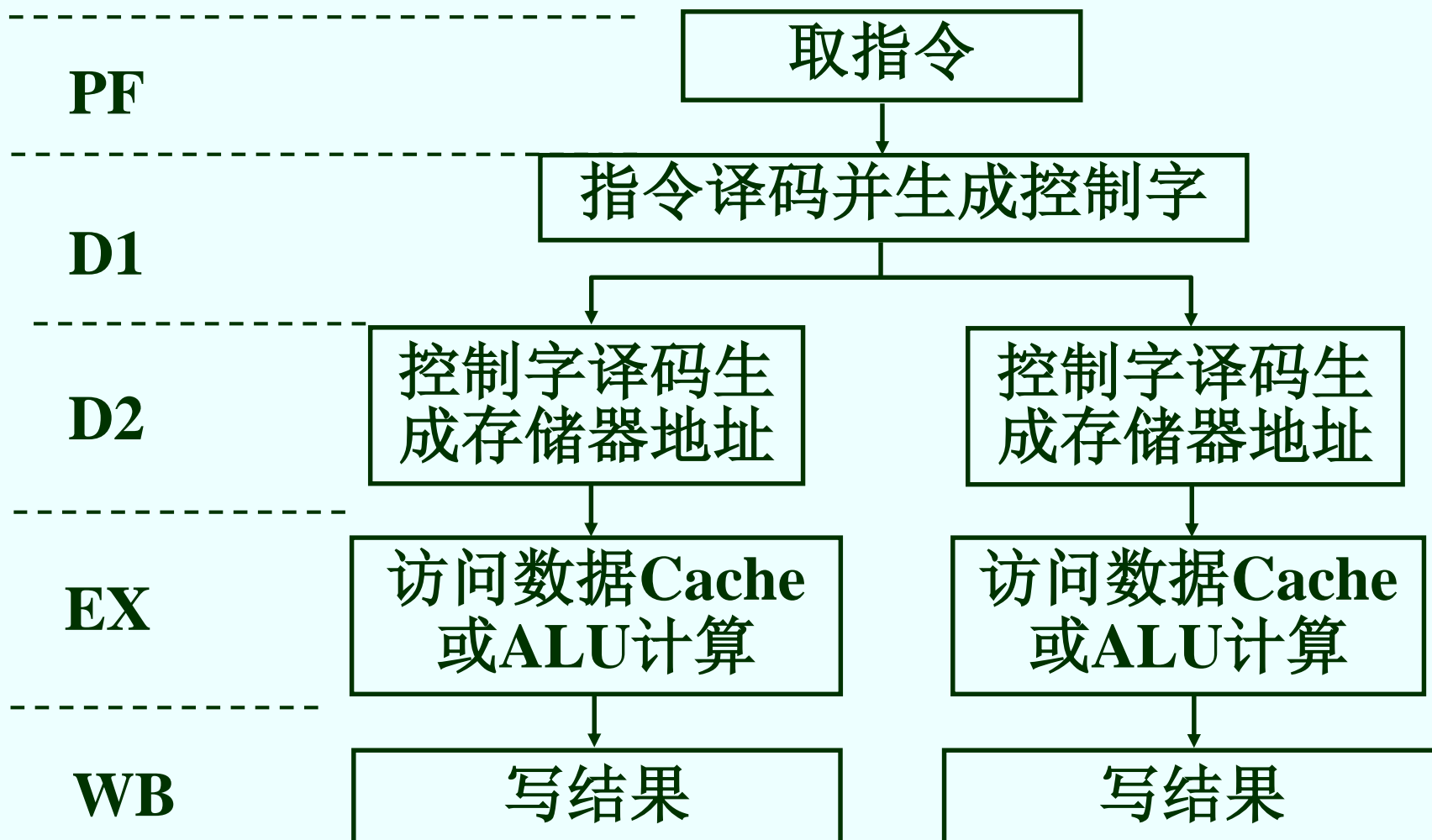
指令执行(EX)

写结果和设置处理器状态(WB)

– U和V两条流水线同时执行的流水线状况:



以流程的方式, 超标量执行可表示为以下形式:



同时运行两条指令的条件？

如何分配两条流水线上的指令：

可同时执行两条指令,但要满足一定**配对规则**:

- 两条指令必须是“简单”指令;
 硬件逻辑实现(不是微代码实现)、单周期执行
- 两条指令要在指令Cache内;
- 成对指令无寄存器数据相关;

如：“写后读” 操作 $\left. \begin{array}{l} \text{MOV EAX, 8} \\ \text{MOV EBX, EAX} \end{array} \right\} \text{寄存器相关}$

- 指令中不能同时包含偏移量与立即数等。

由此引发Pentium流水线调度策略:

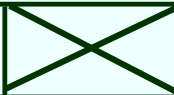



两条流水线的调度策略：按序发射按序完成策略

在译码阶段(D1), 两条连续的指令(如 i_1 和 i_2)先后被译码, 并检查是否可以配对。若是, 这对指令同时发射到U、V流水线的D2段。

要求: 一对同时进入D2阶段的指令也必须同时离开D2阶段, 并进入EX段。如果一条指令在D2段停留, 则另一条指令也必须在D2段停顿。

成对指令进入EX阶段后, 若一条指令EX阶段停留, 则另一条指令也必须在EX阶段停顿。

如下图所示:

i_1	U	PF	D1	D2		EX	WB	
i_2	V	PF	D1	D2	D2	EX	WB	
i_3	U	PF	D1		D2	EX	EX	WB
i_4	V	PF	D1		D2	EX		WB

如果 i_2 需要两个D2, 则 i_1 须停顿一个周期。同时也导致 i_3 、 i_4 在D1结束后不能进入D2。

i_3 、 i_4 的情况是: 由于 i_1 和 i_2 在D2停留, 使 i_3 和 i_4 在D1后不能进入D2而停留(阻塞)一个周期。

如果 i_3 执行所需的时间较长(假设需要2个执行周期), 则V流水线的指令 i_4 必须停顿等待 i_3 执行完毕并一起进入下一阶段WB。

3. 分支预测技术

转移指令对流水线的影响

(1) 条件转移指令

ADD PF D1 D2 EX WB

JC PF D1 D2 EX WB

SUB

MOV

INC

DEC

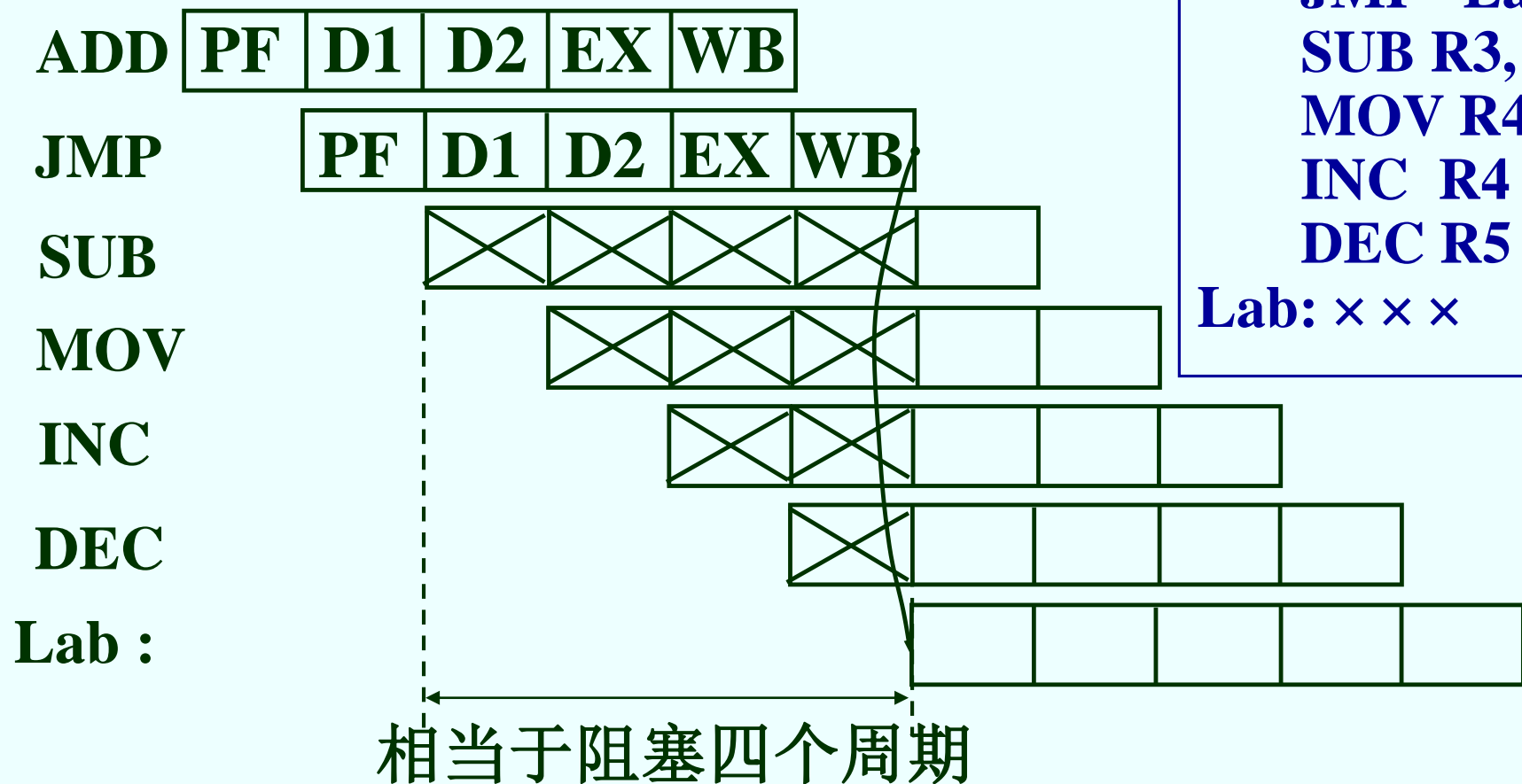
Lab: xxx

如果有进位, 转移
相当于阻塞四个周期

Lab: xxx

ADD R1, R2
JC Lab
SUB R3, R1
MOV R4, R5
INC R4
DEC R5

(2) 无条件转移指令



⋮
ADD R1, R2
JMP Lab
SUB R3, R1
MOV R4, R5
INC R4
DEC R5
Lab: × × ×

说明:

- 转移指令除了对指令执行时间造成不利影响以外, 由于在转移指令与目标指令之间的指令的执行还可能影响一些中间结果和标志位, 从而可能导致程序发生逻辑错误;
- 对于无条件转移指令, 虽然在**D1**结束时便可知程序要转移而不用执行其后面的几条指令, 但是, 在**WB**结束之前的时间, 需要按寻址方式形成目标地址, 因此也必须等到**WB**结束, 才能转移到目标指令执行。

• 解决“条件转移指令”对流水线的影响

① 分支预测技术

- 静态分支预测: 硬件固定选择分支
- 动态分支预测: 在预测点, 根据程序过去分支情况的统计值为依据(或上一次的分支情况)选择分支。

依据: 程序执行的时间局部性规律

Pentium的分支预测: 动态分支预测

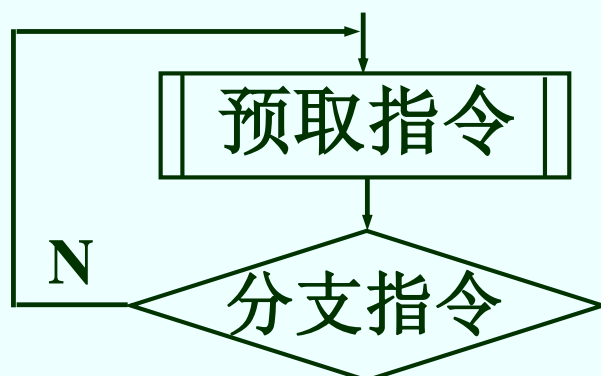
■ 分支情况统计部件:

分支目标缓冲器(**Branch Target Buffer - BTB**)

■ 分支预测时间: 指令预取阶段(**PF**)

■ 分支预测过程: 两个指令预取缓冲区

第一预取缓冲区



BTB预测分支

不转移

预取分支指令
后的指令序列

若预测错误,
清除该缓冲区

第二预取缓冲区

注: 程序按预测方向执行, 如果
分支预测发生错误, 仍会
造成四个周期的时间损失。

转移

预取分支目
标指令序列

若预测错误,
清除该缓冲区

若预测要分支, 则
两个方向的指令
预取同时进行

附：造成Pentium(包括其它CISC处理器)流水线阻塞的其它可能因素

① 指令码太长

即指令码的长度超过了数据总线的宽度。

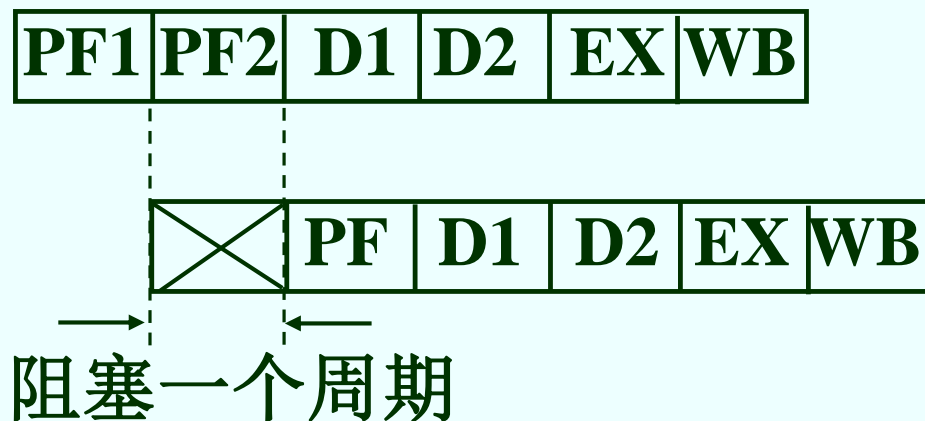
比如, Pentium处理器允许的数据类型有字节、字、双字、4倍字等。

一条指令包含指令码、数据和地址等。此外, 指令的寻址方式中还包含以立即数形式出现的偏移量。因此, 指令码长度就可能超过数据总线的宽度, 这意味着单总线周期不能取出一条指令的全部字节。以80486为例, 指令超过4个字节, 至少需要2个总线周期取指令码, 即需要2个阶段的流水线长度。

ADC **d(DI)**, 立即数
(可达6个字节)

下一条指令:

$(d+DI)+\text{立即数} \rightarrow (d+DI)$



② 执行阶段时间开销太大

流水线的理想状况: 假设了一个**EX**的时间长度能完成指令所要求的全部功能。

指令: **ADC** **d(DI)**, 立即数

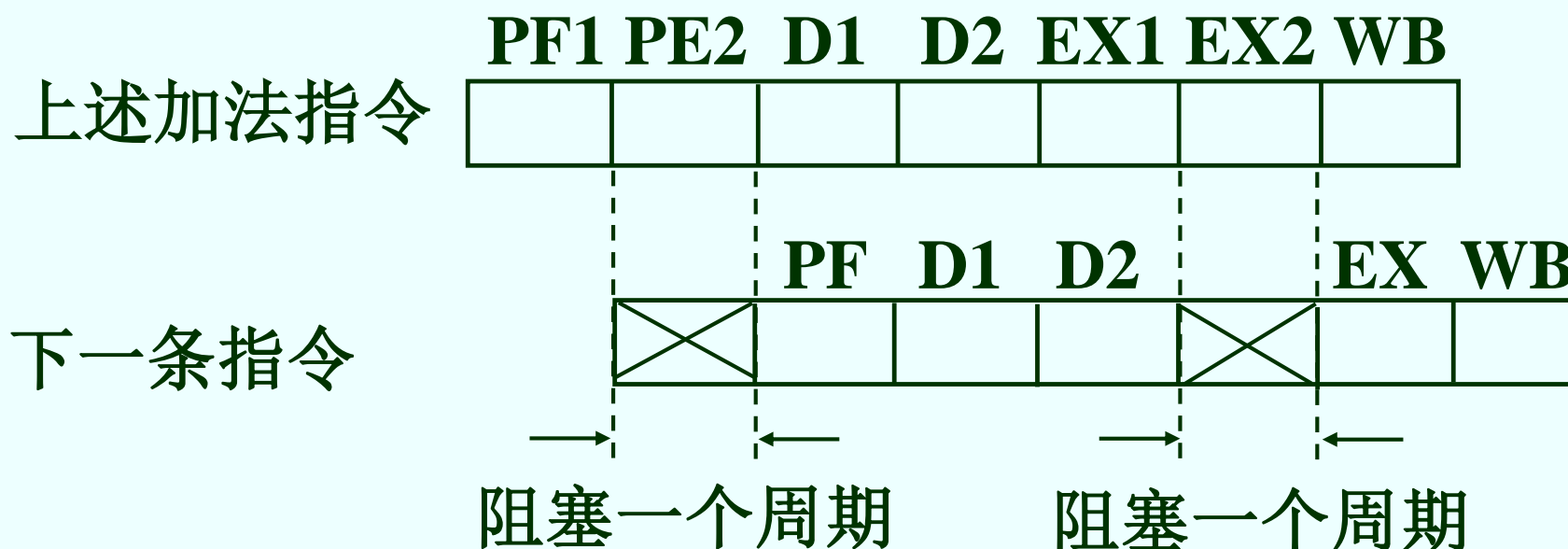
在**EX**阶段的操作:

在EX阶段的操作:

(1) 根据在D2形成的操作数地址访存 取操作数

(2) 计算: 操作数1 + 操作数2 + C(进位位)

如果一个EX周期不能完成上述操作, 则需要两个EX周期。



③ 数据相关: 也称为数据依赖, 如 “写后读”

比如: `MOV AX, 8`

PF1	D1	D2	EX	WB
-----	----	----	----	----

`MOV BX, AX`

PF	D1	D2	X	EX	WB
----	----	----	---	----	----

→ ←
阻塞一个周期

④ 资源冲突: 多条并行的指令需要使用同一资源

计算机系统中存在大量共享的软硬件资源, 如果并行的指令在某一时刻都需要使用同一种共享资源, 则出现资源冲突(比如对总线的访问), 在这种情况下, 至少有一条指令必须等待另外的指令释放该共享资源。

错误案例：根据指令画出流水线执行情况

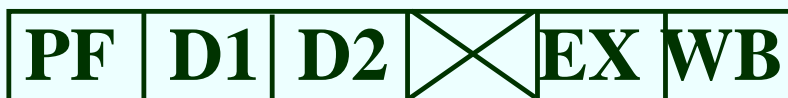
假设指令3和指令4的执行不会发生流水线阻塞。

错误：



.....

