2021-2022春季学期  
《计算机组成原理A(2)》课程报告

严昕宇 20121802

(上海大学 计算机工程与科学学院)

1 引言

计算机组成原理是一门计算机专业的基础核心课。它的先修课程为数字逻辑，后续课程为计算机体系结构、接口技术、操作系统、编译原理等。作为一门承上启下的专业课程，其占有十分重要的地位和作用。

由于受到疫情影响，计算机组成原理A(2)以报告的形式作为考核形式。因此，这份报告应运而生。借此机会，我将在报告中阐述个人在完成计算机组成原理A(2)课本相关知识点的思考、分析以及知识点的实际应用。第二部分并行接口与第三部分虚拟存储器的替换算法为难点与关键技术分析，第四部分“hello, world”程序则为实际场景分析。

在春季学期的学习过程中，通过系统性地学习计算机各个部件，我建立起计算机硬件的基本组成框架，掌握计算机硬件中存储系统、输入/输出系统、总线系统与外围设备（硬盘）的基本原理。

但与此同时，由于计算机组成原理是一门理论性很强的课程，且涉及大量硬件知识。因此在学习过程中，我真切地感受到部分知识点的抽象、难以理解、庞杂而零碎，难度较大，需要利用实验课与课外时间思考并实践以加深理解，也应配合其他课程，加深认知程度。因此，在学习过程中，我也借助思维导图梳理知识脉络，强化记忆。春季学习中所学习的各章节知识点的思维导图，请见压缩包中【思维导图】文件夹。

2 关键技术分析1：并行接口——以可编程并行接口8255为例

2.1 I/O接口

回顾在计算机组成原理A(2)课程中学习到的内容，可以知道：数字计算机使用二进制数，它们或用电位的高、低来表示，或用脉冲的有、无来表示。在前一种情况下，如果电位高时表示数字“1”，那么电位低时则表示数字“0”。在后一种情况下，如果有脉冲时表示数字“1”，那么无脉冲时就表示数字“0”。因此，这影响了信息在计算机中的传送方法。

计算机系统中，传输信息一般采用串行传送或并行传送两种方式之一。在串行传送时，被传送的数据需要在发送部件进行并-串变换，称之为拆卸；而在接收部件又需要进行串-并变换，称之为装配[1]。串行传送的主要优点是只需要一条传输线，对长距离传输显得特别重要，可使其成本比较低廉。

并行数据传输是以计算机的字长为传输单位，利用8、16或32个数据信号线一次传送一个字长的数据。它适合于外部设备与微机之间进行近距离、大量和快速的信息交换，如微机与并行接口打印机、磁盘驱动器等。并行传输方式是微机系统中最基本的信息交换方式，例如，系统板上各部件之间的数据交换。并行数据传输需要并行接口的支持[3]。且并行传送一般采用电位传送。由于所有的位同时被传送，所以并行数据传送比串行数据传送快得多。因此，出于速度和效率上的考虑，系统总线上传送的信息必须采用并行传送方式[1]。

总线接口，即I/O接口或适配器，是指CPU、主存和外围设备之间通过系统总线进行连接的标准化逻辑部件。为了使所有的外围设备能在一起正确地工作，CPU规定了不同的信息传送控制方法。通常在总线和每个外围设备的设备控制器之间使用一个适配器（接口）电路来解决这个问题，以保证外围设备用计算机系统特性所要求的形式发送和接收信息，I/O接口模块的一般结构框图如下图所示：

图示

描述已自动生成

图2-1 I/O接口模块框图[1]

事实上，一个I/O接口模块有两个接口：一是和系统总线的接口。正如之前所提到的，CPU和I/O接口模块的数据交换一定是并行方式；二是和外设的接口。I/O接口模块和外设的数据交换可能是并行方式，也可能是串行方式。因此，根据外围设备供求串行数据或并行数据的方式不同，I/O接口模块分为串行数据接口和并行数据接口两大类。

书本在第8章输入/输出系统的最后介绍了通用I/O标准接口，并以并行I/O标准接口SCSI和串行I/O标准接口IEEE1394为例进行分析。因此在好奇心的驱使下，我对这个知识点进行了调研与思考，并以较为简单的可编程并行接口8255为例，具体分析并行数据接口的实现原理。

2.2 并行接口

并行接口是实现并行通信的接口。其数据传送方向有两种：单向传送（只作为输入口或输出口）、双向传送（既可作为输入口，也可作为输出口）。

并行接口电路有多种，但必须包含最基本的三态缓冲器和锁存器。除此之外，还要有状态寄存器和控制寄存器，以便于接口电路与处理器之间交换信息、与外设间传送信息。接口电路中还要有端口的译码和控制电路，以及用于中断交换方式的有关电路等[3]。

