计算机组成

简单计算机模型

CPU 基本组成

1. 基本知识：负责提取程序指令，并对指令进行译码，然后按程序规定的顺序对正确的数据执行各种操作
2. 基本组成：
    1. 数据通道：它由存储单元（寄存器）和算数逻辑单元（对数据执行各种操作）所组成的网络。这些组件通过总线（传递数据的电子线路）连接起来，并利用时钟来控制事件。
    2. 控制单元：该模块负责对各种操作进行排序并保证各种正确的数据适时地出现在所需的地方。

Register - 寄存器

- 寄存器 是一种存储二进制数据的硬件设备，位于处理器的内部，因此处理器可以快速访问寄存器中存储的各种信息。
- 计算机通常处理的是一些存储在寄存器中，具有固定大小的二进制字的数据。
- 可以向寄存器写入或读取信息，信息也可以在不同的寄存器中传递。寄存器的编址方式与存储器不同，寄存器是由 CPU 内部的控制单元进行编址和处理。
- 现代计算机系统中由各种不同类型的专用寄存器，或者只能存储数据，只能存放地址或各种控制信息。还有一些通用寄存器，可以在不同时刻存储不同信息。

ALU - 算术逻辑单元

- 算术逻辑单元 在程序执行过程中用于进行逻辑运算。ALU 的各种操作存储会影响状态寄存器的某些数据位的数值。通过控制单元发出的信号，控制 ALU 执行各种运算。

Control Unit - 控制单元

- 控制单元 负责监视所有指令的执行和各种信息的传送过程。
- 控制单元负责从内存提取指令，对这些指令进行译码，确保数据适时出现在正确的位置。
- 控制单元还负责通知 ALU 应该使用哪个寄存器，执行哪些中断服务程序，以及对所需执行的各种操作接通 ALU 中的正确电路。

Bus - 总线

- 总线 是一组导电线路的组合，它作为一个共享和公用的数据通道将系统内各个子系统连接在一起。
- 总线可以实现 **点对点**（***point-to-point***）的方式连接两个特定设备，或者将总线用作一条 **公用通道**（***common pathway***）来连接多个设备，作为 **多点总线** 使用。
- 系统总线：指 CPU、主存、I/O 设备各大部件之间的信息传输线
    - 典型系统总线：
        1. **数据总线（*data bus*）**：用于数据传递的总线，数据总线传递的是必须在计算机的不同位置之间移动的实际信息。
        2. **控制总线（*control line*）**：计算机通过控制总线指示哪个设备允许使用总线，以及使用总线的目的。 > 控制总线也传递有关总线请求、终端和时钟同步信号的相应信号。
        3. **地址总线（*address line*）**：指出数据读写的位置

- 由于总线传递的信息类型不同，使用设备不同，因此总线还能细分成不同种类。
  1. 处理器 - 内存总线 是长度较短的高速总线，这种总线连接处理器和与机器匹配的内存系统，并最大限度地扩充数据传递的带宽。
  2. I/O 总线 通常比处理器 - 内存总线长，可以连接带宽不同的各种设备，兼容多种不同的体系结构。
  3. 主板总线 可将主板上的所有部件连接在一起，让所有设备共享一条总线。

总线结构

1. 单总线结构：将 CPU、主存、I/O 设备（通过 I/O 接口）都挂在一组总线上，允许 I/O 设备之间、I/O 设备与 CPU 之间或 I/O 设备与主存之间直接交换信息。这种结构简单，便于扩充，但所有的传输度通过这组共享总线，易造成计算机系统的瓶颈。
2. 多总线结构：双总线结构的特点是将速度较低的 I/O 设备从单总线上分离出来，形成主存总线与 I/O 总线分开的结构。

总线控制

1. 总线判优控制
   - 在任意时刻，只能有一个设备使用总线。这种共享总线的方式可能会引起通信上的瓶颈。更通 常的情况下，各种设备分为 **主设备** 和 **从设备**。主设备是最初启动的设备，从设备是 响应主设备请求的设备。
   - 对于配备不止一个主控设备的系统，则需要 **总线仲裁**（***bus arbitration***）机制。总线仲裁机制时必须为某些主控设备设定一种优先级别，同时又保证各个主控设备都有机会使用 总线。常用的总线仲裁方案有 4 种：
     1. 菊花链仲裁方式
     2. 集中式平行仲裁方式
     3. 采用自选择的分配式仲裁方式
     4. 采用冲突检测的分配式仲裁方式
2. 总线通信控制
   - 各种信息的传递都发生在一个 **总线周期**（***bus cycle***）中，总线周期是完成总线信息传送所需的时钟脉冲间的时间间隔。
   - **同步总线**（***synchronous***）：由时钟控制，各种事件只有在时钟脉冲到来时才会发生，也就是事件发生的顺序由时钟脉冲来控制。这种通信的优点是规定明确统一，模块间的配合简单一致；缺点是主、从设备间的配合属于强制性同步，具有局限性。
   - **异步总线**（***asynchronous***）：各种控制线都是使用异步总线来负责协调计算机的各种操作，采用较为复杂的 **握手协议**（***handshaking protocol***）来强制实现与计算机其他操作的同步。

Clock - 时钟

- 每台计算机中都有一个内部时钟，用来控制指令的执行速度；时钟也用来对系统中的各个部件的操作进行协调和同步。
- CPU 的每条指令执行都是使用固定的时钟脉冲数目。因此计算机采用 **时钟周期**（***clock cycle***）数目来量度系统指令的性能。

