CPU 与主存间的数据交互

20121034 胡才郁

(计算机工程与科学学院)

摘　要　数据流动是计算机组成原理中的重要内容，在计算机中的数据交互发挥重要作用。本文梳理了 CPU 与主存的数据交互过程，并且分析了计算机内部在数据传递、数据存储两个阶段之中的流程与操作细节，并以此为线索展开相关结构组成的介绍。

关键词　数据流动；数据通路；CPU；主存；Cache

1 引言

CPU 与主存都是冯・诺伊曼体系结构中计算机系统的重要组成部分。CPU 是计算机中负责读取指令，对指令译码并执行指令的核心部件。在计算机体系结构中，CPU 是对计算机的所有硬件资源（如存储器、输入输出单元）进行控制调配、执行通用运算的核心硬件单元。计算机系统中所有软件层的操作，最终都将通过指令集映射为 CPU 的操作。而主存是计算机与 CPU 进行沟通的桥梁，主存负责数据的交换，其作用是用于暂时存放 CPU 中的运算数据，以及与硬盘等外部存储器交换的数据。

本文以 CPU 与主存之间的交互为线索，将此过程中数据传递、数据存储两部分。依次对应于 CPU 与主存之间的数据通路、主存内部数据的储存。并且在每一部分举出生动例子帮助理解其中的原理，通过数据这条线索，梳理了计算机中数据处理与存储的知识。本文结合计算机组成原理（1）/（2）中的内容，以数据流动这一实际场景展开分析。

2 数据通路

数据流动的第一阶段为数据的传递，CPU 与主存交互时依赖于内部总线。

2.1 数据通路的功能

数据在功能部件之间传送的路径称为数据通路。运算器与各寄存器之间的传输路径就是中央处理器内部数据通路。数据通路描述了信息从什么地方开始，中间经过哪个寄存器或多路开关，最后传送到哪个寄存器，这些都要加以控制。建立数据通路的任务是由“操作控制部件”来完成的，数据通路的功能是实现 CPU 内部的运算器与寄存器以及寄存器之间的数据交互。

2.2 数据通路的基本结构

2.2.1 CPU 内部单总线

将所有寄存器的输入端和输出端都连接到一条公共通路，这种结构比较简单，但数据传输存在较多的冲突现象，性能较低。当连接各部件的总线只有一条时，称为单总线结构；CPU 中有两条或更多的总线时，构成双总线结构或多总线结构。

2.2.2 CPU 内部三总线

将所有寄存器的输入端和输出端都连接到多条公共通路，相比之下单总线中一个始终只允许传一个数据，因而指令执行效率很低，因此采用多总线方式，同时在多个总线上传送不同的数据，提高效率。

2.2.3 专用数据通路方式

根据指令执行过程中的数据和地址的流动方向安排连接线路，避免使用共享的总线，性能较高，但硬件量大。

表 1 数据通路结构对比

结构方式	物理实现	优点	缺点
CPU 内部单总线	将所有寄存器的输入端和输出端都连接到一条公共通路上	结构简单易于实现	数据传输时存在较多的冲突现象
CPU 内部三总线	将所有寄存器的输入端和输出端都连接到多条公共通路上	同时传送不同的数据	实现较难
专用数据通路	根据指令执行过程中的数据和地址的流动方向安排连接线路，避免使用共享的总线	性能较高	硬件量大

2.3 主存与CPU之间的数据传送

主存与 CPU 之间的数据传送需要借助 CPU 内部总线完成。此处以 CPU 内部单总线结构，CPU 从主存读取指令为例，分析说明数据在数据通路中的传输过程，单总线结构如下图所示：

