微型计算机原理

教
案

数学与计算机科学学院

目　录

第一章 微型计算机概述

一、计算机的定义

计算机是一种信息处理的工具，是一种在程序的控制下完成预定任务的电子仪器。包括硬件和软件两部分。

信息处理：包括信息的收集、加工、存储和传送。

二、微型计算机的组成

硬件：构成计算机系统的物理实体。计算机的硬件设备主要有：运算器、控制器、存储器、输入输出设备等部件。

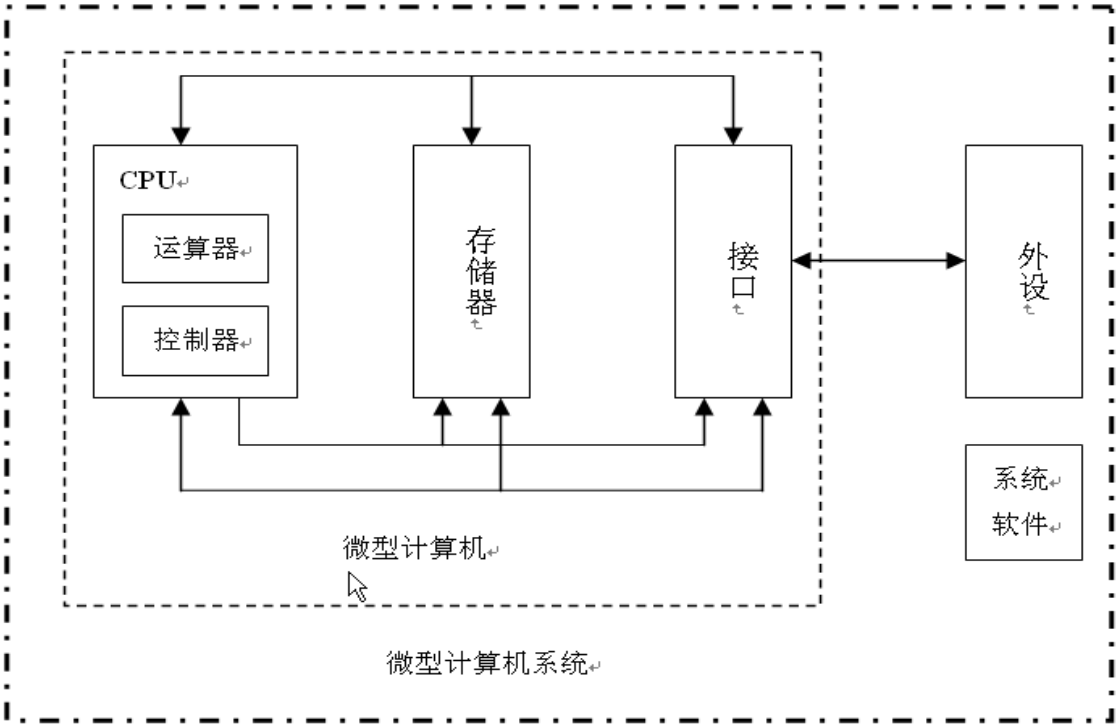


图 1.1 微型计算机组成图

1. 微处理器

运算器：完成算术及逻辑运算

算术运算：加、减、乘、除的定点、浮点运算。

逻辑运算：与（·或∧）、或（＋或∨）、非、异或

控制器：用于解释并协调整个系统完成指令的部件。控制器由指令寄存器、指令译码器、时序和控制电路，以及中断机构组成。指令寄存器存放当前正在执行的指令，而指令译码器对指令进行译码，此时，产生相应的控制信号送到时序

和控制电路，从而组合成 CPU 外部的其他部件所需要的时序和控制信号。这些信号送到微型计算机的其他部件，控制这些部件协调工作。

总之，微处理器是微型计算机的核心，它有两个指标：字长和主频。

2. 存储器

计算机中用于存储程序及数据的物理装置，分为内存和外存两大类。

内存包括 RAM 和 ROM，容量有限，用来存放经常使用的程序和数据，除必要的系统程序外，一般程序是存放在外存当中，只有在运行时才调入内存的某个区域。

外存比内存的容量大的多，但速度较慢，用来存放不常使用的程序和数据，通常作为某个外部设备。

3. 接口：用于计算机主机和外设进行匹配的电子部件。

4. 外设

输入设备：计算机从外部世界获取信息的入口。

输出设备：计算机向外部世界输出信息的出口，把运算结果或其它信息以数字、字符、图形等形式表示出来。

5. 总线

总线为 CPU 和其它部件之间提供数据、地址和控制信息的传输通道。有了总线结构以后，系统中各功能部件之间的相互关系就变为各个部件面向总线的单一关系。一个部件要符合总线标准，就可以连接到采用这种总线标准的系统中。

6. 系统软件

系统软件包括操作系统，一些语言处理程序和数据库。

其中操作系统是系统软件的核心，它管理计算机系统的全部硬件和软件资源，使计算机有条不紊的运行，为用户提供操作界面。

三、微型计算机的四个发展时期

1. 1971—1972 年 典型产品 Intel4004/8008，字长 4 位/8 位，主频 1MHz
2. 1973—1977 年 典型产品 Intel8080，字长 8 位，主频 2MHz。
3. 1978—1984 年 典型产品 Intel8086，字长 16 位，主频 5—10MHz。
4. 1985—至今 典型产品 Intel80386，字长 32 位，主频在 20MHz 左右。

第二章 微处理器 8086

一、16 位微处理器 8086

8086：字长 16 位，主频 5—10MHz，16 根数据总线和 20 根地址总线，可寻址 1MB 的内存储空间和 64KB 的 I/O 端口。

8088：准 16 位微处理器，内部寄存器、运算器以及内部数据总线都是按照 16 位来设计的，外部数据总线只有 8 条。

1.8086 的编程结构

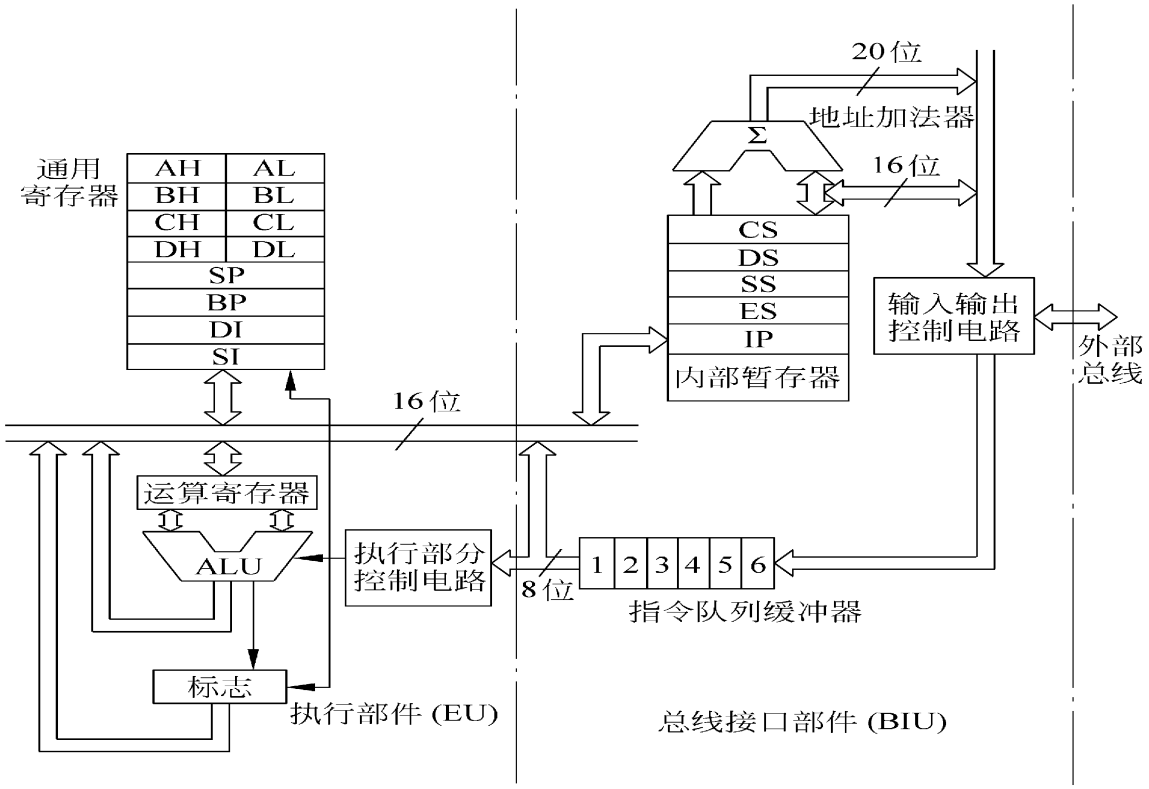


图 2.1 8086 的编程结构图

⑴总线接口部件 BIU

功能：负责与存储器、I/O 端口传送数据。

①BIU 要从内存取指令送到指令队列缓冲器。

②CPU 执行指令时，总线接口部件要配合执行部件从指定的内存单元或者外设端口取数据，将数据送给执行部件。

③把执行部件的操作结果传送到指定的内存单元或外设端口中。

8086 有 6 个字节的指令队列缓冲器，8088 有 4 个字节的指令队列缓冲器。

采用“先进先出”的原则，都会在执行指令的同时，从内存中取下面一条或

几条指令按顺序填入指令队列中。这样就保证了 8086/8088 执行完一条指令后可以立即执行下一条指令，也就是说执行指令和取指令的时间可以重叠，从而提高了 CPU 的利用率。

