第6章 存储器

存储器是计算机系统重要组成部分，主要功能就是存储程序和数据，存储器存储信息的最小单位是二进制数的一位，仅可以存储0和1。本章首先介绍存储器的功能与分类，然后再介绍随机存储器、只读存储器等几种重要的存储器类型，最后再给出有关存储器的实验，以进一步加深读者对存储器的理解。

6.1 存储器的功能与分类

6.1.1 按存储介质分类

存储器按存储介质可分为半导体存储器、磁存储器、光存储器和Flash存储器。

半导体存储器可分为双极晶体管存储器（TTL型）和MOS晶体管存储器。TTL型存储器存取速度较快、功耗较大、集成度低，多用来制作高速缓存；MOS型存储器功耗小，集成度高，存取速度略低，大多用来作为主存。

磁表面存储器是用磁性材料做成的，典型代表有磁盘。常用于存放操作系统、程序和数据，是主存储器的扩充。

光存储器是基于光学原理的存储器，CD、DVD是其典型代表，用途在逐渐弱化。

Flash存储器（闪存）是一种非易失性存储器，其存储单元为三端器件，与场效应管有相同的名称：源极、漏极和栅极。栅极与硅衬底之间有二氧化硅绝缘层，用来保护浮置栅极中的电荷不会泄漏，使得存储单元具有了电荷保持能力，像是装进瓶子里的水，当倒入水后，水位就一直保持在那里，直到再次倒入或倒出，具有记忆能力。闪存约1990年前后应用于市场，目前成为便携式数字设备的最重要存储介质。

6.1.2 按功能分类

根据存储器在计算机系统中所起的作用，存储器可以分为主存储器、辅助存储器、高速缓冲存储器和控制存储器等。

主存储器又称内存，主要由随机存储器（RAM）组成，用来存放计算机运行期间所需要的程序和数据，是计算机各部件信息交流的中心。

辅助存储器简称外存，用来存储大量暂时不参与运算的程序和数据以及需要长期保存的运算结果。通常外存不直接和计算机的其他部件交换数据，只是成批地与主存交换信息。

高速缓冲存储器简称Cache，用来存放主存中经常使用的内容的备份，它被用在CPU与主存之间，起到速度缓冲的作用。

控制存储器用来存放实现全部指令系统的所有微程序，是一种只读型存储器，一旦微程序固化，机器运行时则只读不写。每次读出一条微指令 ，运行这条微指令；再重复这一过程直到运行结束。

6.1.3 按存取方式分类

对存储器的访问实质上就是对存储单元的访问，根据寻找访问单元的方式，可以将存储器分为随机访问存储器和顺序访问存储器两类。

随机访问存储器（Random Access Memory，RAM）是应用较多的存储器，大多半导体存储器都属于随机访问存储器。这里的随机指的是被访问单元的位置是随机的、独立的。换言之，RAM中的每一个单元都可以独立地直接访问。在不考虑访问局部性原理等统计策略级问题的前提下，每两次连续访问之间的关系是独立的，两次访问单元的位置与访问时间无关。在计算机系统中，RAM主要用作主存和高速缓冲存储器。

顺序访问存储器（Sequential Access Memory，SAM），SAM中的单元一般不能被独立访问。当有存取操作时，SAM先根据待访问单元地址定位至一个储存块（存储单元集合），再按顺序寻找到待访问的单元进行存取操作。最为极限的情况下，SAM会从存储体的第一个单元开始按顺序逐个查找，直至找到待访问单元。因此，相比RAM而言，SAM往往具有较低的访问速度，但在价格、容量等指标上有优势。目前，较少被使用。

6.2 随机存储器与只读存储器

随机存取存储器（RAM）和只读存储器（ROM）都是半导体存储器的典型代表。

6.2.1 RAM

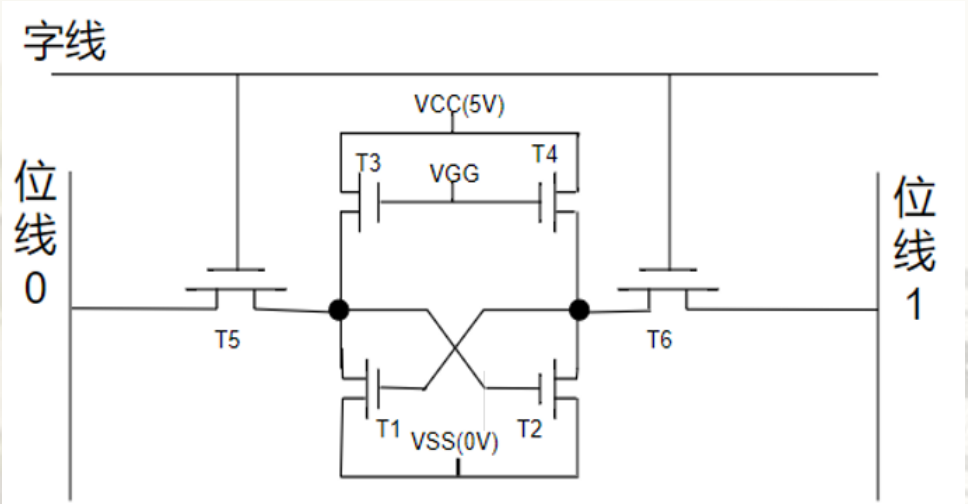
计算机中的内存条即是RAM。RAM可以随时从任何一个指定地址的存储单元中读取数据，也可以随时将数据写到任何一个指定地址的存储单元中。RAM是一种易失性存储器，掉电后将丢失数据，即一旦断开电源，RAM中所存储的信息就会随之丢失，不利于信息的长期保存。一般作为数据存储器使用，暂时性的存取数据。RAM用来存放现场输入的数据或者存放可以更改的运行程序和数据，常用来做堆栈。

根据RAM工作原理不同，可以将其分为两类：基于触发器原理的静态随机存储器（SRAM）和基于分布电容电荷存储原理的动态随机存储器（DRAM）。

1. SRAM

SRAM是当前应用最广的一种随机存储器，该种存储器只要保持有源状态，其中所储存的数据就可以一直保持而不丢失。

SRAM基本原理：由六管组成，T1和T2管构成双稳态正反馈电路，T3和T4管作为负载管，T5和T6管作为控制字线和位线的门控管。当没有访问操作时，字线处于低电平状态，两根位线上呈高电平状态。由于门控管T5和T6截止，T1和T2将保持原有的稳定状态。当存储元进行写入访问时，字线上需加载高电平。