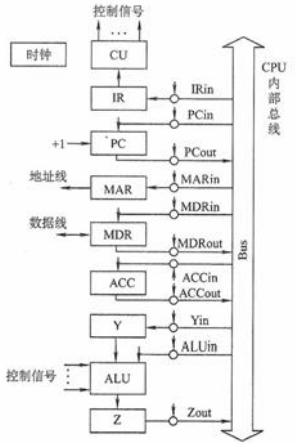

图 1 主存与 CPU 之间的数据传送

1. PC 指明了当前执行指令的地址。因此，将 PC 中的内容放置到总线上。此操作完成后，应当撤消总线上的控制信号。

2. 对主存进行读操作，因此 CU 控制单元应当通过控制总线对主存发出读信号，此时主存需要进行读操作，读操作的地址存放在 MAR 之中，此操作地址通过地址总线传送给主存。

3. 主存根据 MAR 中所指的地址读出相应的数据，再将此数据放入 MDR 之中，此数据的传送是通过数据总线。

4. 此时接通 MDRout 和 IRin 有效完成 MDR 与 IR 之间的数据传送。

表 2 数据通路结构对比

步骤序号	说明	指令
①	PCout 和 MARin 有效，现行指令地址送入 MAR	(PC) -> BUS - > MAR
②	CU 发出读命令	1 -> R
③	MDRin 有效	MEM(MAR) -> MDR
④	MDRout 和 IRin 有效，现行指令送入 IR	MDR -> Bus -> IR

3 主存储器对于数据的储存

数据流动的第二阶段为数据的存储，当 CPU 对于数据处理完成后，通过主线传送给主存后，主存进行对于数据的存储工作。

3.1 多级存储系统

为了解决存储系统大容量、高速度和低成本 3 个相互制约的矛盾，在计算机系统中，通常采用多级存储结构。存储系统层次结构主要体现在“Cache—主存”层次和“主存—辅存”层次。前者主要解决 CPU 和主存速度不匹配的问题，后者主要解决存储系统的容量问题。在存储体系中，Cache、主存能与 CPU 直接交换信息，辅存则要通过主存与 CPU 交换信息。而主存与 Cache、辅存都能交换信息。

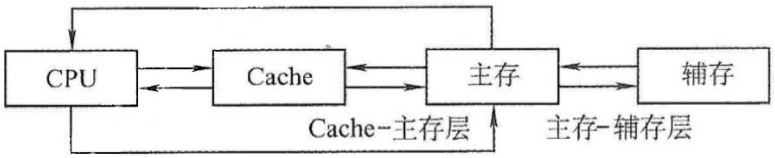

图 2 三级存储系统的层次结构及其构成

存储器层次结构的主要思想是上一层的存储器作为第一层存储器的高速缓存。从 CPU 的角度分析，“Cache—主存”层次速度接近于 Cache，容量与价位却接近于主存。从“主存—辅存”层次分析，其速度接近于主存，容量与价位却接近于辅存。这就解决了速度、容量、成本这三者之间的矛盾。

3.2 RAM

RAM 是随机存取存储器 (Random Access Memory) 的缩写，被用作计算机的短期存储器，用于放置其数据以方便访问。计算机拥有的 RAM 越多，它在给定时间可以存取的数据就越多。

可以将 RAM 看作一个工作场所：一个巨大的工作台显然比一个小茶几更好操作。主存储器有 DRAM 实现，靠近 CPU 的一层则由 SRAM 实现，他们都属于易失性存储器。

3.2.1 SRAM

通常把存放一个二进制位的物理器件称作存储元，它是存储器的最基本的构建。地址码相同的多个存储元构成一个存储单元，若干存储单元的集合构成存储体。

静态随机存储器(SRAM)的存储元是用双稳态触发器(六晶体管 MOS)来记忆信息的，因此，即使信息被读出后，它仍保持原状态而不需要再生(非破坏性读出)。

SRAM 的存取速度快，但集成度低，功耗较大。

3.2.2 DRAM

与 SRAM 的存储原理不同，动态随机存储器(DRAM)是利用存储元电路中栅极电容上的电荷来存储信息的，DRAM 的基本存储元通常只使用单个晶体管，因此，它比 SRAM 的密度要高很多。DRAM 采用地址复用技术，地址线是原来的 1/2，且地址信号分行、列两次传送。

