第 1 章　微型计算机概述

本章首先介绍微型计算机的概念框图，说明微型计算机的主要组成部分及各部分的功能与作用，并说明微型机的简单工作过程，以期从概念上对微型机建立初步的认识。然后，再从 CPU 的外部引线及内部寄存器开始，说明 CPU 的外特性及其工作时序。最后，叙述在此 CPU 的基础上，如何构成微型机的系统总线。

在本章及后面的章节中，要用到一些预备知识，例如数制（二进制、十进制、十六进制等）及其运算、符号数的表示方法、常见的编码（例如 BCD 码、ASCII 码等）以及数字电路和模拟电路的基本知识等。我们认为读者均已学习并掌握了这些知识，在本书中只是应用，不再做任何解释。

1.1　主要内容

1.1.1　微型计算机的组成

微型计算机是由硬件和软件两大部分构成的，下面分别加以说明。

1. 微型计算机的硬件概念框图

从概念上讲，微型计算机都可以认为是由如图 1.1 所示的几部分组成。

从图 1.1 中可以看出，微型计算机硬件以 CPU 为核心，利用 CPU 的信号形成系统总线，并在系统总线上连接内存和接口。其中，内存用来存放程序和数据；接口用于将外设和计算机连接在一起，可利用接口将数据传送到外设，也可将外设的数据通过接口传送到微型机中。CPU 发出的地址信号、控制信号以及数据信号都是通过系统总线传送的。

由此可见，微型计算机硬件主要由 CPU、系统总线、内存和接口这样几个重要部分组成。如果将这些部分集成在一块集成电路芯片中，就构成了人们常说的单片微型计算机。

2. 软件

图 1.1 所示的仅是微型机的硬件，只有硬件的计算机称为“裸机”，它是不能工作的。

为了使微型机完成某种功能，就需要为它配备软件。在微型机中通常配有操作系统、应用程序、用户程序等软件。软件是微型计算机系统的重要组成部分。

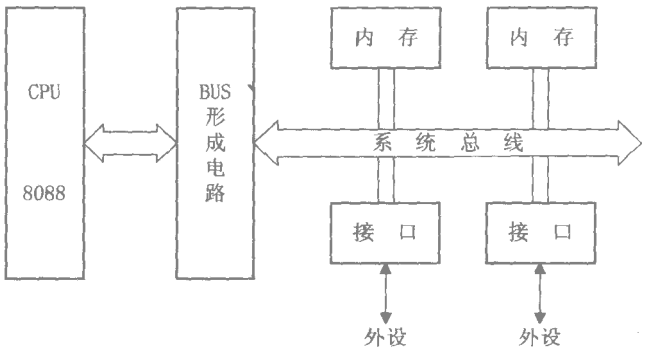

图 1.1 微型计算机概念组成框图

1.1.2 微型计算机的工作过程

微型计算机通过执行程序来实现各种功能。程序是完成某种功能所需指令的集合，而指令则是命令 CPU 完成某种最简单的操作的命令。

通常，我们总是将一个复杂的任务分解成很多最简单的操作，这些最简单的操作可由一条条指令来实现，而这些指令的集合就构成了完成该任务的软件——程序。

由于 CPU 只能识别二进制代码，因此程序最终也必须用二进制代码来表示。将程序代码放到内存中，而后让 CPU 执行这段程序。CPU 从内存中逐条地将指令读到 CPU 内部并识别和执行它。CPU 逐条顺序地执行这些指令，当指令执行完毕，程序的功能也就实现了。

下面我们用一个最简单的例子来说明程序的执行过程。若要计算

5＋7＝？

则可将这种计算写成如下的程序（第一列为各指令的二进制代码）：

```
B005        MOV       AL , 05H
B307        MOV       BL, 07H
02C3        ADD       AL , BL
E63E        OUT       3EH, AL
F4          HLT
```

在上面的程序中，前两条指令分别将 5 和 7 传送到寄存器 AL 和 BL 中，第三条指令将 AL 中的 5 与 BL 中的 7 相加，并将相加的和放到 AL 中，至此就求得了两数之和。最后是一条停机指令，命令 CPU 停止执行指令，原地待命。

如将上面程序左侧的指令代码放入内存中，然后命令 CPU 开始执行这个程序。CPU 首先从内存中读出 B0H，然后再从内存的下一个地址读出 05H，并把它放到 AL 中。这样第一条指令就执行完了。接着 CPU 以同样的方式依次执行下面的指令，第三条指令实现了两数相加并将结果放在 AL 中。执行第四条指令时，CPU 首先从内存中读出该指令的两个操作码 E6H 和 3EH，然后执行该指令，将 AL 中的内容写到地址为 3EH 的接口上。直到 CPU 从内存中取出停机指令 HLT 的指令操作码 F4H，执行该指令 CPU 就处于停机状态。

1.1.3 8088 CPU 的引线及内部寄存器

本书以 8088 CPU 为例进行说明，因为 8088 CPU 在过去的 PC 机中曾得到广泛地应用且具备一般 CPU 的共性。同时，在过去的教材中，都是在讲述了 8088 CPU 之后再简单介绍后来出现的 80386、80486 和奔腾。由于受教学时间的限制，有关保护模式方面的内容通常只作简单介绍，因此，在这里我们主要对 8088 CPU 的外部引线及内部寄存器进行说明。