而早期的 8 位微处理器，取指令和执行指令是循环进行的。

⑵执行部件 EU

①执行单元 EU 的功能就是负责指令的执行。它包括下列部分。

②算术逻辑单元 ALU：ALU 完成 16 位或 8 位的二进制数的算术/逻辑运算，绝大部分指令的执行都由 ALU 完成。

③通用寄存器组和标志寄存器 FLAGS。

④控制电路 EU：接收从 BIU 中指令队列取来的指令，经过指令译码形成各种定时控制信号，向 EU 内各功能部件发送相应的控制命令，以完成每条指令所规定的操作。

2.BIU 和 EU 的动作管理

时钟周期——CPU 的基本时间计量单位，由计算机的主频决定。

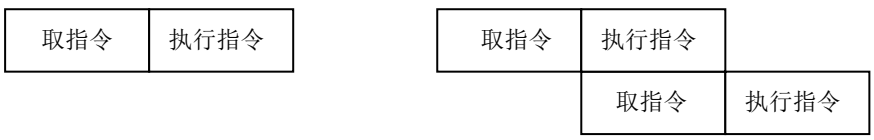
时钟周期=1/主频

例如：8086 的主频为 5MHz

时钟周期=1/(5×106Hz)=0.2×10-6s=200ns

总线周期——CPU 访问一次存储器或 I/O 端口所需要的时间。

由于 8086 的 BIU 中包含了一个 6 字节的指令队列缓冲器，CPU 在执行指令的同时，从存储器中读取下一条指令或下几条指令，取来的指令就放在指令队列中，这样，一般情况下，CPU 执行完一条指令就可以立即执行下一条指令，就够成了取指令和执行指令的二级流水线操作，从而提高了 CPU 的效率。如图而不像以往 8 位 CPU 重复进行先取指令和后执行指令的串行操作。负责取指令的总线接口单元和负责执行指令的执行单元需要按以下原则对流水线进行管理，才能提高 CPU 的执行效率。



（a）串行操作 CPU　　（b）8086 执行执行过程

图 2.2 CPU 执行指令过程

（1）当 8086 的指令队列中有两个空字节时，总线接口单元就会自动把指令取到指令队列中。

（2）当执行部件准备执行一条指令时，它会从总线接口部件的指令队列的队头取出指令代码，然后去执行指令。在执行指令过程中，如果需要访问存储器或 I/O 端口，那么执行部件就会请求总线接口单元，若总线接口单元空闲，则立即响应执行部件的总线请求；若总线接口单元正在取指令，则总线接口单元将先完成取指令的总线周期，再去响应执行部件发出的总线请求。

（3）当指令队列已满，而且执行部件又没有总线访问时，总线接口单元便进入空闲状态。

（4）在执行转移指令、调用指令和返回指令时，下面要执行的指令就不是在程序中的下一条指令了，而总线接口单元往指令队列装入指令时，总是按顺序进行的，这样，指令队列中已装入的指令就没有用。遇到这种情况，指令队列中的原有内容被自动清除，总线接口单元会接着往指令队列中装入另一个程序段中的指令。

3.8086 的总线周期

为了取得指令和数据，BIU 执行一个总线周期。在 8086/8088 中，一个基本的总线周期由 4 个时钟周期组成，将这 4 个时钟周期分别称为 4 个状态，即 T1 状态、T_2 状态、T_3 状态、T_4 状态。

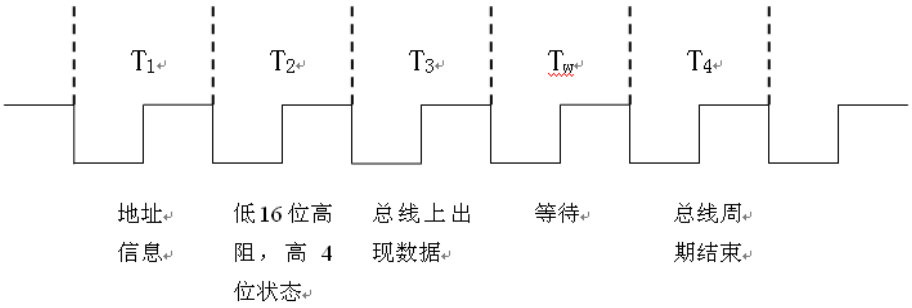


图 2.3 总线周期

二. 8086/8088 存储器和 I/O 编址

1. 8086/8088 存储器组织

8086/8088 的存储器是以字节为单位组织的。具有 20 条地址总线，所以可寻址的存储器地址空间容量为 1 MB（2^{20}B）。每个字节对应一个唯一的地址，地址

范围为 00000H～FFFFFH。

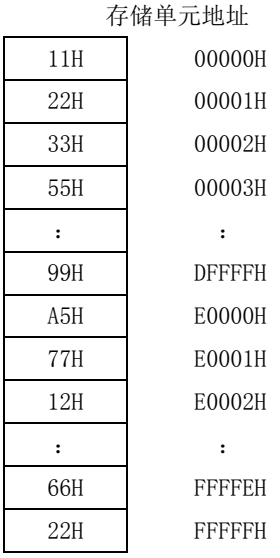


图 2.4 数据在存储器中的存放

一个存储单元中存放的信息称为该存储单元的内容。如 00000H 单元的内容为 11H，记为：(0000H)=11H。

存储器中两个连续的字节，定义为一个字。一个字中的每个字节，都有一个字节地址，字的低字节存放在低地址中，高字节存放在高地址中。如从地址 00002H 开始的两个连续单元中存放一个字，则该数据位 5533H，记为：(00002H)=5533H。

由于 8086/8088CPU 内部数据总线和寄存器均为 16 位，内部 ALU 只能进行 16 位运算，因此 8086/8088CPU 对地址只能进行 16 位运算，其寻址范围为 2^{16}=64KB，但处理器地址总线的位数是 20 位。所以为了获得 20 位地址，需要引入“分段”概念，将寻址范围扩大到 1MB。

将段寄存器内容左移 4 位（低 4 位补 4 个 0）与偏移量相加，即形成 20 位的物理地址。

物理地址=段地址×10H+段内偏移量

例如段寄存器（CS）=1120H, 指令指针（IP）=2008H，则逻辑地址表示为 CS:IP=1120H:2008H，物理地址则为 1120H×10H+2008H=13208H。

2. I/O 编址

8086/8088 系统和外部设备之间都是通过 I/O 接口来联系的。每个 I/O 接口都有一个或几个端口，微机系统要为每个端口分配一个地址，此地址称为端口号。一个端口通常对应一个或一组寄存器。

8086/8088 的 I/O 端口地址总线为 16 位，即提供 2^{16}=65536（64K）个 8 位的 I/O 端口，两个编号相邻的 8 位端口可以组合成一个 16 位端口。

三、8086 的引脚信号

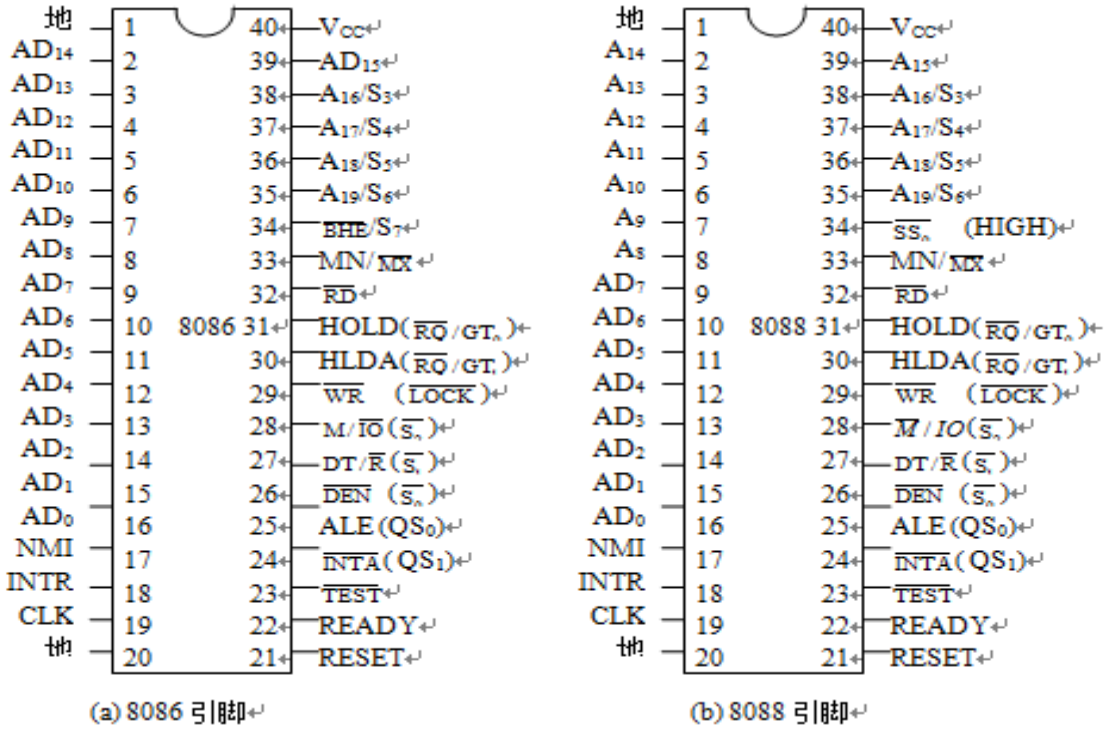


图 2.5 8086/8088 引脚图

1. 地址/数据复用引脚

AD_0～AD_{15}：地址/数据分时复用输入输出信号线。分时复用，T_1 时刻出现地址，T_3 时刻出现数据。

2. 地址/状态复用总线

A_{19}/ S_6～A_{16}/ S_3:每个总线周期开始 T_1 状态时做地址线用，其余状态输出状态信息。

当访问存储器时，T1 状态输出 A_{19}～A_{16}，与 AD_{15}～AD_0 一起构成访问存储器的 20 位物理地址。

当 CPU 访问 I/O 端口是，不使用这 4 个引脚，A_{19}～A_{16} 保持为 0。

在 T_2～T_4 状态，S_6 为 0 表示 8086CPU 当前与总线相连；S_5 表示中断允许标志位 IF 的当前设置；S_4S_3 用来指示当前正在使用哪个段寄存器。