相比于 SRAM 而言，DRAM 具有容易集成、价位低、容量大和功耗低等优点，但 SRAM 的存储速度比 SRAM 慢，一般用来组成大容量主存系统。

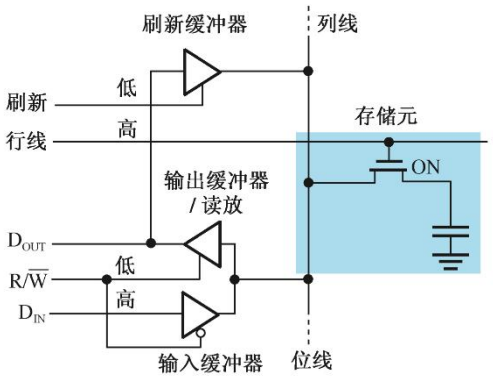

图 3 单管 DRAM 存储元的工作原理

表 3 SRAM 与 DRAM 对比

类型特点	SRAM(静态 RAM)	DRAM(动态 RAM)
存储信息	触发器	电容
破坏性读出	非	是
读出后需要重写	不需要	需要
运行速度	快	慢
集成度	低	高
发热量	大	小
存储成本	高	低
送行列地址	同时送	分为两次 (地址线复用)
用途	Cache	主存

4 高速缓冲存储器 Cache

对于主存与 CPU 的连接而言，由于主存与 CPU 间速度差异较大，因此主要有两种情况提升访问与存储的效率：

(1) 提高存储器本身的工作速度

例如，将 m 个模块组成低位交叉编址存储器，可以将带宽提升为 m 倍。尽管此方法可以提高存储器的工作速度，但优化后，主存与 CPU 之间的差距仍然很大，并不能有效的满足工作需求。

(2) 使用更高速的存储单元设计

CPU 访问到的存储器的部分只是很小的一块，将很小的一块放入贵的存储器之中，从而降低成本，提升速度。

由于程序的转移概率不会很低，数据分布的离散性较大，所以单纯依靠并行主存系统提高主存系统的频宽是有限的。这就必须从系统结构上进行改进，即采用存储体系。通常将存储系统分为"Cache-主存"层次和"主存-辅存"层次。

4.1 程序访问的局部性原理

高速缓冲技术就是利用程序访问的局部性原理， CPU 从主存取指令或取数据，在一定时间内，只是对主存局部地址区域的访问（如循环程序、一些常数等等）。因此，如果将 CPU 近期需要的程序提前存放在一个高速的、容量较小的 Cache 中，使 CPU 的访存操作大多数针对 Cache 进行,这样 CPU 只需访问 Cache 就可以得到所需要的数据,从而大大提高程序的执行速度。一般 Cache 采用高速的 SRAM 制作（主存一般使用 DRAM），其价格比主存高，容量远比主存小。

程序访问的局部性原理包括时间局部性和空间局部性。

(1) 时间局部性

时间局部性是指在最近的未来要用到的信息，很可能是现在正在使用的信息。

(2) 空间局部性

空间局部性是指在最近的未来要用到的信息，很可能与现在正在使用的信息在存储空间上是邻近的，因为指令通常是顺序存放、顺序执行的，数据一般也是以数组等形式簇聚地存储在一起的。