1. 8088 CPU 的外部引线

8088 CPU 是一块具有 40 个引脚的集成电路芯片，其各引线定义如图 1.2 所示。

（1）最小模式下的引线信号

在 8088 CPU 的引线信号中有一条 MN/MX 信号线，这是一条输入信号线。当该信号为高电平时，规定 8088 CPU 工作在最小模式之下。所谓最小模式，就是构成的微型机中只有一个 8088 CPU。而当该信号线为低电平时，规定 8088 CPU 工作在最大模式之下，即在构成的微型计算机中，存在着多个 CPU，除了 8088 CPU 之外还有其他 CPU（例如 8087、8089 等）。

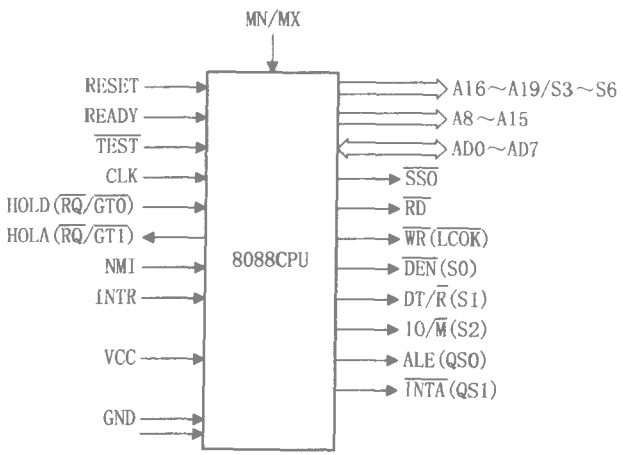

图 1.2 8088 CPU 引脚图

在最小模式下，A16～A19/S3～S6 这 4 条信号线以及 AD0～AD7 这 8 条信号线采用分时复用方式，即某一时刻这些线上送出地址，而另一时刻则送出状态或用于传送数据。信号线 A8～A15 这 8 条线则只用于送出地址信号。由于以上这 20 条信号线从 CPU 输出，最后都经过三态门，因此，它们在特殊情况下会呈现第三态（即高阻状态）。

在此模式下，CPU送出用于内存或接口的控制信号：$\overline{RD}$、$\overline{WR}$、IO/$\overline{M}$信号。当 IO/M 为低电平时，用于读写内存，即对内存寻址；而当它为高电平时用于对接口的寻址。当将数据写入内存或接口时$\overline{WR}$为低电平（有效），而当从内存或接口读出数据时$\overline{RD}$为低电平(有效)。这三个信号为三态输出。

信号线 ALE、$\overline{DEN}$ 和 DT/$\overline{R}$这三个信号是用于形成系统总线的。其中，当地址线上出现有效的地址时需锁存这些地址，ALE 就用作锁存脉冲。$\overline{DEN}$为数据允许信号，用它来控制数据总线驱动器，当DEN有效时使数据总线驱动器有效工作。信号 DT/$\overline{R}$ 用于控制数据总线驱动器的信号传送方向，当 CPU读数据时，信号流向 CPU;而当 CPU 写数据时，方向指向系统总线。这三个信号为三态输出信号。

信号 INTR 和$\overline{INTA}$分别为（可屏蔽）中断请求和中断响应信号。有关中断的概念以及在微机中的具体应用是十分重要的。所谓中断是指当由于微机内部或外部某个事件发生，迫使 CPU 暂时中断正在执行的程序，转向对所发生事件的处理，处理结束后又回到被中断的程序接着中断前的状态继续向下执行的过程。 INTR 是一个外部中断请求的输入端，INTR 的请求是可以屏蔽的，即当标志寄存器中 IF 的状态为 0 时，CPU 不会响应该请求，也就是说它被屏蔽了。而当 IF＝1 时，CPU 才有可能（还有其他条件）对它作出响应。CPU 对 INTR 的响应包括许多内容，其中就有从$\overline{INTA}$引线信号端输出两个负脉冲。该信号为三态输出信号。

引线信号 NMI 为非屏蔽中断请求，顾名思义该中断不可屏蔽，即不受中断屏蔽位 IF 的影响。不管 IF 是 0 还是 1，只要满足其他条件 CPU 就会对它作出响应。当 NMI 为上升沿时，向 CPU 提出请求。

保持请求信号 HOLD 和保持响应信号 HLDA 是一对为完成 DMA 传送而设置的信号，保持请求 HOLD为高电平有效的输入请求信号。当外设与内存需要快速交换数据时，可以通过 DMAC 在 CPU 的 HOLD 输入端加上一个高电平的请求信号， CPU 会在一个总线周期结束时作出响应。CPU 的响应包括两个方面：一方面是从引线 HLDA 上送出高电平加到 DMAC 上，告诉 DMAC 其 DMA 请求 CPU 已经响应。另一方面 CPU 放弃了对系统总线的控制权，将系统总线的控制权交给 DMAC。DMAC 就可以利用系统总线实现 DMA 传送。有关 DMA 传送的详细内容将在后面的章节中加以说明。

在 CPU 的引线中 READY 信号是一个高电平有效的输入信号。当该信号为高电平时，CPU 正常地执行指令，而当其输入为低电平时 CPU 原地踏步，不再向下执行程序。直到 READY 输入为高电平时 CPU才继续向下执行。

输入信号 TEST称为测试信号，当 CPU 执行 WAIT 指令时对$\overline{TEST}$输入电平进行测试。若该输入为高电平，则 CPU 在此指令上原地踏步，直到此输入变为低电平， CPU 才继续向下执行。

复位信号 RESET 是一个重要的输入信号。当它输入为高电平时， CPU 将被复位。复位后 DS、SS、ES、IP、F 等内部寄存器均清零， CS 寄存器为 FFFFH。因此，当 RESET 变低启动时，CPU 的启动入口地址为 FFFF0H。在复位时 CPU 的三态输出信号 AD0～AD7、A8～A15、A16～A19/S3～S6、$\overline{DEN}$ 、$\overline{RD}$、$\overline{WR}$、DT/$\overline{R}$、IO/$\overline{M}$、INTA均处于高阻状态(浮空）；而其他输出信号均为无效状态。