3.控制总线

NMI：非屏蔽中断请求信号，输入，上升沿触发。

INTR：可屏蔽中断请求信号，输入，高电平有效。若此信号有效，表明有外设提出了中断请求，这时若 IF=1，则当前指令执行完成后立即响应中断。

RESET：CPU 的复位信号，输入，高电平有效。复位信号使处理器马上结束现行操作，对处理器内部寄存器进行初始化。8086 要求复位脉冲宽度不得小于 4 个时钟周期。

READY：数据“准备好”信号，输入。高电平有效。当 CPU 对存储器或 I/O 短口进行操作时，在 T_3周期开始采样 READY 信号。若它为高电平，则表示存储器或 I/O 设备已准备好；若为低电平，则表示被访问的存储器或 I/O 设备还未准备好数据，此时在 T_3周期以后插入 Tw 周期，然后再在 Tw 周期中再采样 READY 信号，直到 READY 变为高电平时 Tw 周期才可以结束，进入 T4 周期，完成数据传送。

$\overline{\text{TEST}}$：用于测试的输入信号。当 WAIT 指令执行时，该信号低电平有效，CPU 就进入等待状态

$\text{M/}\overline{\text{IO}}$：用来区分当前操作是访问存储器还是访问 I/O 端口。

$\overline{\text{WR}}$：该引脚输出为低电平时，表示 CPU 正处于写存储器或写 I/O 端口。

$\overline{\text{RD}}$：低电平有效，表示 CPU 正在进行存储器或 I/O 端口读操作。

$\text{DT/}\overline{\text{R}}$：用于确定数据传送的方向，高电平为发送方向，即 CPU 写数据到存储器或 I/O 端口；低电平为接收方向，即 CPU 到存储器或 I/O 端口读数据。

$\overline{\text{DEN}}$：该信号有效时，表示数据总线上有有效数据。

ALE：高电平有效。当它有效时，表明 CPU 送出有效的地址信号，因此，它常作为锁存控制信号将 A_0～A_{19}锁存于地址锁存器中。

$\overline{\text{INTA}}$：CPU 输出的中断响应信号，是 CPU 对外部输入的 INTR 中断请求信号的响应。

HOLD：输入信号高电平有效，用于向 CPU 提出总线保持请求。当某一部件要占用系统总线时，可通过这条输入线向 CPU 提出请求。

HLDA：CPU 对 HOLD 请求的响应信号，高电平有效的输出信号。

$MN/\overline{MX}$：当 $MN/\overline{MX}$ =1 时，CPU 工作在最小模式下，构成的微型机中只包括一个微处理器；当 $MN/\overline{MX}$ =0 时，CPU 工作在最大模式下，构成的微型机系统中除了有 8086CPU 之外，还可以有另外的微处理器。

四、8086 的的操作和时序

1. 总线操作

总线读操作——指 CPU 从存储器或 I/O 端口读取数据。

总线写操作——指 CPU 将数据写入存储器或 I/O 端口。

⑴最小模式下的总线读操作（以读存储器为例）

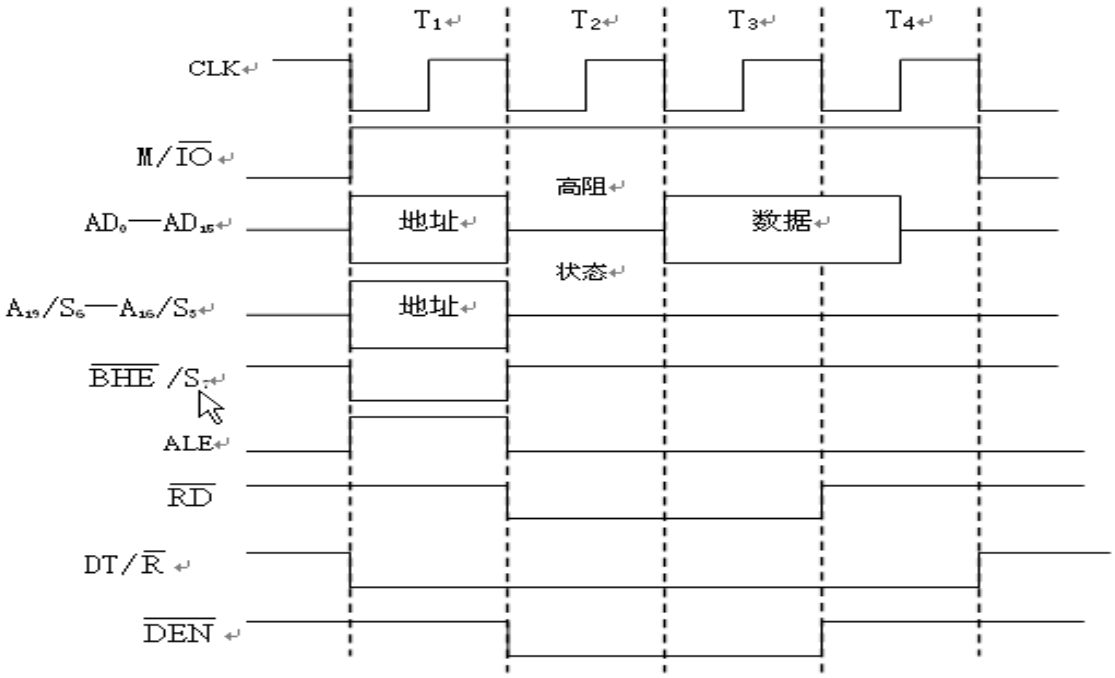


图 2.6 总线读操作时序图

T_1 状态：

首先要用 $M/\overline{IO}$ 信号指出 CPU 是要从存储器还是 I/O 端口读，所以 $M/\overline{IO}$ 信号在 T_1 状态成为有效。$M/\overline{IO}$ 信号的有效电平一直保持到整个总线周期的结束，即 T_4 状态。

8086 的 20 位地址信号通过分时复用总线 A_{19}/S_6 ～A_{16}/S_3 和 AD_{15}～AD_0 输出，

送到存储器或 I/O 端口。

CPU 便在 T1 状态从 ALE 引脚上输出一个正脉冲作为地址锁存信号。在 ALE 的下降沿到来之前，$M/\overline{IO}$ 信号、地址信号均以有效。

$\overline{BHE}$ 信号也通过 $\overline{BHE}/S_7$ 引脚送出，它用来表示高 8 位数据总线上的信息可以使用。

当系统中连接有数据总线收发器时，要用到 $DT/\overline{R}$ 和 $\overline{DEN}$ 作为控制信号。前者作为对数据传输方向的控制，后者允许数据传送。

T_2 状态：

地址信号消失，AD_{15}～AD_0 进入高阻状态，为读入数据做准备；而 A_{19}/S_6～A_{16}/S_3 和 $\overline{BHE}$ /S7 输出状态信息 S_7～S_3。

此时，$\overline{DEN}$ 信号变为低电平，从而在系统中接有总线收发器时，获得数据允许信号。

CPU 在 $\overline{RD}$ 引脚上输出读有效信号，送到系统中所有存储器和 I/O 接口芯片，但是，只有被地址信号选中的存储单元或 I/O 端口，才会被信号从中读出数据，从而将数据送到系统数据总线上。

T_3 状态：

在 T_3 状态前沿，CPU 对引脚 READY 进行采样，如 READY 信号为高电平，则 CPU 在 T_3 状态后沿通过 AD_{15}～AD_0 获取数据；如果为低电平，将插入等待状态 Tw，直到 READY 信号变为高电平。

Tw 状态：

当系统中所用的存储器或外设的工作速度较慢，从而不能用最基本的总线周期完成读操作时，系统中就要用一个电路来产生 READY 信号。低电平的 READY 信号必须在 T_3 状态启动之前向 CPU 发出，则 CPU 将会在 T_3 状态和 T_4 状态之间插入若干个等待状态 Tw，直到 READY 信号变高。在最后一个等待状态 Tw 的后沿处，CPU 通过 AD_{15}～AD_0 获取数据。

