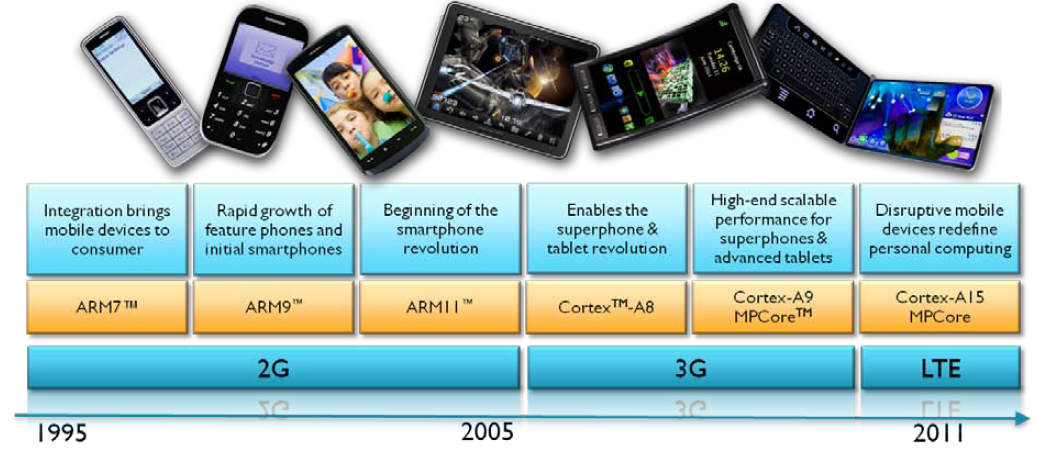
ARMv8架构

# 1 背景：ARM架构

基于ARM® RISC的架构已经发展了20多年，随着ARM7™ 系列处理器的推出，对于ARMv4架构及变体有了广泛的认识。从设计一开始，就专门针对低功耗进行不断优化。那时，没有浮点运算，没有复杂的数学指令，没有SIMD，甚至没有划分2个整数的能力。而到了2011年，所有的ARM Cortex™系列处理器都使用ARMv7架构进行设计。

ARM处理器的市场适用范围也已经扩展到了移动设备之外，所以，尽管ARMv7架构始终没变，但是被分成3个配置文件，更符合市场的细分要求，如A-应用，R-实时，M-微控制器。



Cortex系列处理器很快成为整个行业的首选。使用A系列配置文件的处理器有，Cortex-A8，新引入了SMP多媒体处理功能的Cortex-A9，低成本的Cortex-A5，低功耗的网络互联的ARM926类器件等等。Cortex-A5使开发者有机会以最低的互联网连接设备的价格，使用最新的具有丰富功能的ARM架构。这些Cortex系列处理器共同占据了ARM新业务的大部分。在2011年10月推出的Cortex-A7和能效计划再次改变了业界，使其能够提供具有延长电池寿命的入门级智能手机。

很明显，从ARMv7架构的使用量以及相关生态系统和应用程序商店的增长上来说，新ARM架构必须完全向后兼容。支持向新架构的迁移，同时又必须相对已有架构带来显著优势。

从计算机行业历史来看，都是通过在旧版本上添加新功能来完成增强。这也就是ARMv8设计的重要基础。

基于ARM处理器的设备被认为，在几年后才有进军PC市场的能力。然而，几乎所有平板电脑类设备中使用的基于Cortex-A9的处理器所展示的性能特征无疑已经证明，ARM解决方案的性能能够以更低的功耗提供给绝大多数PC用户所需的体验。该优势意味着ARM将会被应用到其它设备上，尤其是企业级设备和服务器上，在它们身上，基于ARM的设备能够降低每个单位的功耗。

也有一些基本趋势给下一版的架构提出了要求。SoC正在变得越来越集成化，而亚纳米级的制造技术允许在一个单一器件上完成难以想象的集成化。这些器件具有支持更丰富、更复杂软件的性能和需求。承载引用程序的存储器件的需求也在不断增长。单个应用程序的复杂性也需要更多内存地址空间，而32位架构提供的寻址空间已经无法满足。