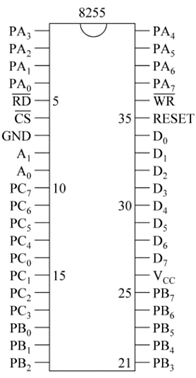
其特点可总结为以下几点：

* 传输速率高——字节、字、双字宽，多根数据线同时传
* 可靠性高——可设置握手联络信号线，易于实现异步互锁协议
* 使用自由——数据格式、传输速率、工作时序无固定规定，根据I/O设备要求决定
* 不做差错检验和速率控制
* 适于近距离传输
* 使用广泛

而8255正是Intel公司生产的为x86系列CPU配套的可编程并行接口芯片。所谓可编程，就是可以通过软件的方式来设定芯片的工作方式。8255的通用性较强、使用灵活，是一种典型的可编程并行接口，因此对其具体分析有普适性的理论价值，可以帮助我更好理解I/O接口。

2.3 外部引线与结构

关于8255内外部结构的部分，由于涉及芯片的原理与定义，因此此处较多地参考了书籍与文献。

8255的外部引线如右所示，共有40个引脚，其功能如下[4-5]。

(1) D0~D7：双向数据信号线，用来传送数据和控制字。

(2)：读信号线，低电平有效。与其他信号线一起实现对8255接口的读操作，通常接系统总线的信号。

(3)：写信号线，低电平有效。与其他信号一起实现对8255的写操作，通常接系统总线的。

(4)：片选信号线，低电平有效。当系统地址信号经译码产生低电平时选中8255芯片，使能够对8255进行操作。

(5) A0、A1：口地址选择信号线。

8255的内部包括3个独立的输入输出端口（A口、B口和C口）以及一个控制寄存器。A0、A1地址信号经片内译码可产生4个有效地址，分别对应A、B、C这3个口和内部控制寄存器，具体规定如表2-1所示。

图2-2 8225的外部引线图[4]

表2-1 各地址信号组合功能

手机屏幕截图

中度可信度描述已自动生成

在实际使用中，A0、A1通常接系统总线的A0和A1，它们与CS一起来决定8255的接口地址。

(6) RESET：复位输入信号。通常接系统的复位RESET端。当它为高电平时使8255复位。复位后，8255的A口、B口和C口均被预设为输入状态。

(7) PA0~PA7：A口的8条输入输出信号线。这8条线是工作于输入、输出还是双向（同时为输入或输出）方式可由软件编程来决定。

(8) PB0~PB7：B口的8条输入输出信号线。利用软件编程可指定这8条线是作输入还是输出。

(9) PC0~PC7：C口的8条线，根据其工作方式可作为数据的输入或输出线，也可以用作控制信号的输出或状态信号的输入线。

2.4 内部结构

图示

描述已自动生成8255的内部结构框图如图2-3所示，它由以下几个部分组成[4-5]。

图2-3 8255内部结构框图[6]

(1) 数据端口

8255有A、B、C共3个8位数据端口，可以通过编程把它们分别指定为输入口或输出口。A口和B口的输入输出都具有数据锁存能力，C口输出有锁存能力，而输入没有锁存能力。A、B、C这3个口作输出时，其输出锁存器的内容可以由CPU用输入指令读回。在使用中，A、B、C这3个口可作为三个独立的8位数据输入输出口；也可只将A、B口作为数据输入输出口，而使C口的各位作为它们与外设联络用的状态或选通控制信号的输入输出。C口的主要特点是可以对其按位进行操作。

(2) A组和B组控制电路

从图2-3中可以看到，A组和B组控制电路一方面接收读写控制逻辑电路的读写命令，另一方面接收由数据总线输入的控制字，分别控制A组和B组的读写操作和工作方式。A组包括A端口的8位和C端口的高4位（PC4~PC7），B组包括B端口的8位和C端口的低4位（PC0~PC3）。编程写入的控制字输入到内部控制寄存器，控制A组和B组的工作方式。

(3) 读写控制逻辑

读写控制逻辑负责管理8255的数据传送。它接收来自系统总线的A0、A1和以及读()、写()和复位信号()，并将这些信号进行逻辑组合，形成相应的控制命令，发送到A组和B组控制电路，以控制信息的传送。

