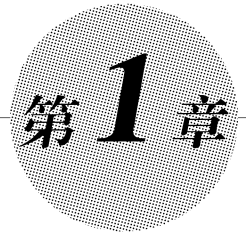



# 引　言

## 1.1　微处理器进入64位时代

随着信息技术的发展，计算机的应用已经渗透和深入至政治、经济、科学技术、社会生活和人们日常生活的各个方面。网络时代的来临以及多媒体信息的数字化等，都使信息量爆炸般增长。信息的存储、处理、交换，强烈地需求和促进微处理器向64位时代过渡。

随着Internet及其各种新的应用(如电子商务)的发展，企业的信息量不断增加，每年增长1～6倍，这使得企业对数据存储的需求急剧增长。调查显示，全球每年存储设备约增长1～10倍(对应于不同的应用环境)，并成为计算机硬件系统购买成本中所占比例最大的部分。

美国加州大学伯克利分校信息管理学院一项研究分析报告中称："全球今后3年内产生的数据将会多于过去4万年中产生的数据。"

数据已成为最宝贵的财富，数据是信息的符号，数据的价值取决于信息的价值。由于越来越多的有价值的关键信息转变为数据，数据的价值也就越来越高。对于很多行业甚至个人而言，保存在存储系统中的数据是最为宝贵的财富。在很多情况下，数据要比计算机系统设备本身的价值高得多，尤其对金融、电信、商业、社保和军事等部门来说更是如此。设备坏了可以花钱再买，而若数据丢失了对于企业来讲，损失将是无法估量的，甚至是毁灭性的。因此，信息存储系统的可靠性和可用性以及数据备份和灾难恢复能力往往是企业用户首先要考虑的问题。为防止地震、火灾和战争等重大事件对数据的毁坏，关键数据还要考虑异地备份和防灾问题。

微处理器是现代计算机系统的核心和引擎，它不仅提供了计算机系统所需的处理能力，而且能够管理缓存、内存和互联子系统，支持整个系统实现多处理器并行计算。

海量的信息，信息的存储、处理和交换，都要求微处理器有更强大的能力，处理器从32位向64位过渡已经成为历史的必然，微处理器已经进入了64位时代。

64位技术揭开了信息时代的新篇章，支持全球性Internet和电子商务的大型7×24网站、破译人类基因密码、广泛应用数字技术、分析预报全球性的天气和灾害、探测外层空间等都离不开各种基于64位微处理器的计算机系统。64位技术的广泛应用促使数据量

爆炸性地增加，推动信息技术的应用发生新革命，进入以存储为中心的新时代。

计算机自1946年问世以来，经历了许多重要的变革，其中最有意义的变革之一是从复杂指令集(CISC)过渡到精简指令集(RISC)体系结构。RISC体系结构和设计思想是20世纪80年代初出现的，它的基本思路是：抓住CISC指令系统指令种类太多(其中80%以上都是程序中很少使用的指令)、指令格式不规范、寻址方式太多的缺点(例如，作为CISC的VAX 780的指令操作类型超过1000种，而作为RISC的Alpha只有不到50种指令)，通过减少指令种类、规范指令格式和简化寻址方式，大量利用寄存器之间的操作，大大简化处理器的结构，优化VLSI(超大规模集成电路)器件使用效率，从而大幅度地提高了处理器性能、并行处理能力和性价比。到20世纪80年代后期，RISC技术已经发展成为支持高端服务器系统的主流技术，各厂商纷纷推出了32位RISC微处理器，如IBM公司的PowerPC和Power2，Sun公司的SPARC，HP公司的PA-RISC 7000和MIPS公司的R系列等。

基于32位RISC芯片的产品取得了很大的成功，应用日益广泛，软件大量积累，在市场上也产生了巨大的影响，这就促进了利用商品化的部件来生产超级计算机。RISC技术使人们能够利用商品化程度很高的RISC微处理器生产出性能可以与低端向量机相媲美的计算机系统。这也启示人们以更高级的RISC技术迈向超级计算的顶峰，孕育了64位RISC计算的新时代。