激励 CPU 工作的是 CPU 的时钟，时钟的每一个周期激发 CPU 内部硬件完成某些细微工作。这些细微工作积累起来就完成了一个总线周期的工作。一个或多个总线周期工作的积累就完成了一条指令的执行。

在最小模式下尚有一个状态输出信号 SSO。它和前面提到的输出信号 DT/R和 IO/M 结合在一起，三个信号的不同状态的编码表示 CPU 的不同的工作状态。

(2) 最大模式下的引线

当 MN/$\overline{MX}$输入端接上低电平时，CPU 将工作在最大模式之下。

最大模式下 8088 CPU 的引线信号大部分与最小模式下的信号相同，只有下面的 10 条引线信号的名称及功能发生了改变，它们是：

SSO 在最大模式下总为高电平。

RD信号不再使用。

$\overline{DEN}$、DT/$\overline{R}$、IO/$\overline{M}$ 这三个信号重新定义为 S0、S1、S2 。这三个信号的不同状态的编码可以表示 CPU 的不同的工作状态。为了在最大模式下利用这些状态， Intel 公司专门研制了总线控制器 8288,利用该芯片对 S0、S1 和 S2 三个状态信号进行译码，产生一系列的总线控制信号，例如$\overline{INTA}$、$\overline{IOR}$、$\overline{IOW}$、MEMR、MEMW等等。

信号 ALE 和$\overline{INTA}$被重新定义为队列状态信号 QS0 和 QS1。这两个信号状态的不同编码用来表示 CPU 内部四个字节的指令预取队列的状态。

保持请求和保持响应信号 HOLD 和 HLDA 重新定义为 $\overline{RQ}$/$\overline{GT0}$和 $\overline{RQ}$/$\overline{GT1}$。功能也由最小模式时的一个请求和一个响应变为二个请求和二个响应，即 $\overline{RQ}$/$\overline{GT0}$和$\overline{RQ}$/$\overline{GT1}$ 每一条引线即可完成请求，在其上加一个宽度为一个时钟周期宽度的负脉冲，而 CPU 响应请求会在同一引线上送出一个时钟周期宽度的负脉冲，而当 DMA 传送结束时，DMAC 在同一引线上输入一个时钟周期宽度的负脉冲表示传送结束。可见，这种功能RQ/GT0 可以实现，RQ/GT1 也同样可以实现。

最大模式下，将$\overline{WR}$信号重新命名为$\overline{LOCK}$。它在 CPU 执行带有‘ LOCK ”前缀指令时输出为有效的低电平，用以封锁总线。

2. 8088 CPU 的内部寄存器

8088 CPU 是一片大规模集成电路芯片，其内部结构十分复杂。我们不可能也没有必要弄清楚其内部的细节，只要知道与我们应用有关的内容。这种思路将适用于本书后面用

到的各种内存和接口芯片，以至于将来的工作中。

(1) 编程中所用到的内部寄存器

在后面学习的过程中，以及指令系统和汇编语言中都会频繁地用到这些寄存器。其中通用寄存器为：16 位的 AX、BX、CX、DX、BP、SP、SI、DI。这 8 个寄存器中前 4 个为数据寄存器，主要用于传送数据和进行算术及逻辑运算。同时，它们还具备其他功能，例如 BX 可存放偏移地址，作为地址指针使用；CX 在循环指令和串操作指令中用作计数单元；DX 在 IO 指令中可存放接口地址，实现接口地址的寄存器间接寻址。AX、BX、CX 和 DX 这 4 个 16 位的寄存器还可以拆成两个独立的 8 位寄存器单独使用，用 AH，AL；BH，BL；CH，CL 和 DH，DL，分别表示一个字的高 8 位和低 8 位。这 8 个独立的 8 位寄存器也是通用的，均可以实现传送、算术逻辑运算及位移、循环等操作。在执行多次循环或位移指令时，必须用 CL 作为循环或位移的次数计数器。

另外 4 个通用寄存器 BP、SP、SI、DI 主要用作地址指针，SI 和 DI 还可以作为通用寄存器使用，用于传送、算术、逻辑、循环、位移等各种指令作为源或目的寄存器之用。

(2) 段寄存器

在 8088 CPU 中有 4 个特殊的 16 位寄存器——段寄存器 CS、DS、SS 和 ES。它们的作用只是为了在只有 16 位内部寄存器的 8088 CPU 中能够形成 20 位地址，以寻址 1MB 的内存空间。在 8088 CPU 内部形成 20 位的物理地址是这样进行的：

$$20\text{位的物理地址} = \text{段寄存器的内容} \times 16 + 16\text{位的偏移地址} \tag{1-1}$$

(3) 专用寄存器

在 8088 CPU 中有两个专用的 16 位寄存器：指令指针 IP 和标志寄存器 F。

在 CPU 执行指令的过程中每从内存中取出一个指令字节，指令指针 IP 的内容自动加 1 并指向下一个要读出的指令字节。这是因为 CPU 读指令字节的内存物理地址为：

$$20\text{位读指令字节的物理地址} = \text{CS} \times 16 + \text{IP} \tag{1-2}$$

标志寄存器 F 也是一个 16 位的寄存器，其各位标志如图 1.3 所示。

×	×	×	×	OF	DF	IF	TF	SF	ZF	×	AF	×	PF	×	CF

图 1.3 标志寄存器 F 的各位标志

标志寄存器中的每一位标志均表示 CPU 执行指令的状态或表示某种功能。读者必须记住标志寄存器各位标志的功能。由于在每一种 CPU 中均有类似的寄存器，因此，学好 8088 CPU 的标志寄存器对今后理解其他 CPU 也是有用的。