(4) 数据总线缓冲器

数据总线缓冲器是一个三态双向8位数据缓冲器，8255通过它和系统的数据总线相连，传递控制字、数据和状态信息。

图2-4给出了8255各引脚及端口在系统中的连接示意图。

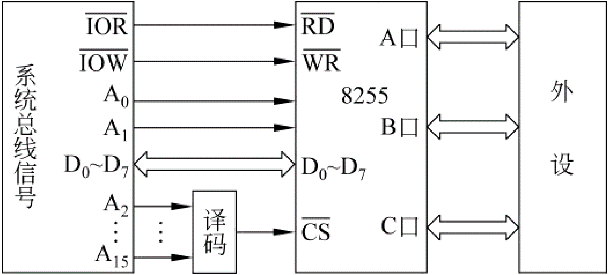


图2-4 8255与系统的连接示意图[6]

2.5 8255的工作方式

8255有3种基本的工作方式:方式0、方式1和方式2。其中A口可以工作在方式0、方式1和方式2；B口只能工作于方式0和方式1；而C口在作为数据输入输出端口时，只能工作于方式0。当A、B口工作在方式1或A端口工作于方式2时，C口的某些位被用作连接相应的选通控制信号。3个端口工作在哪一种方式下，可通过软件编程来设定。

由于工作方式1、2较为复杂，此处仅选取工作方式0作为案例进行分析。

方式0又称为基本输入输出方式。方式0的示意图如图2-5所示。

手机屏幕截图

中度可信度描述已自动生成

图2-5 方式0下的端口示意图[6]

这种方式有如下两点特点：

* A口、C口的高4位，B口以及C口的低4位可分别定义为输入或输出，各端口互相独立，故共有16种不同的组合。例如可定义A口和C口高4位为输入口，B口和C口低4位为输出口；或A口为输入，B口、C口高4位、C口低4位为输出等等。
* 在方式0下，C口有按位进行置位和复位的能力。

方式0最适合用于无条件传送方式，由于传送数据的双方互相了解对方，所以既不需要发控制信号给对方，也不需要查询对方状态，故CPU只需直接执行输入输出指令便可将数据读入或写出。

方式0也能用于程序查询方式，由于没有规定固定的应答信号，这时常将C口的高4位（或低4位）定义为输入口，用来接收外设的状态信号；而将C口的另外4位定义为输出口，输出控制信息。此时的A、B口可用来传送数据。

2.6 8255的应用

8225可应用于打印机接口、键盘接口等。介于篇幅原因与知识深度，8255的应用不以具体案例进行分析，仅分析大致流程。在应用8255可编程接口之前，需要设计为实现相应功能的硬件线路图。在硬件线路设计完成后，需要进行相应的软件设计。

对于8255可编程接口，软件设计包括初始化程序设计和实现数据传输的控制程序设计两部分。作为可编程接口，8255在使用时首先需要初始化，即将适当的控制字写入8255的控制寄存器中。只有在初始化结束后，才能够进行正常的数据传送。在数据传送过程中，CPU还要通过8255向外设发出控制信号并接收外设的状态信息。数据传送的方式可根据外部设备的性质及具体的应用，采用各种输入输出方法[7]。

2.7 总结

8255正是Intel公司生产的为x86系列CPU配套的可编程并行接口芯片。8255的通用性较强、使用灵活，是一种典型的可编程并行接口。通过研究并行接口8255实现原理，我更好的掌握了I/O接口的结构，也了解到并行数据接口的实现原理。在学习过程中，我也对比了串行数据接口与并行数据接口两者之间的区别。但是由于自身能力和知识面所限，对这部分内容的理解只停留在初步，为了更好掌握，我还希望微机原理与接口技术等相关课程与知识内容。

3 关键技术分析2：虚拟存储器的替换算法

当从辅存调页至主存而主存已满时，也需要进行主存页面的替换。为了增加访问命中率，替换策略的选取大为重要，因此存在着不同的的替换算法。

书本在第3章存储系统中介绍了三种替换算法——先进先出（FIFO）算法、最不经常使用（LFU）算法与近期最少使用（LRU）算法。我对这个知识点进行了拓展了解和对比后，在此介绍几类虚拟存储器的替换算法，通过实际案例分析其算法流程，并比较其性能的优劣。

3.1 最佳（Optimal）置换算法

最佳置换算法是由Belady于1966年提出的一种理论上的算法。其所选择的被淘汰页面将是以后永不使用的，或许是在最长（未来）时间内不再被访问的页面。采用最佳置换算法通常可保证获得最低的缺页率。但是还是会面对一个无法实现的问题，我们还是不知道哪些页面会在未来多长一段时间内不会被再次访问[8]。页面无法确认，时间也无法确定。因而该算法是无法实现的，但可以利用该算法去评价其它算法。如果有一种算法其效率最接近最佳置换算法，那么这种算法无疑是优秀的算法。