或：



比如：MOV AX, 8
MOV BX, AX
指令3
指令4

正确:

PF1	D1	D2	EX	WB
-----	----	----	----	----

PF	D1	D2	✕	EX	WB
----	----	----	---	----	----

PF	D1	D2	✕	EX	WB
----	----	----	---	----	----

PF	D1	D2	✕	EX	WB
----	----	----	---	----	----

比如: **MOV AX, 8**
MOV BX, AX
指令3
指令4

在CISC结构,有六种可能的情况造成流水线阻塞:

- 指令太长,一个周期不能取出全部指令码;
 - 指令功能复杂,单执行周期不能完成所要求功能;
 - 无条件转移指令;
 - 条件转移指令,预测错误;
 - 数据相关;
 - 资源冲突。
- 从时间损失的角度,相当于造成流水线阻塞

由于存在上述诸多因素使流水线不能按理想状况运行,因此在实际情况下,单流水线 $IPC < 1$,双流水线 $IPC < 2$ 。

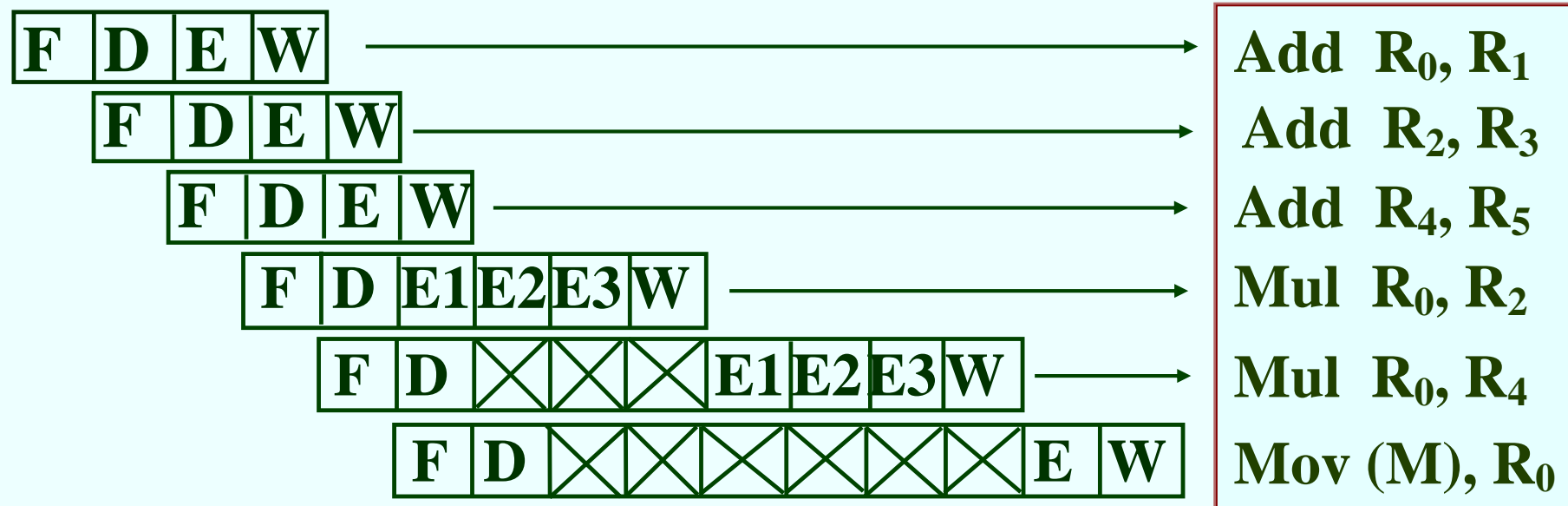
对Pentium处理器,除上述因素,还存在指令配对条件,据统计: Pentium双流水线1000个周期平均执行1300条指令,则 $IPC=1.3$ 。

例：计算流水线性能

计算 $(x_1+y_1)(x_2+y_2)(x_3+y_3)$, 结果存入内存单元。
假设六个参数已分别在 $R_0 \sim R_5$ 个寄存器中; 四级流水线, 仅乘法指令的执行阶段需要 $3\Delta t$, 其余阶段为 $1\Delta t$; 运算类指令: $R_i \text{ op } R_j \rightarrow R_i$
编写以下程序:

```
Add R0, R1      ;  $x_1+y_1 \rightarrow R_0$   
Add R2, R3      ;  $x_2+y_2 \rightarrow R_2$   
Add R4, R5      ;  $x_3+y_3 \rightarrow R_4$   
Mul R0, R2      ;  $(x_1+y_1)(x_2+y_2) \rightarrow R_0$   
Mul R0, R4      ;  $(x_1+y_1)(x_2+y_2)(x_3+y_3) \rightarrow R_0$   
Mov (M), R0     ;  $R_0 \rightarrow (M)$ 
```


流水线运行情况:



(1) 流水线吞吐率 $TP = \frac{n}{T_k} = \frac{6}{15\Delta t}$

(2) 流水线加速比

$$\text{理论值: } S_p = \frac{nk}{k+n-1} = \frac{6 \times 4}{4+6-1} = 2.666$$

$$\begin{aligned} \text{实际值: } S_p &= \frac{T_s \text{ (顺序方式执行所用时间)}}{T_k \text{ (流水线方式执行所用时间)}} \\ &= \frac{3 \times 4\Delta t + 2 \times 6\Delta t + 1 \times 4\Delta t}{15\Delta t} \\ &= \frac{28\Delta t}{15\Delta t} = 1.866 \end{aligned}$$

(3) 流水线效率

$$E = \frac{T_s}{k \times T_k} = \frac{28\Delta t}{4 \times 15\Delta t} = 0.466$$

4. Cache的一致性协议

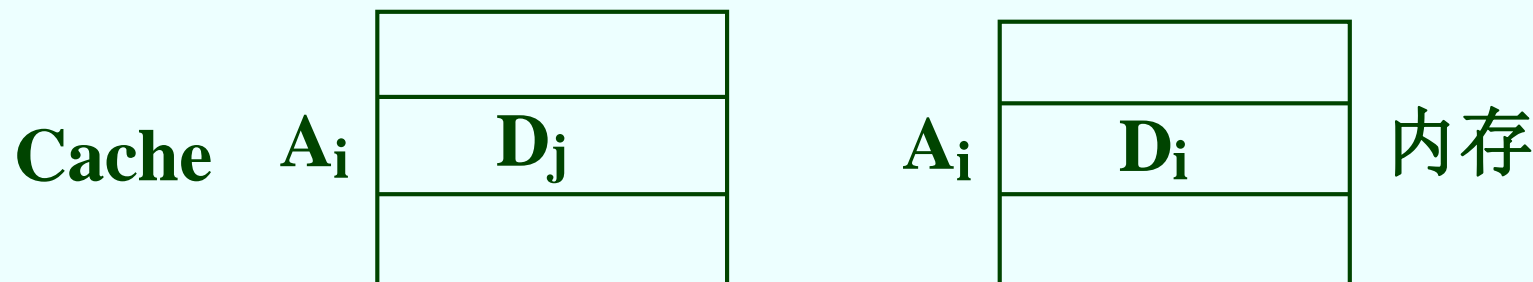
为支持多处理器系统, 针对Cache内容丢失、过时、Cache淘汰、 Cache填充等, Pentium提供的使用Cache的一套规则, 称为Cache的一致性协议。

MESI一致性协议

该协议为每一个Cache行规定了四种状态, 一个Cache行在任何时刻处于某种状态之一。

M — Modified(已修改)

一个Cache行相对于所对应的内存地址单元, 其内容已被修改。比如, 当写Cache 命中后:

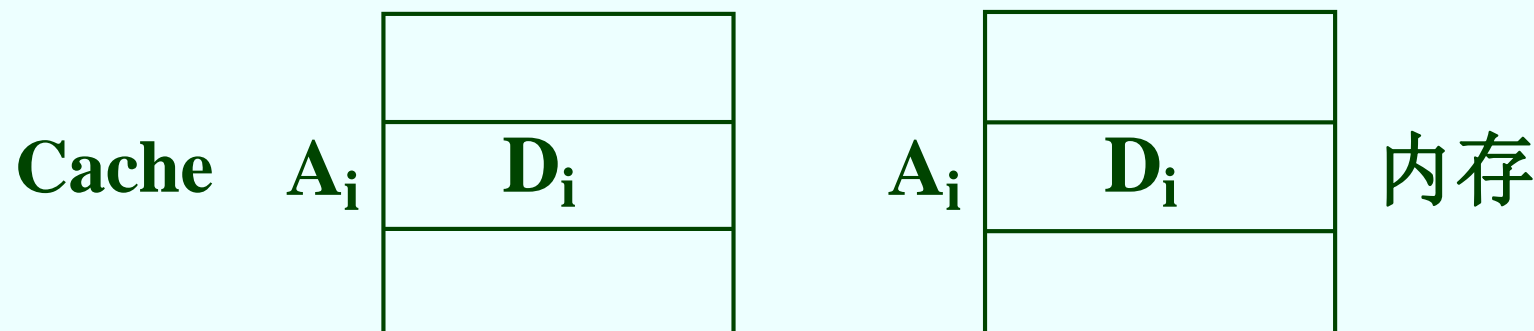


处于M状态的Cache行, 可直接进行读/写操作, 无需启动访问内存的总线周期。

M状态相当于为避免Cache内容丢失而设立的“更新位”

E — Exclusive(独占)

此时的Cache行与所对应内存单元内容相同。如：



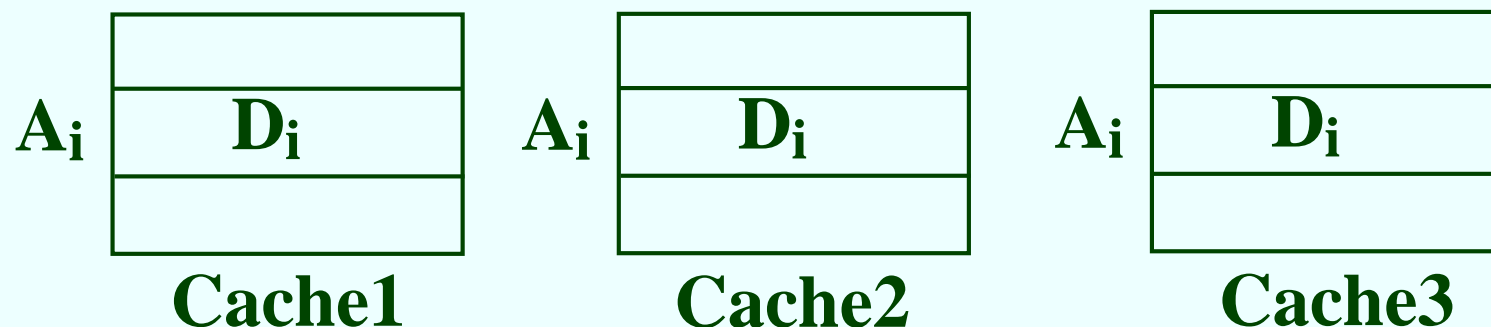
对处于E状态的Cache行的读/写操作，无需启动访问内存的总线周期。

对处于E状态的Cache行实施写操作后，该Cache行状态将变为M状态。

S — Shared(共享)

指示本Cache行与其它Cache行可以共享。

即同一个Cache行的内容可以放在多个Cache内。



对处于S状态下的Cache行进行读操作, 无需启动访问内存的总线周期; 而写操作必须启动访问内存的写周期(直写), 使内存相应单元持有共享Cache行的更新值。(总线监视, 避免过时)

I — Invalid(无效)

该Cache行内容无效。

- ◆ 对I状态的Cache行进行读操作：
不命中Cache, 访问内存, 执行Cache填充。
- ◆ 对I状态的Cache行进行写操作：
写入Cache, 同时写入相应内存单元(直写)。

(三) Pentium 的引脚

1. 数据线 $D_{63} \sim D_0$
2. 数据奇偶校验位 $DP_7 \sim DP_0$
3. 数据校验检查位 \overline{PCHK}
4. 地址线 $A_{31} \sim A_3$ $\overline{BE}_7 \sim \overline{BE}_0$ (对应 $D_{63} \sim D_0$ 的8个字节)
5. 地址奇偶校验位 AP (双向)
6. 地址奇偶校验检查位 \overline{APCHK}

处理器检测到 $\overline{EADS} \rightarrow$ 读地址 \rightarrow 建立 \overline{APCHK}

7. 流水线状态指示信号 IU 、 IV

IU 指示U流水线当前指令执行结束

IV 指示U流水线当前指令执行结束

(四) 两种与Pentium基本型同期处理器比较

1. PowerPC

- RISC结构
- 总线: 数据64位, 地址32位
- 寄存器: 32个32位通用寄存器, 32个64位浮点寄存器
- 32K片内Cache、支持片外Cache
- 三条指令流水线

2. Alpha 21164(1994年)

- RISC结构
- 片内8K指令Cache, 8K数据Cache, 96K片内二级Cache; 支持片外Cache
- 寄存器: 32个64位通用寄存器, 32个64位浮点寄存器
- 四条指令流水线(两条整数, 两条浮点)
- 速度: 1200 MIPS

(1995年Pentium Pro约440MIPS

1997年Pentium II 约466MIPS

1999年Pentium III达到1000MIPS)

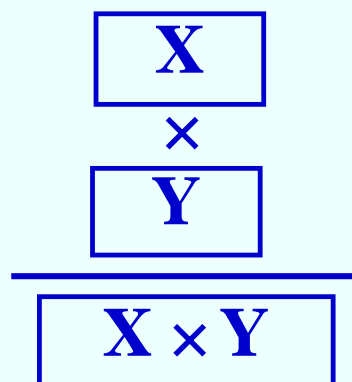
二、Pentium MMX(多能Pentium)



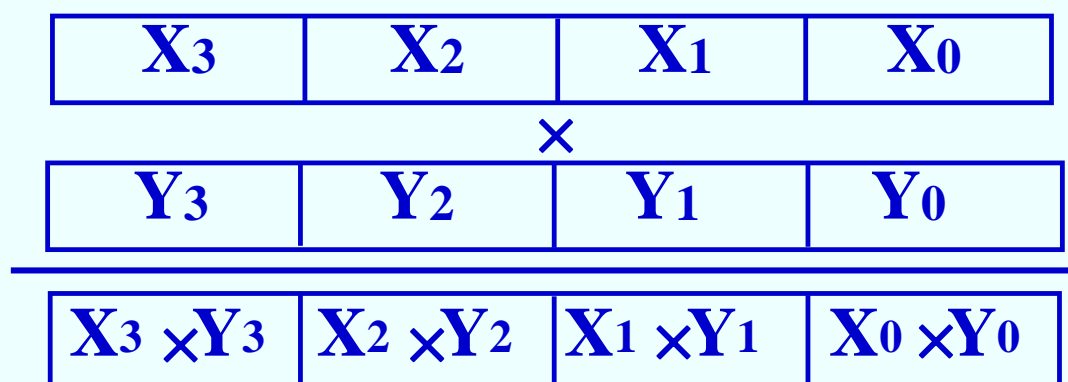
1、基本特征

- 新增加57条多媒体信息处理指令
- 片内Cache容量从16K增加到32K
- SIMD技术: 一条指令同时处理多个数据

传统标量处理



SIMD



- 积和运算功能(如 $X = a_0 \times b_0 + a_1 \times b_1 + \dots + a_7 \times b_7$)
- 饱和计算功能