1.1.4 8088 CPU 的工作时序

1. 指令周期与周期

前面已经提到，由指令集合而成的程序放在内存中，CPU 从内存中将指令逐条读出并执行。我们将 CPU 完整地执行一条指令所花的时间叫做一个指令周期。在后面的章节中我们可以看到，有的指令很简单，执行时间就比较短，而有的指令很复杂，其执行时间比较长，但我们都称为一个指令周期，只不过时间长短不同而已。

如果再细分，一个指令周期还可以分成若干个总线周期，即一条指令是由若干个总线周期来完成的。那么什么是总线周期呢？8088 CPU 通过总线对外部（存贮器或接口）进行一次访问所需的时间称为一个总线周期。这里主要是指 8088 CPU 将一个字节写入一个内存单元或写入一个接口地址或者 8088 CPU 由内存或接口读出一个字节到 CPU。尽管中断响应及 DMA 传送也有它们的总线周期，但在本章里我们主要说明 CPU 读内存中的指令字节、读内存数据字节、读接口字节、将字节写入内存和将字节写入接口这 5 种总线周期。

现在我们已经明确，一条指令可以由若干个总线周期来完成。同时，要进一步明确，在正常情况下，上述 5 个总线周期均由 4 个时钟周期来完成。时钟周期就是加在 CPU 芯片上 CLK 时钟信号的周期。

至此，可以看到一条指令的执行时间可以用总线周期来说明，还可以用时钟周期来度量。这就是指令周期、总线周期和时钟周期间的关系。

2. 总线周期的工作时序

在正常情况下，前面提到的 5 个总线周期均由 4 个时钟周期来完成。为了便于说明问题，我们首先以 CPU 向内存写入一个字节的总线周期来简要说明。该总线周期从第一个时钟周期 T1 开始，在 T1 时刻 CPU 从 A16～A19/S3～S6这 4 条引线上送出 A16～A19、从 A8～A15 这 8 条线上送出 A8～A15、从 AD0～AD7 这 8条线上送出 A0～A7 可见，在这个时钟周期里，CPU 从它的 20条引线上送出了 20 位地址信号 A0～A19，而且时钟 T1 之后，A16～A19 及 AD0～AD7 上的信号将变为其他信号。因此，CPU 在 T1 周期里送出 ALE 地址锁存信号，我们可以用这个信号将 A0～A19 锁存在锁存器中，使地址信号在整个总线周期里保持不变。当然，地址信号 A8～A15 可以不锁存，因为这些信号在整个总线周期里保持不变。在此 T1 期间 CPU 由 IO/$\overline{M}$送出低电平并在整个总线周期中一直维持低电平不变，表示该总线周期是一个寻址内存的总线周期。

在时钟周期 T2 里，CPU 将写入内存的数据从 AD0～AD7 上送出来，加到数据总线 D0～D7 上。同时 CPU 还会送出 $\overline{WR}$控制信号，在地址信号 A0～A19、IO/M 及 $\overline{WR}$ 的共同作用下，将 D0～D7 上的数据写入相应的内存单元中。写入内存的操作通常是在 $\overline{WR}$的后沿（其上升沿）来实现的。这时的地址、数据信号均已稳定，写操作的工作就很可靠。

以上就是在最小模式下正常的内存写入过程。若在实际应用中内存的写入时间要求较长而 CPU 提供的写入时间却较短（最长也只有 4 个时钟周期），则在这样短的时间里数据无法可靠地写入。为了能解决这个问题，可以利用 CPU 的 READY 信号。当 CPU 的总线周期里的时钟周期 T3 开始时(下降沿)，CPU 内部硬件测试 READY 信号的输入电平。若此时 READY 为低电平，则 CPU 在 T3 之后不执行 T4，而是插入一个等待时钟周期 TW。在 TW 的下降沿 CPU 继续检测 READY 输入电平，若其为低电平，则继续插入等待的时钟周期 TW。就这样一直插到 READY 为高电平时，则插入停止并执行总线周期的 T4。这样一来，一个写入内存的总线周期就可以由 4 个时钟周期延长为更多个时钟周期，以满足低速内存的要求。

以上我们仅以最小模式下将数据写入内存的总线周期为例说明 8088 CPU 的时序。对于其他读内存或读、写接口其思路是类似的，在此不再说明。

我们强调时序并不是时序图本身怎么重要，而是其思路及概念对理解本书后面的许多内容是很有用的。这是因为：一方面时序对理解指令的执行很有意义；另一方面时序对于内存和接口的连接及调试是极其有用的。这些问题在本书的后面的综合应用中将详细加以讨论。

1.1.5 系统总线的形成

从前面的图 1.1 中可以看到，微型计算机是在 CPU 形成的系统总线上接上内存和接口来构成的。在后面的章节中，将逐一讨论内存和接口是如何接到系统总线上的。因此，弄懂系统总线的信号怎么产生将是很重要的。在形成系统总线时，我们认为几种常用的数字集成电路(如 74LS244、245、373 等)是读者早已熟悉的。

1. 最小模式下系统总线的形成

在最小模式下，为了形成系统总线上的地址总线 A0～A19，采用 3 片具有三态输出的锁存器 74LS373，利用 CPU 送出的 ALE 信号将出现在时钟 T1 时刻的地址信号 A0～A19 锁存在 74LS373 的输出端，并一直保持到下一个总线周期锁存新的地址信号为止。显然，由于地址信号 A8～A15 在整个总线周期内保持其状态不变，因此，这 8 个信号可以不用锁存器，而用三态门(74LS244 或其他三态门)。