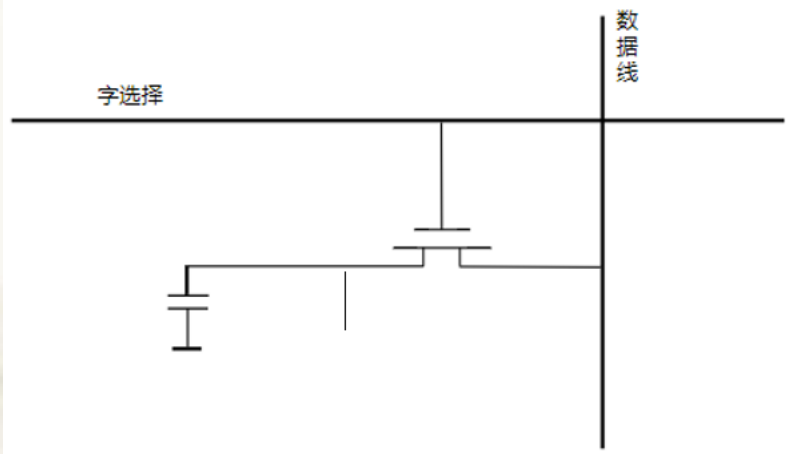
SRAM的优点是性能稳定，不需要外加额外刷新电路，工作速度较快，缺点是每个存储单元都由6个MOS管组成，集成度较低，功耗较大。一般用于规模较小的快速存储器。

2. DRAM

DRAM利用电容内是否储存有电荷来代表一个二进制位是1还是0。

常见有四管MOS型、三管MOS型和单管MOS型。

单管MOS型动态存储单元结构：当存储单元被选中后，字选择线加载高电平，使得控制管T被打开，电流在数据线和存储电容C之间流动。



DRAM的读取是破坏性的读取，必须在读出后进行重写工作，即还原读取前电容的存储状态。

进行周期性的重写工作，否则无法长期保持存储状态。

单管MOS型DRAM的每个存储单元只由一个MOS管和一个电容组成。

优点：芯片集成度高，功耗低；

缺点：芯片本身不具有向电容充电的刷新功能，需外加刷新逻辑电路。

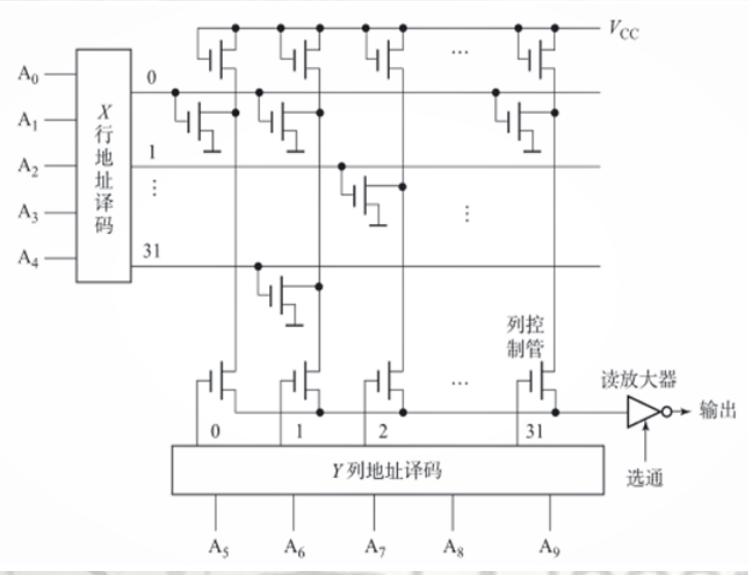
6.2.2 ROM

主板上存储BIOS程序的芯片即是ROM。特点：数据被预先写入，而且一旦被写入，使用时就只能进行数据的读取操作，无法进行更改，且ROM中存储的数据掉电也不会丢失。

ROM的基本存储单元可由二极管、双极型晶体管或MOS管构成。对半导体ROM而言，基本器件分为MOS型和TTL型两种。

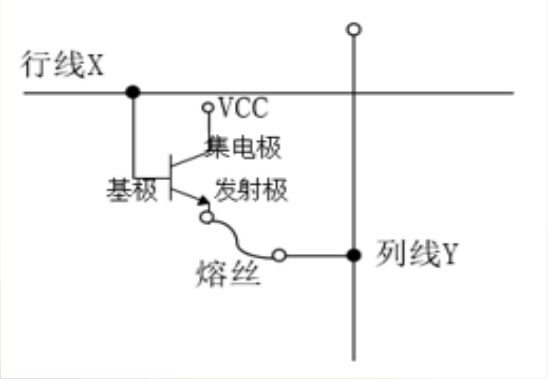
1. 掩模ROM

容量为1K*1位，采用重合法驱动，行、列地址线分别经行、列译码器，各有32根行、列选择线。列选择线各控制一个列控制管，32个列控制管的输出端共连一个读放大器。用行、列交叉处是否有耦合原件MOS管，便可区分原存“1”还是“0”。此ROM制成后不可能改变原行、列交叉处的MOS管是否存在，所以用户时无法改变原始状态的。



2. PROM

PROM时可以实现一次性编程的只读存储器，由双极型电路和熔丝构成的基本单元电路：基线由行线控制，发射级和列线之间形成一条镍铬合金薄膜制成的熔丝（可用光刻计数实现），集电极接到电源VCC。熔丝断和未断可区别其所存信息是“1”或“0”。

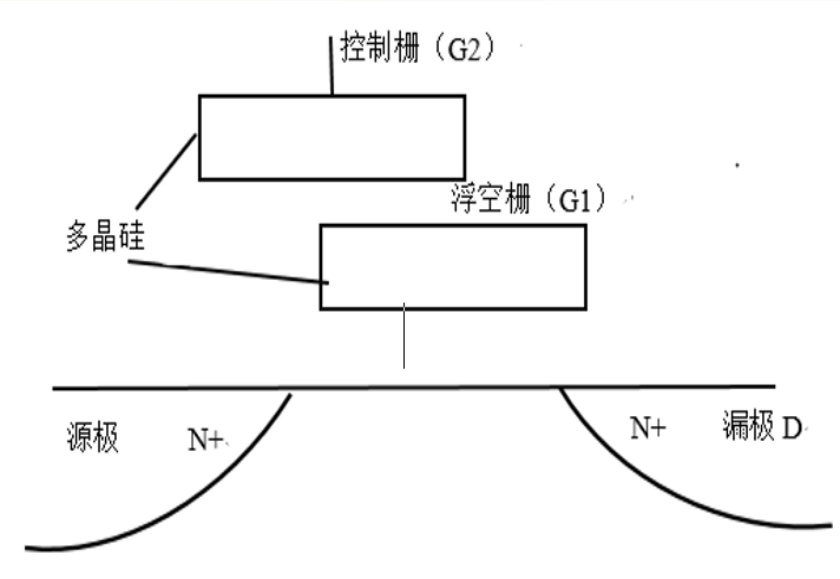


3. EPROM

EPROM是一种可擦除可编程只读存储器。它可以由用户对其所存信息做任意次的改写，目前用的较多的RPROM是由浮动栅雪崩注入型MOS管构成。EPROM的改写有两种方式，一种是用紫外线照射，但擦除时间比较长，而且不能对个别需要改写的单元进行单独擦除或重写；另一种是用电气方法将存储内容擦除再重写；甚至在联机条件下，用字擦除或页擦除方式，既可局部擦写，又可全部擦写，这种EPROM就是EEPROM。

4. EEPROM

EEPROM，带电可编程可擦除存储器，掉电后数据不丢失。EEROM可以在电脑或专用设备上擦除已有信息，重新编程。EEROM具有即插即用的特性。EEPROM的存储元是一个具有两个栅极的MOS管，基本结构：在浮空栅（G1栅）和漏极D之间有一小面积的厚度极薄的氧化层，可产生隧道效应.当控制栅（G2栅）加20V正脉冲时，由隧道效应，电子由衬底注入G1浮栅，利用此方法可将PROM中内容抹为全“1”(即出厂状态)。



