unaligned_access / no_unaligned_access

> 此选项启用或禁用基于 ARM 体系结构的处理器上的未对齐数据访问。
>
> **缺省设置**
> 对于支持未对齐数据访问的基于 ARM 体系结构的处理器，缺省为 `--unaligned_access`。这包括：
> • 基于 ARMv6 体系结构的所有处理器
> • 基于 ARMv7-A 和 ARMv7-R 体系结构的处理器。
> 对于不支持未对齐数据访问的基于 ARM 体系结构的处理器，缺省为 `--no_unaligned_access`。这包括：
> • 基于 ARMv6 以前版本的体系结构的所有处理器
> • 基于 ARMv7-M 体系结构的处理器。

apcs

> APCS，全称为ARM 过程调用标准(ARM Procedure Call Standard)，它定义了：
>
> - 对寄存器使用的限制。
> - 使用栈的惯例。
> - 在函数调用之间传递/返回参数。
> - 能够被'回溯'的基于栈的结构的格式，用来提供从失败点到程序入口的函数(和给予的参数)的列表。

diag_suppress

> 用于屏蔽KEIL报出的警告，

g

> 该选项可以使生成的可执行程序包含调试信息，用于调试。

O

> 指定编译优化级别，例如 `-O3`，代表开启3级编译优化

cpu

> 该选项用于选择处理器类型
>
> `armcc`编译器一般使用使用`-cpu 7-A`或者`-cpu Cortex-A8`来指定指令集架构和CPU类型。
>
> `gcc`编译器一般使用处理器选项`-mfpu=neon`和`-mcpu`来指定`cpu`类型。如`-mcpu=cortex-a5`

fpu

> 设置目标FP架构版本。
>
> 一般来说，对于不带 FPU 的处理器，ARM提供了一个浮点支持软件库用于计算浮点数：fplib，这也就是所谓的软件浮点，本质上就是把浮点运算转成**浮点运算的函数调用和库函数调用**(即用整数运算模拟浮点运算)，没有FPU的指令调用，也没有浮点寄存器的参数传递。与之对应的则是硬件浮点，编译器将代码直接编译成硬件浮点协处理器(浮点运算单元FPU)能识别的指令，这些指令在执行的时候ARM核直接把它转给协处理器执行。FPU 通常有一套额外的寄存器来完成浮点参数传递

和运算。使用实际的硬件浮点运算单元(FPU)会带来性能的提升。而如果需要使用硬件浮点，就会牵扯到编译选项的设置了，主要有以下三种：

- `-mfpu =name`（neon or vfp）指定FPU 单元
- `-mfloat-abi= name`（soft、hard、 softfp）：指定软件浮点或硬件浮点或 兼容软浮点调用接口
- `-mfpu`可以指定使用FPU单元类型，以ARM A76为例有两种：`VFP`和`Neon`

VFP与Neon

- VFP (vector floating-point)

VFP是一个按顺序工作的浮点协处理器, 它对一组输入执行一个操作并返回一个输出。目的是加快浮点计算,支持单精度和双精度浮点。

- NEON

Neon是SIMD (Single Instruction Multiple Data)， 支持单指令多数据操作，意味着在执行一条指令期间，将对多达16个数据集并行执行相同的操作，支持整数和单精度浮点数向量化(并行)操作。

- 二者区别

VFP 支持单精度和双精度浮点，顺序执行，目的加快浮点计算

Neon 是SIMD，支持单指令多数据操作，支持整数和单精度浮点数向量化(并行)操作，不支持双浮点 。Neon最大的好处是如果想要执行矢量操作，如对视频的编码/解码， 它可以并行执行单精度浮点（float）操作。 从而提高对视频编码/解码性能

VFP 和Neon是共用浮点寄存器，只是执行的指令不同。

soft、hard、 softfp

在有fpu的情况下，可以通过`-mfloat-abi`来指定使用哪种，有如下三种值：

- **soft**：不用fpu计算，即使有fpu浮点运算单元也不用。
- **armel**：(arm eabi little endian) 也即**softfp**，用fpu计算（即会将浮点运算编译成对应的浮点指令），但是传参数用普通寄存器传，这样中断的时候，只需要保存普通寄存器，中断负荷小，但是参数需要转换成浮点的再计算。
- **armhf**：(arm hard float)也即**hard**，用fpu计算，传参数用fpu中的浮点寄存器传，省去了转换性能最好，但是中断负荷高。

而arm64，64位的arm默认就是hard float的，因此不需要hf的后缀。**kernel、rootfs和app编译的时候，指定的必须保持一致才行。**

使用softfp模式，会存在不必要的浮点到整数、整数到浮点的转换。而使用`hard`模式，在每次浮点相关函数调用时，平均能节省20个CPU周期。

mthumb

Assemble Thumb instructions，汇编 Thumb 指令

使用这个编译选项生成的目标文件是Thumb指令的，与之对应的则是`-marm`,它的意义是，使用编译选项生成的目标文件是ARM指令的。

mthumb-interwork

> 编译选项生成的目标文件就是Thumb指令的，但是可以被其他的ARM的目标文件交叉调用。
>
> 一般如果工程中需要一部分文件目标文件编译为ARM指令，一部分目标文件编译为Thumb指令时， 可以在这两部分的编译选项中都加入"`-mthumb-interwork`"选项，这样就可以在后面将这两部分链接为一个可执行文件。

endian

> 指定目标字节序，有一下两种模式：
>
> - little 小端模式，高位字节排放在内存的低地址端，低位字节排放在内存的高地址端。
> - big 大端模式，低位字节排放在内存的低地址端，高位字节排放在内存的高地址端。

ffunction-sections / fdata-sections

> gcc链接操作是以 section 作为最小的处理单元，只要一个 section 中的某个符号被引用，该 section 就会被加入到可执行程序中去。
>
> 因此，gcc在编译时可以使用 `-ffunction-sections` 和 `-fdata-sections` 将每个函数或符号创建为一个 sections，其中每个 sections 名与 function 或 data 名保持一致。而在链接阶段， `-Wl,-gc-sections` 指示链接器去掉不用的section（其中-wl, 表示后面的参数 -gc-sections 传递给链接器），简而言之，就是只把实际被引用的section添加到可执行程序中去，减少最终的可执行程序的大小。