## ARM9/ARM10/ARM11与Intel x86处理器的架构对比分析: ARM体系结构的特点

张建夫

10235101477

2025/03/03

East China Normal University

## 目录

1 ARM与x86简单介绍 …………………………………………………………………2

2 ARM与x86架构发展概述 …………………………………………………………….2

2.1 ARM架构发展概述 ..……………………………………………………………..2

2.2 x86架构发展概述 ………………………………………………………………...3

3 ARM与x86指令集比较 ………………………………………………………………...3

3.1 ARM的精简指令集（RISC）特性 ……..…………………………………………3

3.2 x86的复杂指令集（CISC）特性 …………………………………………………4

3.3 二者指令集的对比分析 ...………………………………………………………....4

4 ARM与x86流水线架构比较 …………………………………………………………..5

4.1 ARM系列的流水线设计 ………………………………………………………….5

4.2 x86的流水线设计 …………………………………………………………………6

4.3 ARM与x86流水线设计对比分析 ……………………………………………….6

5 ARM与x86中断机制比较 ……………………………………………………………..7

5.1 ARM的中断机制 ………………………………………………………………….7

5.2 x86的中断机制 ……………………………………………………………………8

5.3 二者中断机制的对比分析 ………………………………………………………….9

6 结论 ………………………………………………………………………………………9

7 引用(Reference) …………………………………………………………………………11

## ARM9/ARM10/ARM11与Intel x86处理器的架构对比分析: ARM体系结构的特点

ARM与x86是两大主流的处理器架构。ARM架构以其精简指令集（RISC）、低功耗和高效性能闻名，广泛应用于嵌入式系统和移动设备；x86架构则以其复杂指令集（CISC）和强大的兼容性在PC与服务器市场占据主导地位。本文以ARM9、ARM10和ARM11系列处理器为代表，全面对比ARM与x86架构在指令集设计、流水线深度与优化、中断处理机制、条件执行策略等关键技术方面的异同，并探讨其各自的技术演进及应用场景。

**ARM与x86架构发展概述**

**ARM架构发展概述**

ARM架构的起源可以追溯到1980年代，由英国Acorn公司为开发个人计算机而设计。其最初目标是打造低成本、低功耗且高效的精简指令集（RISC）处理器（Furber, 2000），以区别于当时流行的复杂指令集（CISC）架构。1990年，Acorn公司分拆成立了Advanced RISC Machines Ltd.（即ARM公司），正式确立了ARM架构的商业化发展路径（ARM Ltd., 2001）。ARM的早期处理器（如ARM2和ARM3）就展示出了RISC架构在功耗和性能之间的优异平衡。

进入21世纪后，ARM架构迅速崛起，广泛应用于嵌入式系统、智能手机、平板电脑和物联网设备。ARM9、ARM10和ARM11分别在流水线深度、内存管理单元（MMU）和多媒体指令集扩展方面不断改进。ARM11更是首个支持对称多处理（SMP）的ARM架构（ARM Ltd., 2005），为后续Cortex系列奠定了基础。ARM架构的核心优势包括低功耗

设计、高代码密度以及良好的软件兼容性，这些特性使其在移动端和嵌入式领域占

能计算领域，形成了与x86架构并驾齐驱的竞争态势。

**x86架构发展概述**

x86架构最早由Intel公司于1978年推出，最初是16位的8086处理器，为个人计算机奠定了基础（Intel Corporation, 2011）。8086采用CISC（Complex Instruction Set Computer）设计理念，提供丰富而灵活的指令集，使得软件开发者无需关注底层硬件细节即可实现复杂功能（Patterson & Hennessy, 2016）。在随后推出的80286和80386中，x86逐步扩展为支持24位和32位寻址，并引入了分段内存管理与虚拟内存机制，为多任务操作系统的运行提供了底层支持（Jacob, Ng & Wang, 2010）。

1990年代，x86架构迎来重要转折，Intel推出了Pentium系列，引入超标量流水线（superscalar pipeline）和分支预测技术，大幅提升了指令并行度和处理性能（Yeh & Patt, 1991）。进入21世纪后，x86架构进一步引入多级缓存、多核并行和硬件加速虚拟化技术，以适应高性能计算和服务器市场的需求（Borkar, 2011）。x86-64扩展则在原有32位指令集基础上增加了64位寻址与寄存器扩展，极大提升了数据处理能力与大内存支持（Intel Corporation, 2011）。