3.2 先进先出（FIFO）置换算法

图表, 条形图

描述已自动生成FIFO算法是最早出现的置换算法，其思想与队列相同。该算法总是淘汰最先进入内存的页面，即选择在内存中驻留时间最久的页面予以淘汰。该算法实现简单，只需把一个进程已调入内存的页面按先后次序链接成一个队列，并设置一个指针，称为替换指针，使它总是指向最老的页面。但该算法与进程实际运行的规律不相适应，因为在进程中，有些页面经常被访问，比如含有全局变量、常用函数、例程等的页面，FIFO算法并不能保证这些页面不被淘汰[8]。

如右图所示，假设内存只能保存4个页面，此时的访问请求按照时间顺序是1-2-3-4-5，那么按照时间顺序，当访问到4号页面时队列正好填满，当要访问5号页面时，会将最先入队的1号页面替换。

FIFO的实现很简单，但是从实现上来看，性能和最佳置换算法差距最大。因为被替换出去的页面很有可能是最常使用的页面，而且FIFO算法会出现Belay异常，因此这个算法很少见出现在数据库缓冲池管理中的。

图3-1 FIFO替换算法案例

3.3 近期最少使用（LRU）置换算法

通过之前FIFO的分析，可以发现：FIFO置换算法的性能之所以较差，是因为它所依据的条件是各个页面调入内存的时间，而页面调入的先后并不能反映页面的使用情况。近期最少使用（LRU）的页面置换算法是根据页面调入内存后的使用情况做出决策的。由于无法预测各页面将来的使用情况，只能利用“最近的过去”作为“最近的将来”的近似[9]。

因此，LRU置换算法是选择近期最少使用的页面予以淘汰。该算法赋予每个页面一个访问字段，用来记录一个页面自上次被访问以来所经历的时间。当需淘汰一个页面时，选择现有页面中其值最大的，即最近最久未使用的页面予以淘汰。

图表, 条形图

描述已自动生成

图3-2 LRU替换算法案例[10]

假设按照1-5-2-2-6-5-4的顺序访问，内存和访问示意图如上：其中最接近顶端的页面我们认为其t最小，最接近底部，我们认为其t最大。访问6号页面的时候，内存被填满，下一次访问5号页面的时候，会将5号页面提升到顶部，也就是t最小，之后访问4号页面，因为原先内存中没有4号页面，因此会选择替换一个页面。此时1号页面在底部，其t最大，因此被替换。因此，相比FIFO，LRU置换算法的效率更接近最佳置换算法。

3.4 最不经常使用（LFU）置换算法

最不经常使用（LFU）算法认为应将一段时间内被访问次数最少的页面换出。为此，需要在内存中为每个页面设置一个寄存器作为计数器。新页调入后从0开始计数，每访问一次，被访页的计数器增1。当需要替换时，对这些特定页的计数值进行比较，将计数值最小的页换出，同时将这些特定页的计数器都清零。这种算法将计数周期限定在两次替换之间的间隔时间内，因而不能严格反映近期访问情况[9]。

一般情况下，LFU能够避免周期性或者偶发性的操作导致缓存命中率下降的问题。但LFU需要记录数据的历史访问记录，一旦数据访问模式改变，LFU需要更长时间来适用新的访问模式，即：LFU存在历史数据影响将来数据的“缓存污染”效用。

3.5 简单时钟（Clock）置换算法

虽然LRU是一种较好的算法，但由于它要求有较多的硬件支持，使得其实现所需的成本较高。因此，我设想是否存在一种算法，有着类似于LRU的性能，但可以使用较少的硬件呢？经过资料查找后，我发现在实际应用中，简单Clock算法就是用得较多的一种LRU近似算法。

当利用简单Clock算法时，只需为每页设置一位访问位，再将内存中的所有页面都通过链接指针链接成一个循环队列。当某页被访问时，其访问位被置1。置换算法在选择一页淘汰时，只需检查页的访问位。如果是0，就选择该页换出；若为1，则重新将它置0，暂不换出，给予该页第二次驻留内存的机会，再按照FIFO算法检查下一个页面。当检查到队列中的最后一个页面时，若其访问位仍为1，则再返回到队首去检查第一个页面。图5-8示出了该算法的流程和示例。由于该算法是循环地检查各页面的使用情况，故称为Clock算法。但因该算法只有一位访问位，只能用它表示该页是否已经使用过，而置换时是将未使用过的页面换出去，故又把该算法称为最近未用算法或NRU（Not Recently Used）算法[8]。

