汇编语言

类目：计算机类
书名：汇编语言
主编：阎 双 秦 朗 邓国记
出版社：中国建设科技出版社
开本：大 16 开
书号：978-7-5160-4388-2
使用层次：通用
出版时间：2025 年 10 月
定价：49.80 元
印刷方式：双色
是否有资源：有

项目统筹：李亚博
责任编辑：汪永涛
封面设计：旗语书装

汇编语言

专·精·志·远
为您提供专业服务
编　辑　部：010-63567684
事业发展中心：010-63567692
网 上 书 店：www.jskjcbs.com
本社微信公众号
本社天猫旗舰店

ISBN 978-7-5160-4388-2
定价：49.80元

计算机类创新融合精品规划教材
“互联网+”教育改革新理念教材
汇编语言
主编◎阎双　秦朗　邓国记

中国建筑科技出版社有限责任公司

计算机类创新融合精品规划教材
“互联网+”教育改革新理念教材

汇编语言
主编◎阎双　秦朗　邓国记

中国建筑科技出版社有限责任公司
China Construction Science and Technology Press Co., Ltd.

计算机类创新融合精品规划教材
“互联网＋”教育改革新理念教材

汇编语言

主　编◎阎　双　秦　朗　邓国记

中国建设科技出版社有限责任公司
China Construction Science and Technology Press Co., Ltd.
北　京

图书在版编目(CIP)数据

汇编语言 / 阎双，秦朗，邓国记主编. --北京 ：中国建设科技出版社有限责任公司，2025. 10. -- ISBN 978-7-5160-4388-2

Ⅰ. TP313

中国国家版本馆 CIP 数据核字第 2025VF9684 号

汇编语言

HUIBIAN YUYAN

主　编　阎　双　秦　朗　邓国记

出版发行：中国建设科技出版社有限责任公司

地　　址：北京市西城区白纸坊东街 2 号院 6 号楼

邮　　编：100054

经　　销：全国各地新华书店

印　　刷：唐山唐文印刷有限公司

开　　本：880mm×1230mm　1/16

印　　张：14.5

字　　数：360 千字

版　　次：2025 年 10 月第 1 版

印　　次：2025 年 10 月第 1 次

定　　价：49.80 元

本社网址：www.jskjcbs.com，微信公众号：zgjskjcbs

请选用正版图书，采购、销售盗版图书属违法行为

版权专有，盗版必究。本社法律顾问：北京天驰君泰律师事务所，张杰律师

举报信箱：zhangjie@tiantailaw.com　　举报电话：(010) 63567684

本书如有印装质量问题，由我社事业发展中心负责调换，联系电话：(010) 63567692

编　委　会

主　编　阎　双　秦　朗　邓国记

副主编　周　佳　王鸿丹　甘　霖

张红霞　王　春

前　言

汇编语言能够直接、有效地控制硬件，可用于编写代码量小、运行速度快的高效程序，在计算机及相关专业的教学和许多应用场合中具有不可或缺的作用。汇编语言使用处理器指令编程，是一种底层程序设计语言。学习汇编语言的目的是从软件角度理解计算机硬件工作原理，为相关课程提供基础知识，同时全面认识程序设计语言，体会底层编程特点，以便更好地应用高级语言。

本书系统地介绍了汇编语言程序设计的相关知识，全书共有八章，包括汇编语言概述、数据表示和寻址、通用数据处理指令、程序结构、模块化程序设计、32 位指令及其编程、Windows 编程、浮点与 64 位指令。为了让读者能够及时检查自己的学习效果，把握自己的学习进度，每章后面都附有相应的习题。

本书既可以作为各专业汇编语言课程的教材，又可以作为汇编语言培训或技术人员自学的参考资料。

限于学术水平有限，本书不妥之处在所难免，敬请广大读者批评指正。

编　者

2025 年 6 月

目　录

第一章

汇编语言概述

学习目标

1. 理解汇编语言的硬件基础。
2. 掌握汇编语言的程序格式。
3. 掌握基于 DOS 操作系统和微软 MASM 汇编程序的程序开发方法。

程序设计语言是人与计算机沟通的语言，程序员利用它进行软件开发。通常人们习惯使用类似自然语言的高级程序设计语言，例如 C、C++或者 Basic、Java 语言等。高级语言需要翻译为计算机能够识别的指令，即机器语言，才能被计算机直接执行。机器语言是一串由 0 和 1 组成的二进制代码，对程序员来说艰涩难懂，被称为低级语言。将二进制代码的指令和数据用便于记忆的符号——助记符——表示就形成汇编语言，所以汇编语言是一种面向机器的低级程序设计语言，或称为低层语言。

第一节 个人计算机系统概述

计算机系统包括硬件和软件两大部分。硬件是指构成计算机的实在的物理设备，是人们看得见、摸得着的物体，就像人的躯体。软件一般是指在计算机上运行的程序(广义的软件还包括由计算机管理的数据以及有关的文档资料)，是指示计算机工作的命令，就像人的思想。

微型计算机，简称微机，是最常使用的一类计算机，在科学计算、信息管理、自动控制、人工智能等领域有着广泛的应用。工作、学习和娱乐中使用的个人计算机(PC)是我们最熟悉、也是最典型的通用微型计算机。

一、计算机的硬件

源于冯·诺依曼设计思想的计算机由五部分组成：控制器、运算器、存储器、输入设备和输出设备。控制器是整个计算机的控制核心；运算器是对数据进行运算处理的部件；存储器是用来存放数据和程序的部件；输入设备将数据和程序变换成计算机内部所能识别和接收的信息方式，并把它们送入存储器中；输出设备将计算机处理的结果以人们能接受的或其他机器能接受的形式送出。

现代计算机在很多方面都对冯·诺依曼计算机结构进行了改进，五大部件演变为三个硬件子系统：处理器、存储系统和输入输出系统。运算器和控制器被制作在一块大规模集成电路芯片上，称为处理器，也常被称为中央处理单元 CPU(Central Processing Unit)。传统的存储器也发展成为存储系统，由寄存器、高速缓冲存储器、主存储器及辅助存储器构成。处理器和存储系统在信息处理中起主要作用，是计算机硬件的主体部分，通常被称为主机。输入设备和输出设备统称为外部设备，简称为外设或 I/O 设备；输入输出系统的主体是外部设备，但还包括外设与主机之间相互连接的 I/O 接口电路。

为简化各个部件的相互连接，现代计算机广泛应用总线结构，见图 1-1。采用总线连接系统中各个功能部件使计算机系统具有了组合灵活、扩展方便的特点。

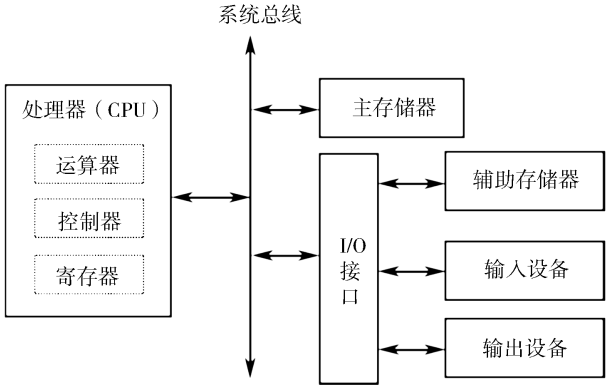


图 1-1　计算机系统的硬件组成

1. 处理器

处理器是计算机的运算和控制核心，微机中一般称为微处理器。现代通用微处理器功能非常强大，人们已经习惯称为处理器或者 CPU。

(微)处理器芯片内集成了控制器、运算器和若干高速存储单元(寄存器)。高性能处理器内部非常复杂，例如运算器中不仅有基本的整数运算器，还有浮点处理单元甚至多媒体数据运算单元，控制器还会包括存储管理单元、代码保护机制等，为提高存储器的性能还会集成高速缓冲存储器。处理器及其支持电路构成了计算机系统的处理和控制中心，对系统的各个部件进行统一的协调和控制。

PC 采用美国英特尔(Intel)公司的 80×86 系列处理器或与其兼容的处理器，例如常用的奔腾系列处理器或者酷睿系列多核处理器。之所以称为 Intel 80×86 系列处理器，是因为它们都源于 16 位结构的 Intel 8086 处理器，而 8086 具有的所有指令，即指令系统，是整个 Intel 80×86 系列处理器的基本指令集。

2. 存储器

存储器是计算机的记忆部件，用来存放程序和数据。存储系统由处理器内部的寄存器、高速缓冲存储器、主板上的主存储器和以外设形式出现的辅助存储器构成。

按所起作用，存储器可分为主存储器和辅助存储器。主存储器(简称主存或内存)由半导体存储器芯片组成，安装在机器内部的电路板上，相对辅助存储器来说速度快，但容量小、造价高，主要用来存放当前正在运行的程序和等待处理的数据。辅助存储器(简称辅存或外存)主要由磁盘、光盘存储器等构成，以外设的形式安装在机器上，相对主存储器来说造价低、容量大，信息可长期保存，但速度较慢，主要用来长久保存程序和数据。一般来说，程序和数据以文件形式保存在辅存上，只有使用它们时才读入主存。

按读写功能，存储器可分为可读可写存储器和只读存储器 ROM(Read Only Memory)。半导体存储器具有按指定位置访问，即随机存取的特点，所以可读可写的半导体存储器常被称为 RAM(Random Access Memory)。构成主存既需要 RAM，也需要 ROM，但需要注意的是，存放在 RAM 芯片上的信息断电后将会丢失，而 ROM 芯片中的信息则可在断电后保存。通常作为辅存的磁盘存储器和 CD-ROM 光盘都可以在断电后长期保存信息，它们二者的不同在于，CD-ROM 光盘是只读的，而作为辅存的磁盘存储器是可读可写的。不过，由于读写时涉及磁头或光头的移动、磁盘或光盘的旋转，