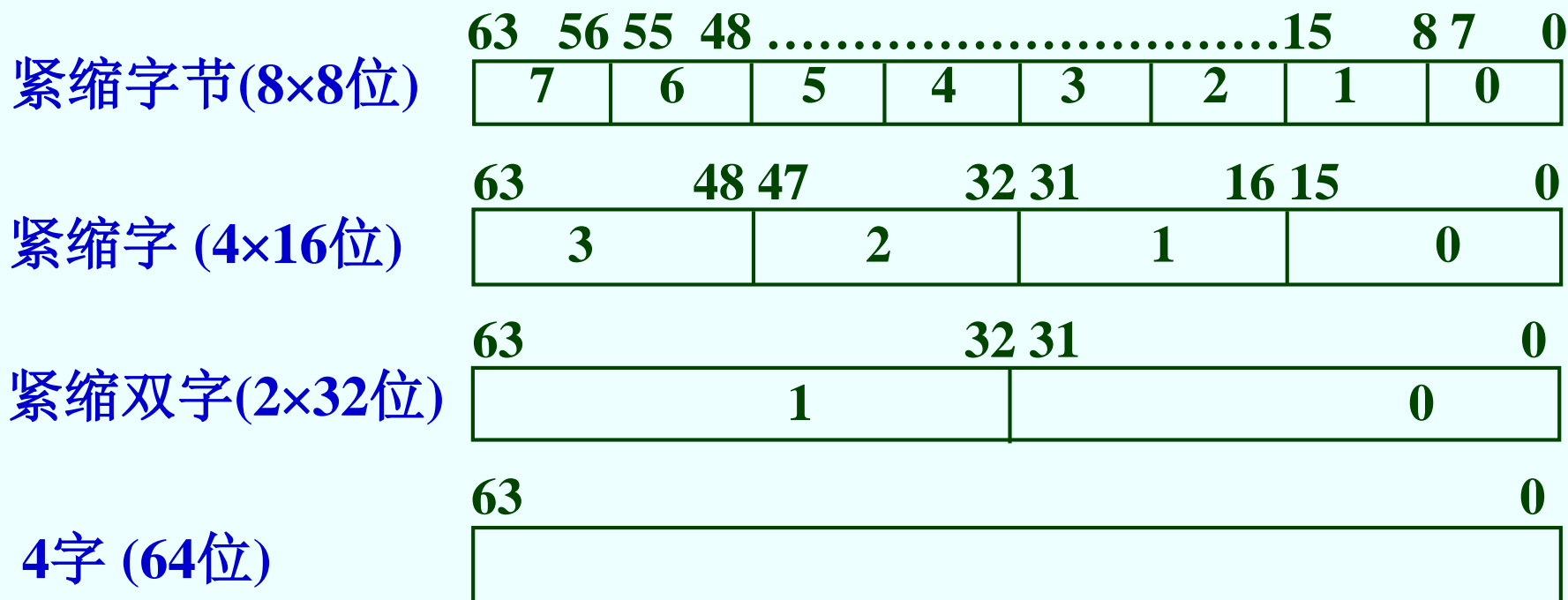
2、MMX技术对编程环境的扩展

(1) MMX寄存器

由8个64位寄存器MM0~MM7组成, 直接使用寄存器名MM0 ~ MM7访问。但只能用来对数据进行运算, 不能用于寻址。MM0 ~MM7非独立寄存器, 是由FPU的R0~R7映射而来。

(2) MMX数据类型

MMX技术定义了以下64位数据类型:



紧缩数据在存储器中以连续地址方式存放

(3) 单指令多数据执行方式

用SIMD技术对64位的MMX寄存器中的字节、字或双字实现算逻运算。例如，指令PADDSB将源操作数中的8个带符号字节加到目标操作数中的8个带符号字节上，并将8字节的结果存储到目标操作数中。

指令例:

字乘加指令 **PMADDWD** MM_i, MM_j

功能:

MM_j	b3	b2	b1	b0
	×	×	×	×
MM_i	a3	a2	a1	a0
	+		+	
MM_i	$a3 \times b3 + a2 \times b2$		$a1 \times b1 + a0 \times b0$	

可以加快如 $\sum a_i \times b_i$ 运算的速度并方便编程

Pentium基本型和Pentium MMX
第一代奔腾处理器(P5架构)

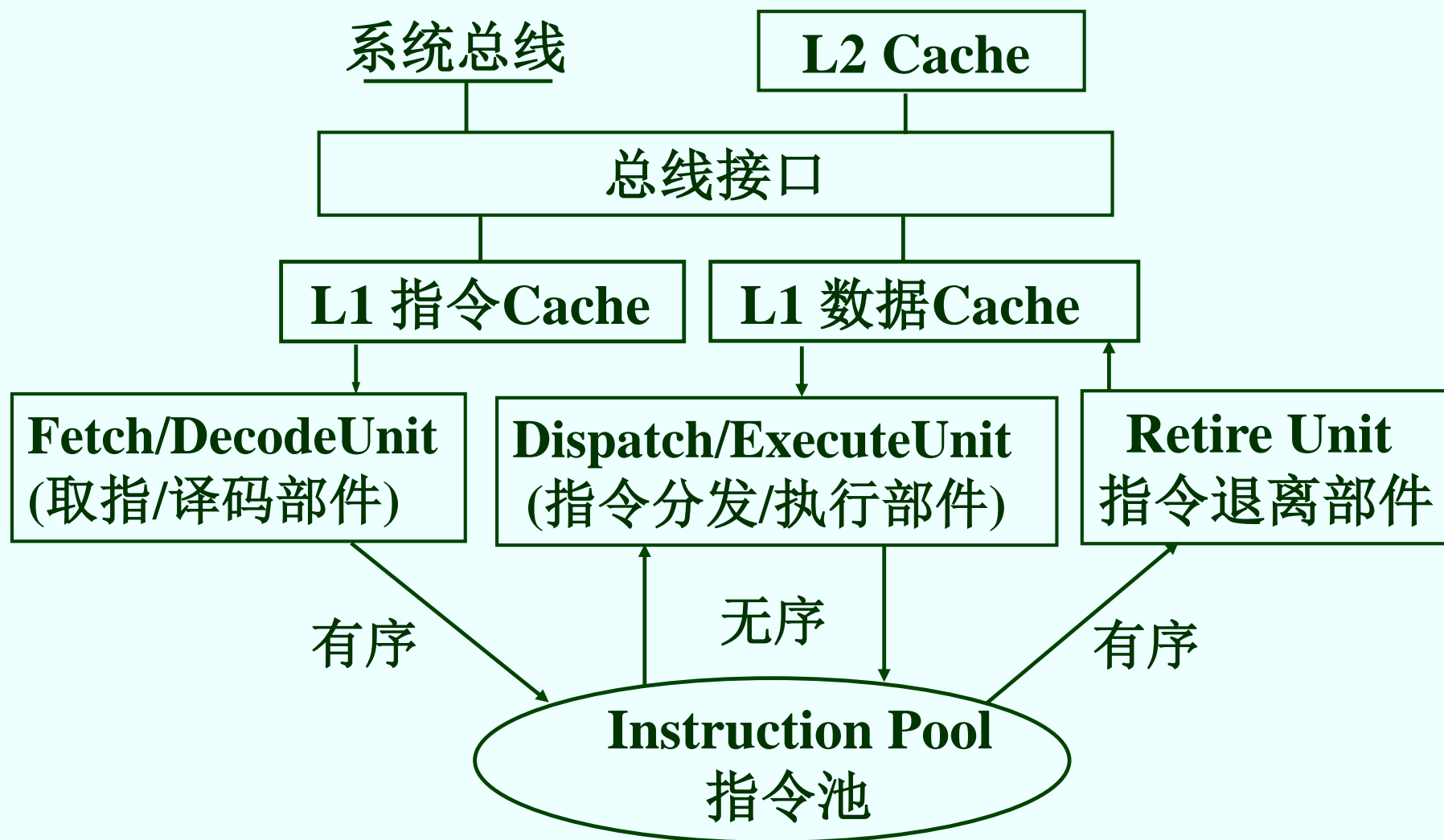
三、 Pentium Pro

在Pentium基本型上的主要改进:

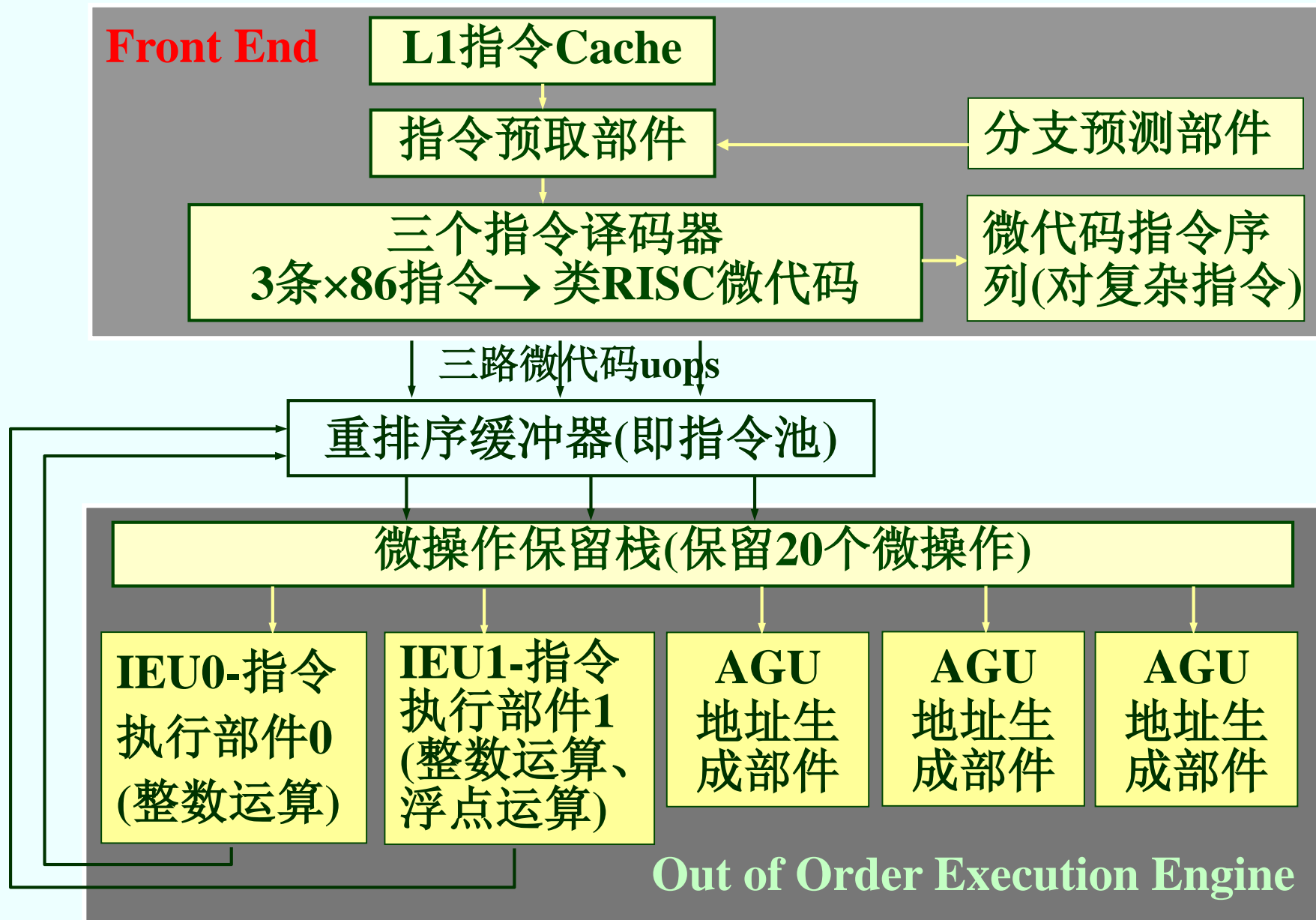
- 将CISC指令集转换为RISC指令集
- 双穴封装技术: 将L2Cache(256K)封装在一个模块中, L1Cache和L2Cache之间64位的独立总线连接
- 指令乱序执行: 分析并重排指令(指令重调度), 优化指令的顺序执行(称为指令流分析技术)。
- 14级流水线
- 寄存器重命名(Register renaming): 数据重新定位到一个内部寄存器(速度与流水线的考虑)



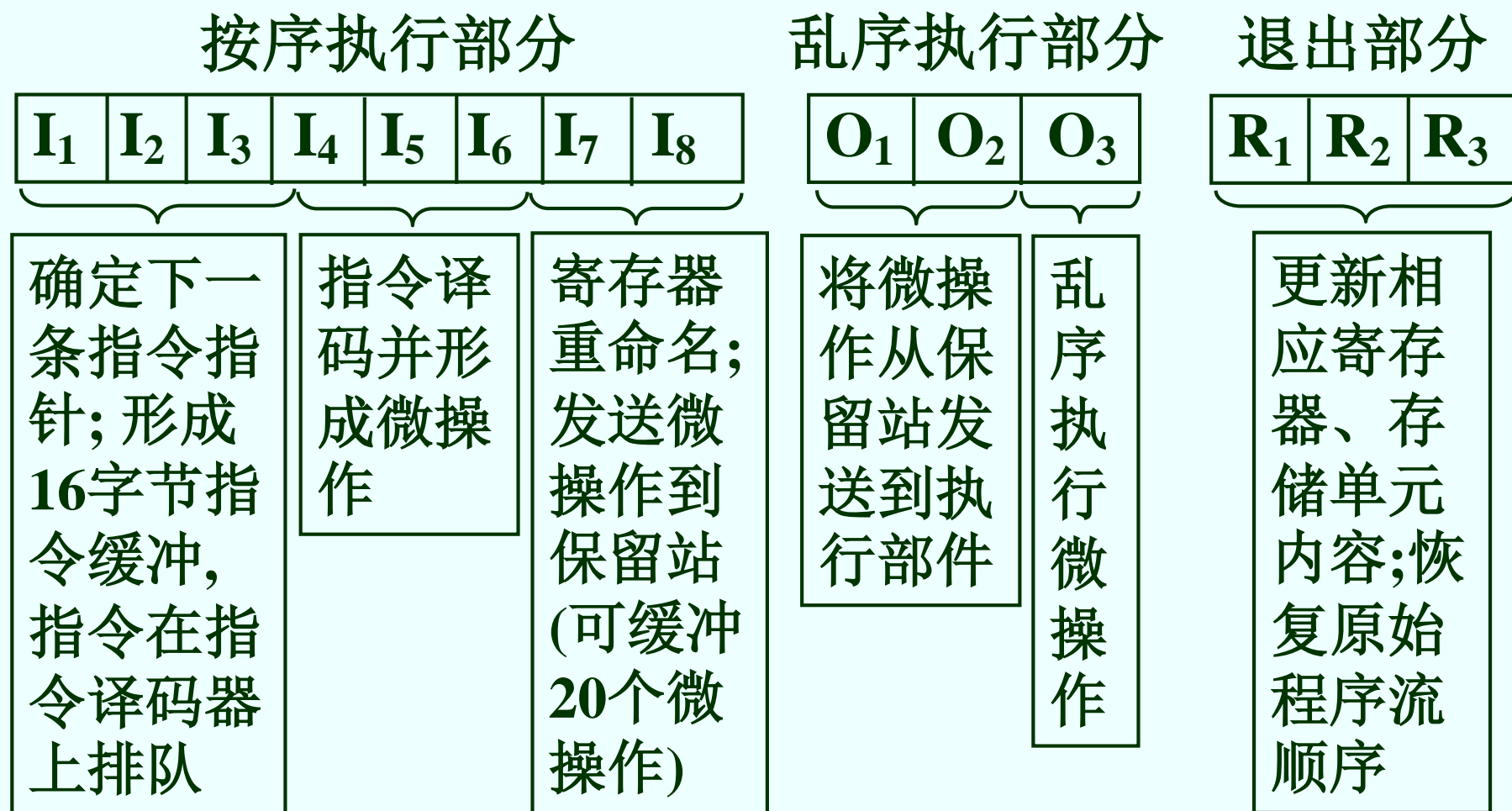
1. Pentium Pro的结构框图



更详细的内部结构:



2. Pentium Pro流水线结构



3. 通过乱序来优化指令流水线

例1: 调整数据相关

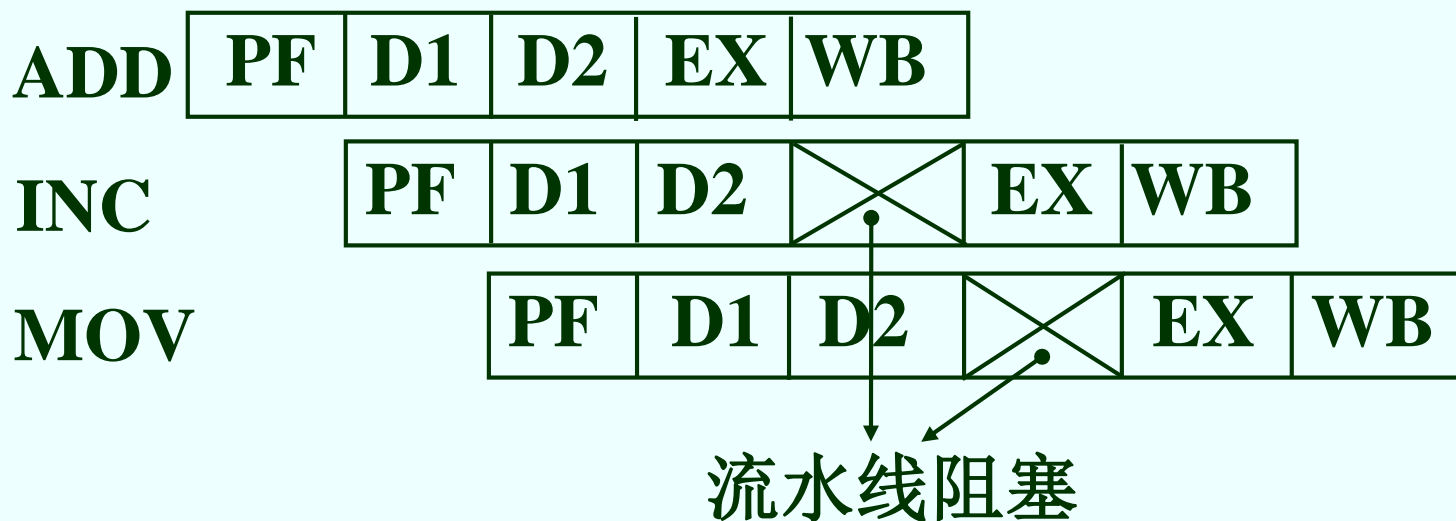
ADD AX, BX ; AX+BX→AX

INC AX ; AX+1→AX

MOV CX, DX ; DX→CX

...