I/O 输入/输出子系统

- 输入 / 输出是指各种外围设备和主存储器（内存）之间的数据交换。通过输入设备可以将数据输入计算机系统；而输出设备则从计算机中获取信息。

- 输入输出设备通常不直接与 CPU 相连，而是采用某种 **接口**（***interface***）来处理数据交换。接口负责系统总线和各外围设备之间的信号转换，将信号变成总线和外设都可以接受的形式。
- CPU 通过输入输出寄存器与外设进行交流，这种数据的交换通常有两种工作方式。
  1. 在 内存交换的输入输出（***memory-mapped I/O***）方式中，接口中的寄存器地址就在内存地址的分配表中。在这种方式下，CPU 对 I/O 设备的访问和 CPU 对内存的访问是完全相同的。这种转换方式的速度很快，却需要占用系统的内存空间。
  2. 在 指令实现的输入输出（***instruction based I/O***）方式中，CPU 需要有专用的指令来实现输入和输出操作。尽管不占用内存空间，但需要一些特殊的 I/O 指令，也就是只有那些可以执行特殊指令的 CPU 才可以使用这种输入输出方式。

存储器组成和寻址方式

- 存储器的地址几乎都是采用无符号整数来表示。正常情况下，存储器采用的是 **按字节编址**（***byte-addressable***）的方式，也就是说每个字节都有一个唯一的地址。计算机的存储器也可以采用 **按字编址**（***word-addressable***）的方式，即每个机器字都具有一个自己的唯一地址。
- 如果计算机系统采用按字节编址的体系结构，并且指令系统的结构字大于一个字节，就会出现 **字节对齐**（***alignment***）的问题。在这种情况下，访问所查询的地址时无需从某个自然对齐的边界线开始。
- 存储器由随机访问存储器（RAM）芯片构成。存储器通常使用符号 L × W（长度 × 宽度）来表示。长度表示字的个数，宽度表示字的大小。
- 主存储器通常使用的 RAM 芯片数目大于 1，通常利用多块芯片来构成一个满足一定大小要求的单一存储器模块。
  - 单一共享存储器模块可能会引起存储器访问上的顺序问题。存储器交叉存储技术（***memory interleaving***），即从多个存储器模块中分离存储器单元的技术。

Interrupt - 中断

- 中断：反应负责处理计算机中各个部件与处理器之间相互作用的概念。中断就是改变系统正常执行流程的各种事件。多种原因可以触发中断：
  1. I/O 请求
  2. 出现算术错误（除以 0）
  3. 算数下溢或上溢
  4. 硬件故障
  5. 用户定义的中断点（程序调试）
  6. 页面错误
  7. 非法指令（通常由于指针问题引起）
  8. …
- 执行各种中断的操作称为中断处理，不同中断类型的中断处理方法各有不同。但是这些操作都是由中断来操作的，因为这些过程都需要改变程序执行的正常流程。
- 由用户或系统发出的中断请求可以是 **屏蔽中断**（***maskable***）（可以被禁止或忽略）或 **非屏蔽中断**（***nonmaskable***）（高优先级别的中断，不能被禁止，必须响应）。
- 中断可以出现在指令中或指令之间，可以是同步中断，即与程序的执行同步出现；也可以是异步中断，即随意产生中断。中断处理可能会导致程序的终止执行，或者当中断处理完毕后程序会继续执行。

MARIE

寄存器和总线

- CPU 中的 ALU 部件负责执行所有进程（算数运算、逻辑判断等）。执行程序时，寄存器保存各种暂存的数值。在寻多情况下，计算机对寄存器的引用都隐含在指令中。
- 在 MARIE 中，有 7 种寄存器；
  1. AC：累加器（***accumulator***），用来保存数据值。它是一个 通用寄存器（***general purpose register***），作用是保存 CPU 需要处理的数据。
  2. MAR：存储器地址寄存器（***memory address register***），用来保存被引用数据的存储器地址。
  3. MBR：存储器缓冲寄存器（***memory buffer register***），用来保存刚从寄存器中读取或者将要写入存储器的数据。
  4. PC：程序计数器（***program counter***），用来保存程序将要执行的下一条指令的地址。
  5. IR：指令寄存器（***instruction register***），用来保存将要执行的下一条指令。
  6. InREG：输入寄存器（***input register***），用来保存来自输入设备的数据。
  7. OutREG：输出寄存器（***output register***），用来保存要输出到输出设备的数据。
     - MAR、MBR、PC 和 IR 寄存器用来保存一些专用信息，并且不能用作上述规定之外的其他目的。另外，还有一个 标志寄存器（***flag register***），用来保存指示各种状态或条件的信息。
- MARIE 需要利用总线来将各种数据或指令传入和移出寄存器。在 MARIE 中，使用一条公共总线，每个设备都被连接到这条公用总线。
- 在 MAR 和存储器之间设计一条通信路线，MAR 通过这条电路为存储器的地址线提供输入，指示 CPU 要读写的存储器单元的位置。
- 从 MBR 到 ALU 还有一条特殊的通道，允许 MBR 中的数据也可以应用在各种算数运算中。同时，各种信息也可以从 AC 流经 ALU 再流回 AC，而不需要经过公用总线。
  - 采取这三条额外通道的好处是，在同一个时钟周期内，既可以将信息放置在公用总线，又可以同时将数据放置到这些额外的数据通道上。

指令系统体系结构

- 计算机的 指令系统体系结构（***instruction set architecture ISA***）详细规定了计算机可以执行的每条指令及其格式。从本质上来说 ISA 是计算机软件和硬件之间的接口。
  1. Load X 将地址为 X 的存储单元中的值存入 AC
  2. Store X 将 AC 中的值存到地址 X 中
  3. Add X AC -= X 中的值
  4. Subt X AC -= X 中的值
  5. Input 从键盘输入一个值到 AC 中
  6. Output 将 AC 中的值输出
  7. Halt 终止程序
  8. Skipcond 一定条件下跳过下一条指令
  9. Jump X 将 X 的值存入 PC
- 指令由 操作码 和 地址 构成。大多数 ISA 由处理数据、移动数据和控制程序的执行序列的指令组成。
- 计算机的输入和输出是比较复杂的操作。现代计算机都是采用 ASCII 编码字符的方式进行输入和输出操作，先以字符形式读入，再转换成相应数值后，才能被存放再寄存器 AC 中。