它们的存取性能低于半导体存储器。

个人计算机的主存由半导体存储芯片 RAM 和 ROM 构成。在 16 位 PC 系列机时代，RAM 容量不过是 64kB 或 1MB。32 位 PC 的 RAM 容量从最初的 4MB，逐渐发展直到 8GB 或 16GB。由于大量应用程序都会占据一定 RAM 主存空间，因此 PC 的主存主要由 RAM 构成，俗称主存条(内存条)。

个人计算机的 ROM 部分主要是固化的 ROM-BIOS。BIOS(Basic Input/Output System)表示“基本输入输出系统”，是 PC 软件系统最底层的程序。它由诸多子程序组成，主要用于驱动和管理诸如键盘、显示器、打印机、磁盘、时钟、串行通信接口等基本的输入输出设备。操作系统通过对 BIOS 的调用驱动各硬件设备，用户也可以在应用程序中调用 BIOS 中的许多功能。ROM 空间还包含机器复位后初始化系统的程序，它将操作系统引导到 RAM 存储器中执行。

3. 外部设备

外部设备是指计算机上配备的输入(Input)设备和输出(Output)设备，也称 I/O 设备或外围设备，简称外设(Peripheral)，其作用是让用户与计算机实现交互。

个人计算机上配置的标准输入设备是键盘、标准输出设备是显示器，二者合称控制台(Console)。个人计算机还可使用鼠标、打印机等 I/O 设备。作为外部存储器驱动装置的磁盘驱动器，既是输出设备，又是输入设备。

由于各种外设的工作速度、驱动方法差别很大，无法与处理器直接匹配，所以不可能将它们直接连接到主机。这里就需要有一个 I/O 接口来充当外设和主机之间的桥梁，通过该接口电路来完成信号变换、数据缓冲、联络控制等工作。在个人计算机中，较复杂的 I/O 接口电路通常制成独立的电路板，也常被称为接口卡(Card)，例如显示卡，使用时需要将其插在主板的总线插槽上。

4. 系统总线

总线(Bus)是用于多个部件相互连接、传递信息的公共通道，物理上就是一组公用导线。例如，处理器芯片的对外引脚(Pin)常被称为处理器总线。这里的系统总线(System Bus)是指计算机系统中主要的总线，例如处理器与存储器和 I/O 设备进行信息交换的公共通道。

16 位 PC 采用 16 位工业标准结构 ISA(Industry Standard Architecture)系统总线连接各个功能部件。32 位 PC 上使用外设部件互连 PCI(Peripheral Component Interconnect)总线连接 I/O 接口卡。系统总线除了作为主板上处理器、主存和 I/O 接口的公共通道外，主板上还设置有许多系统总线插槽，主要用于插接 I/O 接口电路以扩充系统连接的外设，故也被称作 I/O 通道。

对汇编语言程序员来说，处理器、存储器和外部设备依次被抽象为寄存器、存储器地址和输入输出地址，因为编程过程中将只能通过寄存器和地址实现处理器控制、存储器和外设的数据存取及处理等操作。

二、计算机的软件

计算机的软件

完整的计算机系统包括硬件和软件，软件又分为系统软件和应用软件。

1. 系统软件

系统软件是为了方便使用、维护和管理计算机系统的程序及其文档，通常由计算机生产厂商或者软件公司提供，其中最重要的是操作系统。

操作系统 OS(Operating System)管理着系统的软硬件资源，为用户提供使用机器的交互界面，为程序员使用资源提供可供调用的驱动程序，为其他程序构建稳定的运行平台。

程序员采用某种程序设计语言编写源程序，利用语言翻译程序将源程序转变成可运行的程序。例如，用汇编语言编写源程序的方法，但源程序必须利用“汇编程序”完成翻译才可被计算机执行。高级语言则采用编译类或解释类程序来完成这项工作。辅助程序开发的程序设计语言的翻译程序等一般也归类为系统软件。

2. 应用软件

应用软件是解决某个问题的程序及其文档，大到用于处理某专业领域问题的程序，小到完成一个非常具体的工作程序。它覆盖了计算机应用的所有方面，每个应用都需要相应的应用程序。

个人计算机系统具有多种多样的应用软件。例如，进行程序设计时要采用文本编辑软件编写源程序，带有丰富格式的字处理软件帮助你书写文章，排版软件则用于书刊出版。

大型的程序设计项目往往要借助软件开发工具(包)。这个开发工具是进行程序设计所用到的各种程序的有机集合，所以也被称为集成开发环境，包括文本编辑器、语言翻译程序，有用于形成可执行文件的连接程序，以及进行程序排错的调试程序等。

三、程序设计语言

利用计算机解决实际问题，一般要编制程序。程序设计语言就是程序员用来编写程序的语言，它是人与计算机之间交流的工具。程序设计语言可以分为机器语言、汇编语言和高级语言。

1. 机器语言

计算机能够直接识别的是由二进制数 0 和 1 组成的代码。机器指令就是用二进制编码的指令，指令是控制计算机操作的命令，是处理器不需要翻译就能识别(直接执行)的“母语”，通常一条机器指令控制计算机完成一个操作。每种处理器都有各自的机器指令，某处理器支持的所有指令的集合就是该处理器的指令集或指令系统。指令集及使用它们编写程序的规则被称作机器语言。

用机器语言形成的程序是计算机唯一能够直接识别并执行的程序，而用其他语言编写的程序必须经过翻译、变换成机器语言程序，因此，机器语言程序常称为目标程序(或目的程序)。

例如，完成两个数据 100 和 256 相加的功能，在 8086 处理器的二进制代码序列如下：

```
10111001 01100100 00000000
00000101 00000000 00000001
```

几乎没有人能够直接读懂该程序段的功能，因为机器语言就是看起来毫无意义的一串代码。用机器语言编写程序的最大缺点是难以理解，因而极易出错，也难以发现错误。所以，只是在计算机发展的早期或不得已的情况下，才采用机器语言编写程序。现在，除了有时在程序某处需要直接采用机器指令填充外，几乎没有人采用机器语言编写程序了。

二进制虽然只有 0 和 1 两个数码，但表达信息时会很长。为了简化表达，常用到十六进制。因为一个十六进制位就可以表达 4 位二进制数，并且易于相互转换，即二进制数 0000，0001，0010，…，1001，1010，1011，1100，1110，1111 用十六进制表达依次是 0，1，2，…，9，A，B，C，D，E，

F，其中 A～F 依次表示十进制 10～15。这样，上述二进制代码序列用十六进制代码表示为：

```
B8 64 00
05 00 01
```

汇编语言中，习惯使用后缀字母区别不同进制的数据。例如，使用字母 B(或小写字母形式 b，来自二进制的英文单词 Binary)表示二进制数，使用字母 H(或 h，来自十六进制的英文单词 Hexadecimal)表示十六进制数，而十进制数通常不需要特别说明(或者用后缀字母 D 或 d，以示强调)。另外，涉及计算机硬件原理的技术文献中，所谓的“位”常指二进制位，也会表示十六进制位或者十进制位，根据上下文予以分辨，否则可能产生误解。

2. 汇编语言

为了克服机器语言的缺点，人们采用便于记忆并能描述指令功能的符号来表示机器指令。表示指令功能的符号称为指令助记符，或简称助记符(Mnemonic)；助记符一般采用表明指令功能的英语单词或其缩写。指令操作数同样也可以用易于记忆的符号表示。用助记符表示的指令就是汇编格式指令。汇编格式指令以及使用它们编写程序的规则形成汇编语言(Assembly Language)。用汇编语言书写的程序就是汇编语言程序，或称汇编语言源程序。

例如，实现 100 与 256 相加的 MASM 汇编语言程序片段如下：

```
mov   ax,100
add   ax,256
```

第一条指令的功能将数据 100 传送给名为 AX 的寄存器，MOV 是传送指令的助记符，实现赋值功能。该指令对应的机器代码就是机器语言举例的第一个二进制串。

第二条指令实现加法操作，ADD 是加法指令的助记符，对应机器语言举例的第二个二进制串。

如果熟悉有关助记符及对应指令的功能，就可以读懂上述程序片段了。

汇编语言是一种符号语言，它用助记符表示操作码，比机器语言容易理解和掌握，也容易进行调试和维护。但是，汇编语言源程序要翻译成机器语言程序才可以由处理器执行。这个翻译的过程称为“汇编”，完成汇编工作的程序就是汇编程序。

3. 高级语言

汇编语言虽然较机器语言直观一些，但仍然烦琐难记。于是在 20 世纪 50 年代，人们研制出了高级程序设计语言（High Level Language)。高级语言比较接近于人类自然语言的语法习惯及数学表达形式，它与具体的计算机硬件无关，更容易被广大计算机工作者掌握和使用。利用高级语言，即使一般的计算机用户也可以编写程序，而不必懂得计算机的结构和工作原理。

目前，计算机高级语言已有上百种之多，得到广泛应用的有十几种，每种高级语言都有其最适用的领域。用任何一种高级语言编写的程序都要通过编译程序翻译成机器语言程序（称为目标程序）后计算机才能执行，或者通过解释程序边解释边执行。