1994年6月，Intel公司和HP公司签署合作协议，为服务器和工作站市场共同开发全新的64位架构。1997年11月，Intel和HP公司宣布推出基于EPIC(explicitly parallel instruction computing，显性并行指令计算)的Itanium体系结构，并推出产品代号为“Merced”的IA-64处理器系列的计划。EPIC既不是RISC也不是CISC，它实质上是一种吸收了两者长处的体系结构。IA-64的第一代芯片名为Merced，第二代芯片名为McKinley，第三代芯片名为Madison (Deerfield是它的缩小型号)，目前统称为Itanium处理器系列(IPF系列)。2001年5月，经过7年的艰苦努力，IA-64体系结构IPF系列的第一代产品Itanium终于正式上市。

Itanium体系结构的设计实现基于如下的原则，使得IPF系列处理器不但能够实现持续高性能，而且具有随着技术发展进一步提高性能的潜力：

- 支持显性并行指令计算(EPIC)；
- 提供一系列有利于增强指令级并行的特性；
- 把重点放在提高应用软件实际运行的性能，面向广泛范围的应用上。

众所周知，人们主要通过提高IPC(每个周期执行的指令数)和主频来提高芯片的性能。为了提高IPC，必须提高处理器指令级并行(ILP)的能力。所谓ILP是指处理器同时执行多条指令的能力，即处理器在每个时钟周期内发送和执行尽可能多条指令的能力。为此要求处理器：(1)能够找到和标识程序中可以并行执行的指令段；(2)具有充分的资源在最短时间内发送和同时执行可并行执行的指令段。这就要求处理器具有足够的智能和资源来完成这两项任务，要求人们不断探索更快速、更经济的途径完成这两项任务，推

动处理器技术向前发展。

传统的 RISC 设计师们希望通过在芯片上增加更多的逻辑和智能(“聪明的处理器”)来提高指令并行度,同时又不必采用太高的工艺、增加太多的资源。他们把指令级并行分为静态和动态两类。静态并行在编译时由编译程序发现和处理;动态并行在运行时由处理器发现和处理。大多数现代的 RISC 处理器都具有这两种并行功能。首先通过编译程序把程序改造成一个由许多可并行执行的指令段组成的记录(静态并行)。但是,许多有关程序执行过程的信息只有处理器能够在运行时了解到,例如内存访问是否命中缓存,比较指令的结果和转移指令的方向等。因此,处理器还具有无序指令发送机制,使得处理器能够根据程序的运行实际结果改变指令发送和执行的次序,而不会阻塞处理器的运行。这种无序执行技术的主要优点是能够在有限的工艺和资源条件下,大大提高指令并行度。

虽然无序执行技术已经成为当前 64 位 RISC 芯片设计思想的主流,取得了很大的成功,但是这种技术也有其缺点,主要缺点如下:

① 无序执行技术要求处理器具有较高的智能和复杂的逻辑,使得芯片的结构越来越复杂,同时也妨碍了主频和性能的提高。

② 设计难度越来越大,使得许多 RISC 芯片的设计周期越来越长,而且经常不能按期上市,难以满足应用发展的需要。

③ 处理器在运行时没有能够充分利用编译程序所产生的许多有用的信息来提高指令并行度,也就是说,传统的 RISC 技术没有充分发挥硬件和软件相结合的合力。

IA-64 的 EPIC 体系结构在吸收这些教训的基础上另辟蹊径,它的基本设计思想是:

① 提供一种新的机制,利用编译程序和处理器协同能力来提高指令并行度。传统的 RISC 体系结构没有能够充分利用编译程序所产生许多有用的信息,如关于程序运行线路的猜测信息;也没有充分利用现代编译程序强大的对程序执行过程的调度能力。EPIC 体系结构采用创新的技术充分利用编译程序提供的信息和调度能力来提高指令并行度,同时保证在程序运行过程中发现猜测和调度有错时,处理器仍然给出正确的结果,并且尽量减少由此而带来的延迟和惩罚。

② 在此基础上简化芯片逻辑结构,为提高主频和性能开辟道路,在工程上有一条基本原则,不是越复杂越好,而是越简洁越好,事实上,简洁的构思比复杂的构思更困难。

③ 提供大量的资源来实现 EPIC,包括存储编译程序提供的信息以及提高并行计算效率所需的处理单元、缓存和其他资源。

IA-64 体系结构引入 64 位寻址和新的指令集,它还包含一个 IA-32 模式的指令集,所有 IA-64 处理器都能够执行 IA-32 程序。

Itanium 体系结构支持两种操作系统环境:

- IA-32 系统环境,支持 IA-32 32 位操作系统。
- Itanium 系统环境,支持基于 Itanium 的 64 位操作系统。

Itanium 体系结构也支持在单个基于 Itanium 操作系统中,IA-32 的应用程序和基于 Itanium 的应用程序的混合。如图 1-1 所示。

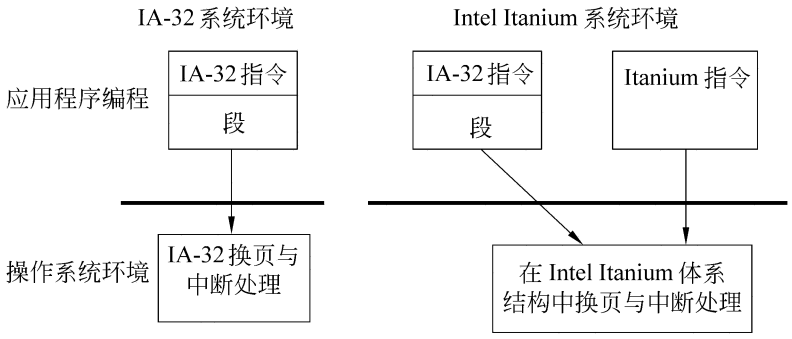



**图 1-1　系统环境**

表 1-1 定义了主要支持的操作系统环境。

**表 1-1　主要操作系统环境**

| 系 统 环 境 | 应用程序环境 | 用　　法 |
|---|---|---|
| IA-32 系统环境 | IA-32 指令集 | IA-32 PM、RM 和 VM86 应用程序和操作系统环境与 IA-32 Intel Pentium、Pentium Pro、Pentium Ⅱ 和 Pentium Ⅲ处理器兼容 |
| | Intel Itanium 指令集 | 不支持，基于 Itanium 的应用程序不能在 IA-32 系统环境中执行 |
| Itanium 系统环境 | IA-32 保护模式 | Intel Itanium 系统环境中 IA-32 保护模式应用程序 |
| | IA-32 实模式 | Intel Itanium 系统环境中 IA-32 实模式应用程序 |
| | IA-32 虚拟模式 | Intel Itanium 系统环境中 IA-32 虚拟 8086 模式应用程序 |
| | Intel Itanium 指令集 | 基于 Intel Itanium 的操作系统与基于 Itanium 的应用程序 |

总之，IA-32 的应用程序可以在 Itanium(IA-64)的 IA-32 系统环境或 Itanium 系统环境的 IA-32 模式下运行。但它不能使用 Itanium 提供的丰富的 64 位处理器的资源。IA-32 与 Itanium 实质上是两套不同的指令系统，它们实质上是不兼容的。

x86 系列的另一个主要生产商 AMD 推出了与 x86 兼容的 64 位处理器——x86-64 体系结构。本书针对 AMD x86-64 和 Intel Itanium 两种处理器，讨论它们的应用编程。