假设按照1-2-3-4的顺序访问页面，则内存缓冲池会以这样的一种顺序被填满：

图片包含 ipod, 钟表, 游戏机

描述已自动生成

图3-3 简单时钟替换算法案例-1[10]

我们可以注意到中间的指针，就像是时钟的指针一样在移动。这样的访问结束后，缓冲池里现在已经被填满了，此时如果要按照1-5的顺序访问，那么在访问1的时候是可以直接命中缓存返回的，但是访问5的时候，因为缓冲池已经满了，所以要进行一次替换操作，示意图如下：

图片包含 ipod, 钟表, 台球, 房间

描述已自动生成

图3-4 简单时钟替换算法案例-2[10]

最初要经过一轮遍历，每次遍历到一个节点发现时，将该标记位置为0，然后遍历下一个页面，一轮遍历完后，发现没有可以被替换的页面，则进行下一轮遍历，这次遍历之后发现原先1号页面的标记位，则将该页面置换为页面5，并将指针指向下一个页面。

3.6 总结

LRU算法和FIFO本质上都是队列的先进先出的思路，只不过LRU时针对页面最近访问的时间来进行排序，所以需要在每一次页面访问的时候动态地调整各个页面之间的先后顺序；而FIFO是针对页面进入内存的时间来进行排序的，这个时间是固定不变的，所以各个页面之间的先后顺序是固定的[11]。

如果一个页面在进入内存后没有被访问，那么他的最近访问时间就是他进入内存的时间。换句话说，如果内存当中的所有页面都未曾访问过，那么LRU算法就退化为FIFO算法。

简单Clock算法则是使用了一些内存中的信息来模拟LRU算法，与LRU算法表现相类似，但对于那些曾经被访问过的页面，则不能像LRU算法准确记录他们的位置。由于自身知识面所限，对这部分内容的了解停留在表层，为了更好掌握，我还需要学习操作系统的课程知识内容。

4 实际案例分析——以“hello, world”程序为例

Kernighan和Ritchie在其撰写的C语言经典教材《C程序设计语言》中，通过所示的“hello, world”程序来向读者介绍C语言。尽管“hello, world”程序非常简单，但是为了让它实现运行，计算机系统的每个主要组成部分都需要协调工作。从某种意义上来说，通过了解在系统上执行“hello, world”程序时，系统发生了什么，以及为什么会这样的过程中，可以串联起计算机组成原理A(1)与(2)中所学习到的知识点，帮助我们更好掌握这些知识。

因此这部分中。我将通过跟踪“hello, world”程序的生命周期——从它被程序员创建开始，到在系统上运行，输出简单的信息，然后终止，对所涉及到的课程关键概念、专业术语和执行流程进行回顾、分析与总结。当然，由于其中涉及到了操作系统以及编译原理课程的相关知识点，对于此部分内容将简化。

4.1 代码的输入与存储形式

“hello, world”程序的生命周期是从一个源程序（或者说源文件）开始的，即程序员通过编辑器创建并保存的文本文件。

为了将组成“hello, world”程序的文字输入进电脑，此时需要使用外部设备——键盘。正如研讨环节中我组对USB设备从接入到使用的介绍中讲述的，键盘采用中断传输方式。该方式传送的数据量很小，但这些数据需要及时处理，以达到实时效果。当按下一个键或释放一个键，键盘都会通过USB接口与USB数据线发送键盘扫描码到主机。比如按下A，键盘就会发送0x1C到主机。当键盘被释放，键盘会发送0xF0加键码到主机，告诉主机键盘的哪个键被释放。键盘的每一个键都有一个特定的键码，无论Shift、Num Lock等按键是否被按下，键盘总是发送相同键码，而键盘BIOS会负责区分Shift、Num Lock的状态。在输入过程中，也涉及到了输入/输出系统、总线系统等相关知识，由于在第二部分中已经介绍了可编程并行接口8255，受篇幅限制，在此不详细展开。