例如，用高级语言表达 100 与 256 相加，就是通常的数学表达形式：100＋256。

4. 学习汇编语言的意义

高级语言简单、易用，而汇编语言复杂、难懂，是否就没有必要再采用汇编语言了呢？

表 1-1 列出了汇编语言和高级语言的特点。

表 1-1　汇编语言和高级语言的对比

汇编语言	高级语言
与处理器密切相关。每种处理器都有自己的指令系统，相应的汇编语言各不相同。所以，汇编语言程序的通用性、可移植性较差	与具体计算机无关，高级语言程序可以在多台计算机上编译后执行
功能有限，又涉及寄存器、主存单元等硬件细节。所以，编写程序比较烦琐，调试起来也比较困难	采用类似自然语言的语法，不必关心标志、堆栈等琐碎问题，容易被掌握和应用
本质上是机器语言，可以直接、有效地控制计算机硬件，因而容易产生运行速度快、指令序列短小的高效率目标程序	不易直接控制计算机的各种操作，编译程序产生的目标程序往往比较庞大，运行速度较慢

由表 1-1 可知高级语言的优势明显。很自然地人们称机器语言和汇编语言为低级语言。但事实上，汇编语言被称为低层语言（Low Level Language）更合适。程序设计语言是按照计算机系统的层次结构区分的，本没有高低贵贱之分，只是某种语言更适合某种应用层面（或说场合）而已。汇编语言便于直接控制计算机硬件电路，在时间和空间两方面最有效，即执行速度快和目标代码小的程序。这些优点使汇编语言在程序设计中占有重要的位置，是不可取代的。汇编语言的主要应用场合有以下几个。

（1）程序具有较快的执行时间，或者只能占用较小的存储容量。例如，操作系统的核心程序段，实时控制系统的软件，智能仪器仪表的控制程序等。

（2）程序与计算机硬件密切相关，程序要直接、有效地控制硬件。例如，I/O 接口电路的初始化程序段，外部设备的低层驱动程序等。

（3）大型软件需要提高性能、优化处理的部分。例如，计算机系统频繁调用的子程序、动态链接库等。

（4）没有合适的高级语言或只能采用汇编语言的时候。例如，开发最新的处理器程序时，暂时没有支持新指令的编译程序。

（5）许多实际应用的情况，例如分析具体系统尤其是该系统的低层软件、加密解密软件、分析和防治计算机病毒等。

当然，无须回避的事实是，随着各种编程技术的发展，单独使用汇编语言开发程序尤其是应用程序的情况越来越少。所以，在实际的程序开发过程中，可以采用高级语言和汇编语言混合编程的方法，互相取长补短，更好地解决实际问题。

另外，编写汇编语言程序，需要使用处理器指令解决应用问题，而指令只是完成将一个数据从存储器传送到寄存器、对两个寄存器值求和、指针增量指向下一个地址等简单的功能。所以，从教学角度来说，汇编语言程序员在将复杂的应用问题翻译成简单指令的过程中，就是从处理器角度解决问题，自然就容易理解计算机的工作原理了。

第二节 8086 处理器

处理器是计算机系统的硬件核心部件，决定着计算机的关键性能，常被称为中央处理单元，简称 CPU。了解处理器的基本结构、熟悉其寄存器作用以及存储器的组织是学习处理器指令的基础。

一、8086 的功能结构

处理器由多个功能部件组成。Intel 公司按两个功能模块描绘了 Intel 8086 处理器的内部结构，见图 1-2。

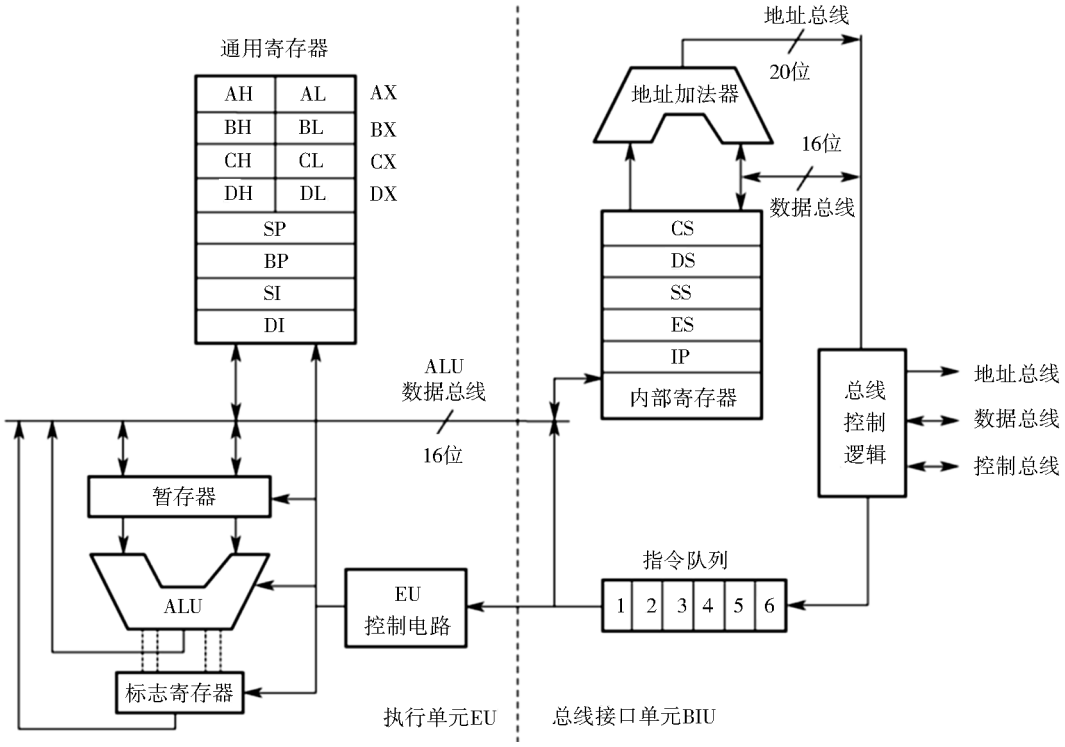


图 1-2　8086 的内部结构

1. 总线接口单元和执行单元

图 1-2 虚线右侧部分是总线接口单元 BIU（Bus Interface Unit）。它由 6 字节的指令队列（指令寄存器）、指令指针 IP（等同于程序计数器的功能）、段寄存器（CS、DS、SS 和 ES）、地址加法器和总线控制逻辑等构成。该单元管理着 8086 与系统总线的接口，负责处理器对存储器和外设进行访问。对外的 8086 处理器引脚由 16 位双向数据总线、20 位地址总线和若干控制总线组成。8086 所有对外操作必须通过 BIU 和这些总线进行，例如从主存中读取指令、从主存或外设读取数据、向主存或外设写出数据等操作。

图 1-2 虚线左侧部分是执行单元 EU（Execution Unit）。它由算术逻辑单元 ALU（Arithmetic

Logic Unit)、标志寄存器、通用寄存器和进行指令译码的 EU 控制电路等构成，负责执行指令的功能。

源程序由语句组成，而可执行程序则由指令组成，运行程序就是执行一条条指令的过程。一条指令的整个执行过程又可以分成两个主要阶段：取指和执行，见图 1-3（a）。

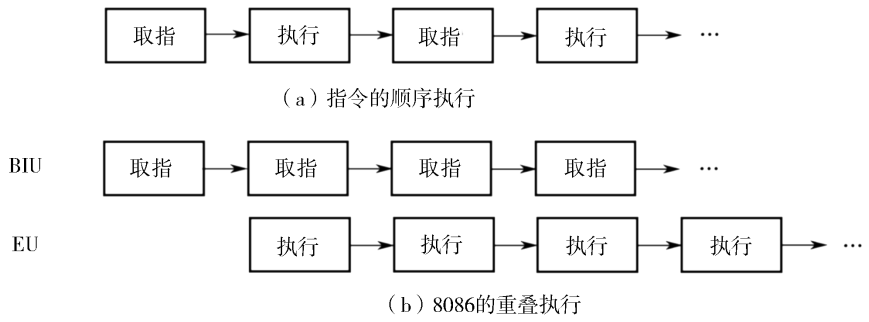
取指阶段是处理器将指令代码从主存储器中取出并进入处理器内部的过程。在 8086 处理器中，指令在存储器中的地址（位置编号）由代码段寄存器 CS 和指令指针寄存器 IP 共同提供，再由地址加法器得到 20 位存储器地址。总线接口单元 BIU 负责从存储器取出这个指令代码，送入指令队列。

执行阶段是处理器将指令代码翻译成它代表的功能（被称为译码）并发出有关控制信号实现这个功能的过程。8086 处理器中，执行单元 EU 从指令队列中获得预先取出的指令代码，在 EU 控制电路中进行译码，然后发出控制信号由算术逻辑单元 ALU 进行数据运算、数据传送等操作。指令执行阶段需要的操作数据有些来自处理器内部的寄存器，有些来自指令队列，还有些来自存储器和外设。如果需要来自存储器或外设的数据，则控制单元 EU 控制总线接口单元 BIU 从外部获取这些数据。

2. 指令预取

8086 处理器维护着长度为 6 字节的指令队列，该队列按照“先进先出”FIFO（First In First Out）的方式进行工作。当指令队列中出现空缺时，BIU 会自动取指填补这一空缺；而当程序不能按顺序执行，即发生转移（出现分支）时，BIU 又会废除已经取出的指令，重新取指形成新的指令队列。

在 8086 处理器中，指令的读取操作在 BIU 单元实现，而指令的执行阶段在 EU 单元完成。因为 BIU 和 EU 两个单元相互独立、分别完成各自操作，所以可以并行操作。也就是在 EU 单元对一个指令进行译码执行时，BIU 单元可以同时对后续指令进行读取。所以，8086 处理器的指令读取，实际上是指令预取（Prefetch），见图 1-3（b）。