## 1.2　术语和记法

**1011b**：一个二进制值。在此例中为一个 4 位二进制值。

**F0EAH**：一个十六进制值。在此例中为一个 2 字节十六进制值。

**[1,2)**：一个范围。它包括左边的值(即左边是闭区间，此例中为 1)，但不包括右边的值(即右边是开区间，此例中为 2)。

**7～4**：位的范围，从位 7 至位 4，包含边界两位。高位先显示。

**128位媒体指令**：使用128位XMM(扩展内存管理程序)寄存器的指令。这些是SSE(流式单指令多数据扩充指令集)和SSE2指令集的组合。

**64位媒体指令**：使用64位MMX™(多媒体扩展指令集)寄存器的指令。这些主要是MMX和3DNow!™(三维Now!技术)指令集的组合，还具有某些从SSE和SSE2指令集来的附加指令。

**16位模式**：一种传统模式或兼容模式，在其中，活动的是16位地址尺寸。见“传统模式”和“兼容模式”。

**32位模式**：一种传统模式或兼容模式，在其中，活动的是32位地址尺寸。见“传统模式”和“兼容模式”。

**64位模式**：长模式的一种子模式。其中，默认的地址尺寸是64位，对于系统和应用软件支持如寄存器扩展等新特性。

**#GP(0)**：指示通用保护异常(#GP)的记法，它具有差错码0。

**绝对**：是指引用码段的基地址而不是指令指针的位移量。与“相对”相对照。

**偏移阶**：对于一具体的浮点数据类型，浮点值的阶和一个偏移常数的和。偏移常数使偏移阶的范围始终是正，这允许互换而不溢出。

**字节**：8位。

**清除**：写一位的值为0。与“设置”相对照。

**兼容模式**：长模式的一种子模式。在兼容模式中，默认的地址尺寸是32位。传统的16位和32位应用程序可以不修改地运行。

**提交**：一个指令的结果按程序的顺序不可逆地写至软件可见的存储器中，如寄存器(包括标志)、数据缓存、内部的写缓冲器或内存。

**CPL**：当前特权级。

**CR0～CR4**：从寄存器CR0至CR4的寄存器范围(包括的)，低寄存器在先。

**CR0.PE = 1**：指示CR0寄存器的PE位有值1的记法。

**直接**：引用地址作为一个立即操作数包含在指令的语法中的内存单元。此地址可以是绝对的或相对的。与“间接”相对照。

**直接数据**：数据保持在处理器的缓存或内部缓冲器中，这比保持在存储器中的复制更近。

**位移量**：一个加至段的基地址(绝对寻址)或指令指针(相对寻址)上的符号值。同于“偏移量”。

**双字**：2个字或4个字节或32位。

**双四字**：8个字或16个字节或128位。也称为八字(octword)。

**DS:rSI**：内存单元的内容，它的段地址在DS寄存器中，它的相对于段的偏移量在rSI寄存器中。

**EFER.LME = 0**：指示EFER寄存器的LME位有值0的操作码的记法。

**有效地址尺寸**：对于当前指令在计算了默认地址尺寸和任何的地址尺寸超越前缀后的地址尺寸。

**有效操作数尺寸**：对于当前指令在计算了默认操作数尺寸和任何的操作数尺寸超越

前缀后的操作数尺寸。

**元素**：见“向量”。

**异常**：作为指令执行的结果发生的反常条件。处理器对异常的响应取决于异常的类型。对于所有异常，除了128位媒体SIMD(单指令多数据)浮点异常和x87浮点异常，控制按异常向量的定义被传送至此异常的处理程序(或服务例程)。对于由IEEE 754标准定义的浮点异常，有屏蔽的响应和未屏蔽的响应两种。当未屏蔽时，调用异常处理程序；当屏蔽时，提供默认的响应以代替调用处理程序。