**ARM与x86指令集比较**

**ARM的精简指令集（RISC）特性**

ARM架构自诞生之初便秉持精简指令集（RISC）的设计理念，强调指令集简洁、解码逻辑简单和硬件实现高效（Furber, 2000）。ARM的指令长度固定为32位（ARMv4及之前），在Thumb模式下则采用16位压缩指令，兼顾代码密度与执行效率（ARM Ltd., 2001）。绝大多数ARM指令具备固定的执行周期，能够在单个周期内完成，从而简化了流水线设计和分支预测逻辑（ARM Ltd., 2005）。此外，ARM指令集强调寄存器操作，减少访存指令的使用频率，从而降低存储器访问的延迟对性能的影响（Sloss, Symes, & Wright, 2004）。ARM还具备独特的条件执行机制，大量普通指令都可带条件前缀，从

而减少了对分支跳转指令的依赖，提高了流水线的连续执行效率（Furber, 2000）。整体来看，ARM的RISC指令集以高效执行和低功耗为核心目标，非常适合移动和嵌入式场景（Patterson & Hennessy, 2016）。

**x86的复杂指令集（CISC）特性**

x86架构是典型的复杂指令集（CISC）代表，其指令集不仅数量庞大、功能丰富，而且指令格式和长度高度可变（Intel Corporation, 2011）。从8086到现代x86-64，x86指令长度可以从1字节到15字节不等，支持丰富的操作数寻址模式和复杂的内存操作（Jacob, Ng, & Wang, 2007）。x86指令集直接支持诸如字符串处理、BCD运算等高级操作，极大简化了应用软件的开发难度（Patterson & Hennessy, 2016）。然而，x86的CISC特性也带来了解码复杂、执行单元设计负担重等问题（Smith & Nair, 2005）。为了兼顾向后兼容性与性能，现代x86处理器普遍采用「前端CISC解码+后端RISC执行」的微架构策略，将复杂指令解码成多个RISC风格的微操作（micro-ops），以便后续流水线乱序执行和指令并行（Jacob, Ng, & Wang, 2010）。这种设计确保了x86架构的广泛软件兼容性，同时通过复杂硬件逻辑提升了执行性能，但也使得硬件规模和功耗显著高于典型RISC处理器（Patterson & Hennessy, 2016）。

**二者指令集的对比分析**

ARM的RISC和x86的CISC分别代表了两种不同的设计哲学，各有优劣（Patterson & Hennessy, 2016）。RISC强调**指令简单、长度固定、解码高效、执行周期固定**，从而降低硬件实现复杂度和功耗（Furber, 2000）。RISC架构依赖软件编译器将复杂操作分解为多个简单指令，因此指令条数较多，但流水线执行效率更高，整体性能和能效比非常突出（ARM Ltd., 2001）。相对而言，CISC强调**单条指令功能强大、对软件开发友好、兼容早期代码**，但带来了硬件解码复杂、执行单元负担重等问题（Smith & Nair, 2005）。现代x86处理器通过多级缓存、预测执行和微操作解码等先进技术弥补了部分性能损失

，但功耗和面积始终高于RISC架构（Borkar, 2011）。整体来看，RISC架构更适合移动设备和嵌入式系统，而CISC架构则依托强大的软件生态和向后兼容性，在桌面和服务器领域占据主导地位（Patterson & Hennessy, 2016）。随着硬件技术进步和指令集设计理念的交融（如ARMv8引入高级SIMD指令），RISC与CISC的界限逐渐模糊，未来二者将更多结合各自优势，推动计算架构多样化发展（Borkar, 2011）。