指令的执行过程

取址 - 译码 - 执行周期

- 取址 - 译码 - 执行（***fetch-decode-execute***）表示计算机运行程序时所遵循的步骤。CPU 首先提取一条指令，即将指令从主存储器转移到指令存储器上；接着对指令进行译码，即确定指令的操作码以及提取执行该指令所需的数据；然后执行该指令。

中断和输入 / 输出

- 输入寄存器保持由输入设备传送到计算机的数据；输出寄存器保持准备传送到输出设备的各种信息。
- 在 MARIE 中，使用 **中断控制的输入输出**（***interrupt-drive I/O***）解决冲突。当 CPU 要执行输入或输出指令时，首先通知相应的 I/O 设备，然后处理其他的任务直至 I/O 设备准备就绪。此时，I/O 设备会向 CPU 发送一个中断请求信号。随后 CPU 会响应和处理这个中断请求。完成输入输出操作后，CPU 会继续正常的取址 - 译码 - 执行周期。
- 通常情况下，输入或输出设备使用一个特殊寄存器来发送中断信号。通过在这个寄存器中设置某个特殊的二进制位来表示有一个中断请求产生。若 CPU 发现中断位，就会执行中断处理任务。CPU 处理中断服务程序，也是对各种中断指令执行正常的取址 - 译码 - 执行周期，直至所有中断程序运行完毕。
- CPU 执行中断服务程序后，必须严格返回到原始程序的运行位置。因此它会先存储 PC 中的内容，CPU 的所有寄存器中的内容以及原始程序中原有的各种状态条件，使其能够严格恢复到原始程序运行的真实环境。

存储器

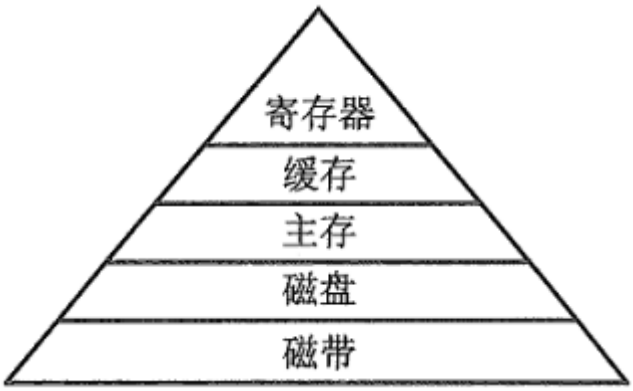
存储器分类

- 计算机的快速发展使得 CPU 的设计不断改进，而要求与之配套的存储器必须在速度上与之匹配。然而存储器的发展已经成为一个瓶颈，但由于采用了 高速缓存存储器的技术，所以改进主存储器的任务不再尤为紧迫。
- 高速缓存存储器是一种小容量、高速度，同时也是高价格的存储器。高速缓存的作用是在频繁存取数据的过程中充当一个缓冲器。
- 尽管存储器类型繁多，但大部分计算机系统都使用两种基本类型的存储器：随机存储器（***random access memory RAM***）和只读存储器（***read only memory ROM***）。RAM 更合适的名字是 **读写（*read-write*）存储器**。同时 **RAM** 也就是主存储器。

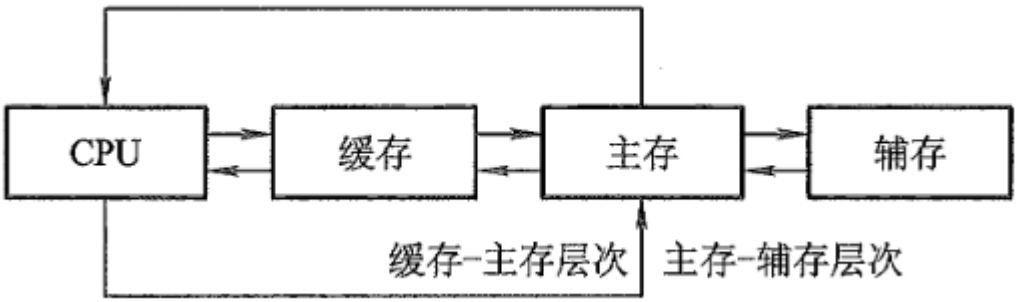
1. 按存储介质分类 > 存储介质是指能寄存 0 1 两种代码并能区别两种状态的物质或元器件
    1. 半导体存储器
    2. 磁表面存储器
    3. 磁芯存储器
    4. 光盘存储器
2. 按存取方式分类
    1. 随机存储器（***Random Access Memory, RAM***）：RAM 是一种可读 / 写的存储器，其特点是任何一个存储单元的内容都可随机存取，而且存储时间与存储单元的物理位置无关。
    2. 只读存储器（***Read Only Memory, ROM***）：只读存储器是能对其存储的内容读出，而不能对其写入的存储器。这种存储器一旦存入了原始信息，在程序执行过程中，只能将内部信息读出，而不能随意重新写入新的信息去修改原始信息。