在形成地址总线信号时，应注意，若将来在系统总线上不进行 DMA 传送，则三态门输出控制端（74LS373 的 $\overline{OE}$ 及 74LS244 的 $\overline{E1}$、$\overline{E2}$）均可接成永远有效，即可接地。若要在系统总线上进行 DMA 传送，则那些三态门的控制端就应当由 CPU 的 HLDA 输出信号或其他译码输出控制信号来控制。

数据总线形成电路可由双向三态门 74LS245 或其他类似的器件来实现。74LS245 的 A 边接 CPU 的 AD0～AD7，而其 B 边接出去即为数据总线 D0～D7。而 74LS245 的允许控制

信号$\overline{E}$接到 CPU 的数据允许输出$\overline{DEN}$信号上，它的方向控制信号 DR 接到 CPU 的数据发送接收(即写入和读出)信号 DT/R上。在这样信号的控制下，形成了数据总线 D0～D7 ，很方便地实现数据总线的双向(读和写)传送。

同样，若系统总线上不进行 DMA 传送，则数据总线就可以如上述方法形成；若要利用系统总线进行 DMA 传送,则加到 74LS245 允许控制端$\overline{E}$上的信号必将 CPU 的DEN和 HLDA 先经过或门，然后将或门的输出加到 74LS245 的$\overline{E}$上。当不进行 DMA 时，HLDA 为低电平，74LS245 受$\overline{DEN}$控制。当进行 DMA 传送时，HLDA 为高电平，经或门后输出的高电平加到 74LS245 上，使数据总线 D0～D7 处于高阻状态。

在最小模式下，CPU 可以输出控制信号 IO/$\overline{M}$、$\overline{RD}$、$\overline{WR}$、$\overline{INTA}$等。在形成系统总线的控制总线时，可以直接利用这些信号。对于小系统这些信号可以不加驱动器；对于规模较大的系统，这些控制信号应用三态门（如 74LS244 或其他器件）进行驱动。同样，在系统总线不进行 DMA 传送时，三态门可接成总是有效状态，即将 74LS244 的控制端 E1 和 E2 接地。若进行 DMA 传送可将 74LS244 的控制端 E1 和 E2 接到 HLDA 上，则在 CPU 响应加在 HOLD 上的 DMA 请求时，会使三态门 74LS244 的输出置成高阻状态。在不采用驱动器时，CPU 响应 DMA 请求会将有关控制信号置成高阻状态。

2. 最大模式下系统总线的形成

在最大模式下，系统总线中的数据总线 D0～D7 和地址总线 A0～A19 的形成方法与最小模式相同。不同的是锁存器所用的锁存信号不是由 CPU 提供的，而是由总线控制器 8288 来提供。同时，双向数据总线驱动器的方向控制信号 DT/R及允许信号 DEN 也均由 8288 来产生，只是 8288 产生的 DEN 为高电平有效。而在最小模式时，$\overline{DEN}$是由 8088 CPU 自己产生且低电平有效。

在最大模式下，形成系统总线时必须采用总线控制器 8288。它除了产生上述三个信号外，还产生多个控制信号：$\overline{IOR}$、$\overline{IOW}$、MEMR、MEMW、INTA 等。这些信号构成了主要的控制总线信号。

同样，在此工作模式下，若系统总线上要实现 DMA 传送，则必需对包括 8288 在内的所有总线形成芯片进行仔细控制。这里不再详细说明。

1.1.6 关于其他 CPU

在前面只介绍了 Intel 早期的 8088 CPU。它是早期的 16 位 CPU 8086 的简化形式。后者与 8088 差别不大，在外部引线上 8086 CPU 的 AD8～AD15 是数据和地址信号复用的，而在 8088 上它们只用于传送地址信号 A8～A15。8088 上的 SSO 信号在 8086 上被定义为 BHE/S7 ,以保证 8086 可以一次读写一个字(16位),这就意味着 8086 是一个真正的 16 位 CPU。8088 的 IO/$\overline{M}$信号在 8086 上变为 M/$\overline{IO}$。在 8086 内部有 6 个字节的指令预取队列，

而在 8088 内部是 4 个字节的预取队列。

从那以后 Intel 陆续推出了 80186、80286、80386、80486 及一系列的奔腾（Pentium）CPU 芯片。从 8086/88 到 80186/188 技术上没有很大的进展，只是通过提高集成度减少了相应的配套芯片。

80286 CPU 在技术上有了实质的进展，在其硬件设计和指令设置上支持多用户、多任务处理、支持虚拟存贮器管理并设置有硬件保护机构。但是， 80286 的上述特殊功能很有限，它只是一个从 16 位向 32 位过渡的产品，很快地被后面出现的 CPU 所代替。

80386 是真正的 32 位 CPU，支持多用户、多任务处理，可访问的物理地址为 4 GB，而虚拟寻址空间可达 64 TB(即 64 MMB)。它具有实地址模式、保护模式和虚拟 8086 三种工作方式。其后的 80486、Pentium 都是在 80386 的基础上发展起来的。

80486 主要是将 80386、80387、8 KB 的高速缓存(Cache）及支持多处理器的硬件集成在一块芯片中，其性能比 80386 更优越。

Pentium 研制出来到现在又先后出现了许多改进、增强的品种。尽管今天的 Pentium Ⅲ CPU 上的数据线已达 64 位，但它仍是 32 位 CPU，因为其内部寄存器仍是 32 位。但是它内部又增加一些新的寄存器和 512 KB 的 Cache,已采用 0.18 μm 的微细加工技术，时钟频率已达 500 MHz，600 MHz，1000 MHz 或更高，其性能也有了极大的提高。