T_4 状态：

CPU 使 $\overline{RD}$ 信号变为高电平，于是，存储器模块上的总线驱动器又处于高阻

状态，从而让出总线。

⑵最小模式下的总线写操作（以写存储器为例）

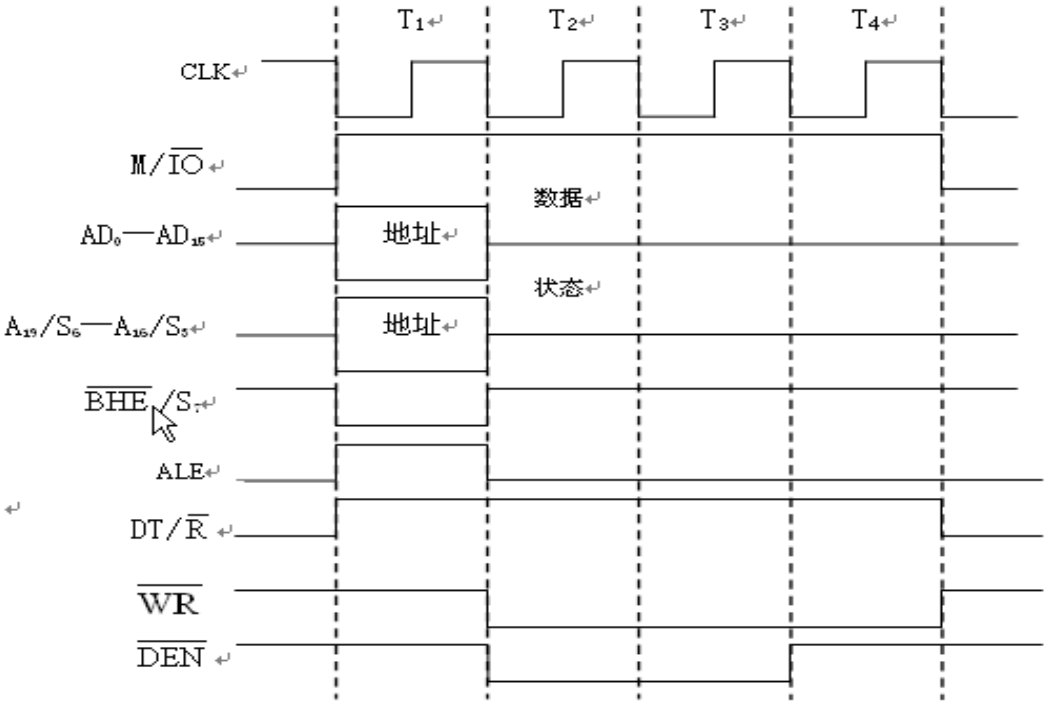


图 2.7 总线写操作时序图

对存储器或 I/O 端口操作时序基本相同。总线读操作中，选通信号是 $\overline{RD}$，而总线写操作中是 $\overline{WR}$。

在 T_2 状态中，AD_{15}～AD_0 上地址信号消失后，AD_{15}～AD_0 的状态不同。总线读操作中，此时 AD_{15}～AD_0 进入高阻状态，并在随后的状态中保持为输入方向；而在总线写操作中，此时 CPU 立即通过 AD_{15}～AD_0 输出数据，并一直保持到 T_4 状态中。

2. 中断系统和中断操作

8086/8088 可以处理 256 种不同的中断，每个中断对应一个中断类型码，中断类型码为 0——255。

⑴中断的分类

硬件中断是通过外部是硬件产生的

非屏蔽中断 NMI；可屏蔽中断 INTR

软件中断是 CPU 根据软件的某条指令或软件对标志寄存器中某个标志的设置而产生的。

比如：除数为 0 引起的中断；中断指令引起的中断

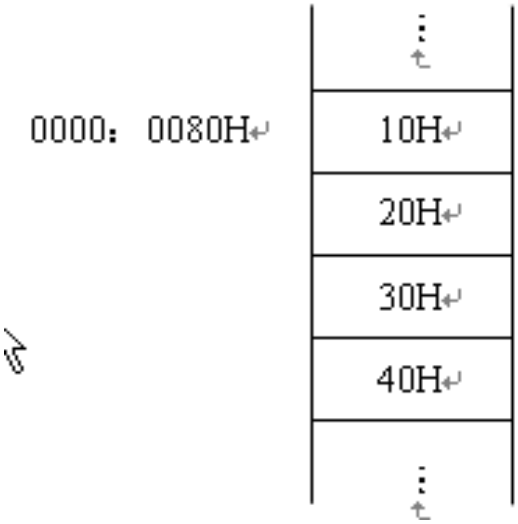
⑵中断向量和中断向量表

中断向量：中断处理子程序的入口地址。

每个中断类型对应一个中断向量。一个中断向量占用 4 个存储单元，其中前 2 个单元存放中断处理子程序入口地址的偏移地址；后 2 个单元存放中断处理子程序入口地址的段地址。

中断向量不是任意存放的。

中断向量表：内存 0 段的 0000—03FFH(1023D)区域。



20H 号中断对应的中断向量为 4030:2010H。

⑶硬件中断的响应和时序

可屏蔽中断的响应过程：

当 CPU 在 INTR 引脚上接收一个高电平的中断请求信号，并且当前中断允许标志为 1 时，CPU 就会在当前指令执行完以后，开始响应外部的中断请求，即 CPU 往发两个连续的负脉冲，外设接口接到第二个负脉冲以后，立即往数据总线上给 CPU 送来中断类型码。之后依次做以下工作：

①从数据总线上读取中断类型码，将其存入内部暂存器。

②将标志寄存器(PSW)的值压入堆栈；

③清楚中断允许标志 IF 和跟踪标志 TF；

④将断点保存到堆栈中；

⑤根据前面得到的中断类型码，到内存 0 段的中断向量表找到中断向量，再根据中断向量转入相应的中断处理子程序。

响应可屏蔽中断时的总线时序：

①执行 2 个中断响应总线周期。CPU 接收中断类型码，将它左移 2 位称为中断向量的起始地址，存入内部暂存器。

②执行一个总线写周期，将 PSW 的值压入堆栈。

③清除 IF 和 TF。

④执行一个总线写周期，将断点处 CS 的内容压入堆栈。

⑤执行一个总线写周期，将断点处 IP 的内容压入堆栈。

⑥执行一个总线读周期，CPU 将从中断向量所在的前 2 个字节中读取中断处理子程序入口地址的偏移地址送入 IP。

⑦执行一个总线读周期，将中断处理子程序入口地址的段地址送入 CS。

⑷中断处理子程序

①保护现场

②开中断

③中断服务——中断处理的具体内容

④关中断

⑤恢复现场

⑥中断返回

第三章 8086 的指令系统

一、计算机语言的发展

机器语言是由计算机能够识别并执行的 0、1 代码构成，是唯一能被机器识别并执行的语言。显然这种语言不便于人们阅读、理解、交流，编写程序复杂困难，并且很容易出错。既然这种语言在使用过程中有诸多不便之处，所以在计算机语言的发展过程中就出现了汇编语言和高级语言。

汇编语言是面向机器硬件的语言，是一种助记符语言。它以指令系统为核心，采用能够帮助理解和记忆的英文单词或其缩写符号来代替机器指令中的操作码，并对所需的数据、寄存器或有关数据的地址用相应的符号表示，并把每一条机器指令都用相应的符号化指令代替，与机器语言一一对应。

高级语言与计算机的硬件结构及指令系统无关，它有更强的表达能力，可方便地表示数据的运算和程序的控制结构，能更好的描述各种算法，而且容易学习掌握。但高级语言编译生成的程序代码一般比用汇编程序语言设计的程序代码要长，执行的速度也慢。

二、8086 寻址方式

所谓寻址方式是指寻找操作数或者操作数地址的方式。

一条汇编语言指令由操作码字段和操作数字段两部分组成。操作码指示计算机所要完成的操作，而操作数则是操作码的操作对象。换句话来讲，一条汇编语言指令需要解决两个问题，其一是要指出完成何种操作；其二是要指出大部分指令涉及的操作数和操作结果存放在何处，不但要指出操作数的值为多少或者存放在什么地方，而且还要指出操作结果送到哪里去。为了使指令形式比较简单，约定将操作结果送回到原来存放操作数的地方。这样，第二个问题，就归结为指出操作数的来源，这就是操作数的寻址方式。

通常，操作数有四个来源：立即数、寄存器、存储器、I/O 端口。

操作数字段可以有一个、两个或者没有，分别称为一地址指令、二地址指令和零地址指令。单操作数指令就是一地址指令，它只需要指定一个操作数。在 8086 指令系统中大部分属于二地址指令，此时这两个操作数分别称为目的操作数和源操作数，其格式如图 3.1 所示。

操作码	目的操作数,源操作数

图 3.1 二地址指令格式

1．立即寻址

操作数直接放在指令中，它是紧跟在操作码后面一个 8 位或 16 位的常数。

例 1：MOV AX，2345H

指令执行后，(AX)=2345H

这种寻址方式主要用来给寄存器或存储单元赋初值，并且只能出现在源操作数字段。

2．寄存器寻址

操作数在寄存器当中，在指令中指定寄存器号。

不同位数的操作数使用不同的寄存器。对于 16 位操作数，寄存器可以是 AX、BX、CX、DX、SI、DI、SP、BP，对于 8 位操作数，寄存器可以是 AH、AL、BH、BL、CH、CL、DH、DL。

例 2：MOV AX，BX

指令执行前(AX)=3088H，(BX)=5678H；则指令执行后(AX)=5678H，BX 保持不变。

3．直接寻址

指令操作码之后直接给出操作数的 16 位偏移地址，适用于处理单个变量。

例 3：MOV AX，[2000H]

设指令执行前(DS)=3000H，(32000H)=56H，(32001)=34H，则

物理地址=(DS)×16D+2000H =30000H+2000H=32000H

指令执行后：(AX)=3456H，寻址过程如图 3.2 所示：

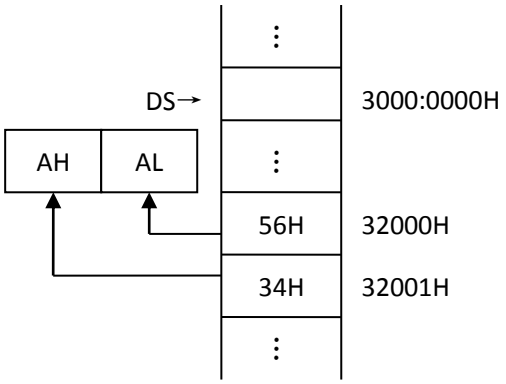


图 3.2 直接寻址示意图

在汇编语言指令中，可以用符号地址代替数值地址，如：

MOV　AX，VALUE

如果数据在附加段当中，应在指令中指明段跨越前缀，如：

MOV　AX，ES：VALUE

4．寄存器间接寻址

操作数的偏移地址在指令中指明的基址寄存器或变址寄存器中，而操作数则在存储器中。在 8086 系统中可以使用的基址或变址寄存器有 BX、BP、SI 和 DI，当使用 BP 时，默认的段寄存器为 SS，否则默认的段寄存器为 DS。