使用时，漏极D加20V正脉冲，G2栅接地，浮空栅G1上的电子通过隧道返回衬底，相当于写“0”。读出时，在G2栅加3V电压，若G1栅上有电子积累则MOS管不导通，相当于读取“0”；若G1栅上无电子积累，MOS管导通，相当于读取“1”。EEPROM可进行上千次的重写，数据可存储20年以上，电可擦、按字擦除等功能的实现使得ROM存储器的使用更加灵活便利，使用领域更加广泛，但它的擦除和重写仍需在专门的编程器(写入器)中进行。

6.3 SD卡与高速缓存

6.3.1 SD卡

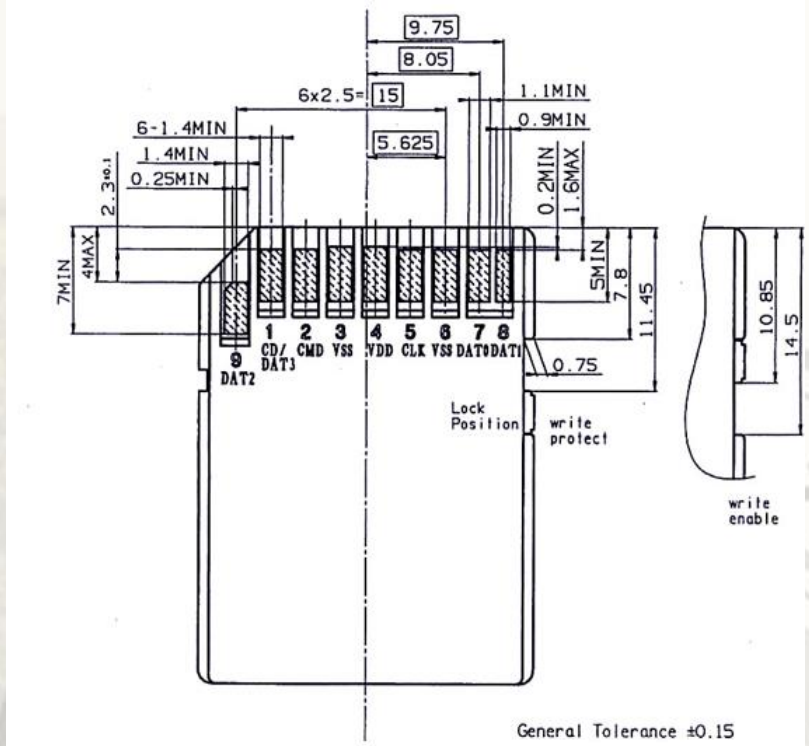
SD卡是一种基于半导体快闪记忆器的新一代记忆设备，容量大、传输速率高、以及安全性高。

外观特征：端子保护；写入保护开关；可正确插入的楔形设计；凹口设计；导槽。

可变时钟频率为0-25MHz通信电压范围，低电压消耗，自动断电以及自动睡醒，智能电源管理，不需额外编程电压，卡片带电插拔保护，支持双通道闪存交叉存取，快写技术。提供超高速闪存访问和高可靠数据存储。

SD卡表面上有9个引脚，目的是把传输方式由串行变成并行，从而提高传输速度。最大的特点：加密功能，保证数据资料的安全保密。

SD卡的接口可以支持SD卡和SPI两种操作模式。



6.3.2 高速缓存

1. 高速缓存的地位和作用

综合成本和容量两方面因素，现代计算机广为采用DRAM构成的内存作为内存实现方法。DRAM的功耗和成本较低，易构成大容量的内存。但DRAM的存取速度相对较慢，很难满足高性能CPU在速度上的要求，同时程序执行所需要使用的指令或数据在存储器中很可能是在同一地址的附近，那么就产生了高速缓冲存储器（Cache）的设计理念，即只将CPU最近需要使用的少量指令或数据以及存放它们的内存单元的地址复制到速度较快的Cache中，以便提供给CPU使用，用少量速度较快的SRAM构成Cache置于CPU和主存之间。这种设计思想利用SRAM的速度优势和DRAM的高集成度、低功耗及低成本的特点。在目前的系统中，均采用了Cache和DRAM内存的组合结构。基于目前的大规模集成电路技术和生产工艺，已经可以在CPU芯片内部放置一定容量的Cache。CPU芯片内部的Cache称为一级（L1）Cache，CPU外部由SRAM构成的Cache称为二级（L2）Cache。目前最新的CPU内部已经可以放置二级甚至三级Cache。

2. 高速缓存的结构及工作原理

当CPU需要数据或指令时，它首先访问Cache，看所需要的数据或指令是否在Cache中，方法是将CPU提供的数据或指令在内存中存放位置的内存地址，首先与Cache中已存放的数据或指令的地址相比较，若相等，说明可以在Cache中找到需要的数据或指令，称为Cache命中；若不相等，说明CPU需要的数据或指令不在Cache中，称为未命中，需要从内存中提取。若CPU需要的指令或数据在Cache中，则不需任何等待状态，Cache就可以将信息传送给CPU；若数据或指令不在Cache中，存储器控制电路会从内存中取出数据或指令传送给CPU，同时在Cache中拷贝一份副本。这样做是为了防止CPU以后在访问同一信息时又会出现未命中的情况，从而降低CPU访问速度相对较慢的内存的概率。CPU访问Cache的命中率越高，系统性能就越好。Cache的命中率取决于三个因素：Cache的大小、Cache的组织结构和程序的特性。就Cache组织结构而言，有三种类型的Cache：全相连映像方式、直接映像方式和组相连映像方式。

1）全相连映像方式

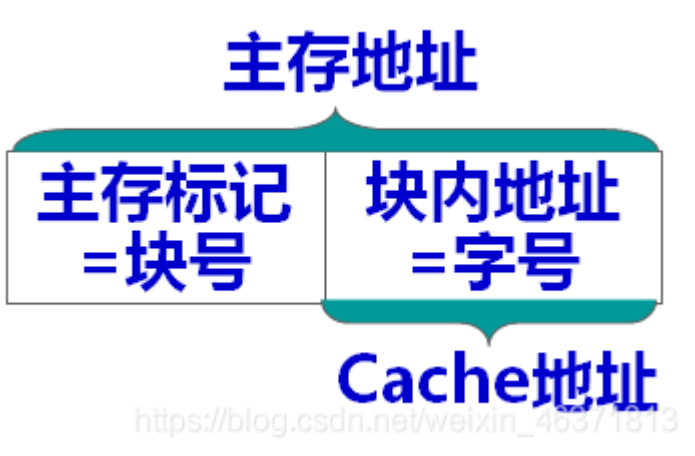
全相连映像方式的Cache中，任意主存单元的数据或指令可以存放到Cache的任意单元中，两者之间的对应关系不存在任何限制。在Cache中，用于存放数据或指令的静态存储器称为内容Cache，用于存放数据或指令在内存中所在单元的地址的静态存储器称为标识Cache。全相联映像的块间映射：对于全相连映像Cache，Cache中存储的数据越多，命中率越高。但增加Cache的容量带来的问题是，每次访问内存都要进行大量的地址比较，既耗时同时效率也低。若Cache的容量太小，由于命中率太低，CPU就要频繁地等待操作系统将Cache中的信息换入换出，因为在向Cache中写入新信息之前，必须将Cache中已有的信息保存在主存中。