3．串行访问存储器：对存储单元进行读 / 写操作时，需按其物理位置的先后顺序寻找地址。显然这种存储器由于信息所在位置不同，使得读 / 写时间均不相同。

存储器的层次结构



- 通常存储器的速度很快，单位信息惯的成本也就越高。通过采用存储器的分层组织结构体，是不同层次的存储器具有不同的访问速度（存取速度）和存储容量。存储器分层结构系统的基本类型包括：寄存器、高速缓存、主存储器和辅助存储器。
- 现代计算机一般都具有一个数目较小的高速存储器系统，称为 高速缓存。高速缓存用来暂存一些存储器单元频繁使用的数据。这种高速缓存会被连接到一个容量较大的主存储器。通常主存储器速度中等，然后再使用一个容量非常大的辅助存储器作为机器存储器系统的补充。采用这种存储器分层结构的组织方式，可以提高对存储器系统的有效访问速度，而且只需使用少量快速的芯片。
- CPU 不能直接访问辅存，辅存只能与主存交换信息，因此辅存的速度比主存慢得多。



- 在讨论存储器层次结构时，会涉及一些术语：
  - 命中（***hit***）：CPU 请求的数据就暂存在要访问的存储器层中。通常，只是在存储器的较高层才会关心命中率的问题。
  - 缺失（***miss***）：CPU 请求的数据不在要访问的存储器层。
  - 命中率（***hit rate***）：访问某个特定的存储器层时，CPU 找到所需数据的百分比。
  - 未命中率（***miss rate***）：在某个特定的存储器层中，CPU 取得所请求的信息所需要的时间。
  - 未命中惩罚（miss penalty）：CPU 处理一次缺失时间所需要的时间，其中包括利用新的数据取代上层存储器的某个数据块所需要的时间，再加上将所需数据传递给处理器所需要的附加时间。

- 通常情况下，处理一次未命中事件所花费的时间比处理一次命中事件所需的时间更多。
- 对于任何给定数据，处理器首先将访问数据的请求传送给存储器中速度最快、规模最小的存储器层。通常是高速缓存存储器，而不是寄存器，原因是寄存器通常用于更专用的用途。若在高速缓存中找到所需的数据，这些数据会很快地被装入 CPU 的寄存器中；若数据没有暂存在高速缓存中，那么访问数据的请求就会被传送到下一个较低层次的存储器系统，继续搜索数据。
- 核心思想是，较低层的存储器系统会响应较高层的存储器对位于存储单元 X 处的访问请求。同时，这些低层存储器也会将位于地址 X-1，X-2… 处的数据送出，也就是将整个数据块返回给较高层的存储器系统。

引用的局部性

1. 时间局部性（***Temporal locality***）：对于最近引用过的内容，很可能在不久的将来再次被访问
2. 空间局部性（***Spatial locality***）：对于最近引用过的内存位置，很可能在不久的将来引用其附近的内存位置
3. 顺序局部性（***Sequential locality***）：访问存储器的指令倾向于按顺序执行

- 这种局部性的原理使系统有机会使用少量速度非常快的存储器来有效加速对系统中主要存储器的访问。通常，在任意给定时刻计算机系统只需要访问整个存储空间中的某一个非常小的部分，而且存放该位置的数值常常被重复读取。
- 利用这种方式组织的存储器系统可以将大量的信息存储在一个巨大的、低成本的存储器中。

高速缓存存储器

- 高速缓存存储器的基本工作原理，计算机系统会将那些频繁使用的数据复制到高速缓存存储器中，而不是通过直接访问主存储器来重新获取数据。
- 计算机与人不同，没有办法得知哪些数据使最有可能被计算机访问读取的。因此，计算机需要使用局部性原理，在访问主存储器时会将整个数据块从主存储器中复制到高速缓存存储器。
- 高速缓存对这种新数据块的定位操作取决于两个因素：高速缓存的映射策略和高速缓存的大小。高速缓存的大小会直接影响到是否有足够的空间存放新的数据块。
- 对于不同的计算机，高速缓存存储器的容量有很大的差别。通常个人计算机的第二级（*L2*）高速缓存的大小为 256KB 或 512KB；而第一级（*L1*）高速缓存要小一些，通常为 8KB 或 16KB。L1 高速缓存通常集成在微处理器中，而 L2 高速缓存则位于 CPU 和主存储器中。因此 L1 高速缓存要比 L2 高速缓存的速度快。
- 使用高速缓存存储器的目的是为了加快存储器的访问速度。高速缓存将最近使用过的数据存放在靠近 CPU 的地方，而不是将这些数据存放到主存储器中。通常主存储器由 DRAM 组成，高速缓存由 SRAM 构成，比主存储器的访问速度快得多。
- 高速缓存存储器不是通过地址进行访问，而是按照内容进行存取。基于这种原因，高速缓存存储器也被称为 按内容寻址的存储器（***content addressable memory***），简称为 CAM。在大部分高速缓存映射策略的方案中，都必须检查或搜索高速缓存的入口，以确定所需的数值是否放置高速缓存中。

高速缓存的映射模式

- 在访问数据或指令时，CPU 首先会生成一个主存储器地址。此时，若要访问的数据已经被复制到高速缓存中，那么这个数据在高速缓存中的地址与主存储器中的地址不同。CPU 会采用某种特殊的映射模式，将主存储器的地址转换为一个对应的高速缓存存储器的位置。
- 可以通过对主存储器地址的各个位规定某些特殊意义来实现地址的转换。首先将地址的二进制位分成不同组，称为域。根据映射模式的不同，分成多个地址域。如何使用地址域取决于采用的特定的映射模式。高速缓存的映射模式决定了数据最初被复制到高速缓存存储器中时的存放位置。