例 4：MOV　AX，[BX]

设指令执行前：(DS)=2000H，(BX)=1000H，(21000H)=A0H，(21001H)=50H，则

物理地址=(DS)×16D+(BX)=20000H+1000H=21000H

指令执行后：(AX)=50A0H，寻址过程如图 3.3 所示：

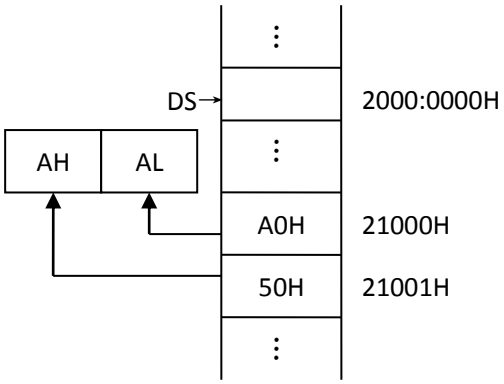


图 3.3 寄存器间接寻址示意图

5．寄存器相对寻址

操作数的偏移地址为基址寄存器或变址寄存器的内容和指令中指定的 8 位或 16 位的位移量之和。同样这里能够使用的基址或变址寄存器有 BX、BP、SI 和 DI，当使用 BP 时，默认的段寄存器为 SS，否则默认的段寄存器为 DS。

例 5：MOV　AX，COUNT[SI]，也可写成 MOV　AX，[COUNT＋SI]

设指令执行前：(DS)=3000H，(SI)=2000H，COUNT=3000H，(35000H)=45H，(35001H)=23H，则

物理地址=(DS)×16D+(SI)+ COUNT=30000H+2000H+3000H=35000H

指令执行后：(AX)=2345H，寻址过程如图 3.4 所示：

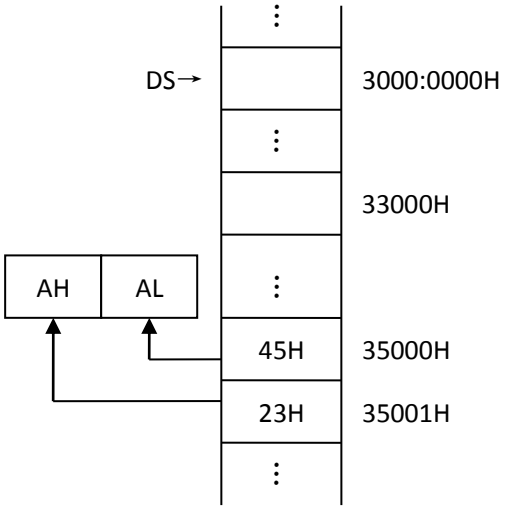


图 3.4 寄存器相对寻址示意图

6．基址变址寻址

操作数的偏移地址是指令中指定的一个基址寄存器和一个变址寄存器的内容之和。在 8086 系统中，能够作为基址寄存器使用的只能是 BX 和 BP，能够作为变址寄存器使用的只能是 SI 和 DI。

例 6：MOV　AX，[BX][DI]，也可写成 MOV　AX，[BX＋DI]

设指令执行前：(DS)=2000H，(BX)=1000H，(DI)=0010H，(21010H)=34H，(21011H)=12H，则

物理地址=(DS)×16D+(BX)+ (DI)=20000H+1000H+0010H=21010H

指令执行后：(AX)=1234H，寻址过程如图 3.5 所示：

7．相对基址变址寻址

指令中指定了一个基址寄存器和一个变址寄存器，同时还给出了一个 8 位或 16 位的位移量，将三者相加就得到操作数在内存中的偏移地址。

例 7：MOV　AX，VAL[BX][SI]，也可写成 MOV　AX，[VAL＋BX＋SI]

设指令执行前：(DS)=2000H，(BX)=1000H，(SI)=0200H，VAL=0050H，(21250H)=34H，(21251H)=12H，则

物理地址=(DS)×16D+(BX)+ (SI)＋VAL=20000H+1000H+0200H+0050H=21250H

指令执行后：(AX)=1234H，寻址过程如图 3.6 所示：

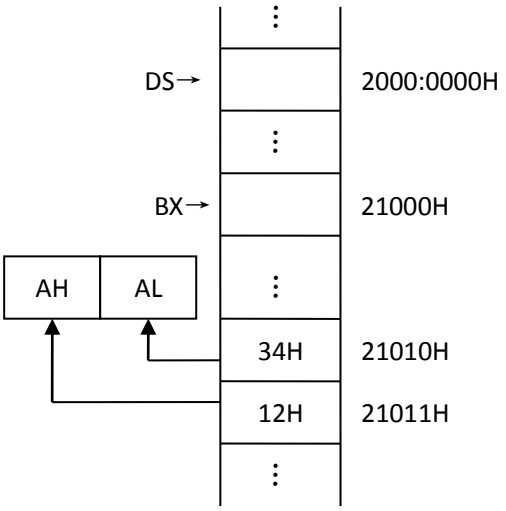


图 3.5 基址变址寻址示意图

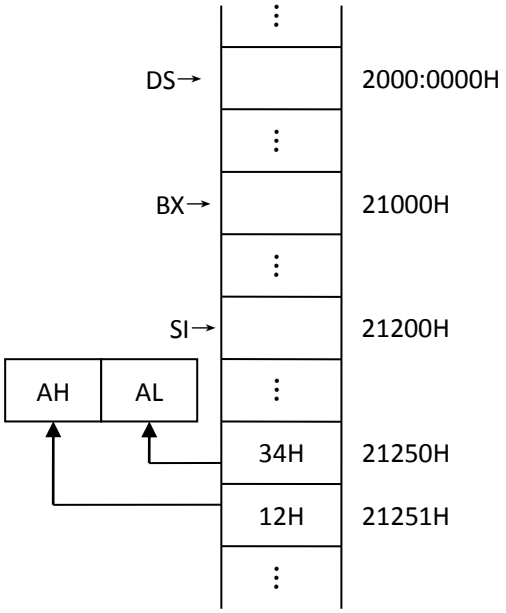


图 3.6 相对基址变址寻址示意图

三、程序转移寻址

程序正常顺序执行时，每取出一条指令执行(IP)＋n→(IP)，其中 n 为取出指令的字节数，然后形成下一条指令的物理地址。

物理地址=(CS)×16D+(IP)

但程序发生转移时，需要计算出偏移地址并修改 IP，有时候还需要修改 CS 的值，据此程序转移寻址可以分为段内转移和段间转移。段内转移是指在同一个代码段内，仅改变 IP 的值而不改变 CS 的值发生的转移。但是如果程序从一个代码段转移到另一个代码段，则不仅改变 IP 的值，同时要改变 CS 的值，这种情形

称为段间转移。

1．段内直接寻址

转向的偏移地址是当前 IP 寄存器的内容和指令中指定的 8 位或 16 位的位移量之和，然后再送入 IP。这种寻址方式适用于条件转移指令和无条件转移指令。当位移量为 8 位时，偏移地址形成一个转移范围在－128～+127 字节之间的短距离转移，则称为短跳转。当位移量为 16 位时，偏移地址形成一个转移范围在 64KB 内的近距离转移，则称为近跳转。

例 8：JZ NEXT

JMP SHORT LOP

JMP NLAB

2．段内间接寻址

转向的偏移地址在某个通用寄存器中或者在存储器的一个字单元中，则将寄存器的内容或存储单元的内容送入 IP，这个寄存器或存储单元的内容可以用操作数寻址方式中除立即寻址以外的任何一种寻址方式取得。这种寻址方式只能用于无条件转移指令。

例 9：指令 JMP BX

设指令执行前：(BX)=1256H，则

指令执行后：(IP)=1256H

例 10：指令 JMP [BX][SI]

设指令执行前：(DS)=2000H，(BX)=1200H，（SI）=0050H，（21250H）=50H，（21251H）=24H，则

物理地址=(DS)×16+(BX)+(SI)=20000H+1200H+0050H=21250H

指令执行后：(IP)=2450H

3．段间直接寻址

在指令中直接提供了转向段地址和偏移地址，所以只要用指令中指定的偏移地址取代 IP 寄存器的内容，用指令中指定的段地址取代 CS 寄存器的内容就可以从一个段转移到另一个段。这种寻址方式只能用于无条件转移指令。

例 11：JMP FAR PTR NEXT

4．段间间接寻址

用存储器中的两个相继字单元的内容来取代 IP 和 CS 寄存器中的原始内容，以达到段间转移的目的，其中第一个字单元的内容作为偏移地址送入 IP，第二个字单元的内容作为段地址送入 CS。这种寻址方式只能用于无条件转移指令。

例 12：JMP DWORD PTR［INTER+BX］

四、数据传送类指令

1.通用数据传送指令

①传送指令 MOV

格式：MOV　DST，SRC

功能：(DST)←(SRC)