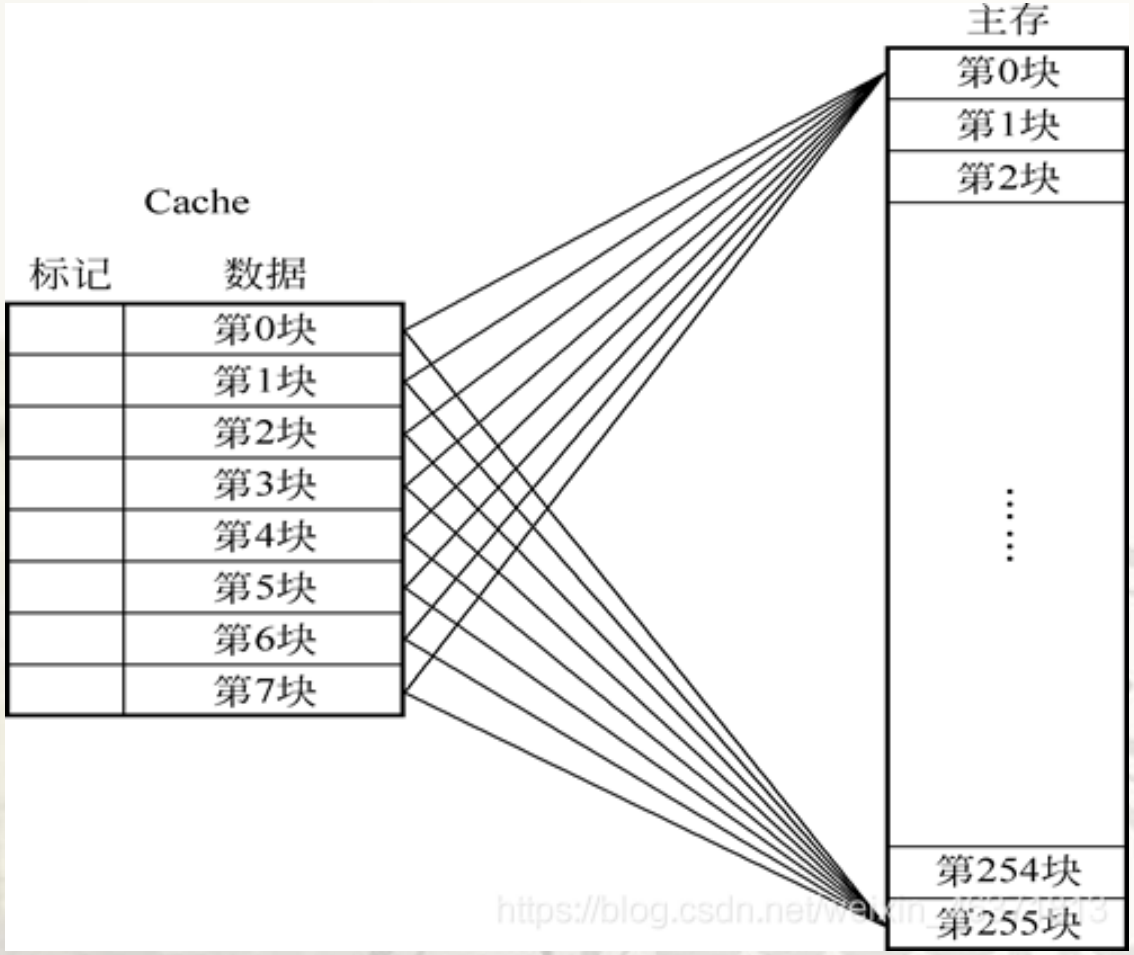
1）全相连映像方式

全相联映射即主存块可以映射到cache中的任意一块，即说白了就是一对多，‘一’指的是主存中的任意一块，‘多’指的是cache的每一行，即主存块可以映射到cache中的任意一块。

直接来看下面主存与cache之间的全相联映射，如右图：

相对应的地址格式：





1）全相连映像方式地址计算举例

若数据在主存和Cache之间按块传送单位为512字节。Cache大小为8KB，主存容量为1MB ,求其主存的地址格式。

解题思路：

字号：按块传送单位为512字节，即512=2^9,故字号为9位

块号或主存标记：主存容量为1MB,即 1M/512=2^11,故标记号11位。

全相联映射总结：

优点：命中率比较高，cache存储空间利用率高；

缺点：访问相关存储器时，每次都要与全部内容比较，速度低，成本高，因而利用少。

2）直接映像方式Cache

直接映像Cache与全相连映像Cache完全相反，它只需要做一次地址比较即可确定是否命中。在这种Cache结构中，地址分为两部分：索引和标识。索引是地址的低位部分，直接作为内容Cache。单元的地址定位到内容Cache的相应单元，而地址的高位部分作为标识存储在标识Cache中。但是，这种方式所需的逻辑电路甚多，成本较高，实际的Cache还要采用各种措施来减少地址的比较次数。每个主存块只与一个缓存块相对应，映射关系式为：