由于 80386 及其以后 CPU 的保护模式描述比较繁杂，因此教材中有关保护模式的详细内容，往往受教学时间的限制而没有仔细讲述。我们认为应重点掌握前面简单的 CPU 的基本概念，而对复杂的保护模式只作一般了解。

1.2 重点及难点

本章所涉及的内容并不复杂，但是，由于是本课程开始的部分，许多同学对这门课程的思想方法和分析问题解决问题的思路还不适应，故一开始会感到比较困难。随着后面章节的介绍，就会很快入门。

1.2.1 微型机的组成及工作

提到微型计算机的组成，要求读者立即反应出来它是由硬件和软件两大部分构成的。硬件上应记住图 1.1 所示的概念框图，了解图中所给出的微型机的几个主要组成部分以及各组成部分的大致功能。在此基础上记住微型机硬件必须在软件的支持下才能工作。微型机的工作就是不知疲倦地运行软件。

微型机的软件（程序）是以二进制编码的形式逐条放在内存中，读者应当了解指令的执行过程，并建立起 CPU 从内存中先取出指令代码，然后再执行该指令的功能的概念。CPU 就是这样逐条指令执行下去，直到全部指令执行完毕。

1.2.2 从 8088 CPU 引线到系统总线形成

8088 CPU 可以工作在最小／最大模式下，应了解在两种模式下信号的功能，不管是加到 CPU 上的外部控制信号还是 CPU 产生的用于形成系统总线的总线接口信号。CPU 的信号都是不可改变的。读者只有在理解的基础上记住它们。

利用 CPU 的信号可以构成系统总线信号，而 CPU 对内存或接口的读写就是通过这些总线信号来实现的。例如，在最小模式下，对内存的读操作所使用的系统总线信号是 A0～A19、IO/M 和$\overline{RD}$信号，在这些信号作用下，数据便从选中的内存单元中读出并加到 D0～D7 的数据总线上，通过数据总线进入 CPU。而在最大模式下，只用总线信号 A0～A19 和$\overline{MEMR}$便可将存贮单元的内容读出，经数据总线 D0～D7 传给 CPU。对于其他情况，内存写及接口的读、写这里不再说明。

1.2.3 内部寄存器及内存地址的形成

在 8088 CPU 中有多个内部寄存器，这些寄存器有些是通用的，有些则具有特殊性，前面已作了说明，这里仅强调两个问题。

1. 标志寄存器

很好地理解标志寄存器各标志位的定义，何时清零而何时又会置 1。

2. 内存地址的形成

在 8088 CPU 中，内存地址的形成很具特色。它是由段寄存器的内容和偏移地址共同决定的。CPU 取指令的内存地址一定是代码段寄存器 CS 的内容乘以 16 再加上 16 位的指令指针寄存器 IP 的内容。而存取数据的内存地址则是由其他的段寄存器 DS、ES 或 SS 的内容乘以 16 再加上 16 位的偏移地址来构成的。我们反复说明这一点的目的是使读者建立起牢固的概念。

1.2.4 时序的概念

时序对于理解 CPU 指令执行过程和硬件调试都是十分重要的。读者必须记住，在 8088 中一条指令是由若干个总线周期来完成的，也就是说是一个总线周期接一个总线周期来完成的。任何时候 CPU 不可能在同一时刻同时执行两个或两个以上的总线周期。进一步可以推论任何时刻地址总线上只能有一个确切的地址 A0～A19 控制信号 IOR、$\overline{IOW}$、$\overline{MEMR}$、$\overline{MEMW}$ 中决不可能有两个以上的信号同时有效，也就是说当 8088 CPU 执

行指令时，任何时候这四个控制信号最多只能有一个有效（低电平）。这对于理解后面章节的内容将是很有用的。

时序对于理解 CPU 的工作是很有帮助的。希望能够记住在一个总线周期里哪些信号在哪个时钟周期里出现，在后面的综合运用中将会频繁地应用这些概念。

1.3 例题分析

本章例题相对比较简单，下面就几个例题进行简要说明。

例题 1　试说明 RESET 信号的功能。

8088 CPU 的 RESET 信号是一个高电平有效的输入信号。当它高电平复位时：

① 会影响到 CPU 的一部分内部寄存器，使 CS 里的内容为 FFFFH，使 DS、SS、ES 及 F(标志)寄存器的内容均为 0000H。

② 使 8088 CPU 所有的三态输出信号线均为高阻状态，使所有的输出控制信号均为无效电平。

③ 当由 RESET 启动时，在 CPU 内部由 CS 的内容为 FFFFH 和 IP 的内容为 0000H 共同形成复位启动的物理地址为 FFFF0H。这个启动入口地址是必须记住的，进一步讲其他 CPU 或单片机的复位启动地址也是应当记住的，这在工程应用中也常用到。

例题 2　说明 8088 CPU 上 READY 信号的功能。

在由 8088 CPU 构成的微型计算机中（参见图 1.1），若是构成微型机的内存或外设接口的读写速度很慢，也就是说要求在一个读或写的总线周期里加到内存或接口上的有关信号（例如地址信号、读写控制信号及写入时的数据信号）要维持较长的时间，而 8088 CPU 在正常情况下对它们的读和写只有 4 个时钟周期，时间过短，不能满足内存和接口的要求，则必然导致 CPU 读写的不可靠。为了解决这个问题，可以利用 CPU 上的 READY 信号。在每一个读写内存或读写接口的总线周期里，我们可以在时钟周期 T3 到来之前产生一个低电平的 READY 信号加到 CPU 上。CPU 在 T3 开始的下降沿检测此信号，发现它是低电平就会插入一个等待的时钟周期 TW。并且，在 TW 的下降沿继续检测 READY 输入，若为低电平则继续插入 TW，直到 READY 变为高电平为止。因此，只要我们适当控制加到 READY 上低电平信号的宽度，想插入几个 TW 就能插入几个。我们总可以利用 READY 信号通过插入若干个 TW，在一个总线周期里使 CPU 提供的读写时间满足内存或外设接口的要求，从而使慢速的内存或接口与快速的 CPU 相适应，实现正确地读写操作，这就是 READY 信号的功能。