（a）指令的顺序执行

（b）8086的重叠执行

图 1-3　指令的执行过程

由于要译码执行的指令已经预取到了处理器内部的指令队列，因此 8086 不需要等待取指操作就可以从指令队列获得指令进行译码执行。而对于简单的处理器来说，在指令译码前必须等待取指操作的完成。取指是处理器最频繁的操作，每条指令都要读取指令代码一到数次（与指令代码的长度有关），所以 8086 的这种结构和操作方式节省了处理器的许多取指时间，提高了工作效率。这就是最简单的指令流水线技术。同时也看到，程序转移将使预取指令作废，从而降低了流水线效率。

二、8086 的寄存器

处理器内部需要高速存储单元，用于暂时存放程序执行过程中的代码和数据，这些存储单元被

称为寄存器（Register）。处理器内部设计有多种寄存器，每种寄存器还可能有多个，从应用的角度可以分成两类：透明寄存器和可编程寄存器。

有些寄存器对应用人员来说无法通过指令直接编程控制，例如：保存指令代码的指令寄存器。它们对应用人员来说好像看不见一样，被称为透明寄存器。这里的“透明”（Transparency）是计算机学科中常用的一个专业术语，表示实际存在但从某个角度看好像没有；运用“透明”思想可以使人们抛开不必要的细节，而专注于关键问题。

底层语言程序员需要掌握可编程（Programmable）寄存器。它们具有引用名称，供编程使用，还可以进一步分成通用和专用寄存器。

（1）通用寄存器：这类寄存器在处理器中数量较多，使用频度较高，具有多种用途。例如，它们可用来存放指令需要的操作数据，又可用来存放地址以便在主存或I/O接口中指定操作数据的位置。

（2）专用寄存器：这类寄存器各自只用于特定目的。例如，程序计数器 PC（Program Counter）只用于记录将要执行指令的主存地址，标志寄存器保存指令执行的辅助信息。

8086 的寄存器有 8 个 16 位通用寄存器、4 个 16 位段寄存器、1 个 16 位标志寄存器和 1 个 16 位指令指针寄存器，见图 1-4（图中数字 15、7、0 等依次用于表达二进制位 D15、D7、D0）。

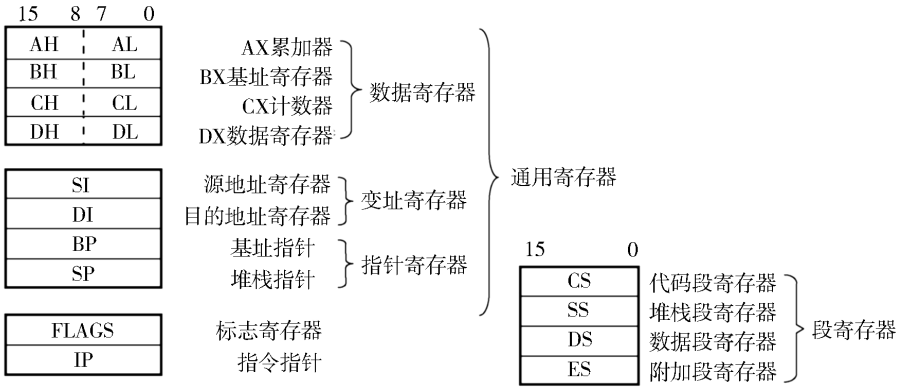


图 1-4　8086 的寄存器

扫一扫
通用寄存器

1. 通用寄存器

通用寄存器（General-Purpose Register）一般是指处理器最常使用的整数通用寄存器，可用于保存整数数据、地址等。8086 处理器只有 8 个通用寄存器，数量有限。

8086 处理器的 8 个 16 位通用寄存器，分别被命名为 AX、BX、CX、DX、SI、DI、BP 和 SP。其中前 4 个通用寄存器 AX、BX、CX 和 DX 还可以进一步分成高字节 H（High）和低字节 L（Low）两部分，这样又有了 8 个 8 位通用寄存器：AH 和 AL、BH 和 BL、CH 和 CL、DH 和 DL。前 4 个通用寄存器在编程中，可以整个使用 16 位寄存器（例如 AX），也可以分成两个 8 位使用：D15～D8（例如 AH）和 D7～D0（例如 AL），对其中某 8 位的操作，并不影响另外对应 8 位的数据。

通用寄存器的用途很多，可以保存数据、暂存运算结果，也可以存放存储器地址、作为变量的指针。但在 8086 处理器中每个寄存器又有它们各自的特定作用，并因其而得名。程序中通常也按照其含义进行使用（表 1-2）。

表 1-2　8086 处理器的通用寄存器

名称	中英文含义	作用
AX	累加器（Accumulator）	使用频度最高，用于算术、逻辑运算以及与外设传送信息等
BX	基址寄存器（Base）	常用作存放存储器地址，以方便指向变量或数组中的元素
CX	计数器（Counter）	常作为循环操作等指令中的计数器
DX	数据寄存器（Data）	可用来存放数据，在输入输出指令存放外设端口地址
SI	源地址寄存器（Source Index）	用于指向字符串或数组的源操作数
DI	目的地址寄存器（Destination Index）	用于指向字符串或数组的目的操作数
BP	基址指针寄存器（Base Pointer）	默认情况下指向程序堆栈区域的数据，主要用于在子程序中访问通过堆栈传递的参数和局部变量
SP	堆栈指针寄存器（Stack Pointer）	专用于指向程序堆栈区域顶部的数据，在涉及堆栈操作的指令中会自动增加或减少

许多指令需要表达两个操作数（操作对象，例如加法指令的被加数以及加法结果）。

（1）源操作数是指被传送或参与运算的操作数（例如加法的被加数）。

（2）目的操作数是指保存传送结果或运算结果的操作数（例如加法的和值结果）。

SI 和 DI 是变址寄存器，常通过改变寄存器表达的地址指向数组元素。SI 常用于指向源操作数，而 DI 常用于指向目的操作数。

堆栈（Stack）是一个特殊的存储区域，它采用先进后出 FILO（First In Last Out）也称为后进先出 LIFO 的操作方式存取数据。它用于调用子程序时暂存数据、传递参数、存放局部变量，也可以用于临时保存数据。BP 和 SP 是指针寄存器，用于指向堆栈中的数据。其中，SP 堆栈指针会随着处理器执行有关指令自动增大或减小，所以 SP 不应该再用于其他目的，实际上可归为专用寄存器；SP 可以像其他通用寄存器一样灵活地改变。

2. 标志寄存器

标志（Flag）用于反映指令执行结果或控制指令执行形式。许多指令执行之后将影响有关的状态标志位；不少指令的执行要利用某些标志；当然，也有很多指令与标志无关。处理器中用一个或多个二进制位表示一种标志，其 0 或 1 的不同组合表达标志的不同状态。Intel 8086 支持的 9 个标志，分为状态标志和控制标志两类，采用一个 16 位的标志寄存器 FLAGS 保存，见图 1-5（图中数字表示该标志在标志寄存器中的位置）。

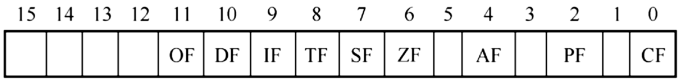


图 1-5　标志寄存器 FLAGS

状态标志是最基本的标志，用来记录指令执行结果的辅助信息。加减运算和逻辑运算指令是主要设置它们的指令，其他有些指令的执行也会相应地设置它们。状态标志有 6 个，处理器主要使用其中 5 个构成各种条件，分支指令判断这些条件实现程序分支。它们从低位到高位是：进位标志 CF（Carry Flag）、奇偶标志 PF（Parity Flag）、调整标志 AF（Adjust Flag）、零标志 ZF（Zero Flag）、符号标志 SF（Sign Flag）、溢出标志 OF（Overflow Flag）。

控制标志用于控制处理器执行指令的方式，可由程序根据需要用相关指令设置。8086 的控制标

志有 3 个：方向标志 DF（Direction Flag），仅用于串操作指令中，控制地址的变化方向；中断允许标志 IF（Interrupt-enable Flag），也称中断标志，用于控制外部可屏蔽中断是否可以被处理器响应；陷阱标志 TF（Trap Flag），也称单步标志，用于控制处理器是否进入单步操作方式。

3. 指令指针寄存器

程序由指令组成，指令存放在主存储器中。处理器需要一个专用寄存器表示将要执行的指令在主存的位置，这个位置用存储器地址表示。在 8086 处理器中，这个存储器地址保存在 16 位指令指针寄存器 IP（Instruction Pointer）中。

指令指针寄存器 IP 是一个专用寄存器，具有自动增量的能力。处理器执行完一条指令，IP 就加上该指令的字节数，指向下一条指令，实现程序的顺序执行。需要实现分支、调用等操作时需要修改 IP，它的改变将引起程序转移到指定的指令执行。但 IP 寄存器不能像通用寄存器那样直接赋值修改，需要执行控制转移指令（如跳转、分支、调用和返回指令）、出现中断或异常时被处理器赋值而相应改变。

4. 段寄存器

在一个程序中，有可以执行的指令代码，还有指令操作的各类数据等。遵循模块化程序设计思想，希望将相关的代码安排在一起，相关的数据安排在一起，于是（区）段（Segment）的概念自然出现。一个段安排一类代码或数据。程序员在编写程序时，可以很自然地把程序的各部分放在相应的段中。对应用程序来说，主要涉及 3 类基本段：存放程序中指令代码的代码段（Code Segment）、存放当前运行程序所用数据的数据段（Data Segment）和指明程序使用的堆栈区域的堆栈段（Stack Segment）。