流水线情况:

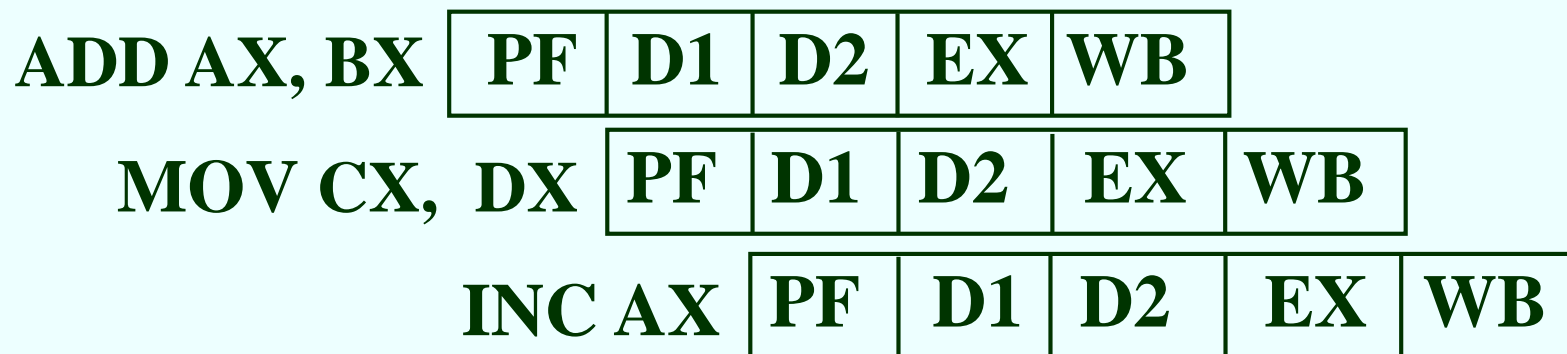


改变指令顺序前:

ADD AX, BX	;
INC AX	;
MOV CX, DX	;
....	

改变指令顺序后:

ADD AX, BX	;	AX+BX→AX
MOV CX, DX	;	DX→CX
INC AX	;	AX+1→AX
...		



流水线无阻塞

即通过非相关指令的执行来填充了流水线的执行部件，从而避免了流水线的损失。

4. 寄存器重命名

优点:

1. 在寄存器数量有限的情况下,可提高指令执行的速度;
2. 在乱序执行的情况下,解决“指令伪相关”。

指令伪相关 – 乱序情况, 两条或多条指令同时向某一个寄存器写数据。

指令真相关 – 前述例子中, 一条指令的输入依赖于上一条指令的输出。

例2: 指令顺序调整结合寄存器(变量)重命名

源程序: 源操作数 目的操作数

ADD R1, R2, R3

ADD R3, R4, R5

MUL R6, R7, R3

MUL R3, R8, R9

$OP \ R_i \ \theta \ R_j \rightarrow R_k$

由于四条指令均与R3有关, 仅通过调整顺序难以优化。因此重新命名一个寄存器(变量)R0

优化后的指令顺序:

ADD R1, R2, R3

MUL R6, R7, R0

ADD R3, R4, R5

MUL R0, R8, R9

将第2条指令与第3条指令交换, 并重新命名一个寄存器R0

四、 Pentium II

- 在Pentium Pro体系结构中引入MMX功能;
- 二级Cache从256K增加到512K;
- 增加了可重命名的段寄存器
- 提高了主频

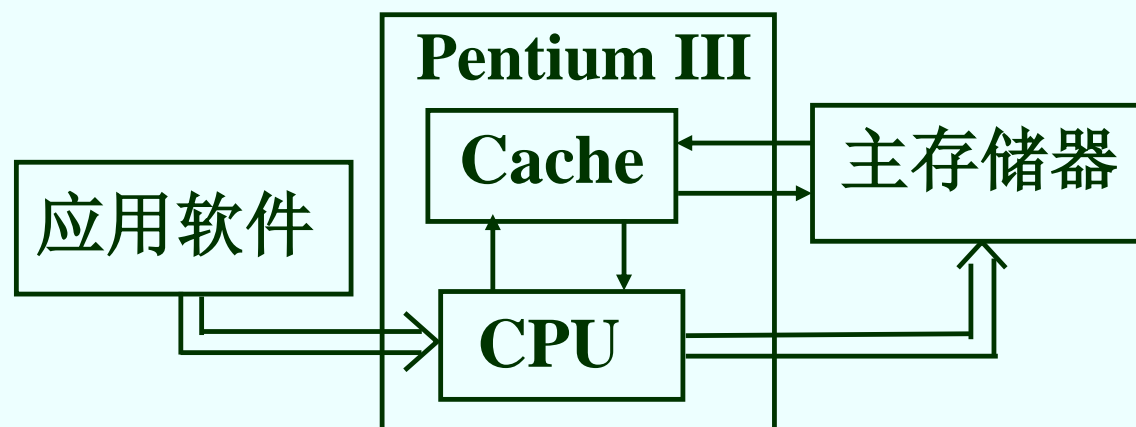


五、 Pentium III

- 增强的SIMD技术: 既能处理整数, 也能处理浮点数;
- 256KB二级Cache集成在芯片核心内(非双穴封装);
- 处理器分离模式: 允许SIMD-FP和MMX并行使用
- 新增SSE指令集(70条指令, Pentium III最重要的技术创新);
(Streaming SIMD Extensions 数据流单指令多数据扩展)

- 50条单指令多数据浮点运算指令
- 12条新的多媒体指令
- 8条存储器连续数据流优化处理指令

该类指令可直接将处理器数据送往存储器,即越过Cache直接传送到主存储器。



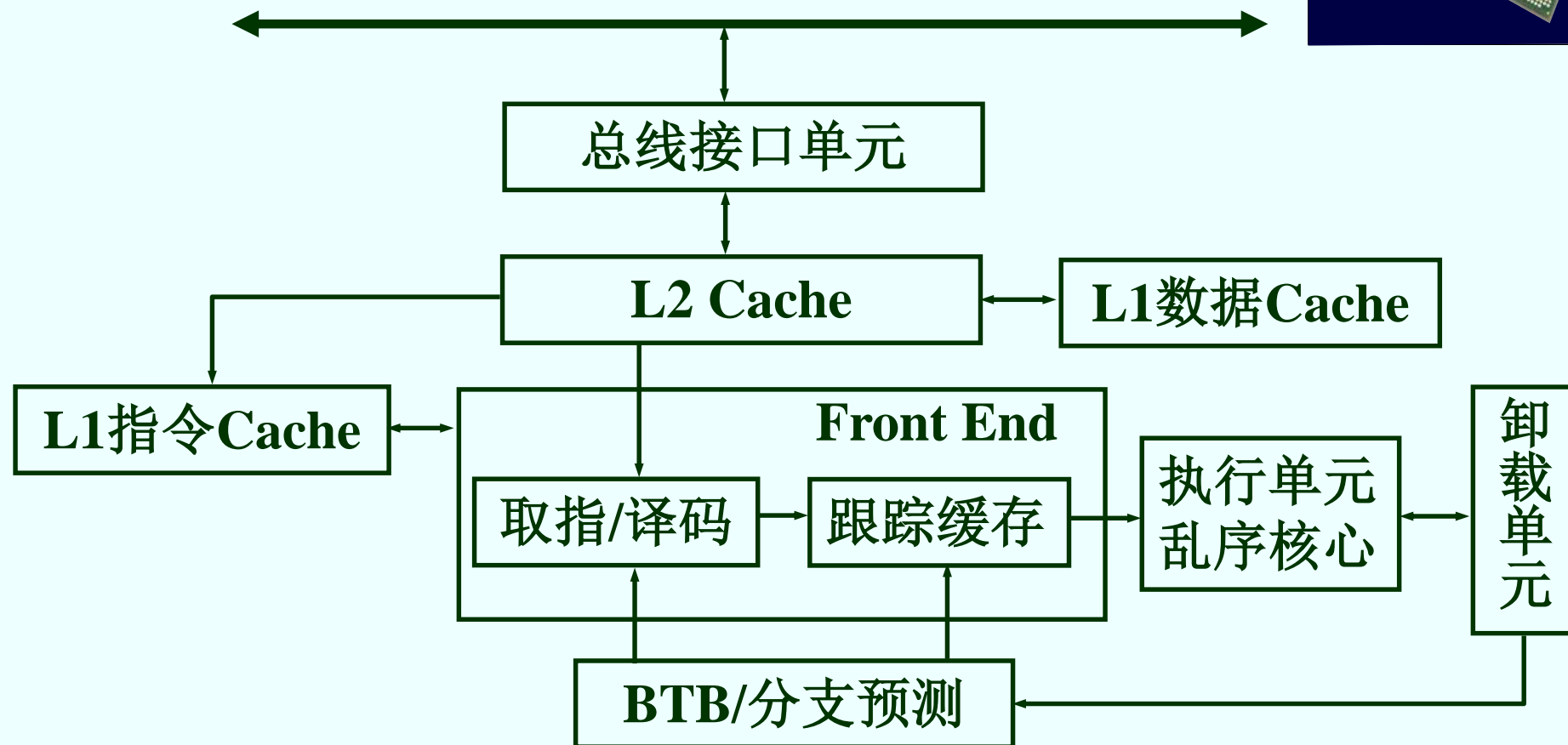
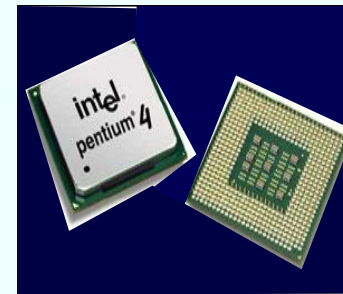
数据流优化处理后Cache控制情况

避免了近期不使用的数据占用Cache空间,或必须清空当前Cache中内容,为新数据流留出空间,提高了Cache的利用率

从PentiumPro到PentiumIII
第二代奔腾处理器(P6架构)

六、 Pentium IV(NetBurst架构)

1. Pentium IV内部结构(简化形式)



2. 主要技术特点

- (1) 增加了144条指令, 进一步增强了对多媒体信息、3维信息、互连网操作的处理能力;
- (2) 主频2G以上, 提高了指令执行的吞吐率;
- (3) 系统总线的速率400M;
- (4) 流水线级数提高到20级;

(5) 高级动态执行(Advanced Dynamic Execution)

可在126条指令范围内, 进行指令重调度, 并配合128个重命名寄存器, 具有更高的预测精度。

与PIII相比, 大约可减少1/3的预测错误。

注: 动态执行一般指集分支预测、指令流分析(即乱序执行)和猜测执行三种技术于一身

猜测执行:

CPU向前查看PC, 并执行后面可能要执行的指令, 执行结果存入缓冲区。如果后面确要执行, 则执行结果有效。其实质是允许CPU预先执行指令, 以后需要时直接获得结果。

(6) 执行跟踪缓存(Execution Trace Cache)

存储已解码的微指令(μ OPs), 以后执行到相同指令时, 不必重复解码, 可直接获取已解码结果并执行即可 (对循环代码段有很高的效率)。

此外, 当分支预测出错, 需回到分支处运行另一路时, 之前的译码阶段已经把另一分支指令缓冲进了 **Trace Cache**, 不用再译码, 可节约1~2个时钟周期。

(7) 快速执行引擎 (Rapid Execution Engine)

采用了称为**Double Pumped**的双重并发技术(即两组 **ALU**), 每个时钟 **ALU** 能执行两次, 效率相应提升一倍, 双 **ALU** 一个周期内可执行4条指令

(8) 超长管道处理技术(超管线技术)

该技术使流水线深度达20级。

但流水线的深度越深,分支预测错误时,损失的指令周期就越多。这也是很多其它许多处理器适当控制流水线长度的原因。

在极端情况下,流水线塞满长延迟、低吞吐量的微操作时,P4会出现100个周期的损失。

(9) SSE2指令集(Streaming SIMD Extension 2)

对原有SIMD的扩展,引入新的数据格式,如:

- 4个单精度浮点数(SSE)
- 2个双精度浮点数(SSE2)
- 16字节数(SSE2) 等

(10) 增加静态分支预测作为动态分支预测的补充

使用静态预测时：

- 如果分支所跳转的地址为逆向跳转，则预测为分支跳转成立，并执行跳转；
- 如果分支所跳转的地址为正向跳转，则预测为分支跳转不成立。

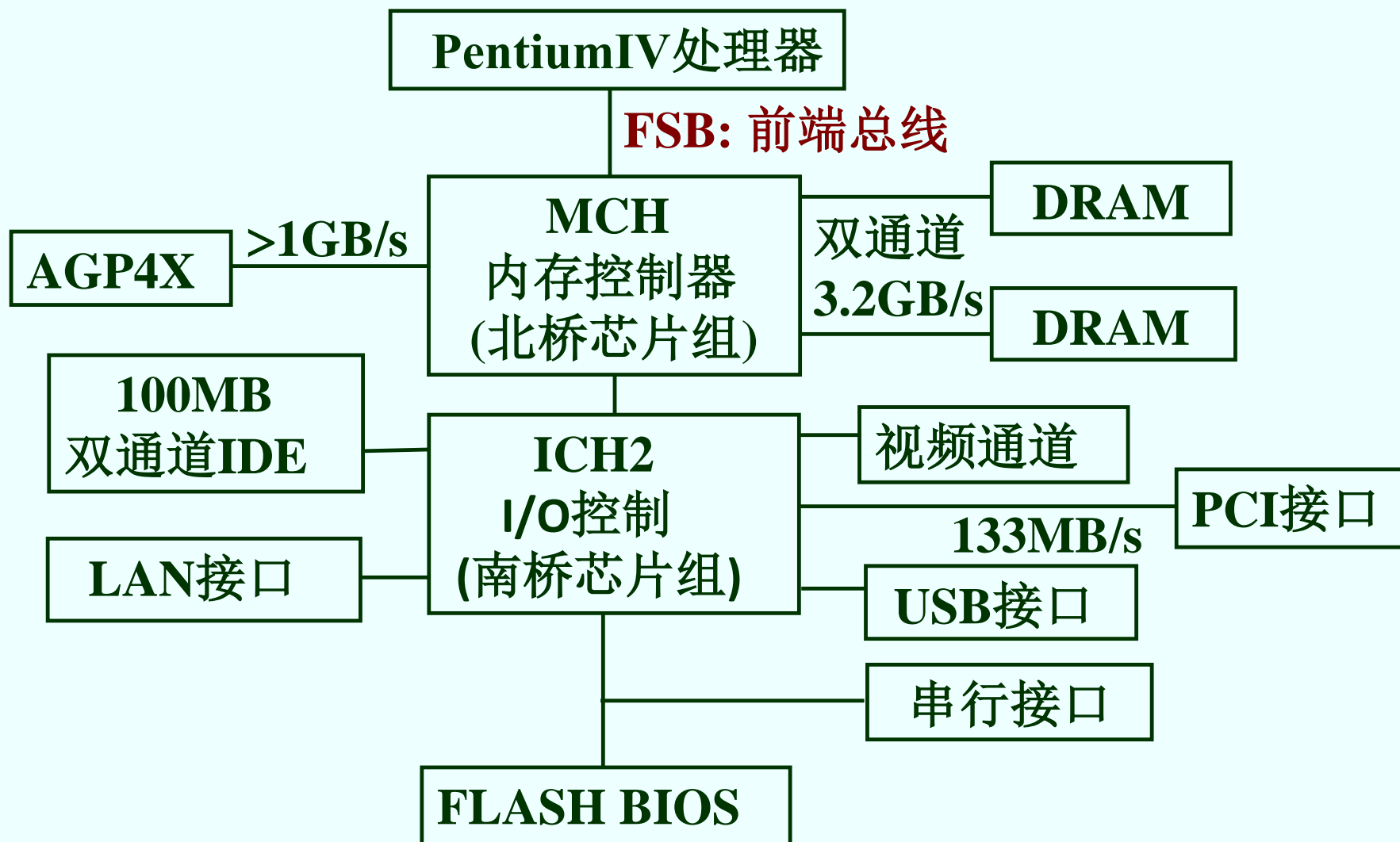
该方式对循环分支较为有效。

(11) 系统总线和内存带宽

前端总线的带宽对系统处理能力有很大影响。

PentiumIV的前端总线通过QDR(Quad Data Rate)技术，使系统总线在1个时钟周期可以四次访问数据，当总线频率为100MHz时，拥有400MHz的效能。