4.2 Cache的基本工作原理

Cache 位于存储器层次结构的顶层，通常由 SRAM 构成，其基本结构如下图：

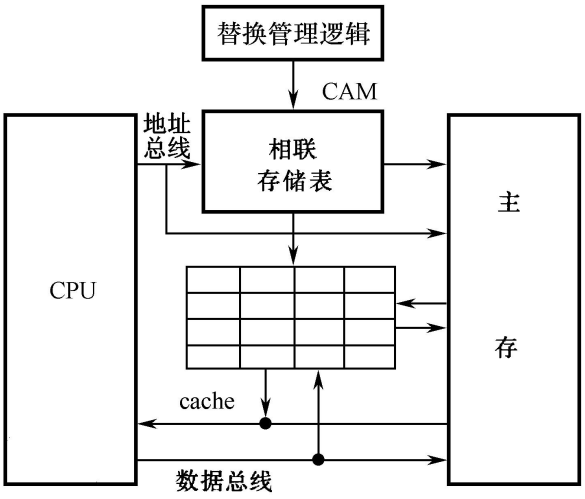

图 4 Cache 基本结构

为便于 Cache 和主存之间交换信息，Cache 和主存都被划分为相等的块，Cache 块又称 Cache 行，每块由若干字节组成，块的长度称为块长（Cache 行长）。由于 Cache 的容量远小于主存的容量，所以 Cache 中的块数要远少于主存中的块数，它仅保存主存中最活跃的若干块的副本。因此 Cache 按照某种策略，预测 CPU 在未来一段时间内欲访存的数据，将其装入 Cache。

当 CPU 发出读请求时，若访存地址在 Cache 中命中，就将此地址转换成 Cache 地址，直接对 Cache 进行读操作，与主存无关；若 Cache 不命中，则仍需访问主存，并把此字所在的块一次性地从主存调入 Cache。若此时 Cache 已满，则需根据某种替换算法，用这个块替换 Cache 中原来的某块信息。在此过程中，CPU 与 Cache 之间的数据交换以字为单位，而 Cache 与主存之间的数据交换则以 Cache 块为单位。

CPU 欲访问的信息已在 Cache 中的比率称为 Cache 的命中率。设一个程序执行期间，Cache 的总命中次数为N_{c}，访问主存的总次数为N_{m}，则命中率H为：

$$H = N_{\mathrm{c}}/(N_{\mathrm{c}} + N_{\mathrm{m}})$$

4.3 Cache和主存的映射方式

Cache 行中的信息是主存中某个块的副本，地址映射是指把主存地址空间映射到 Cache 地址空间，即把存放在主存中的信息按照某种规则装入 Cache。

由于 Cache 行数比主存块数少得多，因此主存中只有一部分块的信息可放在 Cache 中，因此在 Cache 中要为每块加一个标记，指明它是主存中哪一块的副本。这个标记的内容相当于主存中块的编号。为了说明 Cache 行中的信息是否有效，每个 Cache 行需要一个有效位。

地址映射不同于地址变换。地址变换是指 CPU 在访存时，将主存地址按映射规则换算成 Cache 地址的过程。地址映射的方法有以下 3 种：

(1) 直接映射

(2) 全相联映射

(3) 组相联映射

下面以老师在教师上课为例，生动地分析三种映射方式。

4.3.1 直接映射

直接映射可以理解为对号入座的方式，在课堂上，班级如果按照学号排座位并制作座位表，那么当老师点出某一位同学的学号时，即可唯一地确定此同学所在的座位。

主存中的每一块只能装入 Cache 中的唯一位置。若这个位置已有内容，则产生块冲突，原来的块将无条件地被替换出去（无须使用替换算法）。直接映射实现简单，但不够灵活，即使 Cache 的其他许多地址空着也不能占用，这使得直接映射的块冲突概率最高，空间利用率最低。

直接映射的关系公式如下：

$$j = i \bmod 2^c$$

此式中，j是 Cache 的块号（又称 Cache 行号），i是主存的块号，2^c是 Cache 中的总块数。在这种映射方式中，主存的第 0 块、第2^c块、第2^c+1块……只能映射到 Cache 的第 0 行；而主存的第 1 块、第2^c+1块、第$2^{c+1}+1$块……只能映射到 Cache 的第 1 行，以此类推。由映射函数可看出，主存块号的低c位正好是它要装入的 Cache 行号。给每个 Cache 行设置一个长为$t = m - c$的标记，当主存某块调入 Cache 后，就将其块号的高 t 位设置在对应 Cache 行的标记中，如下图所示：