段其实就是主存的一个连续区域，为了表明段在主存中的位置，8086 处理器设计有 4 个 16 位段寄存器：代码段寄存器 CS、堆栈段寄存器 SS、数据段寄存器 DS 和附加段寄存器 ES（Extra Segment）。其中，附加段也是用于存放数据的数据段，专为处理数据串设计的串操作指令必须使用附加段作为其目的操作数的存放区域。

三、8086 的存储器组织

个人计算机主板上的主存条和一个 ROM 芯片构成主存储器，保存正在运行使用的指令和数据。处理器从存储器读取指令，在执行指令的过程中读写数据。对存储器的基本操作是按照要求向指定地址（位置）存进［写入（Write）］或取出［读出（Read）］信息。应该注意的是，存储器内容不会因为被读出而消失，所以更准确地说是获取其副本，而且可以重复取出；只有存入新的信息后，原有信息才会被更改。

1. 存储单元和存储单位

主存储器是一个很大的信息储存库，被划分成许多存储单元。为了区分和识别各个存储单元，并按指定位置进行存取，就给每个存储单元编排一个顺序号码，称为存储单元地址或存储器地址（Memory Address）。现代计算机中，主存储器的每个存储单元都具有一个地址，保存一个字节（8 个二进制位）的信息，称为字节编址（Byte Addressable），也称字节可寻址，因为通过一个存储器地址可以访问到一个字节信息。

计算机中信息的基本单位是一个二进制位（bit，即比特），一般用小写字母 b 表示，一个二进制位可表示一位二进制数：0 或 1。8 个二进制位组成一个字节（Byte），常用大写字母 B 表示，位编号

由右向左（由低位向高位）从 0 开始递增计数为 D0～D7，见图 1-6。8086 是 16 位结构的处理器（字长为 16 位），主要处理的数据长度是 16 位，所以称 16 个二进制位为一个字（Word），位编号自右向左为 D0～D15。一个 16 位字由 2 个字节组成。32 位数据由 4 个字节组成，称为双字（Double Word），位编号自右向左为 D0～D31。

图 1-6　数据的位格式

一个二进制数据的右边最低位称为最低有效位 LSB（Least Significant Bit），即 D0 位；左边最高位称为最高有效位 MSB（Most Significant Bit），对应字节、字、双字长度的数据依次指 D7、D15 和 D31 位。

主存储器使用字节作为基本存储单位，表 1-3 罗列了表达更大的存储容量时常使用的单位及关系。虽然借用了日常生活中千、兆、吉等单位，但没有使用其 10^3 的十进制倍数关系，而是使用 $2^{10}=1024$ 的二进制倍数关系（近似等于 $10^3=1000$）。但有些产品，例如硬盘、U 盘的生产厂商给出容量却采用 10^3 倍数关系，应注意分辨。

表 1-3　常用存储单位

英文符号	中英文名称	二进制倍数关系	十进制倍数关系
K	千（Kilo）	$1KB=2^{10}B=1024B$	$1K=10^3B$
M	兆（Mega）	$1MB=2^{10}KB=2^{10}B$	$1M=10^3K$
G	千兆，吉（Giga）	$1GB=2^{10}MB=2^{30}B$	$1G=10^3M$
T	兆兆，太（Tera）	$1TB=2^{10}GB=2^{40}B$	$1T=10^3G$

2. 物理地址和逻辑地址

主存条和 ROM 芯片构成的主存储器需要处理器通过总线进行访问，被称为物理存储器。物理存储器的每个存储单元有一个唯一的地址，就是物理地址（Physical Address）。物理地址空间从 0 开始顺序编排，直到处理器支持的最大存储单元。

一个二进制位有 0 和 1 两种状态，即 2（$=2^1$）个编码，那么，2 位有 00、01、10 和 11 共 4（2^2）个编码，以此类推，借助排列组合知识，可以得到 N 位有 2^N 个编码。计算机使用数字信号传输信息，一个数字信号也只有低电平和高电平两个状态，可以分别用 0 和 1 表达，所以 N 位数字信号同样也有 2^N 个不同的编码。如果用 N 个数字信号传输访问存储器的地址信息，就可以区别出 2^N 个不同的存储单元，也就是说可以直接访问到 2^N 个存储单元。

8086 处理器具有 20 位地址总线，即用 20 个数字信号访问存储器，故 8086 可以支持 2^{20}，即 1M 个存储单元。由于每个存储单元保存一个字节数据，因此 8086 可以支持 1MB 存储器容量，其物理地址空间是 $0～2^{20}-1$。地址（编号）习惯用十六进制数表达，20 位数字信号对应二进制 20 位，可以

用 5 位十六进制数表达为 00000H～FFFFFH，见图 1-7。

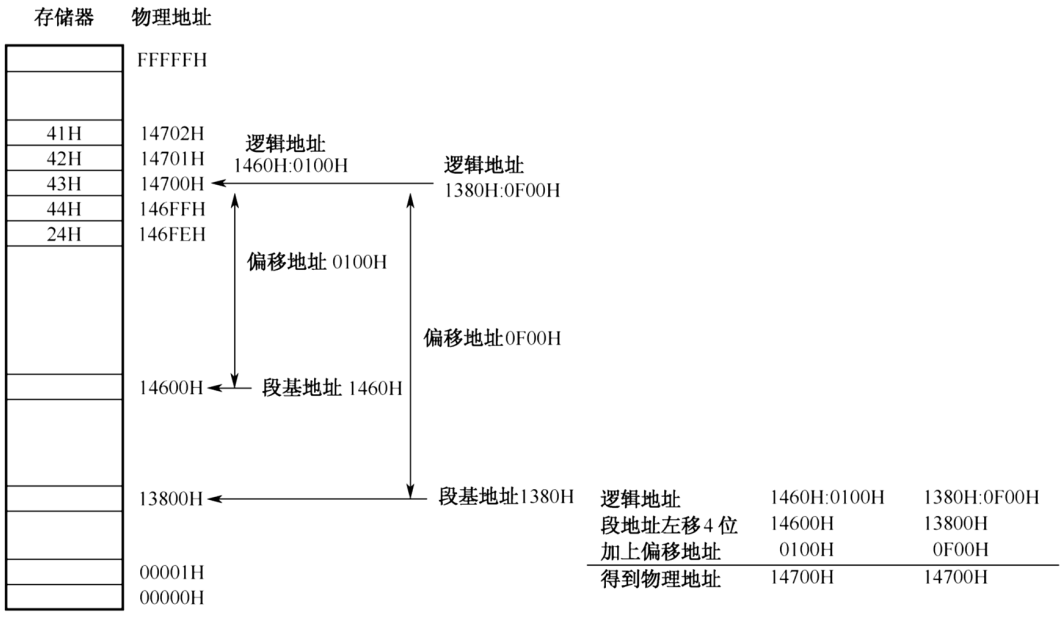


图 1-7　物理地址和逻辑地址

但是，8086 处理器只有 16 位寄存器，没有设计可以完整保存 20 位物理地址的寄存器。而使用二进制 16 位（对应十六进制 4 位）表示存储器地址，可以表达的存储器容量是 64KB（$=2^{16}$ 字节），范围是 0000H～FFFFH。

于是，8086 处理器将 1MB 物理存储器空间分成许多不超过 64KB 的区域进行管理，这种区域被称为（区）段（Segment）。段用于 8086 内部和程序设计中，所以常被称为逻辑段。

由于规定每段最大 64KB，段内的存储单元就可以使用 16 位地址表示。同时，8086 处理器还规定每个段只能起始于低 4 位（二进制）全为 0 的物理地址位置，这种地址是模 16 地址，也就是可以被 16 整除的地址（用十进制表达是：0、16、2×16、3×16……），用十六进制表达是：XXXX0H（X 表示十六进制一位，可以是 0～F 任何值）。既然段起始地址的低 4 位都是 0（对应十六进制是一位 0），可以省略不用表达，那这个 20 位物理地址就可以只表达出高 16 位。

这样，某个存储单元还可以通过它所在的（区）段及在该段中的位置指示，这就是在 8086 内部以及编程中使用的逻辑地址（Logical Address），它包括段基地址和偏移地址，都可以使用 16 位表示。段基地址（常简称段地址）确定该段在主存中的起始地址。以段基地址为起点，段内的位置可以用距离该起点的位移量表示，称为偏移地址（Offset）。

逻辑地址常借用 MASM 汇编程序的方法，使用英文冒号“:”分隔段基地址和偏移地址，形如“段基地址:偏移地址”。并且在 8086 中，段基地址常只表达高 16 位。例如，逻辑地址“1460H:0100H”表示段基地址是 1460H，即该存储单元所在的段起始于物理地址 14600H；该存储单元的偏移地址是 0100H，说明它位于距离段起始位置的 0100H 处，参见图 1-7 中间所示。

编程使用的逻辑地址由 8086 总线接口单元 BIU 的地址加法器电路转换为物理地址，然后通过处理器引脚输出外部访问物理存储器。进行转换时，两个 16 位组成的逻辑地址中的段地址需要左移 4 位（十六进制一位），加上偏移地址才能得到 20 位物理地址。例如，某存储单元的逻辑地址是“1460H:0100H”，其物理地址是 14700H，参见图 1-7 右边所示。

某个存储单元可以处于不同起点的逻辑段中（当然对应的偏移地址也就不同），所以可以有多个逻辑地址，但只有一个唯一的物理地址。或者说，同一个物理地址可以有多个逻辑地址。例如，物理地址 14700H 还可以用逻辑地址“1380H:0F00H”表示，该段起始于 13800H，参见图 1-7 所示。