– 基于Pentium IV处理器的硬件系统结构



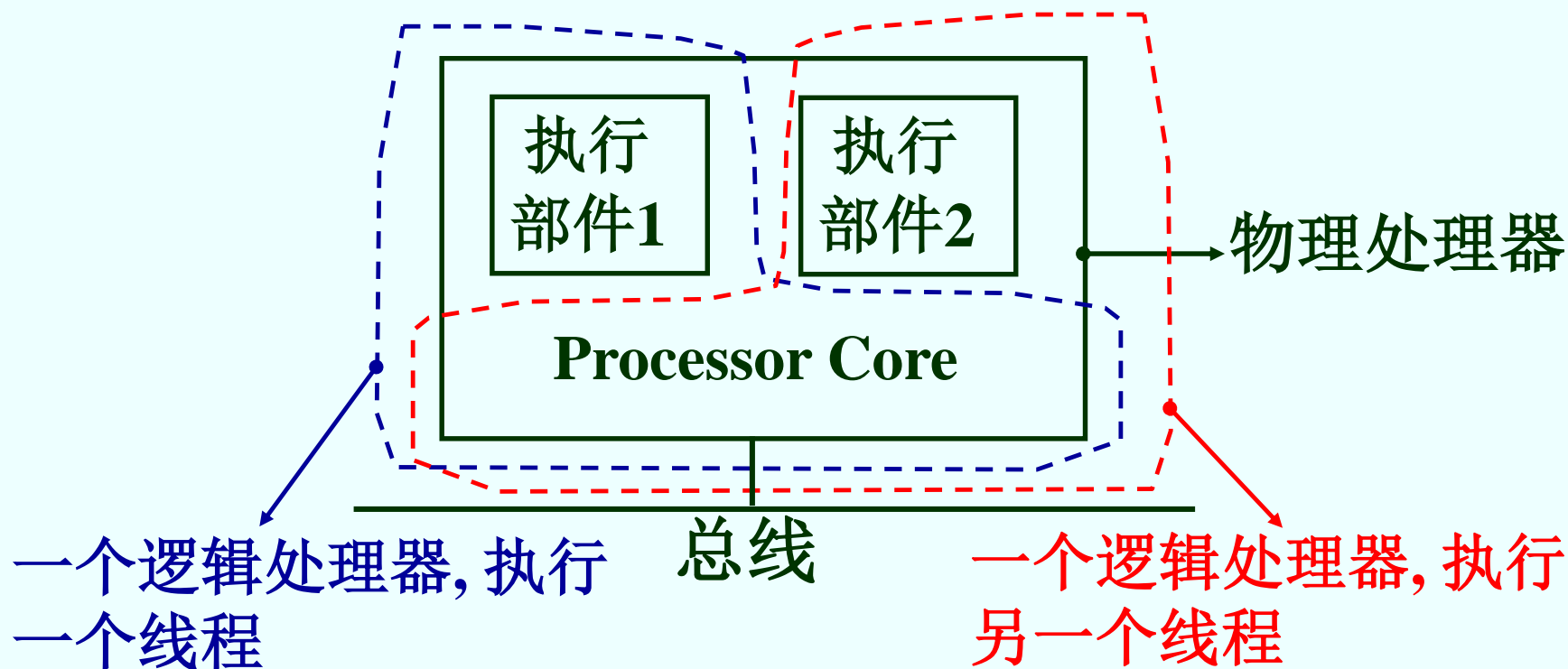
(12) Pentium IV的超线程技术 (3.06G)

- 线程：一段独立的代码流
- 超线程技术(HyperThreading – 简称HT)

利用特殊的硬件指令, 将两个逻辑内核模拟成两个物理芯片, 使单个处理器实现线程级并行计算, 理论上就如两颗CPU在同一时间执行两个线程, 分享一颗CPU内的资源, 减少了CPU的闲置时间, 提高了运行效率。

即: 超线程(HT)技术使一颗物理处理器可以同时处理两段独立的代码流(线程)

如下图所示:



- **Pentium4超线程技术特点:**

- 每个逻辑处理器可以独立执行不同的线程;
- 每个逻辑处理器都可以独立挂起/中断某个的线程;
- 各逻辑处理器共享系统总线接口、BIOS等。

- 新增加了逻辑处理单元(因此具有HT的Pentium4的面积比之前的Pentium4更大)。而其余部分如ALU、FPU、L2 Cache则保持不变(分享)。
- 当两个线程都同时需要某一个资源时, 则其中一个要暂停运行, 并让出资源; 直到这些资源空闲时才能继续。

因此, 当两个线程所要求的相同资源越少, HT技术越能发挥其优势。比如一个线程主要使用整数单元, 另一个线程主要使用FPU。

- **Pentium4 HT的两种运行模式:**

- 单任务模式 (Single Task Mode):**

- 当程序不支持**Multi-Processing**(多处理器作业)时, 系统停止其中一个逻辑**CPU**的运行, 把资源集中到单个逻辑**CPU**中;

- 被停止的逻辑**CPU**处于等待状态仍会占用一定资源。因此**HT**下的**CPU**运行在单任务模式时, 有可能达不到没有**HT**功能的**CPU**性能。尤其在多线程操作系统运行单线程软件时, 超线程技术可能会降低系统性能。

- 多任务模式 (Multi Task Mode):**

- 支持多处理器作业时, 两个逻辑处理器同时工作。

- 超线程技术与超级标量结构的区别
 - 超线程技术以线程为单位来分配逻辑处理器
 - 超级标量技术是以指令为单位将指令分配到指令流水线(Pentium还必须按照指令“配对原则”进行(如两条指令必须是简单指令且在指令Cache内)
 - 超线程技术还必须要有操作系统的支持(特别是要具有识别超线程处理器的BIOS的支持)
- 超线程技术与多处理器系统的区别
 - 多处理器系统无需共享系统总线接口、BIOS等
 - 超线程和超级标量技术都需要共享系统总线接口、BIOS等

七、Pentium 4的后续处理器

1. Prescott 处理器

- 16K一级缓存和1MB二级缓存
- 支持更高级的超线程技术
- 支持SSE-3指令集
- 既可运行64位程序, 也能运行现有32位软件
- 流水线级数由以前的20级增加到了30级(31级)
(称为超长流水线, 是流水线级数最高的处理器)
- 数据预取技术的改进
在数据需要处理之前完成查询缓存, 向内存中取数据, 并将数据预取到缓存中。



- **Prescott**添加了一个单独的整数乘法单元

原**Pentium IV**由浮点单元进行整数乘法运算,给浮点单元增加了负担,而且还需要把待处理的数据在浮点单元与整数单元之间相互传递,占用处理器内部的数据传输带宽。

- 改进了分支预测技术

增加分支目标缓冲器**BTB**的长度(由原来的**512**条增加到**2000**条)。

BTB越大,能够纪录的分支指令数就越多,对于分支指令较密集、分支指令跳转情况变化较复杂的程序,使处理器可以有更高的预测精度;

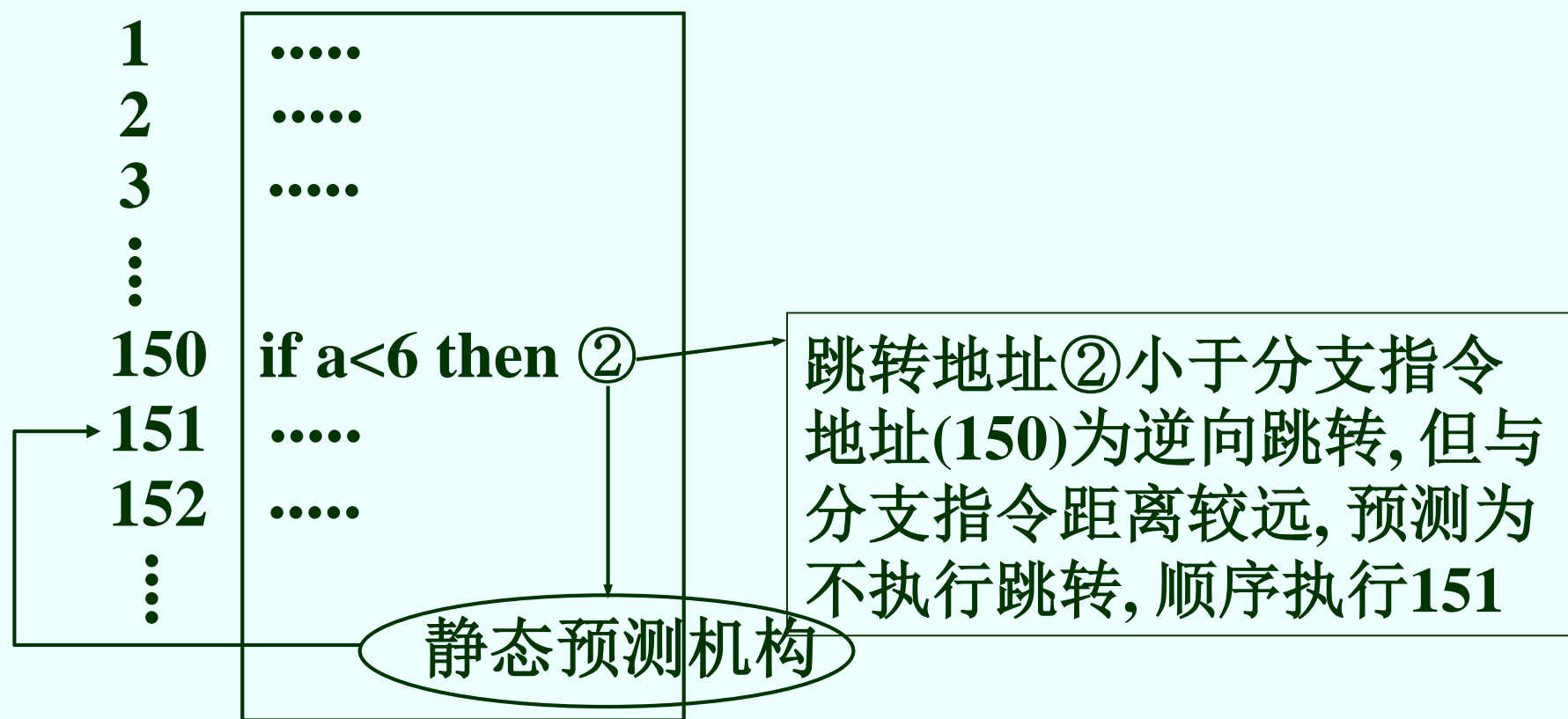
- 改进了静态分支预测技术

Prescott之前的**P4**处理器的静态分支预测方式对非循环分支预测的效率不高。

Prescott在静态分支策略中加入距离判断算法, 仅在跳转的地址为逆向跳转, 且目标地址距离分支指令的地址偏移量较小时, 才预测为执行跳转。

依据是:

多数情况下, 循环体所包含的指令数目都比较小(意味着循环跳转的地址偏移量不会很大)。例:



关于Prescott流水线长度的说明:

在有些实际应用中,证明了处理器流水线的长度与性能成反比。

流水线深度增长的同时,功耗和成本也显著提高。特别是功耗,在 Prescott 处理器上尤为突出。

Netburst架构无法解决这一问题,原计划:

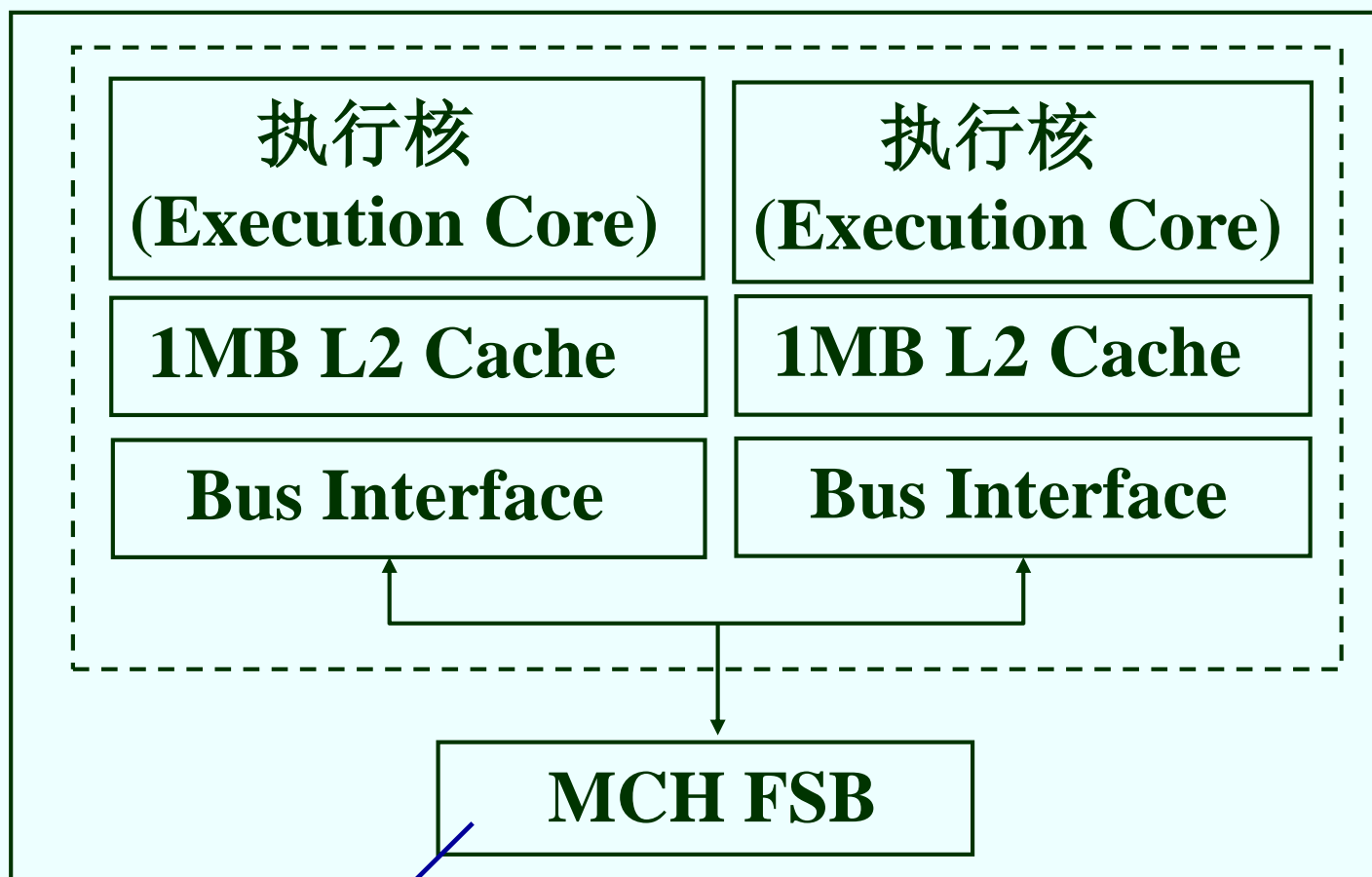
基于Netburst架构的下一代处理器Tejas突破5GHz,但终止步于3.8GHz (英特尔曾计划在2007年推出10GHz Pentium 4) 。