源程序实际上就是一个由值为0和1组成的位（比特）系列，8个位被组成一组，称之为字节，每个字节表示程序中的某些文本字符。在计算机组成原理A(1)学习的第2章2.1.3字符与字符串的表示方法中曾介绍过，大部分的现代计算机系统都采用ASCII标准来表示文字符号，即用一个唯一的单字节大小的整数值来表示每个字符。下图给出了源程序hello.c的ASCII码表示[1]。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **#** | **i** | **n** | **c** | **l** | **u** | **d** | **e** | ***SP*** | **<** | **s** | **t** | **d** | **i** | **o** | **.** |
| 35 | 105 | 110 | 99 | 108 | 117 | 100 | 101 | 32 | 60 | 115 | 116 | 100 | 105 | 111 | 46 |
| **h** | **>** | **\n** | **\n** | **i** | **n** | **t** | ***SP*** | **m** | **a** | **i** | **n** | **(** | **)** | **\n** | **{** |
| 104 | 62 | 10 | 10 | 105 | 110 | 116 | 32 | 109 | 97 | 105 | 110 | 40 | 41 | 10 | 123 |
| **\n** | ***SP*** | ***SP*** | ***SP*** | ***SP*** | **p** | **r** | **i** | **n** | **t** | **f** | **(** | **“** | **h** | **e** | **l** |
| 10 | 32 | 32 | 32 | 32 | 112 | 114 | 105 | 110 | 116 | 102 | 40 | 34 | 104 | 101 | 108 |
| **l** | **o** | **,** | ***SP*** | **w** | **o** | **r** | **l** | **d** | **\** | **n** | **”** | **)** | **;** | **\n** | ***SP*** |
| 108 | 111 | 44 | 32 | 119 | 111 | 114 | 108 | 100 | 92 | 110 | 34 | 41 | 59 | 10 | 32 |
| ***SP*** | ***SP*** | ***SP*** | r | e | t | u | r | n | ***SP*** | 0 | ; | \n | } | \n |  |
| 32 | 32 | 32 | 114 | 101 | 116 | 117 | 114 | 110 | 32 | 48 | 59 | 10 | 125 | 10 |  |

图4-1 hello.c的ASCII文本表示

4.2 从源文件到目标文件

“hello, world”程序的生命周期是从一个高级C语言程序开始的，因为这种形式能够被人读懂。然而，为了在系统上运行hello.c程序，每条C语句都必须被其他程序转化为一系列的低级机器语言指令。然后这些指令按照一种称为可执行目标程序的格式打好包，并以二进制磁盘文件的形式存放起来。目标程序也称为可执行目标文件。

在Unix系统上，从源文件到目标文件的转化是由编译器驱动程序完成的：

linux> *gcc -o hello hello.c*

计算机组成原理A(1)课程对C语言的转换层次进行了介绍。结合其中知识，可以发现GCC 编译器驱动程序读取源程序文件hello.c后，把它翻译成一个可执行目标文件hello。翻译过程可分为四个阶段完成，如图所示。而执行这四个阶段的程序（预处理器、编译器、汇编器和链接器）一起构成了编译系统。

图示

描述已自动生成

图4-2 编译系统

特别的，在编译阶段中，编译器（ccl）将文本文件hello.i翻译成文本文件hello.s，它包含一个用特殊语言写成的程序——汇编语言。该程序包含函数main的定义，如下所示：

|  |
| --- |
| main:  subq $8, %rsp  movl $.LC0, %edi  call puts  movl $0, %eax  addq $8, %rsp  ret |

在第4章4.2.4指令助记符中，我学习过：由于硬件只能识别1和0，所以采用二进制操作码是必要的，但是我们用二进制来书写程序却非常麻烦。为了便于书写和阅读程序，每条指令通常用3个或4个英文缩写字母来表示。这种缩写码称为指令助记符。由于指令助记符提示了每条指令的意义，因此比较容易记忆，书写起来比较方便，阅读程序容易理解。因此，我们可以使用其编写程序，而汇编语言也正采用了此类助记符号。

4.3 从磁盘加载可执行文件到主存

编译完成的程序存储在外部设备，如固态硬盘SSD或机械硬盘HDD中。因此，当运行运行“hello, world”程序时，CPU需要从磁盘读取数据。此处以一个虚拟场景为例，进行分析。

CPU使用一种称为内存映射I/O的技术来向I/O设备发出命令。在使用内存映射I/O的系统中，地址空间由一部分是为与I/O设备保留的，每个地址称为一个I/O端口。当一个设备连接到总线时，它与一个或多个端口相关联（或它被映射到一个或多个端口）。在这学期计算机组成原理A(2)课程中，我学习过磁盘的基本原理与各种方式。其中，硬盘的电路板结构中存在一个芯片——磁盘控制器，它是主机与磁盘驱动器之间的接口。作为主机与驱动器之间的控制器，它需要有两个方面的端口：一个是与主机的端口，控制外存与主机总线之间交换数据；另一个是与设备的接口，根据主机命令控制设备的操作。前者称为系统级接口，后者称为设备级接口[1]。