3. 应用程序的基本段

8086 设计有 4 种类型的逻辑段实现存储器的分段管理，有 4 个段寄存器保存对应段的段基地址（注意，只是保存高 16 位），见表 1-4。

表 1-4　8086 的 4 种逻辑段

段名称	段的作用	段基地址	偏移地址
代码段	存放程序的指令序列	CS	IP
堆栈段	确定堆栈所在的主存区域	SS	SP
数据段	存放当前运行程序所用的数据	DS	EA
附加段	附加的数据段，也用于保存数据	ES	EA

分段管理存储器空间也符合程序的模块化思想，应用程序通常需要使用 3 种类型的段：代码段、堆栈段和数据段，8086 还设计了一个属于数据段类型的附加段。

程序的指令代码必须安排在代码段，否则将无法正常执行。程序利用代码段寄存器 CS 获得当前代码段的段基地址，指令指针寄存器 IP 保存代码段中指令的偏移地址。处理器利用 CS：IP 取得下一条要执行的指令。

程序使用的堆栈（临时存放数据的区域）一定在堆栈段。程序利用 SS 获得当前堆栈段的段基地址，堆栈指针寄存器 SP 保存堆栈栈顶的偏移地址。处理器利用 SS：SP 操作堆栈数据。

一个程序可以使用多个数据段，便于安全有效地访问不同类型的数据。例如，程序的主要数据存放在一个数据段，只读的数据存放在另一个数据段，动态分配的数据安排在第 3 个数据段。8086 规定程序当前访问的数据默认安排在数据段（应用中无须特别说明），也允许将数据安排在其他段（需要明确指明），数据的偏移地址由各种存储器寻址方式计算出有效地址 EA。

4. 存储器的分段管理

8086 规定段基地址低 4 位均为 0，每段最大不超过 64KB，但并没有要求每段必须是 64KB，各段之间可以完全分开，也可以部分重叠，甚至完全重叠。当然，各段的内容是不允许发生冲突的。图 1-8 说明了这种情况。

图 1-8（a）是各段独立的分配示例。CS＝0150H、DS＝4200H、SS＝1CD0H、ES＝B000H，它们分别为代码段、数据段、堆栈段和附加段的段地址。自每个段地址开始，各段均占 64KB 字节的范围，各段之间互不重叠。

图 1-8（b）则是相互重叠段的分配示例。CS＝0200H、DS＝0400H、SS＝0480H，这样代码段、数据段和堆栈段的物理起始地址分别为 02000H、04000H 和 04800H。其中代码段大小为 8KB，数据段占 2KB，而堆栈段只有 256 字节，SP＝0100H。由于该程序没有使用附加段，因此没有设置 ES 值。从这里可以看出，各段大小应根据实际需要来分配，可以重叠。有时，甚至可以将所有 4 种段都集中在一个逻辑段内，形成一个短小紧凑的程序，其大小不超过 64KB。例如，在图 1-8（b）所示的情况下，如果设置 CS＝DS＝SS＝0200H；这时，代码段将占据该逻辑段从偏移地址 0000～1FFFH 的 8KB，而数据段在偏移地址 2000H～27FFH 位置，堆栈顶指针 SP＝2900H。

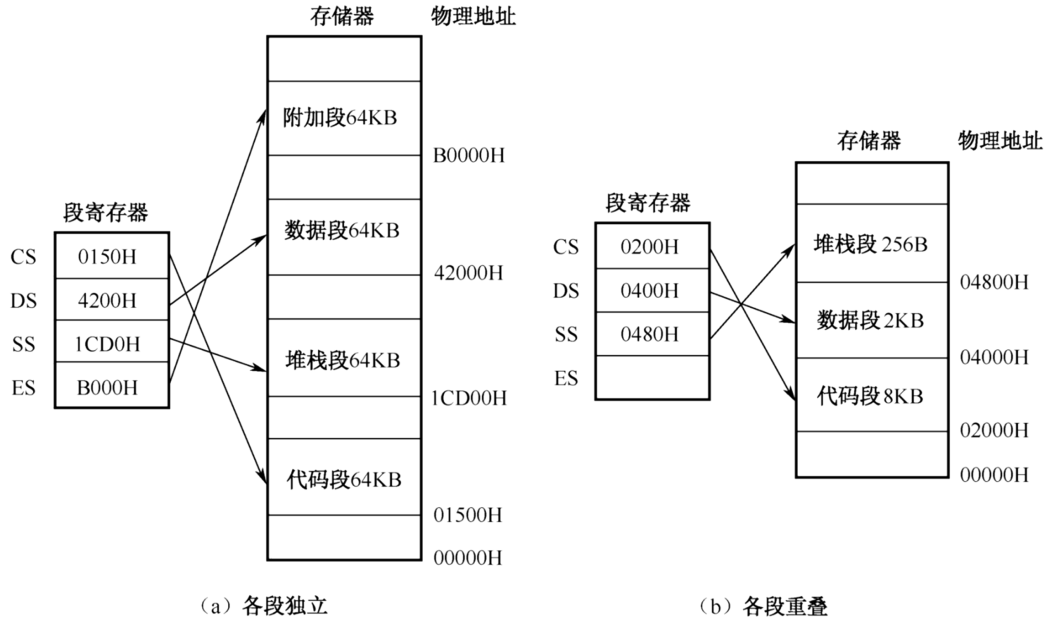


图 1-8　存储器的分段管理

第三节　汇编语言程序的格式

前文介绍基本的硬件基础，本节主要了解汇编语言的语句格式和源程序框架，编写第一个汇编语言程序。

一、指令代码格式

汇编语言程序的主体是处理器指令，了解指令的代码格式有助于理解指令功能。

指令的代码格式（Instruction Format）说明如何用二进制编码指令，也称机器代码（Machine Code）格式，它由操作码和地址码组成。指令的操作码（Opcode）表明处理器执行的操作，例如数据传送、加法运算、跳转等操作。操作数（Operand）是参与操作的数据，即各种操作的对象，主要以寄存器或存储器地址、I/O 地址形式指明数据的来源，所以也称为地址码，例如，数据传送指令的源地址和目的地址，加法指令的加数、被加数及和值都是操作数。有些指令不需要操作数，通常的指令都有一个或两个操作数，也有个别指令有 3 个甚至 4 个操作数。多数操作数需要显式指明，有些操作数隐含使用。

8086 机器代码的一般格式如图 1-9 所示。操作码占 1 或 2 字节，后面的各字节指明操作数。其中，“mod reg r/m”字段表明寻找操作数的方法，“位移量”字段给出相对基地址的偏移量，“立即数”字段给出操作数本身。

下面通过程序中使用最多的，也是指令系统中最基本的数据传送指令为例简单加以说明。

数据传送指令的助记符是 MOV（取自 Move），功能是将数据从一个位置传送到另一个位置，类

似高级语言的赋值语句，表达如下：

```
mov dest,src                    ;dest←src
```

src 表示要被传送的数据或数据所在的位置，称为源操作数（Source），书写在逗号之后。DEST 表示数据将要传送到的位置，称为目的操作数（Destination），书写在逗号之前。注意，后面分号是汇编语言使用的注释符号，表示其后内容是注释，可用于说明指令功能。

图 1-9　8086 机器代码的一般格式

例如，将 16 位寄存器 BX 的数据传送到 AX 寄存器的指令，可以写为：

```
mov ax,bx                       ;指令功能:AX←BX,指令代码:89 D8(十六进制表示)
```

指令中，第 1 个字节“89”是操作码，表示传送 16 位数据的 MOV 指令；第 2 个字节“D8”表示源操作数 BX，目的操作数 AX。

再如，将寄存器 BX 内容加寄存器 SI 内容，再加 6 的值作为存储器地址，从该地址单元传送一个字节数据给 AL 寄存器，可以写为：

```
mov al,[bx+si+6]                ; 指令功能：AL←[BX+SI+6]
                                ; 机器代码：BA 40 06(十六进制表示)
```

其中，第 1 个操作码字节“8A”，表示传送 8 位数据的 MOV 指令；第 2 个字节“40”表示目的操作数是 AL、源操作数是通过寄存器 BX 和 SI 以及 8 位位移量相加得到存储器偏移地址；第 3 个字节“06”正是这个位移量。

二、语句格式

像其他程序设计语言一样，汇编语言对其语句格式、程序结构及开发过程等有相应的要求，它们本质上相同、方法上相似，具体内容各有特色。

汇编语言程序由语句序列构成，每条语句一般占一行，每行不超过 132 个字符（从 MASM 6.0 开始可以是 512 个字符）。汇编语言的语句一般都由分隔符分成的 4 个部分组成，有相似的两种格式，对应表达处理器指令和汇编程序伪指令。

（1）执行性语句——表达处理器指令的语句

标号：处理器指令助记符 操作数，操作数 ；注释

（2）说明性语句——表达汇编程序命令的语句

名字伪指令助记符 参数，参数，……；注释

一条执行性语句通常对应一条处理器指令，一般书写在代码段，是程序设计的主体部分。而说明性语句指示源程序如何汇编、变量怎样定义、过程怎么设置等。相对于真正的处理器指令（也称

为真指令、硬指令），汇编程序命令也称为伪指令（Pseudoinstruction）、指示符（Directive）。

1. 标号与名字

标号与名字

执行性语句中，冒号前的标号表示处理器指令在主存中的逻辑地址，主要用于指示分支、循环等程序的目的地址，可有可无。说明性语句中的名字可以是变量名、段名、子程序名等，反映变量、段和子程序等的逻辑地址。标号采用冒号分隔处理器指令，名字采用空格或制表符分隔伪指令，由此也分为两种语句。