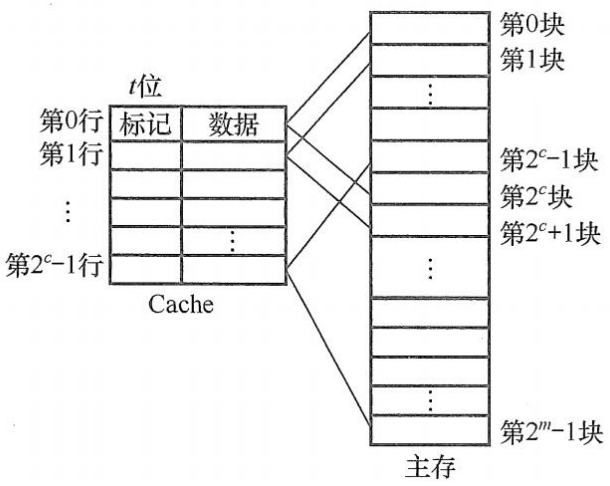

图 5 Cache 和主存之间的映射关系

直接映射的地址结构格式如下图所示：

标记	Cache行号	块内地址

图 6 直接映射的地址结构格式

4.3.2 全相联映射

全相联映射可以理解为空位随意放的方式，全相联映射，即“谁都能映射”。以课堂为例，任意一名学生可以坐在任意的座位上。此方式优点为灵活，所有学生座位都可以选择，而缺点是老师找到某位学生的座位较为困难，速度较慢。

主存中的每一块可以装入 Cache 中的任何位置，每行的标记用于指出该行取自主存的哪一块，所以 CPU 访存时需要与所有 Cache 行的标记进行比较。全相联映射方式的优点是比较灵活，Cache 块的冲突概率低，空间利用率高，命中率也高；缺点是标记的比较速度较慢，实现成本较高，通常需采用昂贵的按内容寻址的相联存储器进行地址映射，如下图所示：

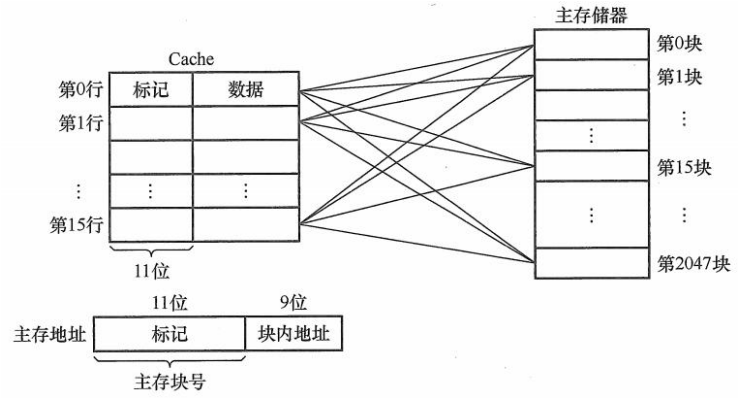

图 7 Cache 和主存之间的全相联映射方式

全相连映射的地址结构格式如下图所示：

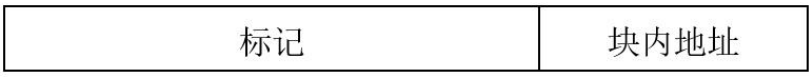

图 8 全相连映射的地址结构格式

4.3.3 组相联映射

组相联映射的方式可以理解为按号分组，组内随意放的方式。在课堂上，将所有同学分为小组，并且让小组长坐在每组的第一个位置，而组内随意选择座位。此方法是对于全相联映射的改进。老师通过寻找小组长的位置，可以很方便的确认该组的位置，并同时找到其余小组成员。

将 Cache 空间分成大小相同的组，主存的一个数据块可以装入一组内的任何一个位置，即组间采取直接映射，而组内采取全相联映射。它是对直接映射和全相联映射的一种折中，当 Q = 1 时变为全相联映射，当 Q = Cache 块数时变为直接映射。假设每组有 r 个 Cache 行，则称之为 r 路组相联，下图的设置中每组有 2 个 Cache 行，因此称为 2 路组相联。