$$I=j \bmod C \text{ 或 } I=j \bmod 2C$$

其中，I为缓存块号；j为主存块号；C为缓存块数。映射结果表明每个缓存块对应若干个主存块，直接映射方式主存块和缓存块的对应关系如表所示。

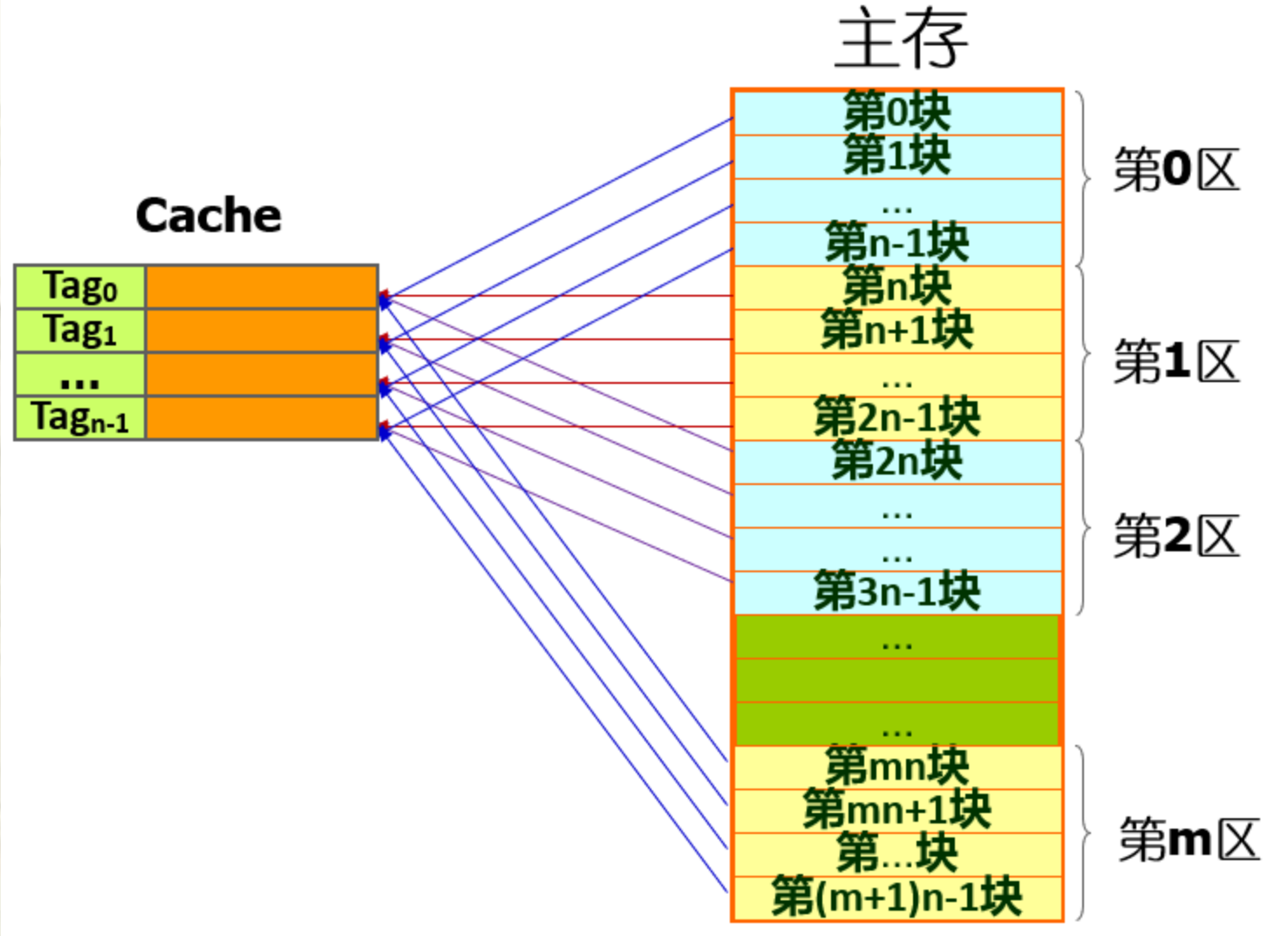
缓存块	主存块
0	0，C，2C，……，2^n-C
1	1，C+1，2C+1，……，2^n-C+1
……	……
C-1	C-1，2C-1，3C-1，……，2^n-1

2）直接映像方式Cache

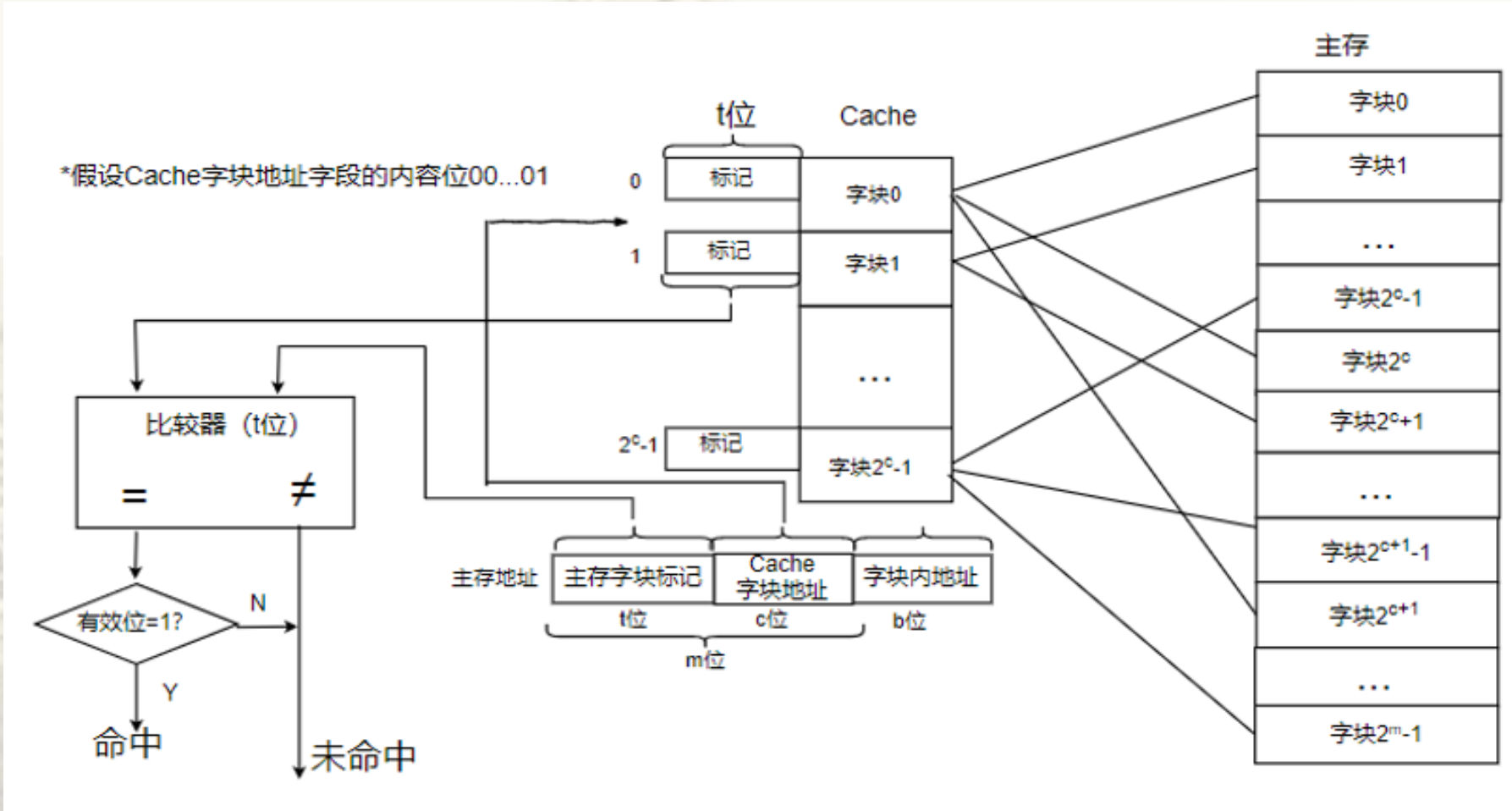
直接映射：说白了就是多对一的映射方式。‘多’指的是主存中的不同分区下的块号，‘一’指的是cache中特定的某一个块。

直接来看下面主存与cache之间的直接映射，如右图：

地址构成格式：



直接映射方式Cache中，Cache字块与主存块对应的关系



1）直接映像方式地址计算举例

有一处理机，主存容量1MB，字长1B，块大小16B；Cache容量4KB，若cache采用直接映射，请给出2个不同标记的内存地址，它们映射到同一个cache行。

解题思路：

(1) 首先要写出内存地址，我们就首先要求出主存的地址格式：区号、块号、字号

(2) 区号：主存容量1MB，Cache容量4KB，故：1MB/4KB=2^8,即区号或标记位8位

(3) 块号：Cache容量4KB，块大小16B，故：4KB/16B=2^8，即块号8位。

(4) 字号：字长1B，块大小16B，故：2^4,即字号4位。

所以：题目中的映射到同一个cache行，即只要块号相同即可满足。

直接映射总结：

优点：地址映射方式简单，数据访问时只需要检查块号是否相等即可，因而能得到比较快的访问速度。

缺点；替换操作频繁，命中率比较低。

3）组相连映像Cache

组相连映像Cache是介于全相连映像Cache和直接映像Cache之间的一种结构。在直接映像Cache中，每个索引在Cache中只能存放一个标识。而在组相连映像中，对应每个索引，在Cache中能存放的标识数量增加了，从而增加了命中率。组相联映射是对直接映射和全相联映射的一种折中。它把Cache分为Q组，每组有R块，并有以下关系：

