# 数据存储方式说明 副本



# 字节顺序

byte ordering：大于一个字节的数，采用little-endian（小端字节序），即Least Significant Byte（LSB）放于存储低地址处

bit ordering：在一个字节内的位，采用小端序，即Least Significant bit（LSb）放于低地址

## 浮点数存储顺序


E7和M22分别为指数和尾数的最小位，如同十进制中的个位

- 从高到低依次存储符号位S，指数E和尾数M
- 指数S和尾数M都是小端序，MSb在地址最高处，LSb在地址最低处
- （上图中的E0和M0是都指MSb，可能与一般记号不同）

# 读写方式

Core Memory的每个地址（Cell）为16B，一次读写的位宽支持16B

# 存储形式

**本芯片可以处理一维到四维的数据**。而存储器（SRAM和DRAM）只有深度Depth和宽度Width这两个维度，可以视为一个二维的表格。每个地址（存储中的一个Cell）代表表格的一行，每个地址中存储的不同数即为一行中的不同列。所谓的“存储方式”即不同维度数据到这个二维表格的一种映射。下面对各种类型的数据在Core Memory中的存储方式和本文档中的表示方法进行讨论。

一个 $N$ 维的数据形状可以表示为 $(D_1, D_2, D_3, ..., D_n)$ ， $Di$ 代表维度的名称；这一 $N$ 维数据中每个数的坐标可以用 $(d_1, d_2, d_3, ..., d_n)$ 表示，其中 $d_i$ 代表这个数在维度 $D_i$ 上的位置。在本芯片文档中， $(D_1, D_2, D_3, ..., D_n)$ 的表示方法中维度的顺序代表了这一数据在Core Memory中的存储方式。

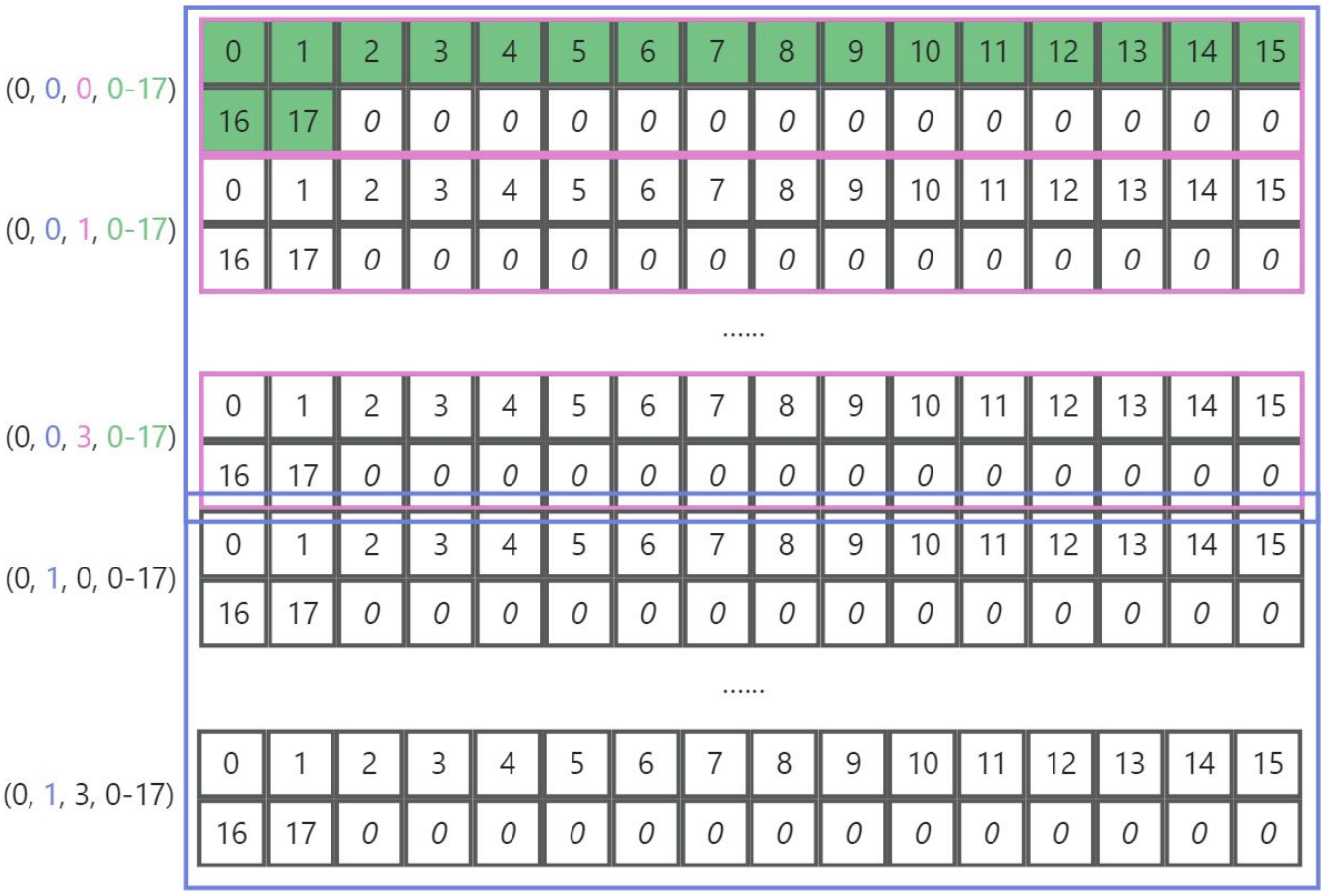
具体而言， $(D_1, D_2, D_3, ..., D_n)$ 这一表示方法中越靠后的维度上相邻的数在存储中的位置越接近：