**ARM与x86流水线架构比较**

**ARM系列的流水线设计**

ARM9系列处理器采用经典的5级流水线，包括取指（Fetch）- 译码（Decode）- 执行（Execute）- 存储（Memory）- 写回（Write Back）（Sloss, Symes, & Wright, 2004）。这种流水线结构相对简单，便于控制与调度，同时兼顾指令并行和低延迟。在ARM9架构中，流水线阶段长度均衡，且结合了RISC架构固定长度指令和条件执行特性，极大降低了分支预测失败带来的性能损失（Furber, 2000）。ARM9流水线以其低功耗和高实时性，广泛应用于嵌入式设备和移动终端（ARM Ltd., 2005）。

ARM10系列进一步深化流水线设计，采用6级流水线，在取指和译码阶段进一步细分，以提高分支预测的准确度和访存操作的效率（ARM Ltd., 2005）。ARM10首次引入了更精细的分支预测机制，根据历史分支行为动态预测分支走向，从而减少流水线空泡。此外，ARM10的访存阶段采用非阻塞加载技术，在等待数据返回时允许后续独立指令继续执行，提升流水线整体利用率（Sloss et al., 2004）。这种深度流水线设计兼顾了性能提升和功耗控制，适合更多中高端嵌入式应用场合。

ARM11架构进一步扩展到8级流水线，在ARM9/ARM10基础上额外细分执行和访存阶段，以支持乱序执行（Out-of-Order Execution），显著提高了每周期指令执行数（IPC）（ARM Ltd., 2001）。ARM11还引入静态与动态分支预测结合机制，同时结合条件执行以降低分支错误率（Furber, 2000）。ARM11的流水线深度增加，使得整体频率得

以提升，但同时带来了更高的分支预测和异常处理复杂度。这一代架构标志着ARM逐渐向高性能应用迈进，为后续Cortex系列打下了基础（ARM Ltd., 2001）。

**x86的流水线设计**

与ARM相比，x86处理器的流水线设计呈现出更深的阶段划分与更复杂的控制逻辑。Pentium 4采用了超过20级的超深流水线设计（Jacob, Ng, & Wang, 2007），通过极深流水线提升时钟频率，达到更高的峰值性能。这种超深流水线的代价是分支预测失败的惩罚极为严重，可能导致整个流水线清空，影响性能稳定性（Smith & Nair, 2005）。

x86处理器普遍采用超标量（Superscalar）架构，支持每周期多个指令的同时发射与执行。为了适应x86指令长度不定、指令解码复杂等特性，x86流水线前端将CISC指令解码为等长的微操作（Micro-ops）（Jacob et al., 2007），这些微操作进入独立的微操作流水线运行。这种设计有效降低了解码负担，并为后端乱序执行创造条件（Patterson & Hennessy, 2016）。

x86处理器的分支预测单元极为复杂，结合了全局历史预测（Global History Prediction）、局部分支预测（Local Branch Prediction）和返回地址预测（Return Address Prediction）等多种技术（Smith & Nair, 2005）。预测正确率直接决定了深流水线的效率。此外，x86处理器广泛应用投机执行，即在预测结果未确认前就开始执行后续指令，一旦预测错误则回退状态（Jacob et al., 2007）。这种投机执行结合乱序执行机制，使得x86流水线既灵活又高效，但硬件复杂度远高于ARM。

**ARM与x86流水线设计对比分析**

ARM9/10/11的流水线深度分别为5级、6级和8级，而x86处理器（如Pentium 4）的流水线深度可达20级以上（ARM Ltd., 2005；Jacob et al., 2007）。ARM的短流水线配合RISC指令集设计，解码和控制逻辑相对简单；x86的深流水线虽提高了时钟频率上限，但对分支预测准确性和异常处理提出更高要求（Patterson & Hennessy, 2016）。

ARM11在ARM家族中率先引入乱序执行，但整体执行窗口规模较小，主要是面向嵌入式与移动设备的功耗优化设计（Sloss et al., 2004）。相比之下，x86处理器普遍采用大规模乱序执行和超标量执行，结合硬件重排序与数据旁路技术，最大程度挖掘指令级并行（Jacob et al., 2007）。

ARM流水线前端直接处理等长RISC指令，解码逻辑极为简洁；x86流水线前端必须面对变长CISC指令，解码复杂度高，现代x86普遍采用CISC解码为RISC风格微操作的策略，将复杂指令拆分为多个微操作后再进入流水线执行（Smith & Nair, 2005）。这种微操作流水线模式有效提高了并行性，但也增加了硬件开销和功耗。