2. Pentium D

Pentium D沿用Prescott架构, 内核由两个独立的Prescott核心组成, 每个核心拥有独立的1MB L2缓存以及执行单元。此外, Pentium D与Prescott处理器的功能几乎完全相同。

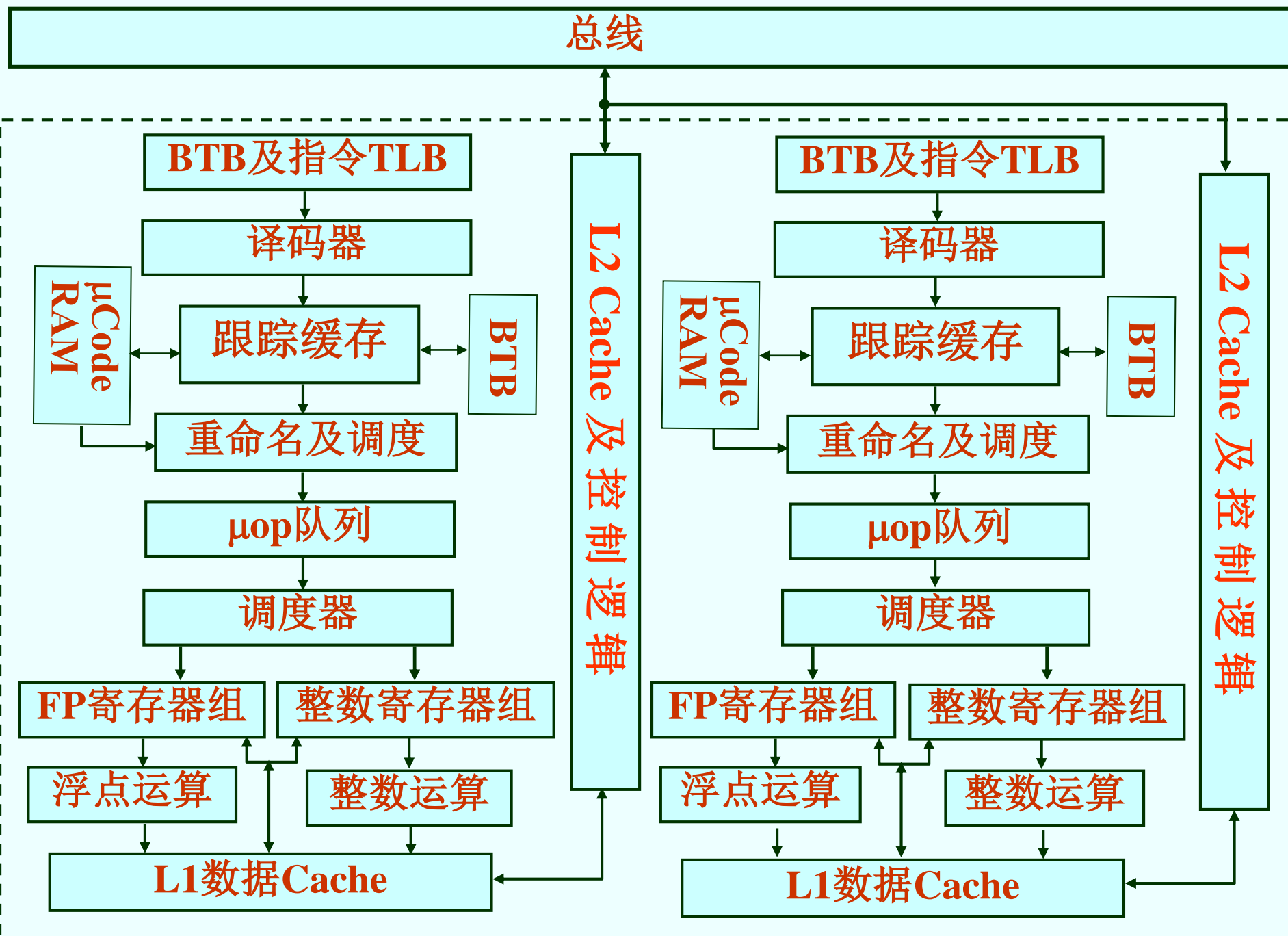
由于随着线程数量的增加, 要使多线程并行执行的难度将会越来越大, HT技术的优势难以发挥。双核技术则还将进一步加大硬件结构的复杂性。因此Pentium D取消了对HT技术的支持。

Pentium D 内部框架结构:



Pentium D 内部框架结构

兼完成两个L2 Cache的一致性工作



Pentium4、 Prescott、 PentiumD NetBurst体系结构

CORE架构的出现终结NetBurst

附：“多处理器技术”

MP技术(Multi-Processing)使用多个CPU协同并行处理，每个CPU在执行队列中选取一个线程执行。

MP系统依据资源共享方式，有多种类型：

1. 松耦合的多处理器技术

通过网络互联与通信，建立多处理器系统架构。不同的处理器称为节点，可共享文件。但不共享CPU和内存，利用软件控制任务分发和并行处理。

2. 紧耦合的多处理技术

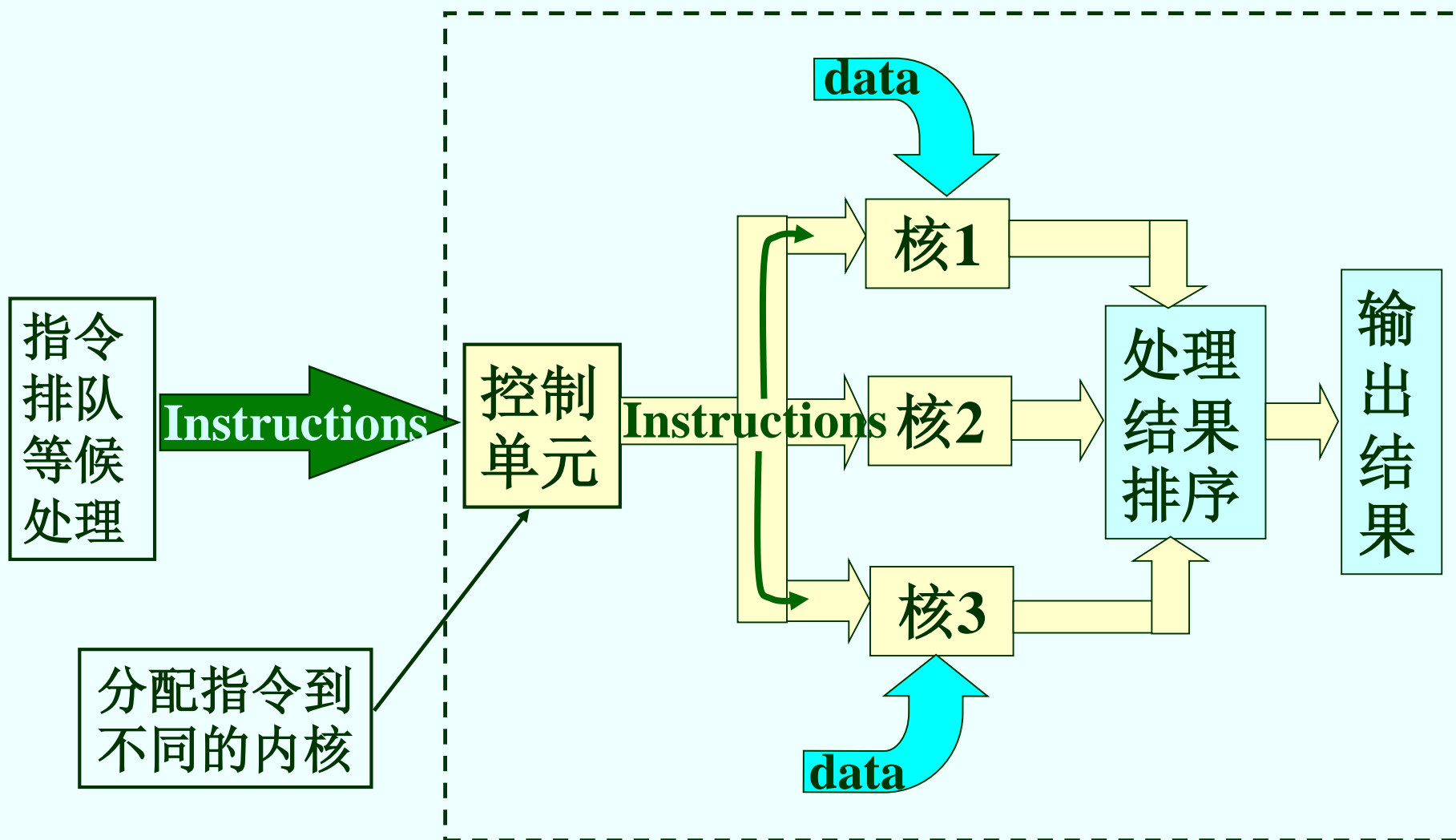
各CPU有自己的独立内存和总线结构，同时共享部分内存，由操作系统控制不同处理器之间协同工作，一般采用“主处理器+从处理器”结构。如果主处理器不能工作，则由某一从处理器升级为主处理器。

3. 对称多处理技术(SMP)

各CPU无差异, 平等地访问内存、外设等。操作系统管理一个任务队列, 每个处理器依次处理队列中的进程。若两个处理器同时请求一个资源(如同一段内存地址), 由硬件或软件的锁机制协调资源竞争问题。

- SMP需要协调各个处理器之间的工作, 因此当处理器数量很多、或任务的并行度差(任务间有复杂的逻辑关联)时, 性能提高的比例递减(即线性度不好)。
- 线性度与CPU设计、操作系统架构及任务类型关系密切, 每个计算机厂商都有自己的理论来支持自己的设计体系, 没有一种理论占绝对优势。

将SMP集成到同一芯片内, 各个处理器并行执行不同进程, 进而形成CMP(Chip multiprocessors)

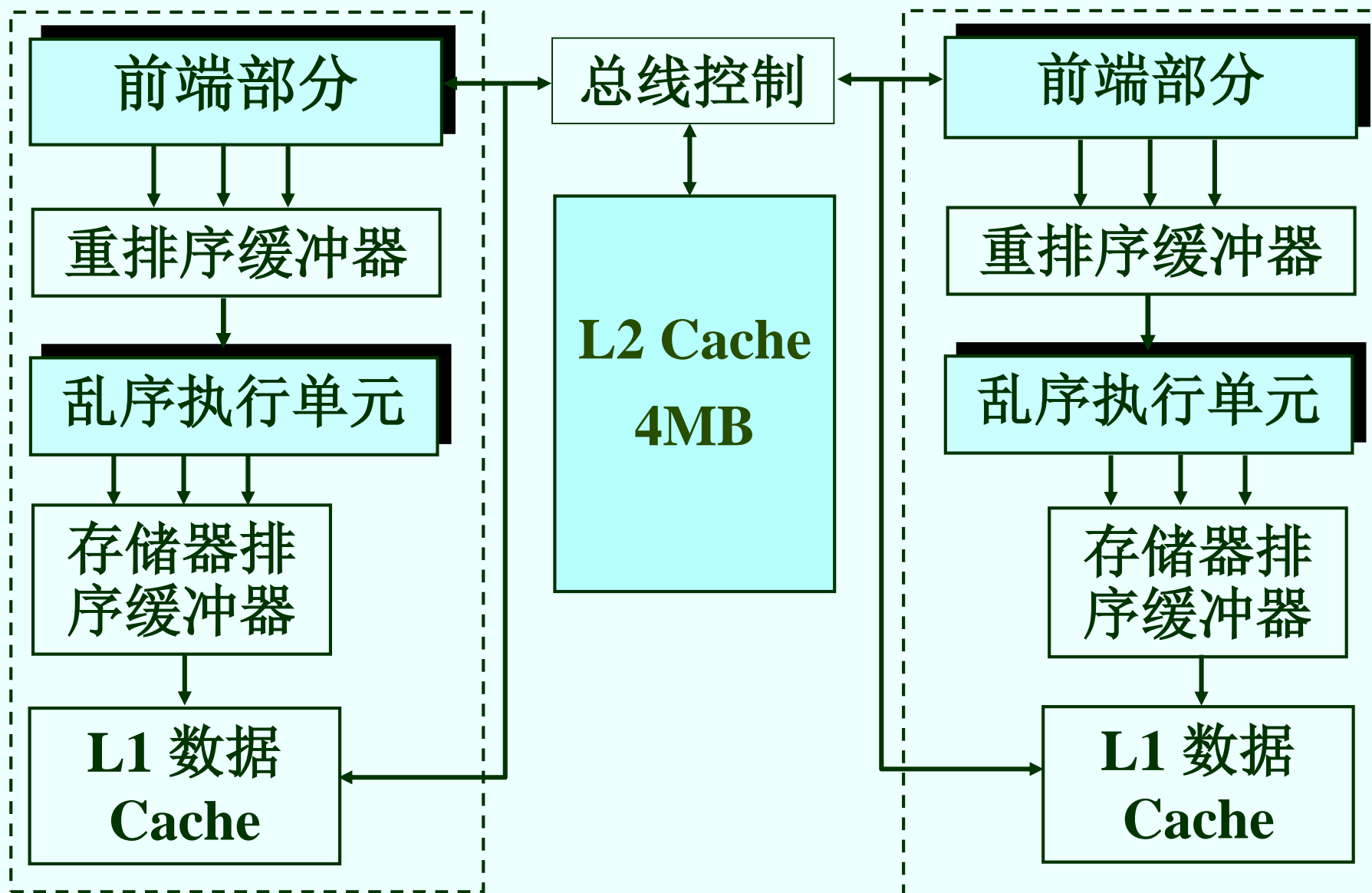


第六节 CORE系列处理器

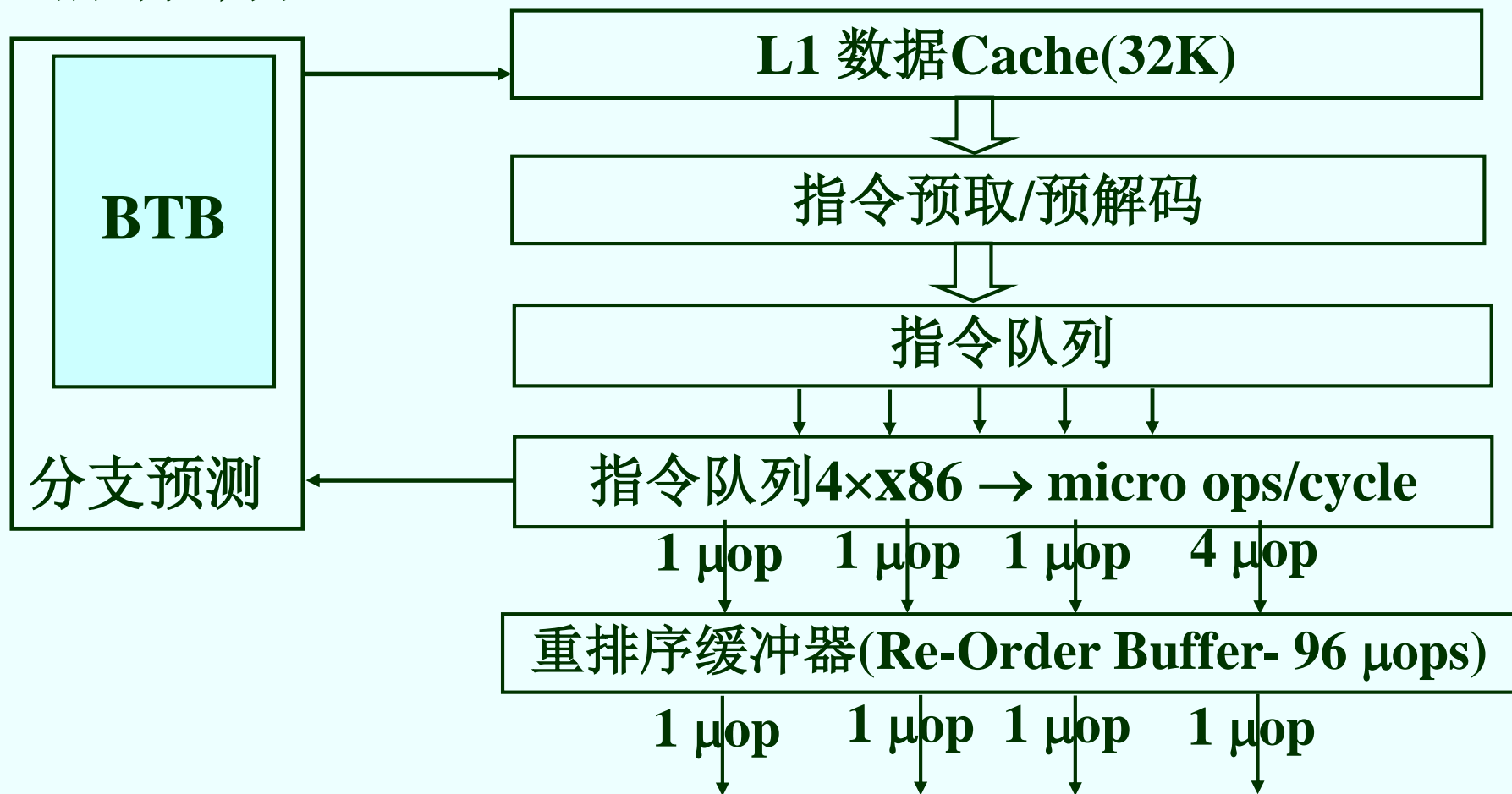
一、CORE2 处理器 (酷睿2)