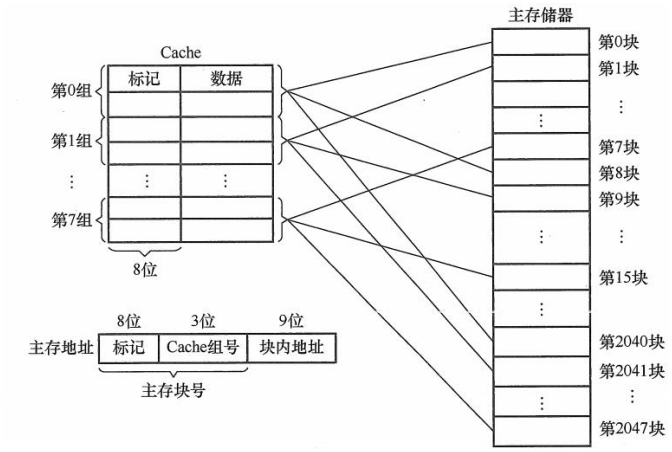

图 9 Cache 和主存之间的全相联映射方式

组相连映射的地址结构格式如下图所示：

图 10 组相连映射的地址结构格式

CPU 访存过程如下：首先根据访存地址中间的组号找到对应的 Cache 组；将对应 Cache 组中每个行的标记与主存地址的高位标记进行比较；若有一个相等且有效位为 1，则访问 Cache 命中，此时根据主存地址中的块内地址，在对应 Cache 行中存取信息；若都不相等或虽相等但有效位为 0，则不命中，此时 CPU 从主存中读出该地址所在的一块信息送到对应 Cache 组的任意一个空闲行中，将有效位置 1，并设置标记，同时将该地址中的内容送 CPU。

4.4 Cache替换算法

在采用全相联映射或组相联映射方式时，从主存向 Cache 传送一个新块，当 Cache 或 Cache 组中的空间已被占满时，就需要使用替换算法置换 Cache 行。而采用直接映射时，一个给定的主存块只能放到唯一的固定 Cache 行中，所以在对应 Cache 行已有一个主存块的情况下，新的主存块毫无选择地把原先已有的那个主存块替换掉，因而无须考虑替换算法。

常用的替换算法有随机(RAND)算法、先进先出(FIFO)算法、近期最少使用(LRU)算法和最不经常使用(LFU）算法。

下面通过商店中货架摆放货物的例子进行介绍：商店的货架中摆满了商品，由于货架容量有限，此时如果想要将新进购的商品摆上货架，则需要与将货架上已有的商品进行交换。而选择将哪一种商品下架，有以下常用算法：

(1) 随机算法(RAND)

换下货架上随机的商品，即为随机算法。

随机地确定替换的 Cache 块。它的实现比较简单，但未依据程序访问的局部性原理，因此可能命中率较低。

(2) 先进先出算法(FIFO)

换下最先摆上货架的商品，即为先进先出算法。

选择最早调入的行进行替换。它比较容易实现，但也未依据程序访问的局部性原理，因为最早进入的主存块也可能是目前经常要用的。

(3) 近期最少使用算法(LRU)

换下近期需求最小的商品，即为近期最少使用算法。

依据程序访问的局部性原理，选择近期内长久未访问过的 Cache 行作为替换的行，平均命中率要比 FIFO 的高，是堆栈类算法。

4.5 Cache的写策略

因为 Cache 中的内容是主存块副本，当对 Cache 中的内容进行更新时，就需选用写操作策略使 Cache 内容和主存内容保持一致，此时分两种情况。

对于 Cache 写命中, 有两种处理方法。

(1) 全写法

当 CPU 对 Cache 写命中时，必须把数据同时写入 Cache 和主存。当某一块需要替换时，不必把这一块写回主存，用新调入的块直接覆盖即可。这种方法实现简单，能随时保持主存数据的正确性。缺点是增加了访存次数，降低了 Cache 的效率。写缓冲：为减少全写法直接写入主存的时间损耗，在 Cache 和主存之间加一个写缓冲,如下图所示。CPU 同时写数据到 Cache 和写缓冲中，写缓冲再控制将内容写入主存。写缓冲是一个 FIFO 队列，写缓冲可以解决速度不匹配的问题。但若出现频繁写时，会使写缓冲饱和溢出。