**FF /0**：指示FF是操作码的第一个字节和在ModR/M字节中有值为0的子操作码的记法。

**冲刷**：一个常常不太明确的术语，有多个含义：

① 若修改，写回，并无效，如flush the cache line(冲刷缓存行)；

② 无效，如flush the pipeline(冲刷管道)；

③ 改变值，如flush to zero(冲刷为0)。

**GDT**：全局描述符表。

**IDT**：中断描述符表。

**IGN**：忽略。字段被忽略。

**间接**：引用一个存储单元而它的地址在一个寄存器或其他内存单元中。地址可以是绝对的，也可以是相对的。

**IRB**：虚拟8086模式中断重定向位图。

**IST**：长模式中断堆栈表。

**IVT**：实地址模式中断向量表。

**LDT**：局部描述符表。

**传统x86**：传统x86体系结构。

**传统模式**：x86-64体系结构的一种传统模式，其中，已存在的16位和32位应用程序和操作系统能不作修改地运行。x86-64体系结构的处理器实现能运行在长模式或传统模式。传统模式有3种子模式：实模式、保护模式和虚拟8086模式。

**长模式**：长模式只相对于x86-64体系结构而言。x86-64体系结构的处理器实现能运行在长模式或传统模式。长模式有两种子模式：64位模式和兼容模式。

**lsb**：最低有效位。

**LSB**：最低有效字节。

**主内存**：物理内存，例如在一个具体计算机系统中安装的RAM和ROM(但不是缓存内存)。

**屏蔽**：① 阻止引用异常处理程序的浮点异常的控制位；

② 用于控制目的的位字段。

**MBZ**：必须是0。若软件企图设置MBZ位为1，则发生通用保护异常(#GP)。

**内存**：主内存，除非另有规定。

**ModRM**：指令操作码之后的规定基于模式(Mod)、寄存器(R)和内存(M)变量的地址计算的字节。

**moffset**：一个直接内存偏移。换句话说，加至码段的基地址（直接寻址）或加至指令指针（相对于指令指针的相对寻址，如在 RIP（相对寻址）中）的位移量。

**msb**：最高有效位。

**MSB**：最高有效字节。

**多媒体指令**：128 位媒体指令和 64 位媒体指令的组合。

**八字**：同于“双四字”。

**偏移量**：同于“位移量”。

**溢出**：浮点数在数量上大于正在使用的数据类型能表示的最大的、有限的、正或负数的条件。

**组合的**：见“向量”。

**PAE**：物理地址扩充（physical address extension）。

**物理内存**：实际内存由主内存和缓存组成。

**探测**：对于在处理器的缓存或内部缓冲器中地址的检查。外部探测针对处理器的外部；内部探测针对处理器的内部。

**保护模式**：传统模式的一种子模式。

**四字**：4 个字或 8 个字节或 64 位。

**RAZ**：按零（0）读，不管写了什么。

**实地址模式**：见“实模式”。

**实模式**：实地址模式的短名，传统模式的一种子模式。

**相对**：从指令指针而不是码段的基地址引用一个位移量（也称为偏移量）。

**REX**：一个指令前缀，它规定 64 位操作数尺寸，提供对附加的寄存器的访问。

**RIP-相对寻址**：相对于 64 位 RIP 指令指针寻址。与“moffset”相对照。

**设置**：写一位的值为 1。

**SIB**：跟随在指令操作码之后的一个字节，它规定基于比例（S）、索引（I）和基（B）的地址计算。

**SIMD**：单指令多数据。见“向量”。

**尺寸（size）**：是地址、寄存器和操作数的大小、长度、宽度、规模的总称。

**SSE**：流式 SIMD 扩展指令集。见“128 位媒体指令”和“64 位媒体指令”。

**SSE2**：对 SSE 指令集的扩展。见“128 位媒体指令”和“64 位媒体指令”。

**粘着位**：由硬件设置或清除的位且保留其状态直至由软件显式改变。

**TOP**：x87 堆栈顶。

**TPR**：任务特权寄存器（CR8）。

**TSS**：任务状态段。

**下溢**：浮点数在数量上小于正在使用的数据类型格式能表示的最小的非零正数或负数的条件。

**向量**：① 一组称为元素的整数或浮点数，它们组合为单个操作数。大多数 128 位和 64 位媒体指令用向量作为操作数。向量也称为组合的或 SIMD（单指令多数据）操作数。

② 在中断描述符表（IDT）中的索引，用于访问异常处理程序。与“异常”相对照。

**虚拟 8086 方式**：传统模式的一种子模式。

**字**：两个字节或 16 位。

**x86**：见“传统 x86”。

**寄存器**：在以下寄存器清单中，名字用于引用一个给定的寄存器或此寄存器的内容。

**AH～DH**：指 AH、BH、CH 和 DH，是 AX、BX、CX、DX 寄存器的高 8 位。与“AL～DL”相对照。

**AL～DL**：指 AL、BL、CL 和 DL，是寄存器 AX、BX、CX、DX 的低 8 位。

**AL～r15B**：AL、BL、CL、DL、SIL、DIL、BPL、SPL 和 R8B～R15B 寄存器的低 8 位，在 64 位模式可用。

**BP**：基指针寄存器。

**CRn**：n 号控制寄存器。

**CS**：码段寄存器。

**eAX～eSP**：16 位 AX、BX、CX、DX、DI、SI、BP 和 SP 寄存器或 32 位 EAX、EBX、ECX、EDX、EDI、ESI、EBP 和 ESP 寄存器。与“rAX～rSP”相对照。

**EFER**：扩展特征允许寄存器。

**eFLAGS**：16 位和 32 位标志寄存器。与“rFLAGS”相对照。

**EFLAGS**：32 位(扩展的) 标志寄存器。

**eIP**：16 位或 32 位指令指针寄存器。与“rIP”相对照。

**EIP**：32 位(扩展的)指令指针寄存器。

**FLAGS**：16 位标志寄存器。

**GDTR**：全局描述符表寄存器。

**GPR**：通用寄存器。对于 16 位数据尺寸，有 AX、BX、CX、DX、DI、SI、BP 和 SP。对于 32 数据尺寸，有 EAX、EBX、ECX、EDX、EDI。ESI、EBP 和 ESP。对于 64 位数据尺寸，有 RAX、RBX、RCX、RDX、RDI、RSI、RBP、RSP 和 R8～R15。