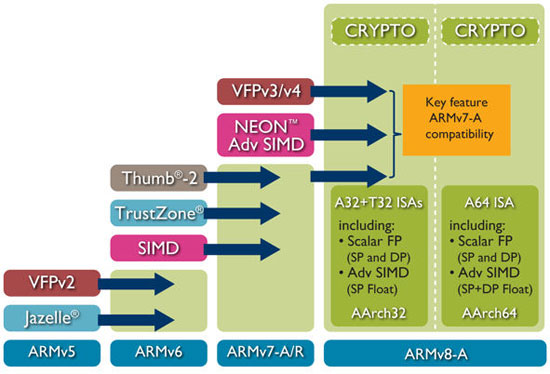
Cortex-A15处理器是在ARMv7架构下发布的；但是也引入了几个扩展功能，以支持关键的市场需求：大物理内存地址扩展（LPAE）和虚拟化。为了在单一内存地址空间上提供更高的器件集成化，需要把每一个器件和它们之间的通信放到公共的地址映射上。不像PC，GPU和CPU具有不同的地址空间，更低功耗、更低成本的解决方案要求共享同一内存地址空间。这个方案就是LPAE，Cortex-15支持新页表格式，它允许将32位的虚拟地址空间映射到40位物理地址上，页大小为4KB。这完美地解决了当今SoC器件正在遭受的问题，日益增加的SoC集成化和地址映射的限制。

那么、如果ARMv7+LPAE已经解决了4GB的限制，那么ARMv8中的变化是什么？ARMv8是否要考虑使用64位指令集架构，带来更大的虚拟地址空间？就目前ARM的市场而言，支持需要超过4GB地址空间的单个应用层序的需求相当小。

趋势。这就是ARM定义新架构时必须考虑的问题。这是我们业务的本质，我们需要展望未来，并制定计划。时至今日，ARMv8是ARM管理的最大程序。创新和改变我们周围的世界的能力是由软件定义的。在这种情况下，硬件只是一个使能者，重要，但也仅仅是软件能够传递体验和价值给消费者的工具。

# 2 ARMv8 架构的基础知识

关于ARMv8首要主要事项就是，仅仅是为Cortex A系列处理器而定义的。ARMv7架构仍然拥有多代生命周期，以解决实时和微控制器市场的需求。事实上，许多A系列需要的32位地址空间之外的扩展已经通过具有LPAE功能的Cortex-A15解决。LPAE定义了一个页表，通过它将虚拟内存映射到40位物理地址空间上。然而，这是以允许我们覆盖掉更大虚拟和物理地址空间的方式完成的。对于ARMv8-A来说，这被定义为高达48位的符号扩展的虚拟地址和高达48位的物理地址，使用这种方法对Cortex-A15上开发的软件影响最小。



对于ARMv8-A架构，必须要提的就是新的指令集，被称为A64；这使得应用程序可以运行在64位机器上。ARM决定使用新的指令集推出64位处理器，而不是在已有的指令上扩展，这有好多原因。最值得注意的是，也许并不奇怪，因为我们可以开发一个新的独立指令集，以低功耗方式执行代码，而不是通过向现有指令集添加指令。当然了，为了兼容性，新的ARMv8架构完全支持整个ARMv7架构，但是当运行64位软件时，机器的这部分没有被使用，也就是说，在64位指令集架构（ISA）中，不需要激活这部分。不像其它的架构，仅仅是在32位机器上对64位的扩展，还残留了32位模式的复杂性和遗留问题。新的指令集架构借鉴了构建不同的微架构实现的多年经验，所以新的架构更加轻松地针对低功耗进行优化-这在基于ARMv4架构的第一个低功耗处理器- ARM7之后，一直没有机会进行。

新的ARMv8-A架构还提供了优化软件异常模型和提高芯片利用率的机会，再一次提供了降低系统功耗的机会。

很明显，这种新的架构需要软件能够支持32位指令集，不像早期的Thumb指令集，我们认为32位和64位执行状态的转换只会发生在异常的边界，称为内部处理（inter-processing），而不是通过传统的ARM到Thumb的互联。这要求在多种支持的执行权限之间具有严格的层次关系。