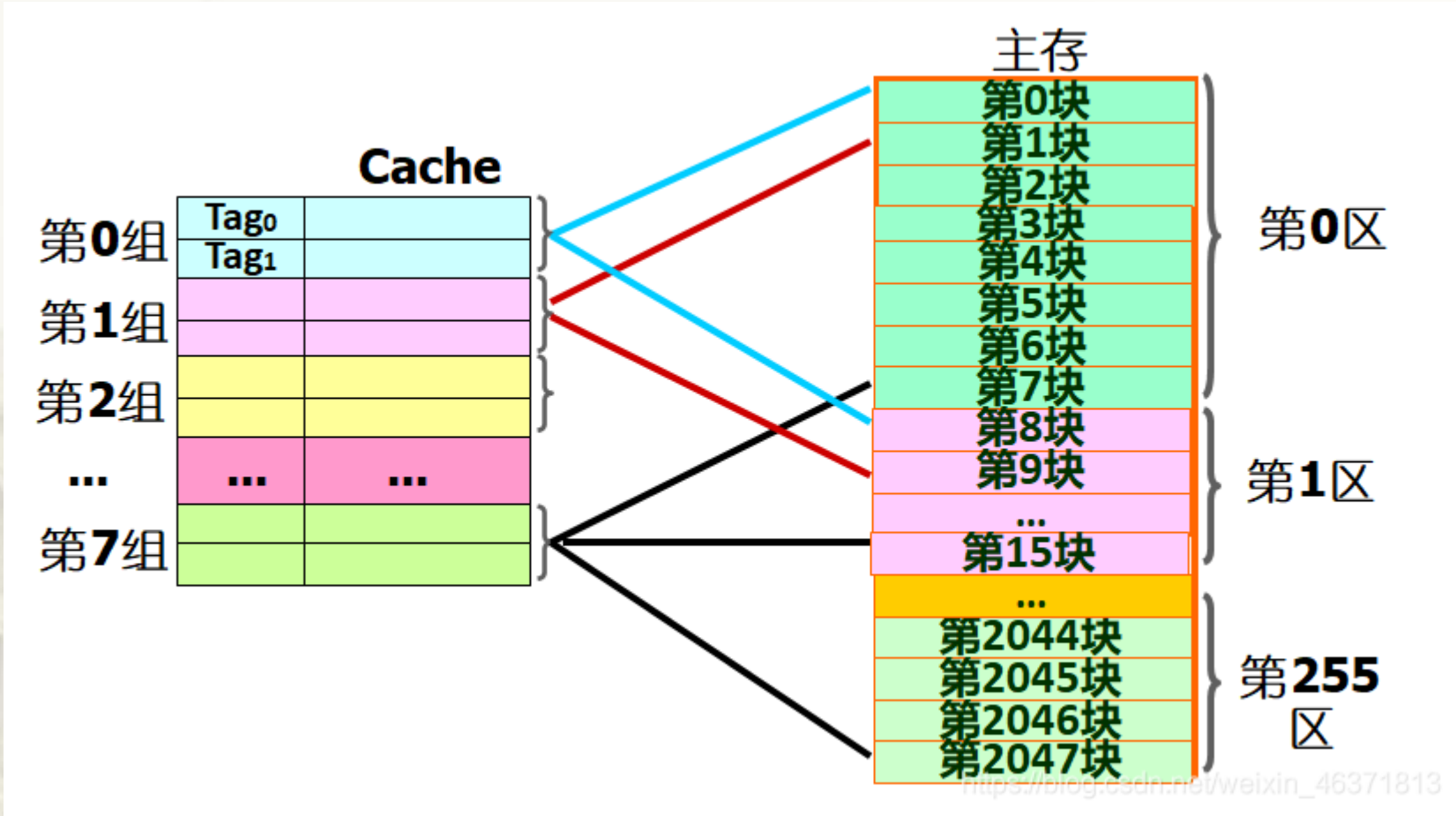
$$i=j \bmod Q$$

其中，i为缓存的组号；j为主存的块号。某一主存块按模Q将其映射到缓存的第i组内。

即组内是全相联映射，组间是直接映射。

3）组相连映像Cache

根据右图：知道cache被分为了8组，每组又分为了2块，右边主存部分被分成了256个区，每个区8块，每个区的8块与cache中的8组不是巧合，而是为了实现组间直接映射，然疑问来了，当主存的某一块按直接映射方式规则进行映射后到了cache中相应的组，而此时在cache中每组有两块，这时候就是实行组内全映射。



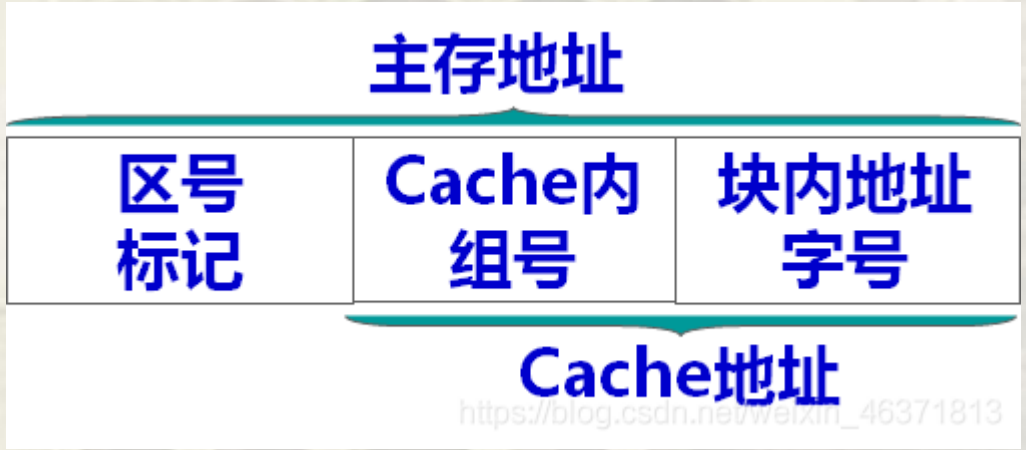
3）组相连映像Cache

因为组相联映射是直接映射和全相联映射的一个折中形式，那么必然也就存在下面两种特殊的形式：

1>当cache中每组内的块数变为1时，这就变成了直接映射；

2>当cache中只有一组时，这就变成了全相联映射。

组相联映射相对应的主存地址格式（和直接映射方式很像，只是块号变成了组号）：



1）组相连映像方式地址计算举例

某计算机按字节寻址，主存有2K个块，每块32个字节。 Cache由64个块组成，每组8块（8路组相联）。请表示主存地址格式。给内存地址为A21FH和C028H两个地址对应的标记、组号和字号。