使用 MOV 指令需要注意的是：

①DST 和 SRC 必须同时为 8 位或 16 位，否则会产生操作数类型不匹配的错误。

②SRC 可以是寄存器、存储单元、段寄存器和立即数，DST 只能是寄存器、存储单元、段寄存器(CS 除外)。

③DST 和 SRC 不能同为两个存储单元，也不能同为两个段寄存器。不能将立即数直接的送段寄存器。

④MOV 指令不影响标志位

⑵堆栈操作指令

格式：PUSH　SRC

　　POP　 DST

功能：8086 堆栈操作总是按字进行的。PUSH 每执行一次，堆栈栈顶指针 SP 减 2，推入堆栈的数据放在栈顶，低位字节放在较低地址单元，高位字节放在较高地址单元。POP 指令正好相反，每弹出一个字，堆栈栈顶指针 SP 加 2。

⑶交换指令 XCHG

格式：XCHG　OPR1，OPR2　 ；OPR 表示操作数

功能：操作数 OPR1 和操作数 OPR2 相交换

注意：XCHG 指令的两个操作数不能同为存储单元，两个操作数任意一个也不允许使用段寄存器，且不影响标志位。

2．累加器专用传送指令

⑴输入输出指令

格式：IN　AL，PORT 或 IN　AX，PORT

　　OUT　PORT，AL 或 OUT　PORT，AX　　；PORT 表示端口号

功能：执行输入指令 IN 时，CPU 可以一个 8 位端口读入一个字节到 AL 中，也可以从两个连续的 8 位端口读一个字到 AX 中；执行输出指令 OUT 时，CPU 可以将 AL 中的一个字节写到一个 8 位端口中，也可以将 AX 中的一个字写到两个连续的 8 位端口中。

输入输出指令可以分为两大类：一类是直接的输入输出指令，另一类是间接的输入输出指令。使用直接输入输出指令时，寻址范围为 0～255(十六进制 0～0FFH)。使用间接输入输出指令时，寻址范围为 0～65535(十六进制 0～0FFFFH)。

⑵换码指令 XLAT

格式：XLAT

功能：(AL)←((AL)+(BX))

使用 XLAT 之前，应先建立一个字节表格，表格的首地址提前存入 BX 寄存器，AL 中为字节表格中某一项与表格首单元之间的位移量，指令执行时，会将 BX 和 AL 中的值相加，把得到的值作为地址，然后将此地址所对应的存储单元的值取到 AL 中。

3．地址传送指令

⑴LEA 指令

格式：LEA　DST，SRC

功能：(DST)←SRC

LEA 装入偏移地址是根据 SRC 寻址方式计算偏移地址，而不是再用偏移地址来取操作数。DST 使用 16 位的寄存器，SRC 只能使用与存储器相关的寻址方式。

例 13：LEA　AX，[2400H]　；将内存单元的偏移地址 2400H 送 AX

　　LEA　BX，[BP+SI]　；指令执行后，BX 中的内容为 BP+SI 的值

4．类型转换指令

⑴CBW 字节转换为字指令

格式：CBW

功能：将 AL 的内容符号扩展到 AH，形成 AX 中的字。如果(AL)的最高位为 0，

则(AH)=0；如果(AL)的最高位为 1，则(AH)=0FFH。

⑵CWD 字转换为双字指令

格式：CWD

功能：将 AX 的内容符号扩展到 DX，形成 DX：AX 中的双字。如果(AX)的最高位为 0，则(DX)=0；如果(AX)的最高位为 1，则(DX=0FFFFH。

五、算术运算类指令

8086 的算术运算指令众多，可归结为双操作数指令和单操作数指令两类。双操作数指令的两个操作数中除源操作数为立即寻址以外，必须有一个操作数在寄存器中。单操作数指令不允许使用立即寻址。

1．加法指令

⑴加法指令 ADD

格式：ADD　DST，SRC

功能：(DST)←(DST)+(SRC)

⑵带进位加法指令 ADC

格式：ADC　DST，SRC

功能：(DST)←(DST)+(SRC)+CF

⑶加 1 指令 INC

格式：INC OPR

功能：(OPR)←(OPR)+1

2．减法指令

⑴减法指令 SUB

格式：SUB　DST，SRC

功能：(DST)←(DST)－(SRC)

⑵带借位减法指令 SBB

格式：SBB　DST，SRC

功能：(DST)←(DST)－(SRC)－CF

⑶减 1 指令 DEC

格式：DEC OPR

功能：(OPR)←(OPR)－1

和加 1 指令类似，一般用在循环程序中修改指针和循环次数。

⑷求补指令 NEG

格式：NEG OPR

功能：(OPR)←－(OPR)

⑸比较指令 CMP

格式：CMP OPR1，OPR2

功能：(OPR1)－(OPR2)

该指令和 SUB 指令一样执行减法运算，但并不保存结果，只是根据结果设置条件码标志。CMP 指令后往往跟着一条条件转移指令，根据比较结果产生不同的程序分支。

例 14：已知数组首地址为 X，段地址已放在 DS 中，末字为 0FFFFH，试统计其中 0 的个数并存放在末字单元，流程图如图 3.7 所示。

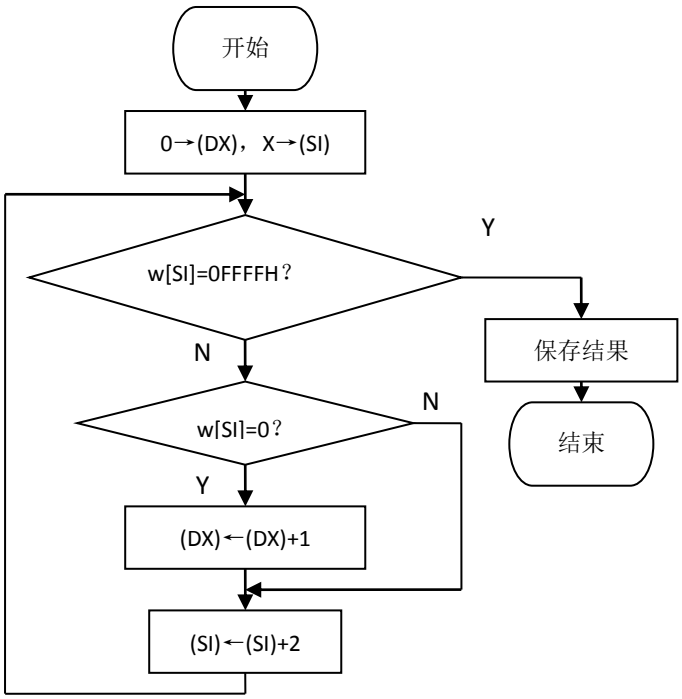


图 3.7 流程图

程序如下：

```
        MOV  DX，0
        LEA  SI，X
LOP：CMP  WORD  PTR [SI]，0FFFFH
```

```
        JZ ENDO
        CMP  WORD  PTR [SI], 0
        JNZ NEXT
        INC DX
NEXT: ADD  SI, 2
        JMP LOP
ENDO: MOV [SI], DX
```

3．乘法指令

⑴无符号数乘法指令 MUL

格式：MUL SRC

功能：(AX)←(AL)*(SRC)　　；字节操作数相乘

(DX，AX)←(AX)*(SRC)　；字操作数相乘

⑵带符号数乘法指令 IMUL

格式：MUL SRC

功能：(AX)←(AL)*(SRC)　　；字节操作数相乘

(DX，AX)←(AX)*(SRC)　；字操作数相乘

在乘法指令里，目的操作数必须是累加器，字运算为 AX，字节运算为 AL。两个 8 位数相乘得到的是 16 位的积存放在 AX 中，两个 16 位数相乘得到的是 32 位的积存放在 DX、AX 中，其中 DX 存放高位字，AX 存放低位字。

4．除法指令

⑴无符号数除法指令 DIV

格式：DIV SRC

功能：当 SRC 为字节操作数，16 位的被除数在 AX 中时，表示为：

(AL)←(AX)/(SRC)的商

(AH)←(AX)/(SRC)的余数

当 SRC 为字操作数，32 位的被除数在 DX、AX 中时，表示为：

(AX)←(DX，AX)/(SRC)的商

(DX)←(DX，AX)/(SRC)的余数

⑵带带符号数除法指令 IDIV

格式：IDIV SRC

功能：与 DIV 指令相同，但操作数必须是带符号数，商和余数也是带符号数，且余数的符号和被除数的符号相同。

六、逻辑运算及移位指令

1．逻辑运算指令

8086 的逻辑运算指令包括 AND(与)、OR(或)、NOT(非)、XOR(异或)和 TEST(测试)指令。

以上五条指令中，NOT 属于单操作数指令，不允许使用立即寻址，其余 4 条指令都属于双操作数指令，除源操作数是立即数外，至少有一个操作数必须放在寄存器中。NOT 指令不影响条件码标志，其余 4 条指令是 CF 和 OF 为 0，AF 位无定义，而 SF、ZF 和 PF 位则根据运算结果设置。

在程序设计中，AND 一般用于对一个数据的指定位清 0；OR 指令常常用来对一些指定位置 1；NOT 指令常常用来将某个数据取反码；XOR 指令指令常用在程序的开头使某个寄存器清 0，以配合初始化工作的完成；TEST 指令执行与运算，但不保存结果，只根据结果设置相应的条件码标志，一般用于检测指定位是 1 还是 0。