- 主存储器和高速缓存的存储空间都会被划分称相同大小的子块，不同系统的存储块大小可能不一样。当生成一个存储器的地址时，CPU 首先会搜索高速缓存存储器，查找所需的数据字是否已经暂存于高速缓存中。如果在高速缓存未找到，那么 CPU 会把主存储器中该字所在的位置的整个块装入高速缓存存储器中。因为访问高速缓存字要比访问主存储器字的速度快得多，所以高速缓存存储器可以加快计算机对存储器的总访问时间。
- 使用主存储器地址的各个地址域：CPU 使用主存储器地址的其中一个域，对暂存在高速缓存中的域请求数据，直接给出数据在高速缓存中的位置，这种情况称为 高速缓存命中（***Cache hit***）；而 CPU 对没有暂存在高速缓存中的请求数据，会指示出数据将要存放在高速缓存中的位置，这种情况称为 **高速缓存缺失**（***Cache miss***）。

- 接下来，CPU 会通过高速缓存块的一个有效位（***valid bit***），来验证所引用的高速缓冲块的合法性。若要引用的高速缓存块不正确，则产生了一次高速缓存缺失，必须访问主存储器。反之引用正确，可能会产生一次高速缓存命中。
- CPU 随后要将属于该高速缓存块的标记与主存储器地址的 标记域（***tag field***）进行比较。标记是主存储器地址中的一组特殊二进制位，用来标识数据块的身份。标记与对应的数据块会一起存放到高速缓存中，若标记匹配，则产生了一次高速缓存命中。
- 在此基础上，还需要再高速缓存块中定位找出所要求的数据字的存放位置。这项工作可以通过使用主存储器地址中一个称为 字域（***word field***）的字地址来完成。字域代表数据字的块内地址，所有的高速缓存映射模式都要求有一个字域。最后，地址中剩余的字段由特定的映射模式来决定。

  1. 直接映射的高速缓存

     - 直接映射的高速缓存采用模块方式来指定高速缓存和主存储器之间的映射关系。因为主存储器块比高速缓存块要多得多，所有很明显主存储器中的块要通过竞争才能获取高速缓存中的对应位置。
     - 直接映射方式是将主存储器中的块 X 映射到高速缓存的块 Y，对应的模量为 N。其中 N 是高速缓存中所划分的存储空间的块的总数。对于高速缓存存储器的第 0 块而言，主存储器中的第 0 块、第 N 块、第 2N 块，都需要竞争映射到第 0 块。
     - 当每个数据块被复制到高速缓存时，它们的身份就已经由 标记（***tag***）确定了，因此计算机可以在任意时刻得知，实际是哪一个数据块驻留在当前高速缓存块中。因此高速缓存中实际存储的信息要比从主存储器复制的信息多。

- 为了实现直接映射功能，二进制的主存储器地址被划分为了 **标记、块、字** 三个不同域。每个域的大小取决于主存储器和高速缓存存储器的物理特性。**字** 域，又称为 **偏移量**（***offset***）域，用来唯一识别以及确定来自某个指定数据块的一个数据字。因此，字域必须具有合适长度的数据位。同样，**块** 域必须选择一个唯一的高速缓存块。剩下的字段为 **标记** 域。在复制主存储器中的数据块到高速缓存时，标记会随着数据块一起被存储到高速缓存中，并可以通过标记唯一识别和确定数据块。(标记确定数据块位置，偏移域确定数据块中的数据位置)

2．全关联高速缓存

- 直接映射的高速缓存方案成本比其他方案低，原因是直接映射方式不需要进行任何的搜索操作。主存储器中的每个数据块都映射到高速缓存的指定的存储位置。当主存储器地址被转换为某个高速缓存地址时，CPU 只需检查地址的块域位，就能准确得知需要到高速缓存中的具体位置。
- 假设不为主存储器中的数据块唯一指定在高速缓存中的位置，而是允许主存储器中的数据块块域存放到高速缓存的任意位置。这种情况下，要找到存储器映射的数据块的唯一方法时搜索全部高速缓存。这样一来，整个高速缓存需要按照 **关联存储器**（***associative memory***）的模式构建，以便 CPU 可以对这种高速缓存存储器执行平行搜索。也就是说，单一的搜索操作必须将所请求的标记与存放在高速缓存中所有的标记进行比对，以确定是否存储有高速缓存中。
- 使用关联映射方式时，需要将主存储器的地址划分为标记域和字域两部分。当要在高速缓存中搜索某个特定的主存储器数据块时，CPU 会将改主存储器地址的标记域与高速缓存中存放的所有合法标记域进行比对。若没有一个标记匹配，表明产生了一个高速缓存缺失，且数据块必须从主存储器中复制到高速缓存中。
- 对于直接映射方式，若一个已被占用的高速缓存单元需要存放新的数据块，则必须移除当前数据块。若数据块的内容已被修改过，则必须将其重新写入主存储器中；否则直接覆盖原来的数据块。对于全关联映射方式，若高速缓存已经装满，则需要一种置换算法来决定将从高速缓存中丢弃哪个数据块。被丢弃的数据块被称为 **牺牲块**（***victim block***）。

3．组关联高速缓存

- 由于直接关联和全关联方案存在许多问题与限制，则有了组关联高速缓存。N 路的组关联高速缓存映射（***N-way set associative Cache mapping***），是上面两种方法的某种组合形式。首先，组关联映射方式类似于直接映射高速缓存。使用地址将主存储器中的数据块映射到高速缓存中的某个在指定存储单元。
- 它与直接映射方式的区别是：这种方式不是将数据块映射到高速缓存的某一个空间块，而是映射到由几个高速缓存块组成的某个块组中。同一个高速缓存中的所有组大小必须相同。一个八路的组关联高速缓存中，每组包含 8 个高速缓存块。由此可见，直接映射的高速缓存是一种特殊形式的 N 路组关联高速缓存，每个组中只有 1 个高速缓存块。
- 在组关联高速缓存的映射方式中，主存储器地址分为三部分：标记域、字域和组域。标记域和字域与直接映射中的作用相同；而组域，与块域类似，表示主存储器中的数据块会被映射到高速缓存中的块组。