- 双核结构
- **Wide Dynmaic Execution** — 4组Decoder, 提高每个周期执行指令的数量
- 双核共享L2 Cache, 减少了共享FSB的时延
- 每颗核心拥有3个预取器(2 Data、1 Instruction), 以及2个L2预取器
- 128位整数SIMD及128位浮点 SIMD
- 1066MHz的FSB
- 改进了功耗管理

CORE 2内部结构



1. 前端部分



(1) 预解码

预解码成可执行的微操作, 主要任务是查找符合某种配对条件的两条指令, 将其合并成一条指令。合并后以一条微指令方式发送, 从而减少微操作数。

比如: 指令**CMP** 和一条**JNE**指令(即比较后判断是否转移), 合并成一条指令。在大部分程序中, 约每10~15条指令会出现1组可以合并的指令。

据估计, 对于含有大量条件转移的程序, 经上述预解码后, $(\mu\text{-ops}/\times 86) < 1$, 即微操作指令的数量少于 $\times 86$ 指令的数量。

该技术称为 “Macro-Fusion技术(宏融合技术)”

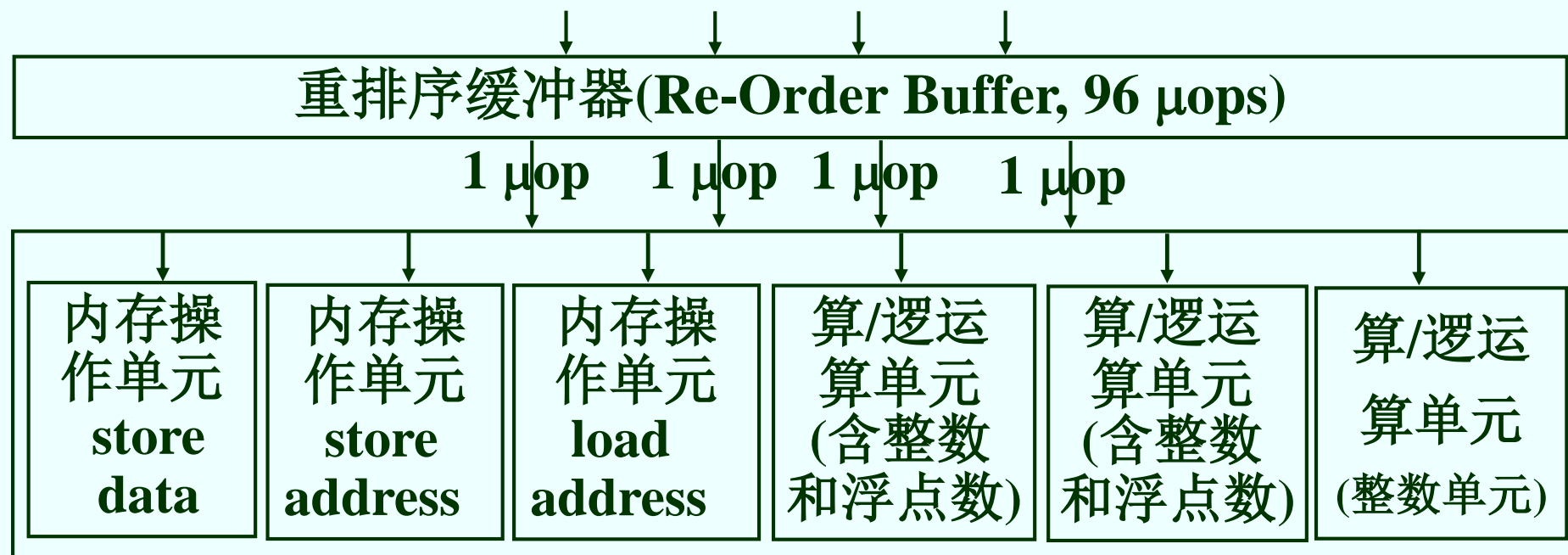
(2) 解码器

具有四个x86指令解码器, 包括三个简单x86指令解码器和一个复杂x86指令解码器。

简单x86指令解码器在每个周期将一个x86指令转换成一条微操作指令;

复杂x86指令解码器每个周期完成2~4条微操作指令(视指令复杂程度), 构成所谓的4+1+1+1解码模式, 即所谓“Wide Dynmaic Execution技术”。

2. 乱序执行单元

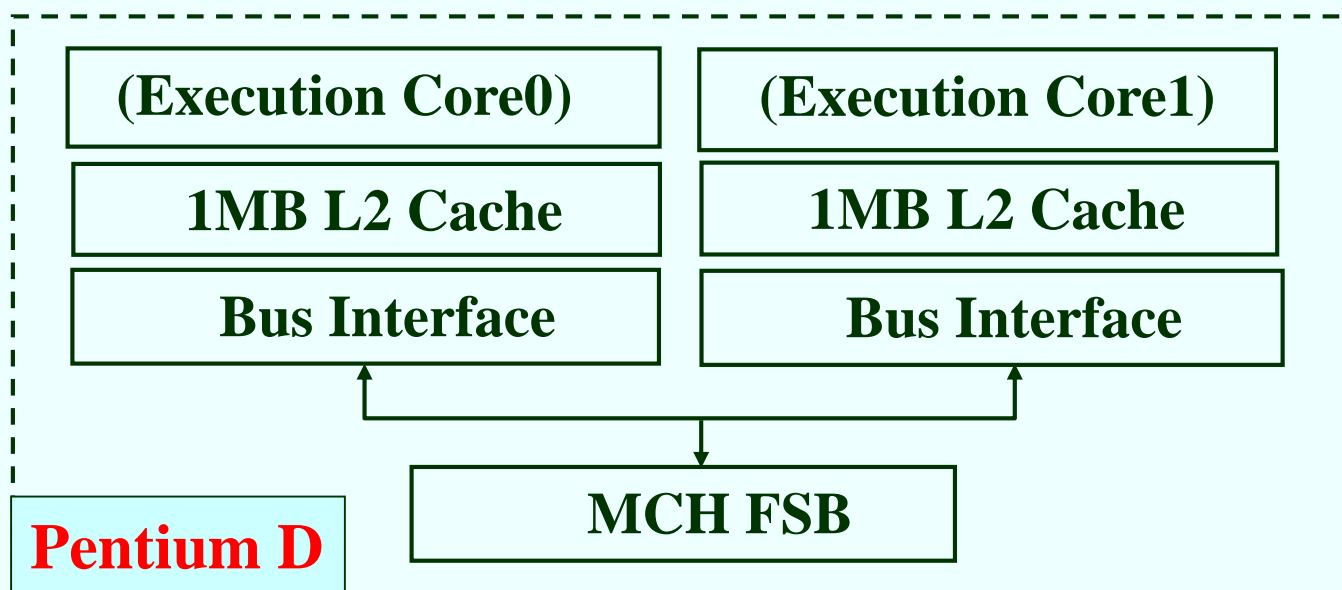


- (1) 三个算/逻单元在理想情况下每一个单元都能在一个周期完成一条简单的64位整数(例如整数加法)指令, 但只有一个单元能执行整数乘法运算;
- (3) 具有比Pentium 4更高效的“Re Order Buffer”;
- (4) 14级流水线。

其它特点:

说明: Pentium D(第一代双核)是将两颗核封装在一起, 共享一个FSB带宽。当某一颗核心要使用FSB时, 需等待另一颗完成; 还需要通过北桥来读取系统内存数据, 并实现双核之间的通信, 加重了FSB负载。

根本原因: 两颗核心没有直接沟通渠道, 如果核0的L2 Cache需要读取核1的L2Cache, 则需经过FSB及北桥才能实现, 从而出现严重的延迟。



(1) Advanced Smart Cache结构

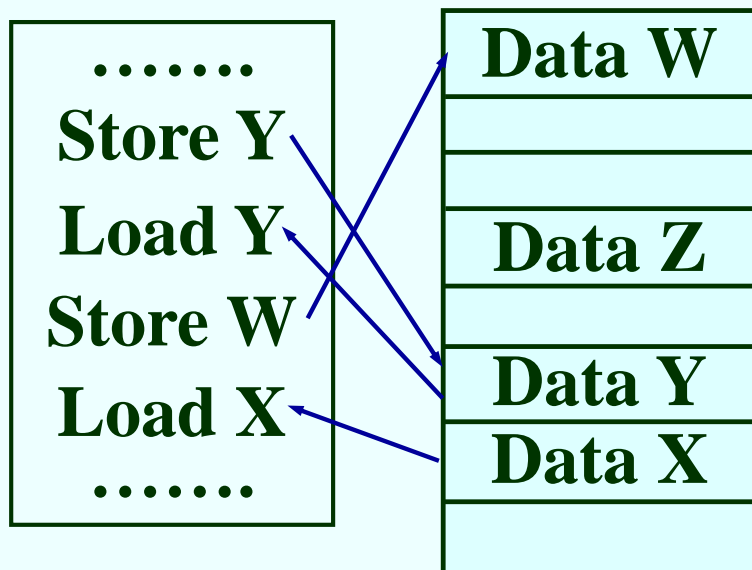
通过核心内部的Shared Bus Router(共享总线路由)共享同一L2 Cache,当“核1”把运算结果存在L2 Cache时,“核2”可通过Shared Bus Router读取“核1”存放在L2 Cache中数据,减低读取的延迟并减少使用FSB带宽。

(2) Advanced Digital Media Boost技术

拥有128Bit-SIMD 整数及128bit SIMD双倍精准度浮点操作能力。之前的处理器只有64位的SIMD整数及浮点操作能力,在执行128位的SSE、SSE2及SSE3指令时,需要将指令分拆为两条64位指令,用两个时钟周期完成,但Core2只需要一个时钟周期即可,执行效率提升达一倍。

(3) 存储器消歧 (Memory Disambiguation)

对内存读取顺序进行分析, 可以让多个load(取数)和store(存数)操作同时执行。比如程序段:



指令**Load X**与之前的数据存取无关, 但也必需要等待**Store 1**、**Load 2**及**Store 3**工作完毕才能执行。

因为处理器不知道前面的指令是否会改变**Data X**的数值, 所以不能提前执行。

存储器消歧技术提供一种智能分析机制, 可以检测是否会发生冲突并决定是否提前读取数据。如上例中能预知**Load X**的**Data X**是独立, 可提前执行。因此减少了处理器等待时间。

(4) Intelligent Power Capability

- 工艺上

采用先进的65nm Strained Silicon(应变硅)等技术、相比上代90nm工艺减少漏电达1千倍。

- 设计技术上

Sleep Transistors技术: 能控制独立开关各运算单元, 需要时开启, 闲置时关闭(避免了不必要的功耗)。比如, 系统负载很轻时, 可以关掉一个处理器。

部分部件采用独立供电方式, 当这些部份部件被闲置未用时, 被置为低功耗模式。

测试结果:

Core体系结构遥遥领先于最接近的对手的整数性能, 浮点性能已经接近于**Itanium 2**和**IBM的Power 5**。

SPEC INT2000测试(对整数), **Core 2 Duo 6700**性能相当于**1.81倍AMD Athlon 64 X2 2.66GHz**的性能;

SPEC FP2000测试(对浮点数), **Core 2 Duo E6700**相当于**1.74倍于AMD Athlon 64 X2 2.66GHz**的性能。

二、Core 四核处理器 – Kentsfield架构

Core 四核处理器仍基于Core微架构, 把两颗酷睿双核心封装在一起。当CPU的2个核心互访时, 分属两个核心的L2缓存(4MB)需要通过前端总线和北桥芯片实现通信, 其设计与奔腾D相似, 存在着明显的瓶颈, 影响了CPU执行的效率。

执行标准软件时, Kentsfield的测评成绩与主频相同的双核处理器基本相同。

三、Core i7四核处理器(Nehalem架构-45nm)

1、微架构的改进

- (1) **Cache**:片内三级Cache,各核心各拥有256KB的L2;片上所有核心共享L3(8MB)。
- (2) **快速通道互联(QPI-Quick Path Interconnect-带宽5倍于FSB)**:取代前端总线。被认为是该架构的最大改进。
- (3) **同步多线程技术(SMT)**:每核心可同时执行2个线程。
- (4) **完整SSE4指令支持**:完整的SSE4指令集,增强了视频编解码/图形处理以及游戏等多媒体应用性能。
- (5) **自动加速技术(Turbo Core技术)**:
可动态提升内核频率来提高性能,并保持运行在限定的功耗/电流/电压和温度范围内(非超频)。

自动加速技术的工作过程:

1. 以1000次/s的速度自动检查处理器以下状态:
 - (1) 处于运行状态的内核数量是否发现变化: 4个内核或3个或2个还是1个处于运行状态;
 - (2) 是否达到处理器散热设计功耗的上限;
 - (3) 是否达到内核电流的上限;
 - (4) 是否达到处理器芯片表面温度的上限;
 - (5) 处于运行状态的内核是否任务繁重;

2. 确定完成的操作

比如, 如果发现仅需要两个核心即可胜任, 则关闭另外两个核心, 使其有提高运行频率的空间。

处理器随时响应操作系统的高性能运算状态的请求, 如操作系统要求处理器处于高性能运行状态(P0状态), 处理器在接到请求后, 随即提高内核运行频率。对于用户来说, 加速技术是通过底层硬件来实现的, 可通过BIOS设置来进行开启和关闭。

比如:

以Core i7 870为例:

默认主频2.93GHz, 如果仅一个内核处于运行状态, 该内核可提速至3.6GHz, 增加了 $666\text{MHz} = 5 \times 133\text{MHz}$ (相当于上5个台阶, 一个台阶为133MHz)。

如果仅2个内核处于运行状态, 则可提速至3.46GHz, 相当于上了4个台阶, $533\text{MHz} = 4 \times 133\text{MHz}$ 。

如果3个或4个内核处于运行状态, 可提速到3.2GHz, 相当于上了2个台阶($266\text{MHz} = 2 \times 133\text{MHz}$)。

四、Core i7 六核处理器(32nm工艺)

- 六个核心集成在一个内核中, 11.7亿个晶体管
- 三级缓存容量扩大到了12MB ;
- 集成内存控制器;
- 超线程技术最高线程数提高到了12个 ;
- 自动加速的核心数量提高到了六核。

- AMD在2009年推出了6核皓龙处理器;
- IBM在2010年, 推出了8核Power 7芯片;
- Intel在2010年, 推出了八核Nehalem-EX处理器;