例题 3　试说明 8088 CPU 上 $\overline{\text{TEST}}$ 信号的功能。

$\overline{TEST}$是一个输入信号。在 8088 CPU 的指令系统中有一条等待指令 WAIT，当 CPU 执行该指令时，CPU 的内部硬件会自动检测$\overline{TEST}$的输入状态。若输入为低电平，则该指令执行结束并继续执行其下面的指令，这时 WAIT 指令相当于一条空操作指令 NOP。若其输入为高电平，则 CPU 在这条指令上等待，不再向下执行，而且在等待的过程中每一个时钟周期都要检测 TEST 的状态，一直检测到发现$\overline{TEST}$的状态变为低电平即逻辑 0 时，CPU 脱离 WAIT 指令，继续向下执行。

利用$\overline{TEST}$输入信号的这一特点，可以实现硬件或软件的调试。例如当需要测试某段软件时，可以在所要测试的位置上插入 WAIT 指令，而后使$\overline{TEST}$ 输入的状态变为高电平并开始执行这段软件。当执行到 WAIT 指令时，CPU 停下来等待$\overline{TEST}$的状态变低。这时，可以测试软件从开始执行到 WAIT 指令段程序的执行结果是否正确。若程序与接口的输出状态有关，可以用仪器测量接口的输出，看其状态与程序应产生的输出状态是否一致。若是正确的，则只要将$\overline{TEST}$的状态变低，CPU 又会向下执行。若结果不对，则需要设计者仔细分析，判断问题出现在程序上还是硬件上。

例题 4　在 8088 CPU 的标志寄存器中，进位标志 CF 的含义及功能是什么？

在 CPU 进行加法运算有进位和进行减法运算有借位时，进位标志 CF 都会被置 1；而没有进位或没有借位时 CF 为 0。还有一些指令在执行后也会影响 CF 标志，例如循环和位移指令、BCD 码加减法校正指令、比较指令等等都会影响到 CF 标志位。读者应记住一些常用的对 CF 有影响的指令，对使用中记忆不清的指令可以去查找厂家给出的指令系统手册。

在实际应用中，经常利用 CF 来实现多字节或多个字长的加减法，在使用时要采用设置了进位标志的加减法（更详细的说明见下一章）。同时，在比较两个无符号数的大小时也会用 CF 来加以判断。可以想象，拿两个无符号数相减，若相减的结果 CF＝1，那么一定说明被减数比减数小；反之，若相减的结果 CF＝0，则说明被减数一定不比减数小。

例题 5　当 8088 CPU 工作在最小模式时，形成系统总线要用到 CPU 上的哪些信号？当利用该系统总线对内存读或对接口写时又分别用到哪些总线信号？

当 8088 CPU 工作在最小模式时，形成系统总线时应利用出现在一个总线周期中第一个时钟周期 T1 由 CPU 送出的地址锁存信号 ALE，由它锁存地址信号形成地址总线信号 A0～A19。利用 CPU 送出的数据允许信号$\overline{DEN}$控制双向数据总线驱动器的允许控制端。利用 CPU 上的 DT/$\overline{R}$ 信号加到双向数据驱动器的方向控制端上，以便控制双向数据总线驱动器信号的传送方向。8088CPU 送出的控制信号如$\overline{RD}$、$\overline{WR}$、IO/$\overline{M}$等可以直接用作控制总线信号，也可以通过三态门驱动器输出作为控制总线信号。当系统总线上不进行 DMA 传送时，这些三态门驱动器可以控制为永远导通。

当 CPU 对内存某单元进行读操作时，系统总线的地址总线 A0～A19 上传送出内存单元的地址、控制总线上的 IO/M 有效(为低电平)、$\overline{RD}$信号有效（为低电平）并加到内存上。在这些信号的共同作用下，被地址选中的内存单元将其存贮的数据传输到数据总线 D0～D7 上，并通过该总线将数据传送给 CPU，由 CPU 读到其内部寄存器中。

当 CPU 对接口进行写操作时，由 CPU 送出的接口地址出现在地址总线 A0～A15 或 A0～A7 上，要写入接口的数据由 CPU 送出放在数据总线 D0～D7 上，同时控制总线 IO/$\overline{M}$ 有效(为高电平)、$\overline{WR}$有效(为低电平)。在这些信号的共同作用下，将数据总线上的数据写到了地址总线所决定的接口地址中。