标号和名字是符合汇编程序语法的用户自定义的标识符（Identifier）。标识符（也称符号 Symbol）一般最多由 31 个字母、数字及规定的特殊符号（如 _ 、$ 、?、@）组成，不能以数字开头（与高级程序语言一样）。在一个源程序中，用户定义的每个标识符必须是唯一的，且不能是汇编程序采用的保留字。保留字（Reserved Word）是编程语言本身需要使用的各种具有特定含义的标识符，也称为关键字（Key Word）。汇编程序中主要有处理器指令助记符、伪指令助记符、操作符、寄存器名以及预定义符号等。

例如，msg、var2、buf、next、again 都是合法的用户自定义标识符。而 8var、ax、mov、db 则是不符合语法（非法）的标识符，原因是 8var 以数字开头，其他是保留字。ax 是寄存器名，mov 是指令助记符，db 是伪指令助记符。

默认情况下，汇编程序不区分包括保留字在内的标识符字母大小写。换句话说，汇编语言是大小写不敏感的。例如，对于寄存器名 AX，还可以书写成 ax、Ax 等。msg 变量名，还可以 Msg、MSG 等形式出现，它们表达同一个变量。处理的原则是：程序中一般使用小写字母形式，功能注释、文字说明通常采用大写字母形式。

用户自定义标识符时，应尽量具有描述性并易于理解，一般不建议使用特殊符号开头，因为特殊符号没有含义，而且常被编译（汇编）程序所使用，例如 C 语言编译程序在内部为函数增加“ _ ”前缀，MASM 大量使用“@”作为预定义符号的前缀。如果不确信标识符是否可用，就不要使用。一个简单的规则是：以字母开头，后跟字母或数字。

2. 助记符

助记符（Mnemonics）是帮助记忆指令的符号，反映指令的功能。

处理器指令助记符可以是任何一条处理器指令，表示一种处理器操作。同一系列处理器的指令常会增加，不同系列处理器的指令系统不尽相同。例如，前面介绍的数据传送指令，其助记符是“MOV”，调用中断服务程序的指令助记符是“INT”（Interrupt），调用子程序的指令助记符是“CALL”。

将数字 9 传送到 AH 寄存器的汇编语言执行性语句写为：

```
mov ah,9                                ;使得 AH=9
```

伪指令助记符由汇编程序定义，表达一个汇编过程中的命令，随着汇编程序版本增加，伪指令会增加，功能也会增强。例如，汇编语言程序中频繁使用的字节变量定义伪指令，其助记符是“DB”（Define Byte），功能是在主存中分配若干的存储空间，用于保存变量值，该变量以字节为单位存取。

用 DB 伪指令定义一个字符串，并取名字 MSG 指示，这个汇编语言说明性语句写为：

```
msg                                     db'Hello, Assembly !', 13, 10,'$'
```

名字 MSG 指示这个字符串在主存的逻辑地址，包含有段基地址和偏移地址，也就是这个字符串

的变量名。可以用一个 MASM 操作符 OFFSET 获得其偏移地址，保存到 DX 寄存器，汇编语言执行性语句如下：

```
mov dx, offset msg                    ;DX 获得 MSG 的偏移地址
```

MASM 操作符（Operator）是对常量、变量、地址等进行操作的关键字。例如，进行加减乘除运算的操作符（也称运算符）与高级程序语言一样，依次是英文符号＋、－、＊和/。

3. 操作数和参数

处理器指令的操作数表示参与操作的对象，可以是一个具体的常量，也可以是保存在寄存器的数据，还可以是一个保存在存储器中的变量等。双操作数的指令中，目的操作数写在逗号前，还可用来存放指令操作的结果；对应地，逗号后的操作数就称为源操作数。

例如，指令“MOV AH，9”中数字 9 是常量形式的源操作数，AH 是寄存器形式的目的操作数。同样，OFFSET MSG 经汇编后转换为一个具体的偏移地址，也是常量。

伪指令的参数可以是常量、变量名、表达式等，可以有多个，参数之间用逗号分隔。例如在“'Hello，Assembly! '，13，10，'$'”示例中，就用单引号表达了一个字符串“Hello，Assembly!”，一个字符“$”，还有常量 13 和 10（这两个常量在 ASCII 码表中表示回车和换行控制字符，其作用相当于 C 语言的“\n”）。

4. 注释和分隔符

汇编语言语句中，分号后的内容是注释，它通常是对指令或程序片段功能的说明，是为了程序便于阅读而加上的，不是必须有的。必要时，一个语句行也可以由分号开始作为阶段性注释。汇编程序在翻译源程序时将跳过该部分，不对它们做任何处理。

在此建议读者养成书写注释的良好习惯。

汇编语言语句的 4 个组成部分要用分隔符分开。标号后的冒号、注释前的分号以及操作数间和参数间的逗号都是按规定采用的分隔符，其他部分通常采用空格或制表符作为分隔符。多个空格和制表符的作用与一个相同。另外，MASM 也支持续行符“\”，表示本行内容与上一行内容属于同一个语句。注释可以使用英文书写。在支持汉字的编辑环境当然也可以使用汉字进行程序注释，但注意这些分隔符都必须使用英文标点，否则无法通过汇编。

良好的语句格式有利于编程，尤其是源程序阅读。在汇编语言源程序中，标号和名字从首列开始书写；通过制表符对齐各个语句行的助记符；助记符之后用空格分隔操作数和参数部分（多个操作数和参数，按照语法要求使用逗号分隔）；再利用制表符对齐注释部分。

三、源程序框架

对应存储空间的分段管理，用汇编语言编程时也常将源程序分成代码段、数据段或堆栈段。需要独立运行的程序必须包含一个代码段，并指示程序执行的起始位置。需要执行的可执行性语句必须位于某一个代码段内。说明性语句通常安排在数据段，或根据需要位于其他段。

随着 MS-DOS 和 MASM 的发展，MASM 各版本支持多种汇编语言源程序格式，本书在此介绍一个最简单的程序格式，使用 MASM 6.X 版本的简化段定义源程序格式，程序模板如下：

```
;example .asm in DOS
        .model small                  ;定义程序的存储模型(small 表示小型模型)
```

```
.stack                  ;定义堆栈段(默认是 1KB 空间)
.data                   ;定义数据段
......                  ;数据定义(数据待填)
.code                   ;定义代码段
.startup                ;程序执行起始,同时设置数据段寄存器 DS 指向程序的数据段
......                  ;主程序(指令待填)
.exit                   ;程序执行结束,返回 DOS
......                  ;子程序(指令待填)
end                     ;汇编结束
```

在简化段定义（Simplied Segment Definition）的源程序格式中，以圆点（英文小数点、句号）开始的伪指令说明程序的结构，必须具有存储模式伪指令 .MODEL。随后，.STACK、.DATA 和 .CODE 依次定义堆栈段、数据段和代码段，一个段的开始自动结束上一个段。代码段中，通过 .STARTUP 语句说明程序从该处开始执行，并含有给 DS 赋值、使其指向该程序的数据段功能，便于后续指令访问数据段中的数据。程序最后利用 .EXIT 指令说明本程序执行结束、返回 DOS 操作系统。

1. 程序的存储模型

存储模型（Memory Model）决定一个程序的规模，也确定进行子程序调用、指令转移和数据访问的默认属性。当使用简化段定义的源程序格式时，必须有存储模型 .MODEL 语句，且位于所有简化段定义语句之前。其格式为

```
.model 存储模型,语言类型
```

.MODEL 语句确定了程序采用的存储模型，MASM 有 7 种可以选择，见表 1-5。

表 1-5　存储模型

存储模型	特点
TINY（微型模型）	创建 COM 类型程序，只有一个小于 64KB 的逻辑段（MASM6.X 支持）
SMALL（小型模型）	创建小应用程序，只有一个代码段和一个数据段，每个段不大于 64KB
COMPACT（紧凑模型）	创建代码少、数据多的程序，只有一个代码段（不大于 64KB），但可有多个数据段
MEDIUM（中型模型）	创建代码多、数据少的程序，可有多个代码段，但只有一个数据段（不大于 64KB）
LARGE（大型模型）	创建大应用程序，可有多个代码段和多个数据段（静态数据小于 64KB）
HUGE（巨型模型）	创建更大的应用程序，可有多个代码段和数据段，对静态数据没有限制
FLAT（平展模型）	创建一个 32 位的程序，运行在 IA-32 微处理器的 32 位 Windows 操作系统

创建运行于 DOS 操作系统下的应用程序，可根据需要选择前 6 种模型，一般的小型程序（例如学习中的小程序）可以选用 SMALL 模型，大型程序选择 LARGE 模型。要创建 COM 程序只能用 TINY 模型，其他模型产生 EXE 程序。FLAT 模型只能用于 32 位 Windows 应用程序中，不能在 DOS 环境执行。

DOS 环境的 COM 类型程序要求将程序的代码、数据和堆栈都安排在一个逻辑段中，大小不超过 64KB，是一种比较紧凑的程序格式。在上述程序模板中，只要将 .STACK 和 .DATA 语句去掉，

并将数据定义填到子程序之后（END之前），就形成了一个COM类型程序的模板文件。

语言类型主要用于与其他语言混合编程时约定调用的规范，例如Windows的系统函数采用标准调用语言类型“STDCALL”。MASM汇编程序还支持C语言调用规范，其关键字是“C”。编写DOS应用程序通常不需要。

2. 逻辑段的简化定义

堆栈段定义伪指令.STACK创建一个堆栈段，段名是STACK。保留字后可书写一个数值型参数指定堆栈段所占存储空间的字节数，默认是1KB（1024B=400H字节）。堆栈段名可用@STACK预定义操作符表示。