# 3 A64 指令集

A64中的每一个指令都是在固定长度的32位指令中定义的。对于硬件，最开始的想法是调整已有的译码表在32位和64位指令集之间共享译码器，但是被拒绝了，而实现了具有连续位操作和立即数的干净译码结构。这不仅在硬件上简化了译码表结构，还提供给JIT编译器重要的加速技术，这对于高性能Web浏览和其它应用程序非常关键。独立的译码还允许一些更高级的分支预测技术。另外通用寄存器的数量也增加了。尽管Cortex-A9中引入的虚拟重命名寄存器池提供给硬件一种自动展开小型循环的方法，但是却没有为编译器提供更多寄存器，这些寄存器可以为日益复杂的算法提供改进的调度选项。因此，A64 ISA推出了31个64位通用目的寄存器。

ISA的另一个变化，实际上，是简化。回顾ARM 指令集架构最初的目标，新的A64指令集架构移除了LDM/STM（加载/载入多个寄存器指令）指令，这些指令在高效处理器内存系统的实现中增加了复杂性。条件指令减少，因为实现的复杂性并没有带来对应的优势。

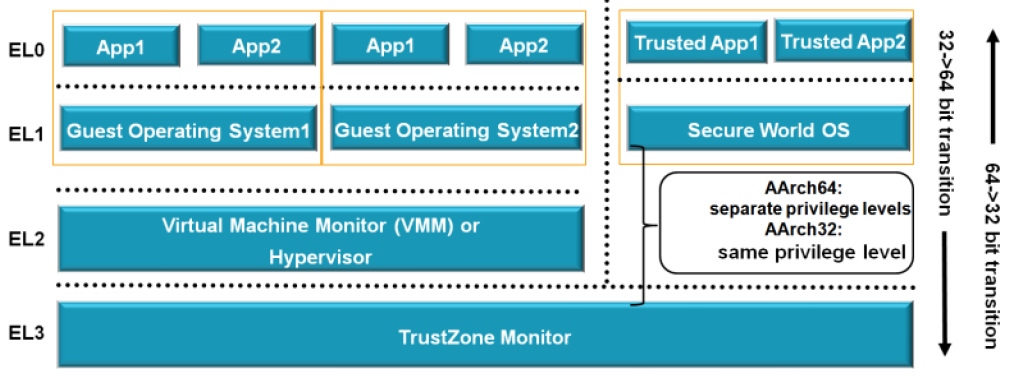
指令集中注释的最后一个方面是浮点单元将始终存在。 这意味着商用软件可以假设FP始终可用，因此不需要检查其是否存在，从而为软件提供底层硬件一致性。

SIMD数据引擎指令集现在也是基础架构的一部分，并针对新的64位架构进行了修改。它为SIMD功能引入了双精度浮点数据处理，同时还进行了简化，以更好地解决根据最新IEEE 754-2008标准所针对的算法。

# 4 异常等级

通过在OS内核模式和TrustZone® 监控模式之间插入Hypervisor模式，Aarch64架构上的异常等级和Cortex-15上的相同。安全模式下仍然只支持一个OS实例，原因就是简化和安全软件的认证之间的关系。

异常体系结构定义了严格的规则，规定32位和64位操作之间的哪些转换是合法的。随着执行的特权等级提升或在提升特权等级的过程中发生异常，操作只能在相同宽度，或者增加的宽度的ISA上执行。例如，32位的管理程序（hypervisor）支持64位的操作系统同时它工作在HYP模式下，这是不可能的。



为了支持这新的异常模型，A64还引入了一个专用的异常链接寄存器（ELR），当从32位模式进入64位异常时，它也将自动进行零扩展。中断掩码也会被自动设置到异常入口上。每一个异常等级都有它自己的向量基址寄存器，每个向量按类型区分；同步（synchronous），IRQ，FIQ或错误。异常的来源也可以从向量中获得，有关异常的其它详细信息在校验位寄存器（sysdrome register）中提供。这是一个特别有用的功能，可以实现IO设备的虚拟化，其中任何虚拟机对设备的访问都会被hypervisor中的一个异常捕获到，因此管理程序可以简单地读取此信息以评估要虚拟化的操作。