ARM流水线普遍依赖条件执行与较浅的流水线降低分支预测错误的代价（Furber, 2000），ARM11进一步引入静态+动态结合预测，兼顾硬件成本与准确率（ARM Ltd., 2001）。x86流水线则必须依赖大规模分支预测单元，结合多种预测策略以及投机执行机制，确保极深流水线的高效运行（Jacob et al., 2007）。分支预测错误对x86性能打击更大，但高预测准确率与投机执行的结合也让x86具备更强的峰值性能。

**ARM与x86中断机制比较**

**ARM的中断机制**

ARM架构的中断系统以**简单高效、低延迟响应**为设计核心，特别是面向嵌入式和实时系统场景。ARM经典架构（ARM7/9/10/11）中，中断分为两类：普通中断（IRQ, Interrupt Request）和快速中断（FIQ, Fast Interrupt Request）（Furber, 2000）。FIQ具有更高优先级，设计初衷是服务实时性要求极高的场合，如音视频数据流处理或通信协议栈的关键中断。

ARM的FIQ和IRQ采用独立的入口向量和寄存器保存策略。FIQ模式专属使用一组独立寄存器（r8-r14），中断处理过程中无需保存这些寄存器，极大降低了中断进入和

退出的上下文切换开销（Sloss, Symes & Wright, 2004）。

当ARM处理器收到中断请求时，硬件自动保存部分关键寄存器（如PC和CPSR），并切换到对应的中断模式（IRQ模式或FIQ模式）。相比x86完全依赖软件保存现场的做法，ARM硬件介入部分状态保存，大幅减少了中断响应延迟（Furber, 2000）。此外，ARM的中断处理过程通常无复杂的特权级转换，进入中断即运行在异常模式下，进一步简化控制流程。

从ARM11开始，ARM架构引入了全局中断控制器（GIC, Generic Interrupt Controller），用于管理多核环境下的中断分发与优先级控制（ARM Ltd., 2001）。GIC支持中断分组、优先级预分类和软中断触发，并可与每核本地中断控制器（PL190）协同工作，形成多层次中断管理体系。这种设计为后续ARM多核处理器（如Cortex-A系列）奠定了基础。

**x86的中断机制**

现代x86处理器采用中断描述符表（IDT, Interrupt Descriptor Table）来存储每个中断向量的详细信息，包括中断服务例程的地址、描述符特权级（DPL）以及中断门、陷阱门等类型（Intel, 2011）。当中断发生时，硬件根据中断号查找IDT，并自动跳转到对应服务例程。x86支持256个中断向量，涵盖硬件中断、异常和系统调用（Patterson & Hennessy, 2016）。

x86中断优先级由硬件与操作系统共同管理。处理器内部存在可屏蔽中断和不可屏蔽中断（NMI）的优先级划分；操作系统则可在IDT中通过向量分组实现软件优先级控制（Intel, 2011）。此外，x86的中断控制器（如APIC）支持可编程中断优先级和中断重定向，特别适合SMP（对称多处理）环境下的中断负载均衡（Smith & Nair, 2005）。

x86中断响应过程中会触发特权级提升（CPL升至0级），并结合段保护机制，切换到受保护的内核栈执行中断处理程序（Jacob, Ng, & Wang, 2007）。硬件负责保存原始

段寄存器和指令指针，确保中断返回时完整恢复上下文。相比ARM的模式切换，x86的特权级转换和段保护机制显著增强了系统安全性，但也增加了中断响应开销（Intel, 2011）。

**二者中断机制的对比分析**

ARM中断机制强调实时响应和低延迟，核心目标是缩短中断处理路径，确保嵌入式系统的实时性能（Furber, 2000）。x86中断机制强调灵活性、兼容性和安全性，尤其是为多任务操作系统和虚拟化环境提供完整的中断隔离与权限控制（Patterson & Hennessy, 2016）。

ARM采用FIQ/IRQ双层中断设计，硬件层面天然支持快速中断通道，部分寄存器专用化极大降低上下文切换成本（Sloss et al., 2004）。x86通过IDT结合APIC实现多层次优先级控制，虽然更加灵活，但硬件不直接参与寄存器保存，导致中断响应时间通常较ARM更长（Jacob et al., 2007）。