解题思路：

(1) 字号：每块32个字节，即 32=2^5,故字号5位。

(2) 组号：Cache由64个块组成，每组8块，即64/8=8=2^3，故组号3位。

(3) 区号标记：主存有2K个块，又因为组间是直接映射(所以把cache中的组数看作直接映射中的块)，即2k/8=2^8,故区号8位。

(4) 所以，主存格式为：区号8位、组号3位、字号5位。

A21FH=1010001000011111B故对应的标记：10100010，组号：000，字号：11111

C028H=1100000000101000B故对应的标记：11000000、组号：001、字号：01000

组相联映射总结：

优点：块的冲突概率比较低，块的利用率大幅度提高；

缺点：实现难度和造价要比直接映射高。

6.4 Flash存储器

6.4.1 Flash在线编程的通用基础知识

- **Flash存储器的特点：**
  - ◆具有固有不易失性、电可擦除、可在线编程、存储密度高、功耗低和成本较低等特点。
  - ◆不需要后备电源来保持数据
  - ◆可在线编程，可以取代电可擦除可编程只读存储器（EEPROM），用于保存运行过程中的参数。
  - ◆Flash存储器已经成为MCU的重要组成部分。

6.4 Flash存储器

6.4.1 Flash在线编程的通用基础知识

- **Flash编程模式**
  - **in-circuit programming (ICP)：监控模式或者写入器编程模式。一般用于通过编程器将程序固化到Flash存储器中，可以整片擦除flash，也可以分扇区擦除。通过JTAG，SWD协议或者boot loader程序装载用户应用程序。**
  - **in-application programming (IAP)：用户模式或在线编程模式。在程序运行过程中，一般用于对Flash存储区的数据或程序进行更新，通过运行Flash内部程序对Flash其他区域进行擦除与写入。通过I/Os、USB、CAN、UART、I2C、SPI等方式。**

6.4 Flash存储器

6.4.1 Flash在线编程的通用基础知识

- **Flash编程特点：**
  - 需要专门的编程过程。
  - 基本操作包括：擦除（Erase）、写入（Program）和读取(read)；擦除操作的含义是将存储单元的内容由二进制的0变成1，而写入操作的含义是将存储单元的某些位由二进制的1变成0。
  - 擦除操作：必须以页为单位擦除或者整个FLash擦除。
  - 写入操作：必须是没有写入过数据的单元，写入之前必须先检测是否已经擦除，如果没有擦除，则会导致写入数据失败。
  - 读取操作：可以随机读取

6.4.2 Flash驱动构件知识要素分析

Flash具有初始化，擦除和写入、读取（按逻辑地址）、读取（按物理地址）、保护、解除保护、判空等8种基本操作。可将初始化，擦除和写入等8种基本操作所对应的功能函数共同放置在命名为flash.c的文件中，并按照相对严格的构件设计原则对其进行封装，同时配以命名为flash.h的头文件，用来定义模块的基本信息和对外接口。

阅读STM32L4XX参考手册关于Flash一章：P77-113

1、Flash擦除操作流程、批量擦除流程

2、Flash标准编程流程和快速编程流程

6.4.2 Flash驱动构件知识要素分析

- **Flash页擦除操作流程：(2Kbyte)**

1. Check that no Flash memory operation is ongoing by checking the BSY bit in the Flash status register (FLASH_SR).

2. Check and clear all error programming flags due to a previous programming. If not, PGSERR is set.

3. Set the PER bit and select the page you wish to erase (PNB) in the Flash control register (FLASH_CR).

4. Set the STRT bit in the FLASH_CR register.

5. Wait for the BSY bit to be cleared in the FLASH_SR register.

```
//=====================================
//函数名称：flash_erase
//函数返回：函数执行执行状态
    ：0=正常；1=异常。
//参数说明：sect：目标扇区号（范围取决于实际芯片，例如 STM32L433:0~127,每扇区2KB;
//功能概要：擦除flash存储器的sect扇区
//=====================================
```

```
uint8_t flash_erase(uint16_t sect){
        //等待之前最后一个flash操作完成
while( (FLASH->SR & FLASH_SR_BSY) != 0U);
        //清闪存即时编程位
         FLASH->CR &= ~FLASH_CR_PG;
         //使能扇区擦除
        FLASH->CR |= FLASH_CR_PER;
        //设置擦除的扇区
        FLASH->CR &= ~FLASH_CR_PNB;
        FLASH->CR |= (uint32_t)(sect << 3u);
        //开始扇区擦除
        FLASH->CR |= FLASH_CR_STRT;
        //等待擦除操作完成
while( (FLASH->SR & FLASH_SR_BSY) != 0U);
        //禁止扇区擦除
        FLASH->CR &= ~FLASH_CR_PER;
    return 0;  //成功返回
}
```

6.4.2 Flash驱动构件知识要素分析

表 6-2 Flash 常用接口函数

序号	函数			形参	
	简明功能	返回	函数名	英文名	中文名
1	flash 初始化	无	flash_init		
2	扇区擦除	uint_16	flash_erase	sect	扇区号
3	向指定扇区写数据	uint_8	flash_write	sect	扇区号
				offset	偏移量
				N	写入数据长度
				buff	写入数组
4	向指定地址写数据	uint_8	flash_write_physical	addr	指定物理地址
				cnt	写入数据长度
				*buff	写入数据
5	从扇区读数据	无	flash_read_logic	*dest	存放读出数据
				sect	扇区号
				offset	偏移量
				N	读出数据长度
6	从物理地址读数据	无	flash_read_physical	*dest	存放读出数据
				addr	目标地址

6.4.2 Flash驱动构件知识要素分析

				N	读出数据长度
7	保护扇区	无	flash_protect	sect	扇区号
8	判断指定区域是否为空	uint_8	flash_isempty	*buff	获取判空区域数据
				N	判断区域大小
说明	（1）关于 flash 初始化的说明：做 flash 模块的其他操作前，必须要先调用 flash_init 初始化 flash 模块。 （2）关于 flash_erase 的说明：擦除成功后，扇区内存储的数据均是 0xFF （3）关于 flash_write 的说明：写之前最好先擦除要写入的扇区 （4）关于 flash_write_physical 的说明：和 flash_write 类似，需要传入物理地址 （5）关于 flash_read_logic 的说明：读指定扇区数据，需要传入扇区号，偏移里 （6）关于 flahs_read_physical 的说明：读指定地址数据，需传入目标地址 （7）关于 flash_protect 的说明：设为保护扇区后，无法对扇区进行写，擦除操作 （8）关于 flash_isempty 的说明：判断指定区域是否为空				

6.4.3 Flash驱动构件的使用方法

Flash头文件中给出了Flash中8个最主要的基本构件函数。

初始化函数直接调用即可，无入口参数及返回值。擦除和写入操作类似，都返回擦除/写入的结果（正常/异常）。写入操作入口参数较多，按扇区号写入还需要传入偏移量，写入数据长度，写入数据的首地址；按物理地址写入只需要传入物理地址和写入数据长度，写入数据的首地址。读取操作与此类似，按物理地址读取需要将直接地址换成扇区号和偏移量。