**IDTR**：中断描述符表寄存器。

**IP**：16 位指令指针寄存器。

**LDTR**：局部描述符表寄存器。

**MSR**：模型特定寄存器。

**r8～r15**：8 位 R8B～R15B 寄存器，或 16 位 R8W～R15W 寄存器，或 32 位 R8D～R15D 寄存器，或 64 位 R8～R15 寄存器。

**rAX～rSP**：16 位 AX、BX、CX、DX、DI、SI、BP 和 SP 寄存器，或 32 位 EAX、EBX、ECX、EDX、EDI、ESI、EBP 和 ESP 寄存器，或 64 位 RAX、RBX、RCX、RDX、RDI、RSI、RBP 和 RSP 寄存器。对于 16 位用空格、32 位用“E”、64 位用“R” 替换占位符 r。

**RAX**：EAX 寄存器的 64 位版本。

**RBP**：EBP 寄存器的 64 位版本。

**RBX**：EBX 寄存器的 64 位版本。

**RCX**：ECX 寄存器的 64 位版本。

**RDI**：EDI 寄存器的 64 位版本。

**RDX**：EDX 寄存器的 64 位版本。

**rFLAGS**：16 位、32 位或 64 位标志寄存器。与“RFLAGS”相对照。

**RFLAGS**：64 位标志寄存器。

**rIP**：16 位、32 位或 64 位指令指针寄存器。与“RIP”相对照。

**RIP**：64 位指令指针寄存器。

**RSI**：ESI 寄存器的 64 位版本。

**RSP**：ESP 寄存器的 64 位版本。

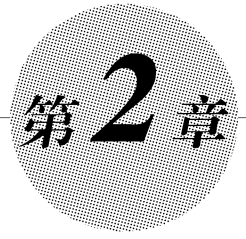
**SP**：堆栈指针寄存器。

**SS**：堆栈段寄存器。

**TPR**：任务优先权寄存器，在 x86-64 体系结构中引入的新寄存器以加速中断管理。

**TR**：任务寄存器。

**结尾顺序**：x86 和 x86-64 体系结构用小结尾字节顺序寻址内存。多字节值存储时用它们的最低有效字节放在最低字节地址和用最低有效字节在右边表示。串用反序表示，因为它们的字节地址从右至左增加。




# AMD x86-64 体系结构概要

## 2.1 引言

AMD x86-64 体系结构是简单的，但它是与工业标准（传统的）x86 体系结构后向兼容的强有力的 64 位扩展。它增加了 64 位寻址，扩展了寄存器资源，已存在的传统的 x86 体系结构的 16 位和 32 位应用程序和操作系统不需要修改或重新编译就能在 x86-64 体系结构下运行，对于重新编译的 64 位程序可提供更高的性能。这种体系结构对于大量已存在的软件和要求更高性能的新的 64 位应用软件都能提供无缝的高性能支持。

x86-64 体系结构的需要是由高性能服务器、数据库管理系统和 CAD 工具等的应用程序，要求大型和高精度数据和大的虚拟和物理存储器的地址范围引出的。同时它们也从 64 位地址和增加的寄存器数得到好处。在传统的 x86 体系结构中可用的寄存器数少，在强计算的应用程序中限制了其性能。寄存器数的增加对许多这样的应用程序提供了性能推进。

### 2.1.1 新特征

x86-64 体系结构引进了以下新特征。

① 寄存器扩展，如图 2-1 所示。

- 8 个新通用寄存器（GPR）；
- 所有 16 个 GPR 是 64 位宽；
- 8 个新 128 位 XMM 寄存器；
- 为所有 GPR 可寻址的统一字节寄存器；
- 一个新指令前缀（REX）可访问所有扩展的寄存器。

② 长模式，如表 2-1 所示。

- 虚拟地址增至 64 位；
- 64 位指令指针（相对寻址）；
- 新的指令指针数据相对寻址模式；
- 平面的（不分段的）地址空间。