置换策略

- 在直接映射方式中，若多个主存储器的块竞争某一个高速缓冲块，则会将现有的数据块从高速缓存这种踢出，为新的数据块预留空间。这一过程称为 **置换**（***replacement***）
- 对于直接映射方式，不需要使用置换策略，因为每个新的数据块对应的映射位置是固定的。但对于全关联高速缓存和组关联高速缓存，需要使用置换策略。当采用全关联映射方式时，主存储器块的映射所对应的可能是任何一个高速缓存块。当采用 N 路组关联映射方式时，主存储器中的数据块可以映射到某个指定高速缓存块组中的任意一块。决定置换哪个高速缓存块的算法称为 **置换策略**（***replacement policy***）。

- 对于置换策略，希望保留高速缓存存储器中将要使用的数据块，丢弃在某段时间内不需要使用的高速缓存块。最佳置换算法的基本思想是，替换掉未来最长时间段内不再使用的高速缓存块。显然，使用最佳算法可以保证最低的高速缓存缺失率。但是，无法从每个程序的单次运行过程中预测未来，因此最佳算法只能作为评估其他算法优劣的量度方式。
  1. 考虑时间局部性，推测最近没有被使用过的数据，且在未来短时间内也不太可能被使用。可以为每个高速缓存块分配一个时间标签，记录它上次被访问的时间，选择最近最少被使用的高速缓存块。这种算法称为 **最近最少被使用算法**（***least recently used***）。但是这种算法要求系统保留每个高速缓存块的历史访问记录，需要高速缓存有相当大的存储空间，同时会减慢高速缓存的速度。
  2. **先进先出**（***FIFO***）是另一种策略，利用这种策略，存放在高速缓存中时间最长的块将会被移除，而不管它在最近何时被使用过。
  3. 另一种策略是 **随机选择**（***random***）。LRU 和 FIFO 会遇到 **简并引用**（***degenerate reference***）的问题，这时会产生对某个高速缓存块的 **重复操作**（***thrash***），即不停重复地将一个数据块移出再移回高速缓存。对于随机选择算法，虽然它不会出现对某个块的重复操作，但有时会把即将需要的数据块移出高速缓存，会降低高速缓存的平均性能。
- 算法的选择通常取决于计算机系统的使用方式。对于所有情况来说，不存在最好的、单一的、实用的算法。

高速缓存的写策略

- 除了要选择进行置换的牺牲块外，还必须对高速缓存的 **脏块**（***dirty block***）进行处理。高速缓存中的脏块是指已经被修改过的数据块。当处理器写入主存储器时，数据可能也会被写入高速缓存中，原因是处理器可能很快会再次读这些数据。若修改了某个高速缓存块的数据，**高速缓存的 写策略**（***write policy***）会决定何时更新对应的主存储器数据块来保证数据的一致性。
  1. **写通（*write through*）策略**
     - 写通策略是指在每次写操作时，处理器会同时更新高速缓存和主存储器中对应的数据块。写通策略速度比较慢，但可以保证高速缓存和主存储器中数据一致。
     - 写通策略的明显缺点是，每次进行高速缓存的写操作都伴随着一次主存储器的写操作，减慢了系统速度。若所有的访问都是写操作，那么存储器系统的速度会减慢到主存储器的访问速度。但实际操作中，大多数存储器的访问都是读操作，因此可以忽略写通策略带来的影响。

2. **回写（*write back*）策略**
   - 当只有某个高速缓存块被选择作为牺牲块而必须从高速缓存中移除时，处理器才更新主存储器中对应的数据块。通常，回写策略的速度比写通策略块，因为对于存储器的访问次数减少了。
   - 回写策略的缺点是，主存储器和高速缓存的对应单元在某些时刻存放着不同数值。若某个进程在回写操作完成前中断或崩溃，会导致高速缓存中的数据丢失。

虚拟存储器

- 高速缓存允许计算机从一个较小规模但速度较快的的高速缓存器中，频繁访问使用过的数据。这种分层组织结构所固有的另一个另一个重要概念是 **虚拟存储器**（***virtual memory***）。
- 虚拟存储器使用硬盘作为 RAM 存储器的扩充，增加了进程可以使用的有效空间。大多数主存储器的容量较小，通常没有足够的存储空间来并发地运行多种应用程序。
- 使用虚拟存储器，计算机可以寻址比实际主存储器更多的空间。计算机使用硬盘驱动器保持额外的主存储器空间。硬盘上这部分区域被称为 **页文件**（***page file***），因为这些页文件在硬盘上保存主存储器的信息块。
- 实现虚拟存储器最常用的方法是使用主存储器的 **分页机制**（***paging***），这种方法是将主存储器划分成固定大小的块，并且程序也被划分成相同大小的块。通常，程序块会根据需要被存放到存储器中，没有必要将程序的连续块也存储到主存储器的连续块中。
- 在高速缓存中，主存储器的地址应该转换为高速缓存的位置；在虚拟存储器中也是如此，每个虚拟地址都必须转换为物理地址。

- 分页实现虚拟存储器的术语：
  1. **虚拟地址**（***virtual address***）：进程所使用的逻辑地址或程序地址。只要 CPU 生成一个地址，就总对应虚拟地址空间。
  2. **物理地址**（***physical address***）：物理存储器的实际地址。
  3. **映射**（***mapping***）：一种地址变换机制，通过映射可以将虚拟地址转换为物理地址，类似于高速缓存映射。
  4. **页帧**（***page frame***）：由主存储器分成的相等大小的信息块或数据块。
  5. **页**（***page***）：由虚拟存储器（逻辑地址空间）划分的信息块或数据块。每页的大小与一个页帧相同。在硬盘上存储虚拟页以供进程使用。
  6. **分页**（***paging***）：将一个虚拟页从硬盘复制到主存储器的某个页帧的过程。
  7. **存储碎片**（***fragmentation***）：不能使用的存储器单元。
  8. **缺页**（***page fault***）：当一个请求页在主存储器中没有找到所发生的事件，必须将请求页从硬盘复制到存储器。
- 因为主存储器和虚拟存储器都被划分为相同大小的页，所以可以把程序进程的地址空间的一些片段移动到主存储器中，但不需要将这些内容连续存储。也不必立即将所有进程全部装入主存储器中；虚拟存储器允许存储器中只需存在特定片段即可运行程序，暂时不用的程序部分会存储在硬盘上的页文件中。

