写在前面

本文档作者是苏芷沅，本文档目的是为了便于自己理解Armv8架构。请务必参照原英文文档进行对照阅读，如有错漏，请务必谅解，并且以原英文文档为准。

本文档并不是一篇原作的翻译，因此，如果有粗疏之处，请不要指责。

本文档接受他人的校注批阅，但如果有更完善的版本，请尽量联系我，同时也非常感谢您对本文档的更改修正。

希望能够对于阅读者了解Armv8架构有更多的帮助。

关于本手册

Processsing element 简称 PE

AArch32，包括了A32和T32

AArch64，只有A64指令集

PE操作的状态机，包括当前的异常级别和安全级别

异常模型

进程间的模型，支持AArch64和AArch32的转换

内存模型

编程者的模型，主要是寄存器什么的

Advanced SIMD指令以及浮点指令

单精度半精度双精度浮点操作

各种浮点数之间的转换

整数浮点的向量操作

安全模型

虚拟化模型

Debug架构

* 1. 各个部分的内容

**Part A**概论

**Part B** 描述 AArch64 执行状态的应用进程级别视图，即来自 EL0 的视图。它描述了编程者模型和内存模型的应用进程级别视图。

**Part C** A64指令集，包括每条指令在EL0下的效果，也就是说非特权级下的执行，也包括它的使用限制，以及在高特权级下的效果。这部分对于编译器汇编器以及其他的生成ARM机器码的人来说是非常重要的

**Part D** AArch64执行状态的系统级视图的描述。包括系统寄存器的细节，这些在EL0状态无法访问，以及在系统级别的视图下的编程者模型和内存模型。以及包括了self-hosted debug。

**Part E** EL0下的AArch32执行状态的应用视图，也就是在User mode。

**Part F**描述了T32和A32，其他和Part C类似

**Part G** AArch32下和Part D类似的内容

**Part H** Debug描述，包括external Debug，以及configuration，breakpoint，以及watchpoint的支持，以及a Debug Communications Channel (DCC) to a debug host

后面的Part看描述貌似不重要。

本次阅读的基础目标是完成A-D，只考虑AArch64。扩展阅读目标是剩余部分，包括了A32和T32的内容。

唔，8k页内容即便只读partA-D，仍然很麻烦，要好几千页。

Part A

这个手册并不试图规定如何实现，也不限制在这些规定之外的一些实现

而且ARM体系结构也包括debug，self-hosted debug和external debug

包括Trace Unit。The Embedded Trace Macrocell Architecture Specification. Self-hosted trace。

RISC架构

具有大的统一寄存器文件。

Load/store架构

简单的寻址模式。

同样定义了PE和内存的交互，包括cache。内存转移系统

* 1. Armv8 架构的一些概念

包括以下四种

* 执行状态
* 指令集
* 系统寄存器
* Debug
  + 1. 执行状态

执行状态有AArch64和AArch32两种

**AArch64**

31个64位寄存器，X29是Frame pointer(FP)，X30是procedure link register(LR)

64位的PC，stack pointer（SP），Exception link register（ELR）

32个128位用于SIMD向量和浮点的寄存器

A64一个单独的指令集

EL0-3的一个Exception模型，提供了一个四级特权级的模型

64位虚拟地址支持

定义了一系列Process状态，其中A64指令集被运行在不同状态上。

使用了一种后缀命名寄存器的方法，来指明，某个寄存器最低能访问的特权级。

**AArch32**

13个32位的寄存器，以及64位的PC，SP，ELR

不同的寄存器，在不同的PEmode下有不同的堆起来的实例

对于Hyp模式下，有一个单独的ELR用于异常返回。

提供了32个64位的寄存器，用于SIMD向量指令和浮点支持

提供了A32和T32两种指令集。

支持基于PE模型的Armv7的异常模型，并且映射到Armv8的基于异常等级的异常模型。

支持32位虚地址。

定义了一系列Process状态，其中A32和T32指令集被运行在不同状态上。并且可以通过Application Program Status Register (APSR)和Current Program Status Register (CPSR)两个寄存器，来访问不同的Process状态。

* + 1. Armv8指令集

**AArch64**

AArch64只支持A64指令集

A64是一个定长指令集，32位编码长度

**AArch32**

AArch32支持A32和T32两种指令集。

A32，是一个定长指令集，32位编码长度

T32，变长指令集，包括16位和32位编码长度。

在其他文档中，A32被称作Arm指令集，而T32被称作Thumb指令集。

Armv8对此进行了扩展，在AArch32模式下，不同的指令集也定义了不同的执行PE状态。

* + 1. 系统寄存器

系统寄存器提供了关于架构的控制和状态信息。

系统寄存器使用了<register\_name>.<bit\_field\_name>的一个命名方式来标识特定的寄存器以及特定的寄存器位。

<register\_name>[x:y]的方式也是可以的。

在AArch64执行状态下。还可以用<register\_name>\_ELx的方式来标识该寄存器可以访问的最低EL等级，其中x可以取0-3

系统寄存器的组成包括：

本文档中描述的内容有

* 通用系统控制寄存器
* Debug寄存器
* 通用时间寄存器
* 可选的，性能监视寄存器
* 可选的，活动监视寄存器

在其他Arm体系结构文档中定义的寄存器

* 系统跟踪寄存器（Trace System registers）
* 可扩展矢量扩展系统寄存器（Scalable Vector Extension System registers）
* GIC系统寄存器

RAS扩展系统寄存器

* + 1. Armv8 Debug

Armv8支持两种Debug模式

**Self-hosted debug**

生成Debug异常，这是在Armv8异常中的一部分。

**External debug**

PE进入Debug状态

* + 1. 支持的数据类型

**整型**

Byte 8 bits.

Halfword 16 bits.

Word 32 bits.

Doubleword 64 bits.

Quadword 128 bits.

**浮点**

半精度（16bit）

单精度

双精度

BFloat16

同时也支持了

word和double word之间的定点转换。

向量类型

Armv8设计了两种register file：

general-purpose register file

SIMD&FP register file

**AArch64**

general-purpose register file

指令可以将其作为64位和32位寄存器访问。后者仅仅使用低32位

**AArch32**