对于内存的写和接口的读操作，读者可自行思考，不难做出说明。

1.4 自测题

1. 当 8088 CPU 工作在最大模式时，在系统总线形成时要用到哪些控制信号？它们是由谁产生的？它们的作用是什么？

2. 在最大模式下，8088 CPU 实现对内存的读、写或对接口的读、写时，各用到系统总线上的哪些信号？

3. 当 8088 CPU 工作在最大模式时，其三个状态输出 S0、S1 和 S2 的状态编码都表示 CPU 的哪些工作状态？

4. 叙述 8088 CPU 内部的标志寄存器中各标志位的定义。

5. 试叙述将一个字节写入内存的过程、一条指令的执行过程及一段程序的执行过程。

6. 说明 8088 CPU 中有哪些内部寄存器？它们是多少位的寄存器？20 位的内存地址是如何利用 16 位的寄存器形成的？

7. 8088 CPU 上的 NMI、INTR、HOLD、$\overline{INTA}$ 及 HLDA 是什么信号？以什么形式为有效？其作用是怎样的？

8. 在 8088 CPU 中，段寄存器的内容如下所列，试写出各段寄存器所决定的内存段的起始地址和终止地址。

(1) 1000H

(2) 2345H

(3) 3330H

(4) D000H

(5) EB00H

9. **在 8088 CPU 中，其 CS 寄存器和 IP 寄存器的内容分别如下所示，试确定被读出执行指令的内存地址。**

(1) CS＝1000H　　IP＝2000H
(2) CS＝2000H　　IP＝1000H
(3) CS＝3400H　　IP＝1A00H
(4) CS＝1A00H　　IP＝C000H
(5) CS＝1234H　　IP＝ABCDH

第 2 章　指令系统与汇编语言程序设计

本章的内容主要包括三大部分：寻址方式、指令系统和汇编语言简介。要求读者了解 8088 CPU 的寻址方式，见到某条指令能知道其寻址方式，并能熟练地写出某种寻址方式的指令。要求熟悉书中的一些常用指令，能读懂几十条指令以内的程序；掌握一些常用的指令，能编写出 50 条指令内的短小程序。要求读者熟悉书中所提到的有关汇编语言的一些规定。

本章所涉及的 8088 CPU 指令系统和汇编语言读者必须牢牢地记住。其指令系统及每一条指令的功能是确定的，再也无法改变，变成了规定。显然，如果你没有记住指令的功能，不但不可能用好它，而且也不可能读懂别人编写的程序。

虽然指令是不可改变的，绝无随意改变的灵活性。但是，用这些硬性规定的指令可以编写出各式各样、灵活多变的程序来。我们认为微型机之所以“无所不能”，在很大程度上取决于其软件。当然用汇编语言编写的程序也在软件之列。读者在学习本章时请注意到这一特点。

2.1　主要内容

2.1.1　寻址方式

通过学习寻址方式，使读者熟悉每一种与数据操作有关的指令，并用它们编写程序；熟悉与程序转移有关的寻址方式，能选择合适的寻址方式进行编程。寻址方式包括如下所述的两个方面。

1. 与操作数有关的寻址方式

我们先以一条最常用的传送指令为例说明指令的基本结构。传送指令如下：

```
MOV AX, BX
```

其中，MOV 称为指令助记符，在汇编时能够形成指令的操作码。操作码的作用就是告诉 CPU 这条指令要做什么。BX 和 AX 分别叫作源操作数和目的操作数。传送指令总是将源

操作数复制到目的操作数的位置中，即数据是由右向左复制的（把 BX 的内容复制到 AX 中），而且 BX 的内容在 MOV 指令执行之后（复制之后）依然保持不变。在指令中，源操作数和目的操作数之间用逗号分开。

在 8088 CPU 中，为了决定操作数的地址，指令中通常给出操作数的偏移地址，而段寄存器或特别指定或隐含默认。与操作数有关的寻址方式有如下 8 种：

①立即寻址：操作数就包含在指令中。

②直接寻址：指令中直接给出操作数的偏移地址。

③寄存器寻址：操作数的地址为某一寄存器。

④寄存器间接寻址：寄存器的内容作为操作数的偏移地址，可以作为寄存器间接寻址的寄存器有 BP、BX、SI、DI。在使用 BP 进行寄存器间接寻址时，默认段寄存器为堆栈段寄存器 SP。而当使用 BX、SI 和 DI 进行寄存器间接寻址时，默认段寄存器为数据段寄存器 DS。若要使用别的段寄存器，则需特别指定。

⑤寄存器相对寻址：操作数的偏移地址由某个基址或变址寄存器（BP、BX、SI 或 DI 中的一个）的内容加上指令中给出的一个 8 位或 16 位带符号的二进制数所表示的位移量，得到段寄存器则与间接寻址时相同：使用 BP 时，默认为 SP；而使用 BX、SI 或 DI 时，默认为 DS。

⑥基址变址寻址：在这种寻址方式下，操作数的偏移地址是由一个基址指针（寄存器）BP 或 BX 和一个变址指针（寄存器）SI 或 DI 的内容之和来产生。请注意它们的组合，一定是一个基址加一个变址。同时，还应记住凡是使用了 BP，则默认段寄存器为 SP；而使用 BX 时默认的段寄存器为 DS。显然，最终形成的物理内存地址就如在第 1 章中所描述的那样：段寄存器的内容乘 16 加上偏移地址。

⑦基址变址相对寻址：在这种寻址方式下，操作数的偏移地址是在上一种寻址方式的基础上，即在基址指针加变址指针的基础上再加上一个 8 位或 16 位带符号的二进制数所表示的位移量。而形成物理地址时所用到的段寄存器如前，即凡是使用了 BP 则段寄存器默认为 SP；而使用 BX 则默认为 DS。

⑧隐含寻址：指令中不给出操作数的地址，只有操作码，操作数就隐含在操作码中。例如压缩 BCD 数加法校正指令 DAA，该指令就只有指令操作码 27H，其中隐含着的操作数一定在 AL 中。

与操作数有关的寻址方式可用图 2.1 来说明。

2. 与程序转移地址有关的寻址方式

在第 1 章中，我们已经给出 8088 CPU 执行程序时，从内存中读取指令的地址是如下确定的：

取指令字节的内存物理地址 $=CS\times16+IP$