2．移位指令

⑴非循环移位指令

格式：SHL/SHR/SAL/SAR OPR，CNT

CNT 表示移位的次数，只能是 1 或 CL，当移位次数大于 1 时，必须在移位指令前将移位次数置于 CL 寄存器中。各指令功能如图 3.8 所示。

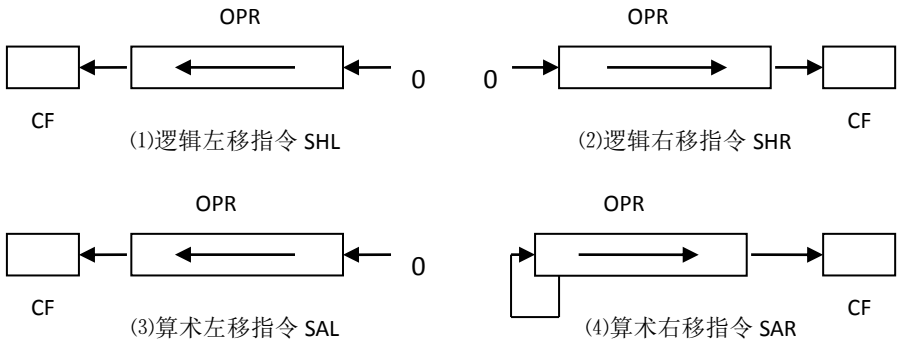


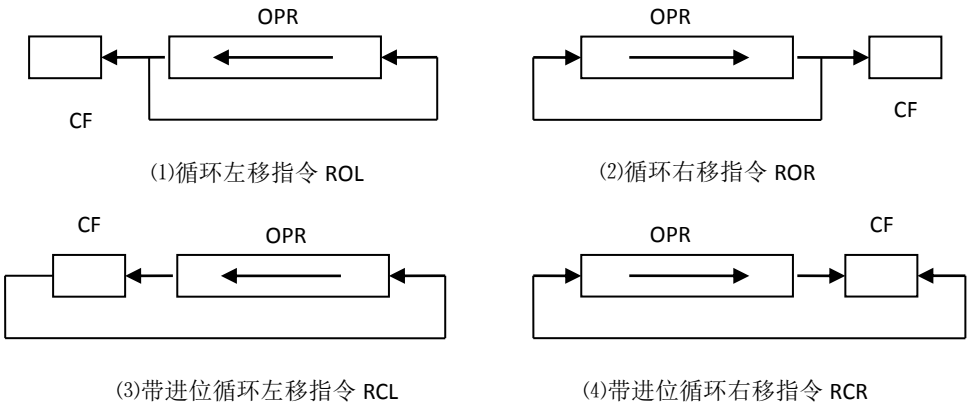
图 3.8 移位指令的功能

用移位指令时，逻辑移位指令适用于无符号数运算，算术移位指令则适用于带符号数运算，左移 1 位相对于将操作数乘 2，右移 1 位相对于将操作数除 2。由于移位指令执行速度快，可以编制一些常用的乘除法程序，比直接使用乘除法指令要快的多。

⑵循环移位指令

格式：ROL/ROR/RCL/RCR　OPR，CNT

CNT 表示移位的次数，和非循环移位指令一样，如果循环移位指令只移动一次，则在指令中直接指出，如果要移动若干位，则必须在 CL 寄存器中指定移位次数。各指令功能如图 3.9 所示。



⑴循环左移指令 ROL　⑵循环右移指令 ROR

⑶带进位循环左移指令 RCL　⑷带进位循环右移指令 RCR

图 3.9 循环移位指令的功能

七、控制转移指令

1．无条件转移指令 JMP

格式：JMP　OPR(目标地址)

功能：无条件地转移到指令指定的地址去执行从该地址开始的指令。

总的来说，转移可以分为段内转移和段间转移。段内转移是指在同一个代码段内进行转移，此时只需要改变 IP 寄存器中的值。段间转移是指程序从一个代码段转移到另一个代码段取执行程序，此时不仅改变 IP 的值，同时还要改变 CS 的值才能达到转移的目的。

2．条件转移指令

格式：JZ　OPR(目标地址)；条件转移指令比较多，以 JZ 为例说明格式

功能：条件转移指令以某一标志位的值或者某个比较结果作为判断是否进行转移的依据，如果满足指令中所要求的条件，则转移到目标地址 OPR 处执行程序，

否则按顺序执行排在条件转移指令后面的一条指令。所有的条件转移指令都是相对形式的，目标地址 OPR 只能在本条转移指令下一条指令地址的－128～＋127 个字节范围之内。

条件转移指令中，有一部分指令是根据对某一条件码标志的判断决定是否转移的，还有相当一部分指令在比较完两个数的大小以后，根据比较的结果决定是否转移的。对于具体的二进制数，将它们看成带符号数或无符号数，比较后得到的结果是不同的。

⑴条件转移指令根据某一条件码标志的值决定是否转移，这类指令如表 3.1 所示。

表 3.1 由条件码标志的值决定是否转移

操作码	含义
JZ/JE	ZF=1 条件成立，则转移
JNZ/JNE	ZF=0 条件成立，则转移
JS	SF=1 条件成立，则转移
JNS	SF=0 条件成立，则转移
JC	CF=1 条件成立，则转移
JNC	CF=0 条件成立，则转移
JO	OF=1 条件成立，则转移
JNO	OF=0 条件成立，则转移
JP	PF=1 条件成立，则转移
JNP	PF=0 条件成立，则转移

⑵条件转移指令根据两个无符号数的比较决定是否转移，这类指令如表 3.2 所示。

表 3.2 由两个无符号数的比较决定是否转移

操作码	含义
JB/JNAE	低于，即不高于且不等于，则转移
JBE/JNA	低于或者等于，即不高于，则转移
JA/JNBE	高于，即不低于且不等于，则转移
JAE/JNB	高于或者等于，即不低于，则转移

⑶条件转移指令根据两个带符号数的比较决定是否转移，这类指令如表 3.3

所示。

表 3.3 由两个带符号数的比较决定是否转移

操作码	含义
JG/JNLE	大于，即不小于且不等于，则转移
JGE/JNL	大于或者等于，即不小于，则转移
JL/JNGE	小于，即不大于且不等于，则转移
JLE/JNG	小于或者等于，即不大于，则转移

例 15：设 2000H 开始的区域中，存放着 100 个字节的无符号数，要求找出其中最大的一个数，并存到 2000H 单元。程序如下：

```
        MOV   BX，2000H
        MOV   AL，[BX]
        MOV   CX，99
LOP：   INC    BX
        CMP   AL，[BX]
        JAE    NEXT
        MOV   AL，[BX]
NEXT：  DEC   CX
        JNZ    LOP
        MOV   [2000H]，AL
```

3．循环控制指令

格式：LOOP OPR(目标地址)

功能：执行 LOOP 指令时，先将 CX 的内容减 1，然后判断 CX 是否为 0。如不为 0，则继续循环；如为 0，则退出循环，执行下一条指令。

因此，在使用 LOOP 指令前，必须用 MOV 指令对 CX 设置初值。

例 16：有一个首地址为 1000H 包含 20 个字的数组，要求求出该数组的内容之和(不考虑溢出)，并将结果存入 2000H 单元中。程序如下：

```
        MOV    SI，1000H
        MOV    AX，0
        MOV    CX，20
LOP：   ADD   AX，[SI]
```

```
ADD  SI，2
LOOP LOP
MOV  [2000H]，AX
```

4．子程序调用和返回指令

为便于模块化程序设计，往往把程序中某些具有独立功能的部分编写成独立的程序模块，称为子程序。程序可由调用程序(或称主程序)调用这些子程序，而在子程序执行完后又返回调用程序继续执行。8086 提供了子程序调用指令 CALL 和返回指令 RET。

⑴子程序调用指令 CALL

格式：CALL DST(子程序的入口地址)

功能：首先把子程序的返回地址存入堆栈，以便子程序返回调用程序时使用，然后转移到子程序的入口地址去继续执行。指令中的 DST 即为子程序的入口地址，也就是子程序的第一条指令的地址。

⑵返回指令 RET

格式：RET

功能：子程序执行完后返回调用程序继续执行。

和调用指令相对应的是返回指令。返回指令总是作为一个子程序的最后一条指令用来返回高一层的程序。返回指令在执行时，会从堆栈顶部弹出返回地址。为了正确的返回，返回指令的类型要和调用指令的类型相对应。

8086 的返回指令还有另外一种形式，叫带参数的返回指令，形式如下：

RET n

其中，n 可以为 0～FFFFH 范围内的任何一个偶数，表示这条指令从堆栈顶部弹出返回地址后，再使 SP 的值加上 n。

5．中断指令

⑴中断指令 INT

格式：INT n

功能：转到中断类型号为 n 的中断服务程序去执行。其中 n 为中断类型号，其值必须是在 0～255 范围内。

执行 INT n 指令时，将使 CPU 转到一个中断处理程序。此时，将标志寄存器

的值推入堆栈，堆栈指针 SP 减 2，然后清除中断允许标志 IF 和单步标志 TF，接着 CPU 将主程序的下一条指令地址即断点地址的段地址和偏移地址推入堆栈，同时堆栈指针 SP 减 4。

要进入中断处理程序，必须找到中断处理程序的入口地址。中断指令中直接给出了中断类型号，按照 8086 中断机制的原理，中断类型号的 4 倍就是中断向量的存放位置的偏移地址，而中断向量已有规则的排列在内存 0 段当中，中断向量的存放位置的段地址默认就是 0000H，中断向量就是中断处理程序的入口地址。因此，由中断类型号乘以 4 得到一个单元地址，由此地址开始的前 2 个单元存放的就是中断处理程序入口地址的偏移地址，后 2 个单元存放的就是中断处理程序入口地址的段地址。随后将中断处理程序入口地址的偏移地址和段地址分别送入 IP 和 CS 寄存器，这样 CPU 就会转入中断处理程序去执行。

⑵中断返回指令 IRET

格式：IRET

功能：返回中断处，执行后续指令。

各个中断处理程序的功能不同，但是，中断处理程序的最后一条指令总是 IRET。执行 IRET 指令时，先从堆栈中弹出 4 个单元的内容送入 IP 和 CS 寄存器，从而恢复断点地址，然后弹出标志寄存器的值。