图示

描述已自动生成

图4-3 磁盘控制器逻辑框图

假设系统级接口中的端口（寄存器）映射到端口0xc0后，CPU通过执行三个对地址0xc0的存储指令，发起磁盘读操作：第一条指令是发送一个命令字，告诉磁盘发起一个读，同时还发送了其他的参数，例如当读完成时，是否中断CPU。第二条指令指明应该读的逻辑块号。第三条指令指明应该存储磁盘扇区内容的主存地址。在磁盘控制器收到来自CPU的读命令操作后，它将逻辑块号翻译成一个扇区地址，磁盘上的信息经读磁头读出以后送读出放大器，然后进行数据与时钟的分离，再进行串-并变换、格式变换，最后送入数据缓冲器。经计算机组成原理A(2)中学习的DMA（直接内存访问）控制将数据传送到主机总线，直接传送到主存中，不需要CPU干涉[12]。

在DMA传送完成，磁盘扇区的内容被安全地存储在贮存中以后，磁盘控制器通过给CPU发送一个中断信号来通知CPU，整个过程如下图所示：

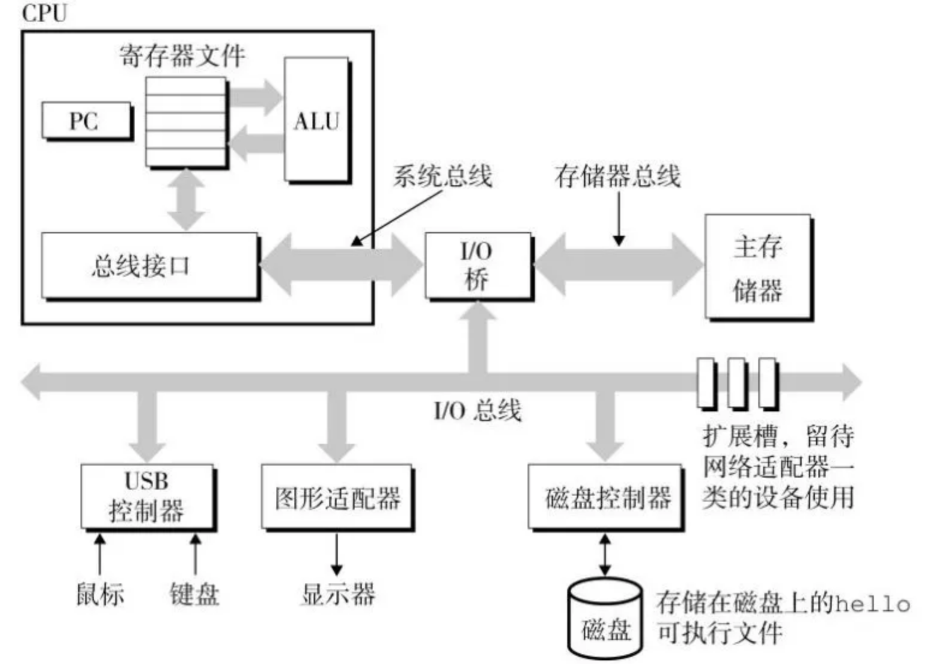


图4-4 从硬盘加载可执行文件至主存

4.4 将字符串输出至显示屏

一旦目标文件hello中的代码和数据被加载到主存，CPU就开始执行hello 程序的main 程序中的机器语言指令。这些指令将“hello, world\n”字符串中的字节从主存复制到寄存器文件，再从寄存器文件中通过数据总线复制到显示设备，最终通过显示在屏幕上。关于屏幕及其显示原理部分，由于在第7章外围设备中有详细介绍，且不为重点内容，因此不过多赘述。