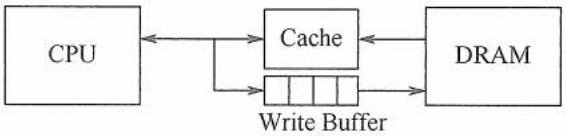

图 11 写缓冲示意图

(2) 写回法

当 CPU 对 Cache 写命中时，只修改 Cache 的内容，而不立即写入 主存，只有当此块被换出时才写回主存。这种方法减少了访存次数，但存在不一致的隐患。采用这种策略时，每个 Cache 行必须设置一个标志位（脏位），以反映此块是否被 CPU 修改过。

全写法和写回法都对应于 Cache 写命中（要被修改的单元在 Cache 中）时的情况。对于 Cache 写不命中，也有两种处理方法。

(1) 写分配法

加载主存中的块到 Cache 中，然后更新这个 Cache 块。它试 图利用程序的空间局部性，但缺点是每次不命中都需要从主存中读取一块。

(2) 非写分配法

只写入主存，不进行调块。非写分配法通常与全写法合用，写分配法通常和写回法合用。

现代计算机的 Cache 通常设立多级 Cache（通常为 3 级），假定设 3 级 Cache，按离 CPU 的远近可各自命名为 L1 Cache、L2 Cache、L3 Cache，离 CPU 越远，访问速度越慢，容量越大。指令 Cache 与数据 Cache 分离一般在 L1 级，此时通常为写分配法与写回法合用。

5 结语

计算机之中的数据离不开 CPU 与主存的数据交互，其中数据传递、数据存储两个阶段分别对应于 CPU 与主存之间的数据通路、主存内部的数据处理。通过分析数据在计算机中流动的过程，能够更好的理解计算机各个组成部分的工作原理。

致谢 十分感谢在百忙之中抽出时间审阅本文的老师。也正有着老师的带领，为我打开了计算机组成原理学习的大门。由于本人的学识和写作的水平有限，在本报告的写作中难免有僻陋，恳请老师多指教。

参 考 文 献

[1] 李梁. 内存数据管理与分析关键技术研究[D].东北大学,2020.DOI:10.27007/d.cnki.gdbeu.2020.000006.

[2] 冯平,尹家宇,宋长坤,余仕湖,李伯阳,陈铖颖,左石凯.非易失性静态随机存储器研究进展[J].半导体技术,2022,47(01):1-8+18.DOI:10.13290/j.cnki.bdtjs.2022.01.001.

[3] 计算机组成原理[M]. 科学出版社 , 白中英主编, 1994

附录 A 课程收获

A.1 知识总结回顾

在本学期《计算机组成原理(2)》的学习之中，我们着重学习了计算及存储系统、总线系统、输入输出等知识。我了解了静态、动态随机存取存储器、虚拟存储器、数据、指令总线等等，也深入体会了数据在计算机中的流通。其中，除了本文重点分析的数据流动过程中CPU 与主存的分工协调之外，并行存储的技术也十分吸引我。

A.2 遗憾与反思

春季学期注定对我们来说是个难以忘记的学期，它有着两个学期的考试都堆在一起的紧张刺激，也有着疫情突如其来的措手不及。不知不觉，这也是我足不出校的整整第 90 天。查询了学校的健康之路我才发现，自 3 月 16 日以来，我已经做过了 40 次核酸，以及数不清楚多少次抗原。

由于学校三学期制度，使得我们的节奏进度较快，导致自己没有充足的时间去细细探究每一个计算机组成部件的设计思想、理念、应用的延申和拓展，让自己可以熟练掌握其应用，并且举一反三，甚至有机会动手亲自组建硬件设施。

在疫情隔离期间，我们有更多的时间去了解学业之外的事，比如在此期间我了解了很多系统结构的设计、汇编语言的学习、不同存储方式的特点，极大地提高了我的兴趣。所以令我比较遗憾的是没有更早了解，这样或许可以更加提高自己春季学期学习积极性。

最后，希望疫情早日结束，大家都能回到正常的生活轨迹之中。