第四章 存储系统

存储器是计算机的记忆部件，用来存放程序和数据。从用途和特点上可以分为两大类：

内存：计算机主机的重要组成部分，用来存放当前正在使用的或者经常使用的程序和数据，CPU 可以直接访问，但是容量受地址总线位数的限制。

外存：容量大，速度慢。用来存放暂时不用的程序和数据，使用时必须先调入内存才能使用。

一、存储器的分类

从功能上可以分为：RAM 和 ROM

1．RAM

静态 RAM（SRAM）：速度快，容量低，功耗大，不需要刷新。

动态 RAM（DRAM）：容量大，功耗低，需要刷新。

2．ROM

掩膜型 ROM：程序和数据由厂家写入，不能再修改。

PROM：用户可以根据自己的需要写入信息，以后不能再修改。

EPROM：可以擦除和重写，但是需要一种产生紫外光的专门设备，专业人士完成。

EEPROM：改变数据是在软件的控制下完成。

二、存储系统的层次结构

人们对存储器的要求是容量大、速度快、成本低，但这三个特性无法在同一存储器中兼得。为了解决这一矛盾，存储体系采用分级存储器结构，通常是将存储器分为高速缓冲存储器、主存储器和外存储器三级，如图 4.1 所示。通过硬件和管理软件组成一个既有足够大的存储空间又能保证满足 CPU 存取速度要求且价格适中的整体。

综上所述，一个较大的存储系统是由各种不同类型的存储设备构成，是一个具有多级层次结构的存储系统，各级存储器承担的职能各不相同。该系统中具有适当的存储容量和存取周期，使它能容纳系统的核心软件和较多的用户程序；高速缓冲存储器解决了存储系统与 CPU 运算速度相匹配的速度问题；辅助存储器则

解决了存储系统极大存储容量的容量问题。采用多级层次结构的存储器系统可以有效的解决存储器的速度、容量和价格之间的矛盾

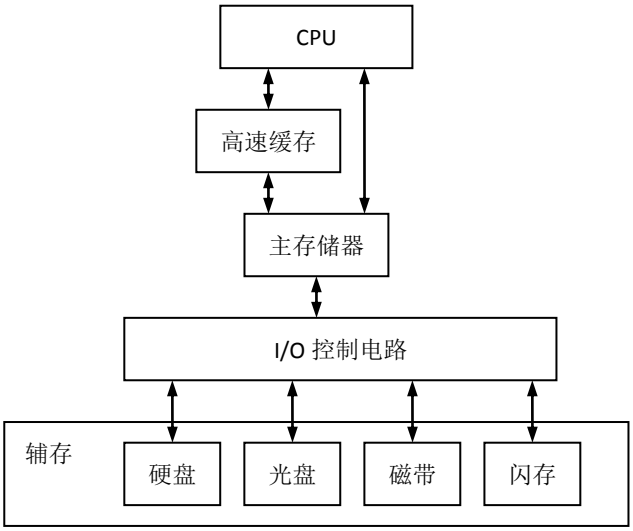


图 4.1 存储系统的多级存储结构

三、存储器扩展方法

由于生产的任何存储器芯片的容量是有限的，其在字数或字长方面和实际应用中对存储器的要求都有差距，所以为了满足实际存储器的容量要求，往往单个芯片不能满足字长或存储单元个数的要求，甚至字长、存储单元数两者都不满足要求。这时，就需要用多个存储芯片进行组合，对存储器进行字向和位向两方面的扩展。这种组合称为存储器的扩展，主要采用的方法有：位扩展法、字扩展法、字位同时扩展法。

1．位扩展

如果单个存储芯片字长（位数）不满足要求，这时就需要进行位扩展，以满足字长的要求。

例：使用 8K×1 的 RAM 存储器芯片，要求组成一个 8K×8 位的存储器。根据要求要用 8 片 8K×1 的存储芯片经位扩展构成 8K×8 位（即 8KB）的存储器，每个单元的 8 位二进制数被分别存在 8 个芯片上，可采用图 4.2 所示的位扩展法。

位扩展的特点：将每个存储芯片的地址线和控制线（包括片选信号线、读/写信号线等）全部并联在一起，而将各个芯片的数据线分别连接至数据总线的不同位上。使用位扩展法构成的存储器的每个单元中的内容被存储在不同的存储芯片上。

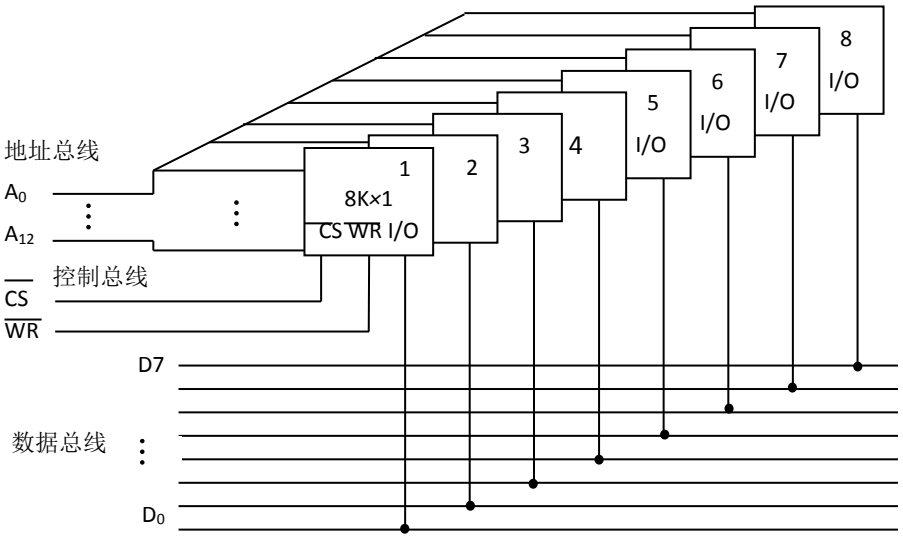


图 4.2 用 8K×1 位芯片组成 8K×8 位的存储器

2．字扩展

存储芯片上每个存储单元的字长已满足要求，只是存储单元的个数不够，需要增加的是存储单元的数量。

例：用 16K×8 位的芯片组成 64K×8 位的存储器，其字长已满足要求，只是容量不够，所以需要进行的是字扩展。显然，需要 4 片来实现，连接图如图 4.3 所示。

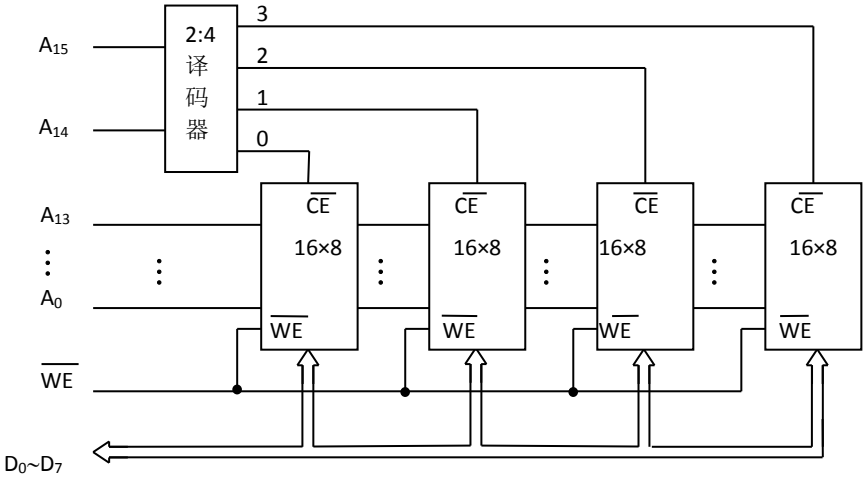


图 4.3　用 16K×8 位芯片组成 64K×8 位的存储器

字扩展的特点：将每个芯片的地址信号、数据信号和读/写信号等控制信号线按名称全部并联在一起，只将片选端分别和地址译码器的不同输出端相连。

第五章 输入输出系统

一、接口概述

接口技术：研究 CPU 和外设之间的数据传送方式、接口电路的工作原理和使用方法。

1．为什么使用接口电路？

①品种繁多； ②工作速度慢； ③信号类型与电平种类不同；④信息结构格式复杂。

存储器：功能单一，品种有限，速度较快，故可以直接连在总线上。

2．CPU 和输入输出设备之间的信号

⑴数据信息

这类信息可以是通过输入设备送到计算机的输入数据，也可以是经过计算机运算处理和加工后，送到输出设备的结果数据。在输入过程中，数据信息由外设经过外设接口之间的数据线进入接口，再到达系统的数据总线，从而送给 CPU。在输出过程中，数据信息从 CPU 经过数据总线进入接口，再通过接口和外设之间的数据线送到外设。

⑵状态信息

状态信息反映了当前外设的工作状态，是外设通过接口往 CPU 传送的。对于输入设备来说，通常用准备好信号(READY)信号来表明输入的数据是否准备就绪；对于输出设备来说，通常用忙(BUSY)信号表示输出设备是否处于空闲状态，如为空闲状态，则可接收 CPU 送来的信息，否则 CPU 要等待。

⑶控制信息

控制信息是 CPU 通过接口传送给外设的，CPU 通过发送控制信息控制外设的工作，外设的启动信号和停止信号就是常见的控制信息。

从含义上说，数据信息、状态信息和控制信息各不相同，应该分别传送。但在微型计算机系统中，CPU 通过接口和外设交换信息时，只有输入(IN)和输出指令(OUT)，所以，状态信息、控制信息也被广义地看成是一种数据信息。即状态信息作为一种输入数据，而控制信息作为一种输出数据。这样，状态信息和控制信息也通过数据总线来传送。