分页

分页的基本思想：

- 按照固定大小的信息块（页帧）为各个进程分配物理存储空间，并且通过将信息写入页表 ***(page table)*** 的方式跟踪记录进程的不同页的存放位置。每个进程都有自己的页表，页表通常驻留在主存储器中，页表存储该进程的每个虚拟页的物理位置。
- 若当前进程某些页不在主存储器中，则页表会通过设置一个 **有效位** ***(valid bit)*** 为 `0` 来指示；若当前进程在主存储器中，则该页的有效位设置为 `1`。因此每个页表的入口目录都由 **有效位和帧数** 组成。

- 通常页表中还会附加一些其他内容，以便传递更多信息。
  1. 修正位 **(*dirty bit* 或 *modify bit*)** 来指示页中内容是否已经发生改变，这样可以使处理器在进行返回页内容到硬盘的操作时更有效率
  2. 使用位 ***usage bit*** 来指示页的使用情况。只要处理器访问过某一页，设置该位为 `1`，一定时间周期后重新设置为 `0`。若某段时间间隔内未使用该页，系统可能把该页从主存储器移除，放到磁盘上。这样可以让系统释放该页占用的存储空间给其他进程需要使用的另外一页。
- 虚拟存储器中页的大小于物理存储器的页帧相同。程序进程的存储空间分为固定大小的页，当把程序的最后一页复制到存储器时，可能产生潜在的 **内部碎片** ***(internal fragementation)*** 现象。进程可能不需要占用整个页帧，但是又没有其他的进程使用该页帧的剩余部分。
- 若进程本身的全部内容需要占用的空间小于一个整页，而在复制到存储器时必须占用一个完整的页帧，这是就会产生存储碎片。对于某个特定的分页，内部碎片会导致一些未使用的存储空间。

分页的工作原理：

- 当某个进程生成了一个虚拟地址时，存储系统必须动态地将虚拟地址转换成数据实际驻留的存储器的物理地址
- 为了实现这种地址之间的转换，可以把虚拟地址分成两个部分：**页域** ***(page field)*** 和 **偏移量域** ***(offset field)***。偏移量表示要使用的数据在页内的位置。这种地址转换过程类似于高速缓存映射算法中将主存储器划分为不同域的过程。
- 要访问给定虚拟地址的数据，系统要执行以下步骤：
  1. 从虚拟地址中提取页码和偏移量
  2. 通过访问页表将页码转换为对应的物理页帧数
     1. 在表中查找页码（使用虚拟地址的页码作为索引），并检验有效位。
        - 若有效位为 `0`，则表示系统产生了一个缺页事件。此时，操作系统必须介入：
          1. 在磁盘上查找所使用的页
          2. 寻找一个空的页帧（若主存储器已满，则必须从主存储器中移除一个牺牲页，然后将内容复制到硬盘，同时修改牺牲页的有效位）
          3. 把要使用的页复制到主存储器的空页帧
          4. 更新页表（新装入的虚拟页在页表中必须有自己的页帧数有效位）
          5. 重新执行引起缺页事件的程序进程

- 若有效位为 1，则表示查找的页已经在主存储器中
  1. 使用实际帧数代替虚拟页码
  2. 对于给定的虚拟页，访问对应的物理页帧中位于对应偏移量的数据。具体的物理地址是 **页帧 + 偏移量**

- 若发生缺页事件，同时分配给进程的主存储器已满，则必须选择一个牺牲页。选择牺牲页需要用到置换算法，与高速缓存中使用的置换算法非常类似。

使用分页的有效存取时间

在分析高速缓存时，引用了有效存取时间的表示方法。使用虚拟存储器时，同样需要地址有效存取时间。

- 处理器每次访问存储器，都必须执行两次对物理存储器的访问操作。一次是引用页表，另一次是访问要请求的实际数据。
- 通过将最近的页查询数据值存放到一个被称为 **转换旁视缓冲器（*translation look-aside buffer, TLB*）** 可以加速页表的查询时间。每个 TLB 的入口目录都由一个虚拟页的页码和对应的物理页帧的帧数组成。
- 处理器通常使用关联高速存储来实现 TLB，并且虚拟页码和物理帧数对可以映射到 TLB 高速缓存的任何位置。当使用 TLB 时，一个地址的查询有如下步骤：

  1. 从虚拟地址中提取页码
  2. 从虚拟地址中提取偏移量
  3. 在 TLB 中搜索虚拟页码
  4. 若在 TLB 中找到虚拟页码和物理页帧数对，则将偏移量加上物理页帧数，并直接访问相应的存储单元
  5. 若发生一个 TLB 缺失，处理器就从页表中获取所需要的帧数。若该页已经位于存储器中，就利用偏移量加上物理帧数生成对应的物理地址
  6. 若请求的页不在存储器中，就会生成一个缺页错误。在出现缺页事件时，需要操作系统介入，重启存储器访问操作。