- 最后一个维度 $D_n$ 优先放在存储中的一个Cell内以及相邻Cell
  - 例如，如果 $d_n^{max} \times bitwidth \leq 16Byte$ ， $(d_1, d_2, d_3, ..., 0 \sim d_n^{max})$ 这些数会放在同一Cell中；
  - 如果 $d_n^{max} \times bitwidths > 16Byte$ 则先按照16Byte分组后再放在相邻的Cell中；
  - 如果 $d_n^{max} \times bitwidths$ 不能被16Byte整除，则在最后一个Cell中补零

（其中 $bitwidths$ 代表数据类型的位宽）

- 固定前 $i-1$ 个维度的坐标 $(d_1, d_2, d_3, ..., d_{i-1})$ ，存储中先排列 $(d_1, d_2, d_3, ..., d_{i-1}, \boldsymbol{d_i}, 0 \sim d_{i+1}^{max}, 0 \sim d_{i+2}^{max}, ..., 0 \sim d_n^{max})$ ，紧接着排列 $(d_1, d_2, d_3, ..., d_{i-1}, \boldsymbol{d_i} + \mathbf{1}, 0 \sim d_{i+1}^{max}, 0 \sim d_{i+2}^{max}, ..., 0 \sim d_n^{max})$

以三维Int8（ $bitwidths = 1Byte$ ）数据为例，假设维度顺序为 $(A, B, C)$ ，具体形状为 $(2, 4, 18)$ ，则它在存储的排列方式如下：


(0, 0, 0, 0-17)
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
16 17 0 0 0 0 0 0 0 0 0 0 0 0 0 0
(0, 0, 1, 0-17)
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
16 17 0 0 0 0 0 0 0 0 0 0 0 0 0 0
.....
(0, 0, 3, 0-17)
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
16 17 0 0 0 0 0 0 0 0 0 0 0 0 0 0
(0, 1, 0, 0-17)
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
16 17 0 0 0 0 0 0 0 0 0 0 0 0 0 0
.....
(0, 1, 3, 0-17)
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
16 17 0 0 0 0 0 0 0 0 0 0 0 0 0 0