向64扇区1字节开始的地址写入32个字节“Welcome to Soochow University!”，参考程序“Exam6_1”

（1）首先，要进行初始化Flash模块。

```
bl flash_init    //初始化flash模块
```

（2）因为执行写入操作之前，要确保写入区在上一次擦除之后没有被写入过，即写入区是空白的（各存储单元的内容均为0xFF）。所以，在写入之前要根据情况是否先执行擦除操作，即擦除64扇区。

```
mov r0,#0x40
bl flash_erase    //擦除0x40（64）扇区
```

（3）通过封装好的入口参数进行传参，进行写入操作。64扇区1字节开始的32个字节内写入“Welcome to Soochow University!”。

```
flash_Content:
    .string  "Welcome to Soochow University!\r\n"    //即将要写入flash的内容
ldr r0,=addr                  //r0=要写入的flash扇区的首地址
mov r1,#0x20                  //r1=写入数据长度
ldr r2,=flash_Content         //r2=写入数据首地址
bl flash_write_physical       //调用物理地址写函数向指定地址写数据
```

（4）按照逻辑地址读取时，定义足够长度的数组变量params，并传入数组的首地址作为目的地址参数，传入扇区号、偏移地址作为源地址，传入读取的字节长度。例如，64扇区1字节开始的地址读取32字节长度字符串。

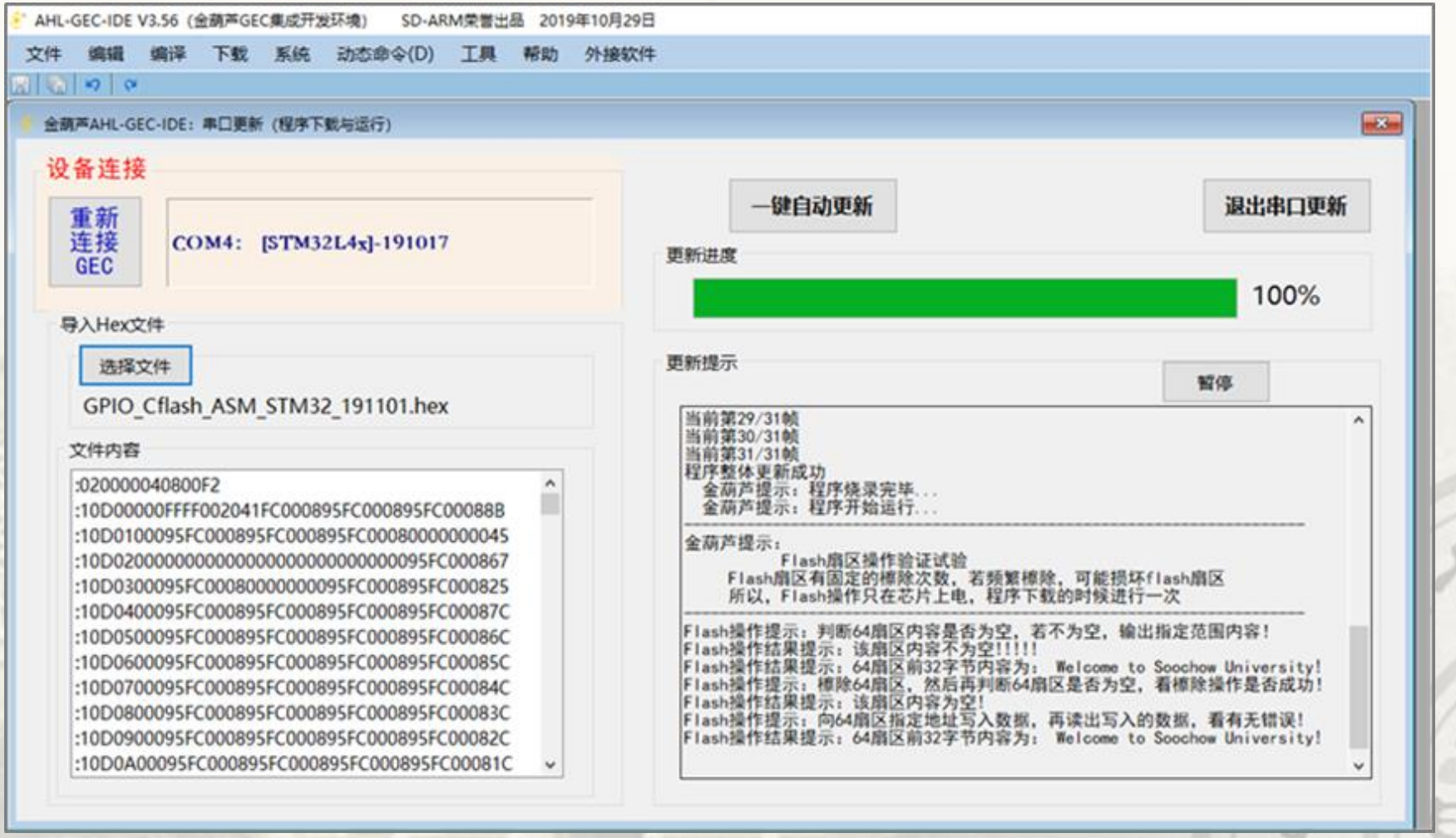
```
ldr r0,=flash_ContentDetail     //r0=要读出的数据存放的首地址
mov r1,#0x40                    //r1=所要读取的扇区的编号
mov r2,#0x0                     //r2=扇区内的偏移量
mov r3,#0x20                    //r3=要读出的内容的长度
bl flash_read_physical          //调用逻辑读函数
```

（5）按照物理地址直接读取时，定义足够长度的数组变量paramsVar，并传入数组的首地址作为目的地址参数，传入直接地址数作为源地址，传入读取的字节长度。例如，从0x00001F00地址处读取存放在此处的1字节长度的全局变量值。

```
flash_ContentDetail:
    .string  "!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!\r\n"   //存储从flash中读出的数据
ldr r0,=flash_ContentDetail              //r0=要读出的数据存放的首地址
ldr r1,=addr                        //r1=要读取的数据所处的地址
mov r2,#0x20                        //r2=要读出的数据长度
bl flash_read_physical                  //调用按物理地址读取函数
```

（6）Flash保护函数的使用非常简单，入口参数为待保护扇区区域号,区域号取值范围为0~127。若需要保护64扇区，仅需要调用函数flash_protect(64)，函数会将64扇区保护起来，函数无返回值。

Flash函数操作结果图



6.5 存储器实验设计举例

本节主要探讨CPU内部寄存器与外部存储器之间的存取速度比较，对于CPU外部存储器，CPU使用三总线方式进行数据存取，增加了操作时间。

本次实验将CPU内部寄存器和外部存储器指定地址存储的值从1加到0x2FFF FFFF（8 0530 6367），外部存储器比内部寄存器存取慢20秒。实验中加1操作都是在内部寄存器进行的，所不同的是为体现CPU使用三总线方式访问外部存储器，因此每次加1后都将具体的数值存到外部存储器的指定地址处，以模仿三总线存取。

具体操作参考程序“Exam6_2”工程

6.6 实验三：存储器实验

作业：1~4