ARM中断模式与普通模式共享同一特权级，切换过程相对简单；x86每次中断都触发特权级提升，并结合段保护与分页机制形成多重防护（Patterson & Hennessy, 2016）。这种设计提升了中断处理的安全性和隔离性，但同时增加了硬件和软件开销，尤其对实时性不友好。

**结论**

ARM与x86作为当前最具代表性的两类处理器架构，分别在嵌入式系统和通用计算领域占据主导地位。通过对ARM9、ARM10、ARM11与现代x86处理器的深入对比可以发现，这两类架构的设计思路高度依赖其主要应用场景。ARM处理器秉承RISC架构的精简高效理念，指令集简洁，流水线控制逻辑相对简单，配合特有的条件执行技术和高效的中断机制，为嵌入式设备和低功耗应用提供了理想的计算平台。尤其是FIQ和IRQ的分离设计，结合硬件辅助的寄存器保存机制，使ARM的实时中断响应能力远超x86，能够很好满足传感器数据处理、通信协议栈控制等对时延高度敏感的场景。

相对而言，x86处理器依托CISC复杂指令集和高度发达的微架构技术，形成了超长流水线、复杂分支预测、乱序执行等一系列性能优化手段，针对高性能通用计算任务展现出极强的处理能力。x86的中断机制结合特权级转换和段保护机制，实现了对操作系统和多任务环境的有效隔离和控制，为虚拟化、服务器计算和个人计算机环境提供了强有力的支持。然而，这种灵活而复杂的中断管理机制也带来了较高的中断响应开销，不适合对实时性有极高要求的应用场景。

从产业演进趋势来看，ARM架构凭借其良好的能效比、不断增强的微架构设计（如乱序执行、深流水线）和完善的多核扩展能力，已经逐渐突破传统嵌入式市场，广泛进入移动终端、高性能计算和数据中心等领域。x86架构则继续巩固其在桌面计算和企业级服务器市场的主导地位，尤其在兼容历史软件资产方面拥有无可替代的优势。总体而言，ARM与x86并非单纯的技术高下之争，而是针对不同应用需求形成的两条差异化技术路径（Blem, E., Menon, J., & Sankaralingam, K., 2013）。未来，随着高性能嵌入式系统、云计算和边缘计算的快速发展，两大架构的相互借鉴与融合趋势将愈发明显，为计算机体系结构的持续演进提供更多可能性。

References

Blem, E., Menon, J., & Sankaralingam, K. (2013, February). Power struggles: Revisiting the RISC vs. CISC debate on contemporary ARM and x86 architectures. In *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)* (pp. 1-12). IEEE.

Borkar, S., & Chien, A. A. (2011). The future of microprocessors. *Communications of the ACM*, *54*(5), 67-77.

Furber, S. B. (2000). *ARM system-on-chip architecture*. pearson Education.

Guide, P. (2011). Intel® 64 and ia-32 architectures software developer’s manual. *Volume 3B: system programming guide, Part*, *2*(11), 0-40.

Jacob, B., Wang, D., & Ng, S. (2010). *Memory systems: cache, DRAM, disk*. Morgan Kaufmann.

Patterson, D. A., & Hennessy, J. L. (2016). *Computer organization and design ARM edition: the hardware software interface*. Morgan kaufmann.

Processor, A. M. (2005). Technical reference manual. *Texas Instruments. Dostopno na: http://www. ti. com/lit/ug/spruh73k/spruh73k. pdf [14.08. 2014]*.

Seal, D. (Ed.). (2001). *ARM architecture reference manual*. Pearson Education.

Sloss, A., Symes, D., & Wright, C. (2004). *ARM system developer's guide: designing and optimizing system software*. Elsevier.

Smith, J., & Nair, R. (2005). *Virtual machines: versatile platforms for systems and processes*. Elsevier.

Yeh, T. Y., & Patt, Y. N. (1991, September). Two-level adaptive training branch prediction. In *Proceedings of the 24th annual international symposium on Microarchitecture* (pp. 51-61).