同构处理器可能出现的极限问题

按照Amdahl定律: 即使能够增加同类型的内核来提升并行处理能力, 但整个系统的性能仍然会受软件中必须串行部分的制约。

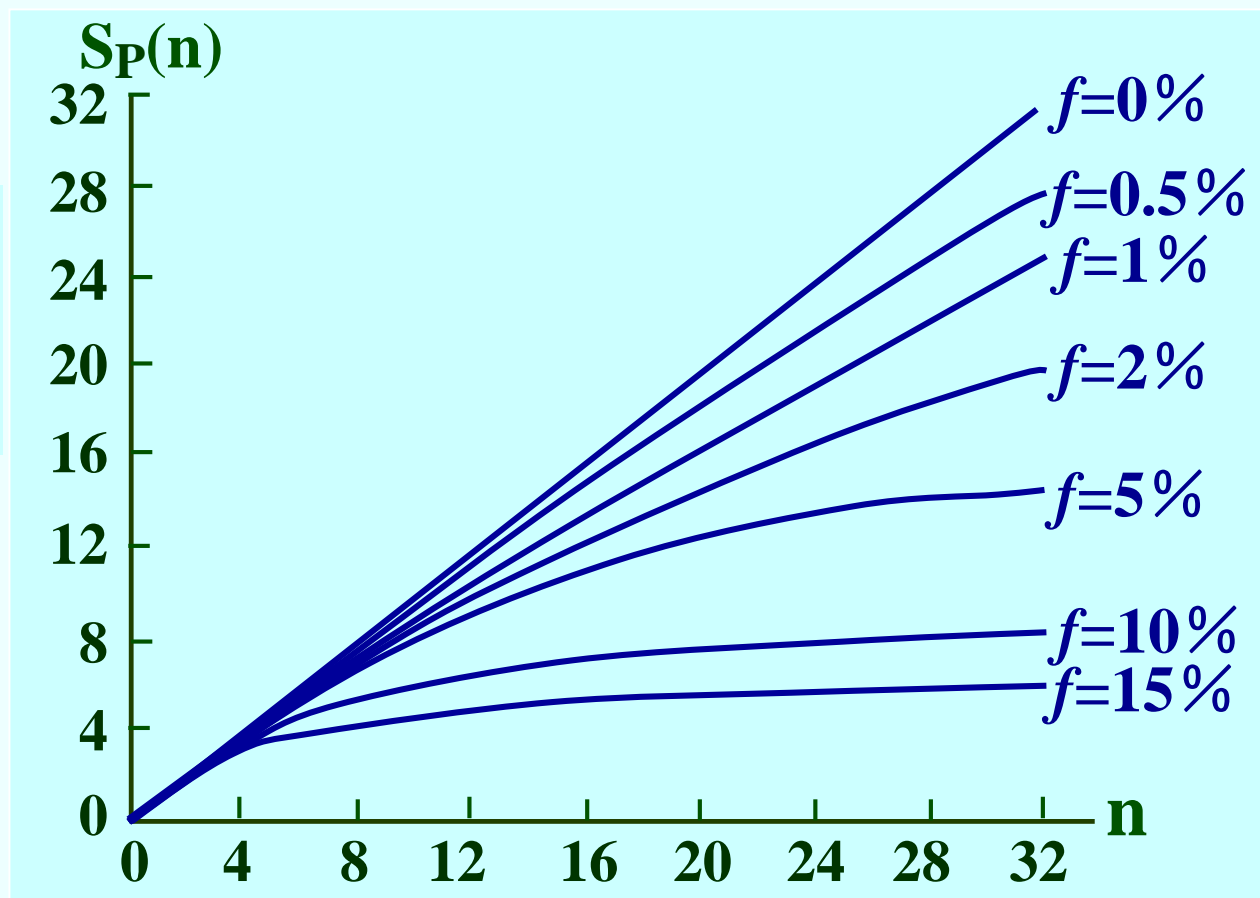
假设程序在单处理器上执行时间为 t_s , 程序中不能分解为并行计算的部分比例为 f , 那么程序在 n 个处理器(或 n 个内核)上执行时, 串行部分所占的时间为 $f \times t_s$, 并行部分的执行时间为 $(1-f) \times t_s / n$, 由此得到加速比:

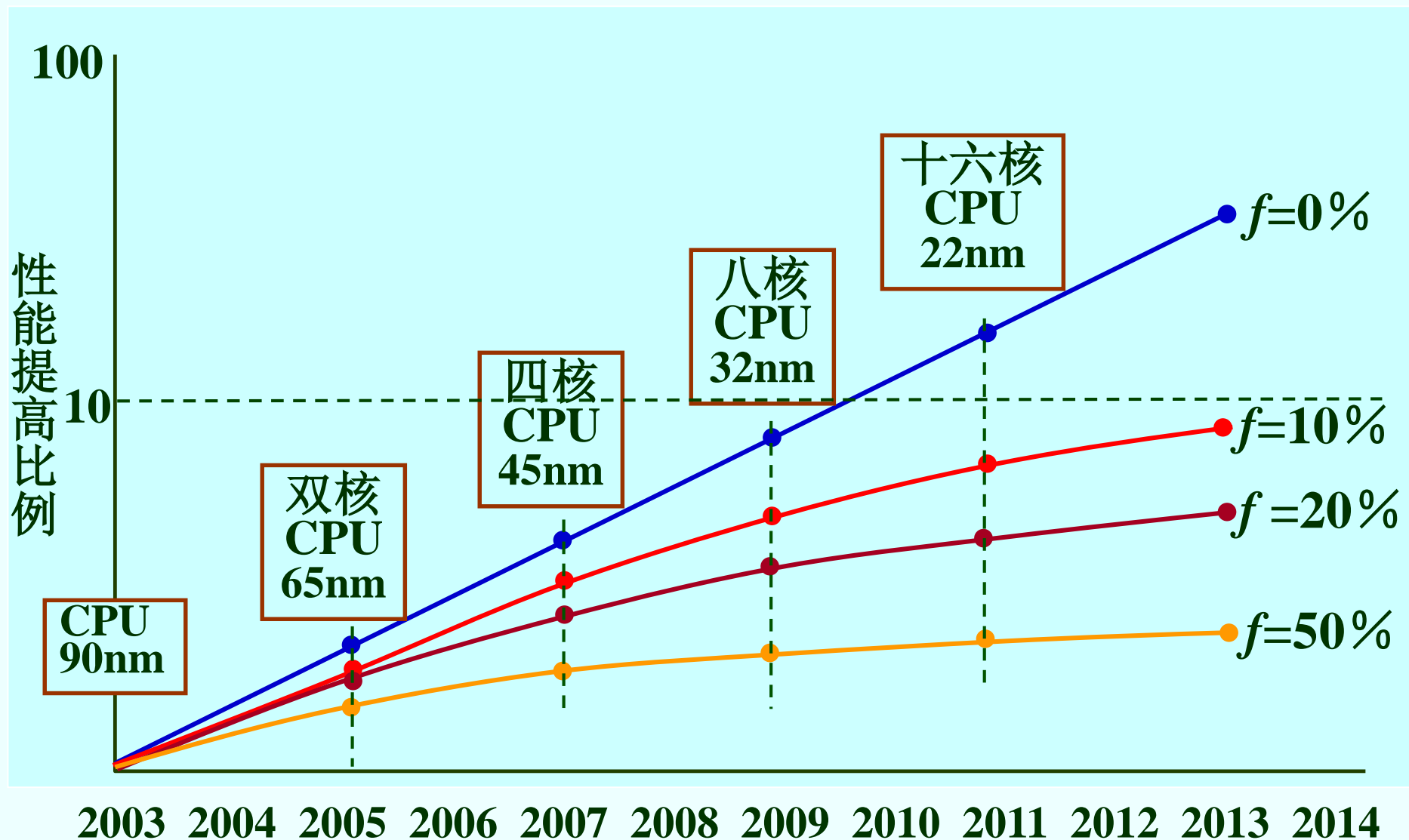
$$S_P(n) = \frac{t_s}{f \times t_s + (1-f) \times t_s / n} = \frac{n}{1 + (n-1)f}$$

显然: $S_P(n) = \frac{n}{1+(n-1)f} < \frac{1}{f}$ (即: 加速比不会超过 $1/f$)

并且: $\lim_{n \rightarrow \infty} S_P(n) = 1/f$ (加速比与处理器或内核数量无关)

f : 程序中不能分解
为并行计算的部分
所占比例





根据: $S_p(n) = \frac{n}{1+(n-1)f} = 1/(f+(1-f)/n)$

如: 假设软件必须串行执行的部分为50% ,内核由1个分别增加到4个、8个、16个、32个, 则有:

性能提高的比例= $1/[0.5+(1-0.5)/4]=160\%$

性能提高的比例= $1/[0.5+(1-0.5)/8]=178\%$

性能提高的比例= $1/[0.5+(1-0.5)/16]=188\%$

性能提高的比例= $1/[0.5+(1-0.5)/32]=194\%$

可看出:

- 内核数量成倍增加, 但性能提升逐步递减;
- 内核数量由1增到32, 性能还未能提升1倍;
- 还可以计算出, 在同样条件下, 若希望性能提升199%, 则需要约200个内核。

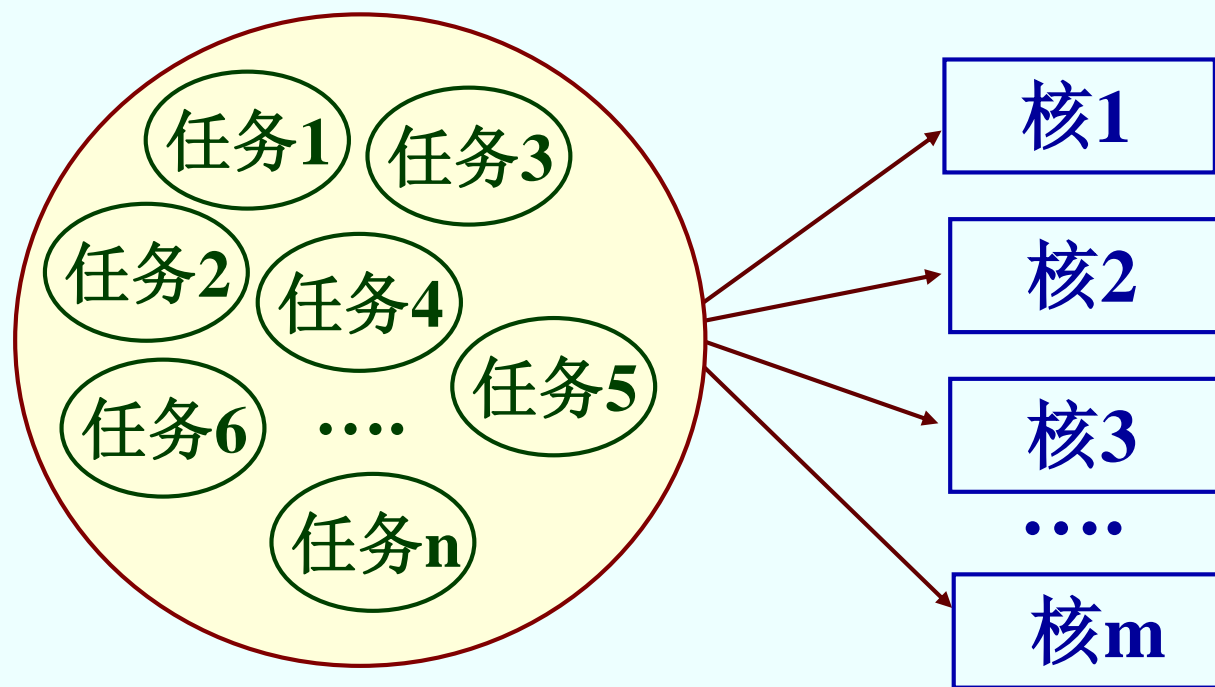
如果再考虑：

- (1) 用于保持高速缓存一致性所需的同步时间；
- (2) 多个内核集中访问内存时的等待时间等；
- (3) 多个内核访问其它共享资源时的互斥所需时间等；
- (4) 不同核之间的通信时间

系统整体处理能力的提升还更低。

* 多核处理器的任务调度

即如何将若干的任务(进程)分配到不同的核上？

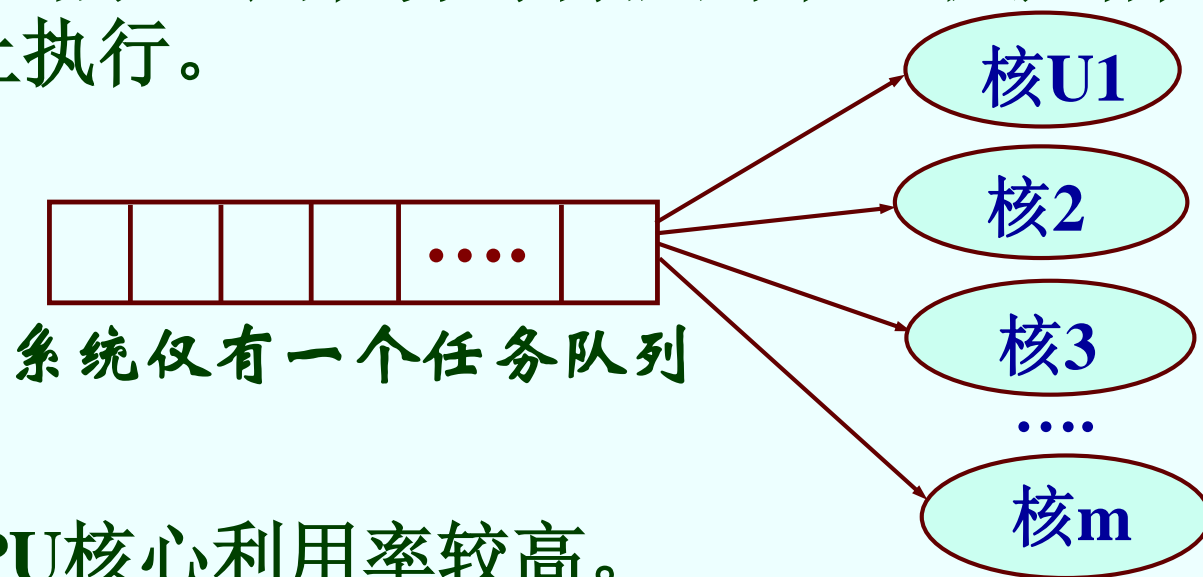


主要有两种任务调度模式：

全局队列调度 和 局部队列调度算法

全局队列调度

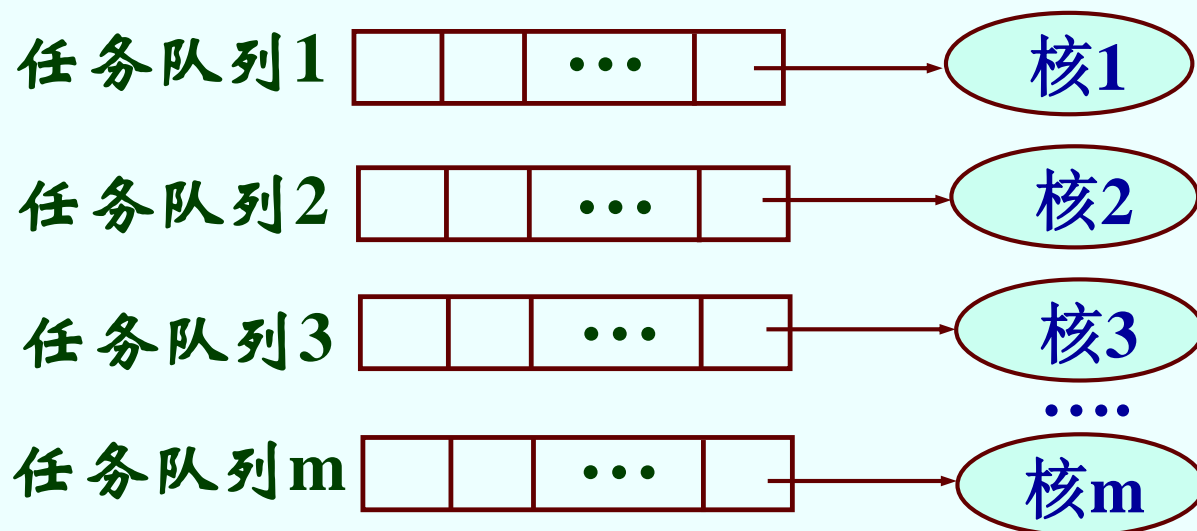
系统维护一个全局任务等待队列, 当有核心空闲时, 按某种策略从全局任务等待队列中选取就绪任务投入该核心上执行。



优点: CPU核心利用率较高。

缺点: 需要计算各CPU负载情况且任务间的通信开销大

局部队列调度



为每个内核维护一个局部的任务等待队列, 当有核空闲时, 从该核的任务队列中选取任务执行。

优点: 任务基本上无需在多个CPU(内核)间切换,
通信量少且无需关心其他CPU负载情况

缺点: 各核负载不均衡

目前多数多核操作系统采用基于全局队列调度算法。

近年来, GPU(图形处理单元)已成为高速发展领域。

GPU的控制相对简单, 对Cache的需求小, 所以大部分晶体管可以组成各类专用电路、多条流水线, 使其计算速度有了突破性的飞跃。浮点性能已大大超越通用处理器。现在CPU的技术进步已慢于摩尔定律, 而GPU的运行速度已超过摩尔定律(约半年性能加倍)。

GPU设计的宗旨是实现图形加速, 因此其设计基本上是围绕图形加速的相关运算优化, 如z-buffering消隐, 纹理映射, 图形的坐标位置变换与光照计算等。这类计算的数据类型比较单一, 单精度浮点占绝大多数。

CPU和GPU存在两方面的重要差异和特色:

1. **CPU的多线程机制通过操作系统提供的API实现, 是一种粗粒度多线程(各线程有自己独立的上下文以及执行资源, 因此各线程可相互独立地各自执行, 大多CPU的线程属于粗粒度线程)。** 当一个线程中断, 或等待某种资源时, 系统就要保存当前线程上下文并装载另一个线程上下文。而**GPU是硬件管理的轻量级线程(进程内的独立代码段), 可实现零开销线程切换。**
- 2、**CPU使用了很多提高指令级并行的技术(如超级标量、深度流水线、乱序执行、分支预测、SSE等)。** 大量晶体管用于这些技术细节的实现, 所以单个核心面积较大, 限制了核的数量。而**GPU包含数目众多的拥有完整前端的小核心, 如HD5870拥有320个VLIW Core。**

采用异构多核处理器成为发展方向之一。将完成不同功能的处理单元集成在同一CPU内核中,以提高多种不同任务运行的并行性。

将GPU与传统CPU集成在一起成为典型的异构处理器方案(如AMD的Fusion方案)。

异构多核架构也可集成如:

- 高运算能力的DSP内核
- DPU(数据并行处理)内核: 适用于视频编/解码等流媒体数据处理应用