整个过程如下图所示：

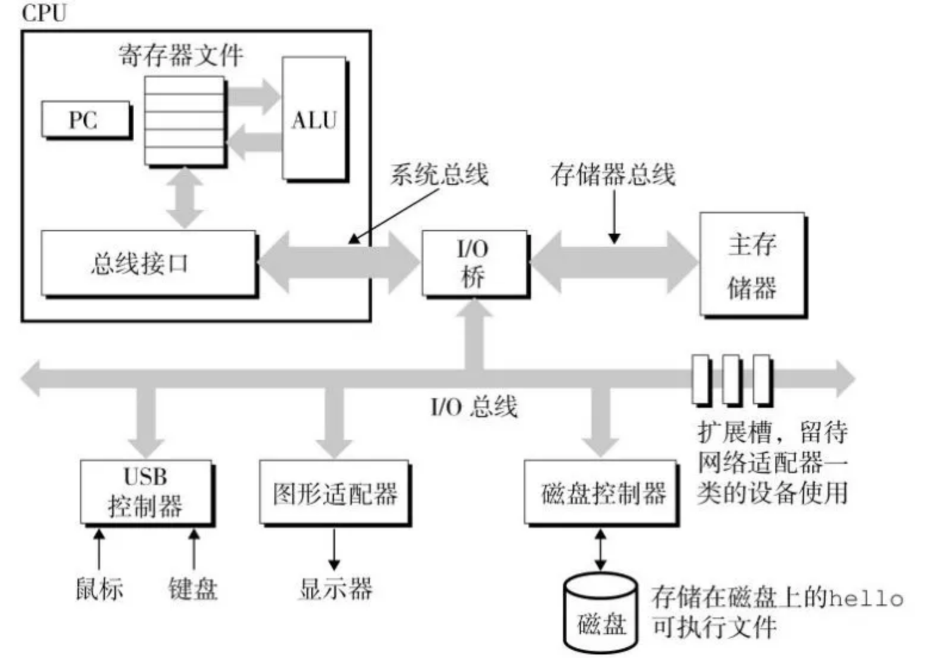


图4-5 将“hello, world”字符串从存储器输出至显示器

5 结语

回顾计算机组成原理A(2)的课程内容，我学习了存储系统、输入/输出系统与总线系统的三大系统，并了解了与之相关的外围设备。通过结合计算机组成原理A(1)的课程内容，“补全”了这块“拼图”，让我对计算机的底层原理有了完整的、全貌的理解。

当然，我也意识到计算机组成原理，或者说计算机硬件相关知识的学习是永无止境的，且自身的能力尚且存在不足。由于缺少实践经验，导致在理解知识点与完成课程作业中遇到诸多困难。在撰写报告的时候，我发现涉及内容的底层原理与其他课程也有联系，例如微机原理、接口技术、汇编语言程序设计、操作系统、编译原理等等。因此，要更好的掌握计算机组成原理，还需要我在课外通过多动手尝试、多学习来增强自身能力。

6 致谢

感谢余老师的认真教授，使我掌握了课程的知识点，并在课后耐心地解答我的困惑。感谢同学们对我的帮助。在大家的帮助下，顺利地完成了计算机组成原理A(2)课程的学习！

参考文献

1. 白中英, 戴志涛. 计算机组成原理[M]. 第六版·立体化教材. 北京: 科学出版社, 2019.
2. 袁春风, 杨若瑜, 王帅, 唐杰. 计算机组成与系统结构[M]. 第2版. 北京: 清华大学出版社, 2015.
3. 钱晓捷, 王义琴, 范喆, 马行进, 马耀峰. 微机原理与接口技术·基于IA-32处理器和32位汇编语言[M]. 第5版. 北京: 机械工业出版社, 2014.
4. 周荷琴, 冯焕清. 微型计算机原理与接口技术[M]. 第6版. 合肥: 中国科学技术大学出版社, 2019.
5. 白莉媛, 肖乐, 钱进. Intel 8255A微机芯片的应用[J]. 光盘技术,2006(6):50-51.
6. 吴宁, 乔亚男, 冯博琴. 微型计算机原理与接口技术[M]. 第4版. 北京: 清华大学出版社, 2016.
7. 刘正红.浅议并行接口芯片8255A的可编程特性[J].电子世界,2012(22):21.
8. 汤小丹, 梁红兵, 哲凤屏, 汤子瀛. 计算机操作系统[M]. 第4版. 西安: 西安电子科技大学出版社, 2018.
9. 博客园. 两种常见的缓存淘汰算法LFU&LRU[EB/OL]. [2019-08-05]. https://www.cnblogs.com/applelife/p/11303881.html.
10. 简书. 页面置换算法之Clock算法[EB/OL]. [2020-02-11]. https://www.jianshu.com/p/23ba78ec4346.
11. 魏文国,赵慧民,庄林凯,许鸿俊.一种基于时钟自适应的改进缓存替换算法[J].中山大学学报(自然科学版),2012,51(06):54-57+62.
12. [美]兰德尔 E.布莱恩特, 大卫 R.奥哈拉伦. 深入理解计算机系统[M]. 原书第3版. 龚奕利, 贺莲译. 北京: 机械工业出版社, 2016.