分页和虚拟存储器的优缺点

- 通过分页和增加一次额外的存储器引用来实现虚拟存储器。通过使用 TLB 高速缓存来存放页表的入口目录，可以减少分页的部分损失。但即使在 TLB 有很高的命中率，这一进程仍然会在地址转换上造成重大开销。
- 使用虚拟存储器和分页的另一个缺点是，需要消耗额外的系统资源，即需要额外的存储器空间存放页表。在极端情况下，页表会占用很大比例的物理存储器。
- 虚拟存储器允许系统运行虚拟地址空间比实际物理存储器空间大的程序。实质上，虚拟存储器允许进程可以和该进程自身共享物理存储器。
- 由于每个程序需要的物理存储器空间较小，因此虚拟存储器可以同时运行更多的程序。这使得系统能够处理总的地址空间远超过系统物理存储器空间的容量，提高了 CPU 的使用率和系统处理能力。
- 从操作系统的角度看，使用固定大小的页帧和页面简化了存储器空间的分配和地址的安排问题；而且分页也允许操作系统以每页为基础实现特定的任务保护

主存储器

- 若要从存储器读出某一信息字时，首先由 CPU 将该字的地址送到 MAR，经地址总线送至主存，然后发出读命令。主存接到读命令后，得知需将该地址单元的内容读出，便完成读操作，将内容读至数据总线上。
- 若要向主存写入一个信息字时，首先 CPU 将该字所在主存单元的地址经 MAR 送到地址总线，并将信息字送入 MDR，然后向主存发出写命令，主存接到写命令后，数据总线上的信息写入对应地址线指出的主存单元中。
- 主存中存储单元地址的分配：主存各存储单元的空间位置是由单元地址号来表示的，而地址总线使用来指出存储单元地址号的，根据该地址可读出或写入一个存储字。通常计算机可以按字寻址也可按字节寻址。

随机存取存储器

- RAM
  - 在计算机执行程序时，RAM 用来存放程序或数据。但是，RAM 是一种易失性的存储器。当存储器系统掉电时，RAM 中所存储的信息会丢失。现代计算机一般采用两种类型的存储器芯片来构建大规模 RAM 存储器：静态随机存储器（***SRAM***）和动态随机存储器（***DRAM***）.
  - 动态 RAM 由一个小电容构建。由于电容会泄漏电荷，所以 DRAM 每隔几毫秒充电才能保存数据。相比之下，静态 RAM 技术只要供电不断，就可以维持它所保存的数据。SRAM 的速度比 DRAM 更快也更贵。使用 DRAM 的原因是因为 DRAM 的存储密度更高，消耗的功耗更低，比 SRAM 产热更少。
  - 因此通常使用两种技术的组合形式：DRAM 作为主存储器，SRAM 作为高速缓存存储器。所有 DRAM 的基本操作都是相同的，但 DRAM 还是可以分成许多类型；SRAM 也是一样，

只读存储器

- 按照 ROM 的原始定义，一旦注入原始信息即不能改变，但随着用户的需要，总希望能随意修改 ROM 内的原始信息。因此出现了 PROM，EPROM 和 EEPROM 等类型。
- - ROM
  - 大部分计算机系统还使用一定数目的 ROM 存储器来存放一些运行计算机系统所需要的关键信息，比如启动计算机系统所需要的程序。ROM 属于非易失性的存储器，可以长久保存它所存放的数据，在电源关闭后还可以保存其中的信息。
  - 有 5 种类型不同的基本 ROM 存储器：ROM、PROM、EPROM、EEPROM 和闪存（***flash memory***）。
    - PROM 称为 可编程只读存储器，它是 ROM 一种改进类型，用户可以使用专门的设备对其进行编程。由于 ROM 采用的是硬连线，PROM 存储器中有许多熔断丝，可以通过烧断的方法进行编程；但是一旦编程完成，PROM 中的数据和指令都不能再更改。
    - EPROM 为可擦除 PROM，是一种可重复编程的可编程存储器。要对 EPROM 进行再编程，首先需要将整块芯片的原有信息擦除干净。

- EEPROM 是电可擦除 PROM，它克服了 EPROM 的许多缺点，而且一次可只擦除芯片上某些部分的信息，例如一次只擦除一个字节。
- 闪存本质也是一种 EEPROM，但是闪存的优点可以按块来擦写数据，不限于一个字节，因此，其擦写速度比 EEPROM 快。

存储器与 CPU 的连接

1. 存储容量的扩展：由于存储芯片的容量总是有限的，很难满足实际的需要，因此必须将若干存储芯片连载一起才能组成足够容量的存储器，称为存储容量的扩展，通常由位扩展和字扩展。
   1. 位扩展：增加存储字长
   2. 字扩展：增加存储器字的数量
   3. 字、位扩展：既增加存储器字的数量，又增加存储字长。
2. 存储器与 CPU 的连接：存储芯片与 CPU 芯片相连时，要注意片与片之间的地址线、数据线和控制线的连接。
   1. 地址线的连接
      - 存储芯片的容量不同，其地址线数也不同，CPU 的地址线数往往比存储芯片的地址线数多。通常将 CPU 地址线的低位与存储芯片的地址线相连。CPU 地址线的高位或在存储芯片扩充时用，或其他用途。
   2. 数据线的连接
      - CPU 数据线数与存储芯片的数据线数也不一定相等。此时，必须对存储芯片扩位，使其数据位数与 CPU 的数据线数相等。
   3. 读/写命令线的连接
      - CPU 读/写命令线一般可直接与存储芯片的读/写控制端相连，通常高电平为读，低电平为写。有些 CPU 的读/写命令线是分开的，此时 CPU 的读命令线应与存储芯片允许读控制端相连，而写命令线与存储芯片的允许写控制端相连。
   4. 选择存储芯片：
      - 通常选用 ROM 存放系统程序、标准子程序和各类常数等。RAM 则是为用户编程而设置的。