EL0->对应原先的User模式

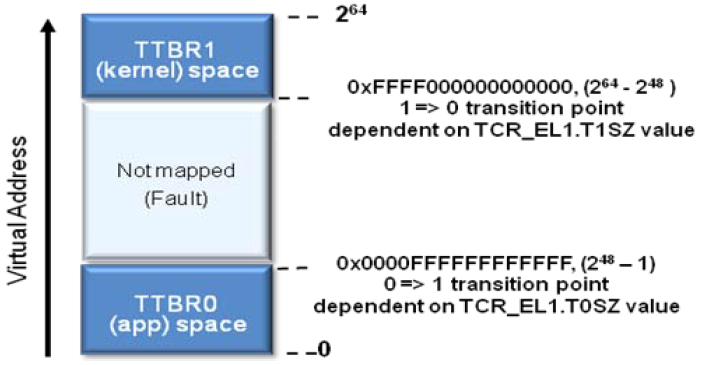
EL1->对应原先的SVC、ABT、IRQ、FIQ、UND、SYS模式

EL2->对应原先的Hypervisor模式

EL3->对应原先的Monitor模式

# 5 内存管理单元

由AArch64定义的内存管理单元基本上与Cortex-A15中使用的相同，除了支持48位虚拟和物理地址以外。对48位的支持具有以下优点：我们可以再次简化硬件，使得它们在需要译码地址时，仅需要支持到4级的页表即可，同时更重要的是限制验证的复杂性范围。 实际上，除了更熟悉的4KB页面之外，AArch64现在本身也支持64KB的最小页面大小，因此可以将所需的内存映射从4级减少到2级，其中42位地址就足够了。



任何32位代码当然都限于在前4GB的地址空间中运行，因此硬件会自动将虚拟地址零扩展到任何提升的64位调用。 为了提供基本的内存映射，该体系结构还提供了两个基本地址，用于访问OS服务和应用程序数据的虚拟地址可以从这两个地址增长。232-248 的范围内分别是64位地址空间的顶部和底部。

与Cortex-A15一样，ARMv8内存管理单元提供从应用程序虚拟地址（VA）到任何虚拟机管理程序使用的中间物理地址（IPA）的两阶段转换，然后到达放置的实际物理地址（PA） 在内存总线上。 IPA和PA的实现定义为支持32位和48位的地址空间。

# 6 寄存器

任何汇编器和编译器的作者将会注意到A64指令集的主要区别之一就是30个通用寄存器的访问。每一个寄存器的宽度都是64位的，在汇编语言中被称为X0-X30，严格意义上，X30并不是一个通用目的寄存器，因为它常被用为程序链接寄存器（PLR）。

不像Aarch32架构中，每一种操作状态都有一些分组寄存器，在Aarch64架构中，只有堆栈指针寄存器（SP），异常链接寄存器（ELR）和保存进程状态寄存器（SPSR）被分组。与Aarch32的当前处理器状态不同，在Aarch64中称为Pstate，其当前进程状态可以分类独立访问，例如，标志位，因此对寄存器中的某一位域的原子级的read/write/modify操作不是必须的。



因此，64位OS或者管理程序可以操作32位环境，这里也有寄存器架构的关系。可以将所有32位寄存器直接映射到每个64位寄存器的低32位。

# 7 Debug

关于debug功能，ARMv7 和ARMv8 的架构是相同的。你仍然可以运行直接从调试器发送的指令，且处理器有另一个通道来提取执行的指令。仍然有2种侵入式的debug类型：自托管和暂停模式。ARMv8 debug类似于ARMv7设备中提供的debug，除了所有断点和观察点（watch points）已经增长到支持64位寻址且可以在调试状态下直接执行的指令仅限于可用的一小部分。

但是应该注意的是，即使处理器仅仅运行32位代码，halting debug view也不再与现在支持ARMv7 暂停调试工具兼容。为了支持debug调试，必须把调试器更新到支持ARMv8架构。但是操作系统使用的自托管Aarch32 debug 没有发生变化。

[原文地址](https://www.arm.com/zh/files/downloads/ARMv8_white_paper_v5.pdf)