数据段定义伪指令.DATA创建一个数据段，段名是_DATA。数据段名可用@DATA预定义操作符表示。

代码段定义伪指令.CODE创建一个代码段，后可选一个标识符型参数指定该代码段的段名。如果没有给出段名，则采用默认段名，例如在TINY、SMALL、COMPACT和FLAT模式下，默认的代码段名是_TEXT。代码段名可用@CODE预定义操作符表示。

3. 程序执行的开始

MASM 6.0引入的.STARTUP指令指明了本程序开始执行的位置，并同时使数据段寄存器DS等于用.DATA伪指令定义的数据段的段基地址。

MASM汇编程序在对源程序的汇编、连接过程中，会根据程序起始位置正确地设置代码段的CS和IP值，根据堆栈大小设置堆栈段的SS和SP值，但并不设置DS和ES值。这样，程序如果使用数据段，就必须在代码段中明确给DS等寄存器赋值。由于大多数程序需要在数据段定义变量，因此通常应该赋值给DS；使用到附加段就一定设置ES。

如果不使用.STARTUP指令，通常可以用如下两条语句代替：

```
start:  mov ax, @data          @DATA表示数据段的段地址,先传送给AX寄存器
        mov ds,ax              ;设置DS等于AX,即数据段的段地址
```

其中标号start也可以使用其他标识符，此处需要利用它在最后的汇编结束END指令作为参数，用于指明程序开始执行的位置。

4. 程序执行的终止

应用程序执行结束，应该将控制权交还给操作系统，.EXIT指令就实现了此功能。它实际上是利用了DOS功能调用的4CH号功能实现的，所以可以用如下两条语句代替：

```
mov ah,4ch
int 21h
```

DOS功能调用是MS-DOS操作系统提供给程序员的一些子程序库，主要以第21H号中断的形式使用（指令“INT 21H”）。

5. 源程序的汇编结束

汇编结束表示汇编程序到此结束将源程序翻译成目标模块代码的过程，而非指程序终止执行。源程序的最后必须有一条END伪指令，END指令之后的任何内容将不会被汇编程序所理会。

END伪指令后面可以有一个“标号”性质的参数，用于指定程序开始执行于该标号所指示的指令。汇编程序将据此设置CS和IP值。如果没有.STARTUP指令说明程序开始执行的位置，就需要

利用这种方法指明。例如，对应前面我们使用的 START 标号，这时就需要用如下语句代替原“END”语句：

```
end start
```

用上述程序格式，编写一个在屏幕上显示信息的程序。

【例 1-1】　信息显示程序。

在数据段给出这个字符串形式的信息，采用字节定义伪指令 DB 实现：

```
        ;数据段
msg     db 'Hello,assembly !',13,10,'$'        ;定义要显示的字符串
```

在代码段编写显示字符串的程序：

```
;代码段
mov dx,offset masg                             ;(1)指定字符串在数据段的偏移地址
mov ah,9                                       ;(2)AH 赋值 9
int 21h                                        ;(3)利用功能调用显示信息
```

这里使用了 DOS 的 9 号功能实现字符串的显示，其要求如下。

(1) 设置入口参数：DS：DX＝字符串在主存逻辑段中的段地址：偏移地址。同时还要求保存在主存的字符串以“$”作为结尾符（类似 C/C＋＋语言中隐含用 NULL 作为字符串结尾）。

(2) 需要将 AH 赋值 9，即使用 9 号 DOS 功能。

(3) 使用指令“INT 21H”实现 DOS 功能的调用。

由于将字符串安排在了数据段，而 .STARTUP 指令已经将 DS 赋值为数据段基地址，因此程序中只要将字符串的偏移地址传送给 DX 就完成入口参数的设置，执行 DOS 功能调用就可以实现输出了。

将例题程序填入程序模板（EXAMPLE. ASM）预留的位置，即将数据段内容填入数据段定义指令 .DATA 之后，将代码段内容填入程序开始执行 .STARTUP 指令之后，就编制了一个完整的 MASM 汇编语言源程序。

```
        ;eg101.asm(文件名)
        .model small
        .stack
        .data
msg     db 'hello, Assembly !',13,10,'$'       ;定义要显示的字符串
        .code
        .startup
        mov dx,offset msg                      ;(1)指定字符串在数据段的偏移地址
        mov ah,9                               ;(2)AH 赋值 9
        int 21h                                ;(3)利用功能调用显示信息
        .exit
        end
```

读者可以基于模板文件的源程序框架创建完整的源程序文件。

现在，对绝大多数读者来说，都是从高级语言开始熟悉计算机程序设计的。虽然汇编语言不是高级语言，但它们都是程序设计语言，有许多本质上相同或相通的方面。所以，学习过程中通过对比，既可以巩固高级语言的知识，也有利于熟悉汇编语言，通过汇编语言还可以进一步加深对高级语言的理解。

6. DOS 功能调用

汇编语言怎么总是使用操作系统的功能调用呢？使用一种编程语言进行程序设计，程序员需要利用其开发环境提供的各种功能，例如函数、程序库。如果这些功能无法满足程序员的要求，还可以直接利用操作系统提供的程序库，否则只有自己编写特定的程序。汇编语言作为一种低级程序设计语言，汇编程序通常并没有为其提供任何函数或程序库，所以必须利用操作系统的编程资源。显然，这是进行程序设计，尤其是采用汇编语言进行程序设计必须掌握的一个重要方面。DOS 提供给程序员的编程资源是以中断调用方法使用的各种子程序，Windows 则以应用程序接口 API 形式提供动态链接库 DLL。

中断是一种增强处理器功能的机制，中断调用是借助中断机制改变程序执行顺序的方法，类似于汇编语言的子程序调用（对应高级语言的函数调用）。8086 处理器支持 256 个中断，每个中断用一个中断编号区别，即中断 0～中断 255。中断调用指令“INT N”实现调用 N 号中断服务程序的功能。DOS 系统中，主要分配 21H 号中断用于程序员调用 DOS 操作系统功能。

调用 DOS 操作系统功能的一般方法如下。

（1）在 AH 寄存器中设置系统功能调用号，说明选择的功能。

（2）在指定寄存器中设置入口参数，以便按照要求执行功能。

（3）用中断调用指令“INT 21H”执行功能调用。

（4）根据出口参数分析功能调用执行情况。

实际上，这就是调用子程序的一般步骤（类似高级语言调用函数）。根据功能不同，有些没有入口参数或出口参数，有些入口参数或出口参数可能较复杂，并且可能有一些特殊要求。

常用 DOS 基本功能调用见表 1-6。

表 1-6　DOS 基本功能调用（INT 21H）

子功能号	功 能	入口参数	出口参数
AH＝01H	从标准输入设备输入一个字符	—	AL＝输入字符的 ASCII 码
AH＝02H	向标准输出设备输出一个字符	DL＝字符的 ASCII 码	—
AH＝09H	向标准输出设备输出一个字符串	DX＝字符串地址（以 $ 结尾）	—
AH＝4CH	程序执行终止	AL＝返回代码	—

例如 4CH 号功能是返回 DOS，它可以有一个入口参数，就是使 AL 等于程序的返回代码（通常用 0 表示程序没有错误）。这样，正常返回 DOS 可以增加一条指令“MOV AL，0”，或者用“MOV AX，4C00H”代替“MOV AH，4CH”指令。这时，它相当于“.EXIT 0”语句。

习　题

1. 说明计算机系统的硬件组成及各部分作用。
2. 什么是标志？说出 8086 状态标志的名称和符号。
3. 将以下 8086 的逻辑地址用其物理地址表示（均为十六进制形式）：

（1）FFFF：0　（2）40：17　（3）2000：4500　（4）B821：4567

4. 应用程序中主要有哪 3 类基本段，各有什么用途？
5. 说明高级语言、汇编语言、机器语言三者的区别，并谈谈你对汇编语言的认识。
6. 区别以下概念：助记符、汇编语言、汇编语言程序和汇编程序。

第二章

数据表示和寻址

学习目标

1. 了解汇编语言如何使用常量表达数据。
2. 掌握汇编语言如何使用变量保存数据。
3. 掌握处理器指令如何寻址数据。

数据(Data)是计算机处理的对象，处理器指令操作的对象也称操作数(Operand)。计算机中的数据需要使用二进制的 0 和 1 组合表示，程序设计语言中使用常量和变量形式来表示和定义，处理器指令则以寻址方式访问数据进行处理。

第一节 数据表示

计算机只能识别 0 和 1 两个数码，进入计算机的任何信息都要转换成 0 和 1 数码。整数指令支持的基本数据类型是 8、16、32、64 位无符号整数和有符号整数，也支持字符、字符串和 BCD 码操作。本节主要介绍这些数据类型的数据表示。

一、数制

人有 10 根手指，所以习惯了十进制计数。计算机的硬件基础是数字电路，它处理具有低电平和高电平两种稳定状态的电平信号，所以使用了二进制。为了便于表达二进制数，人们又常用到十六进制数。

1. 二进制

计算机中为便于存储及物理实现，采用二进制表达数值。二进制数的特点为：逢 2 进 1，由 0 和 1 两个数码组成，基数为 2，各个位权以 2^k 表示。

二进制数：$a_n a_{n-1}\cdots a_1 a_0 \cdot b_1 b_2 \cdots b_m =$

$a_n \times 2^n + a_{n-1} \times 2^{n-1} + \cdots + a_1 \times 2^1 + a_0 \times 2^0 + b_1 \times 2^{-1} + b_2 \times 2^{-2} + \cdots + b_m \times 2^{-m}$，其中 a_i、b_j 非 0 即 1。

二进制数的算术运算类似于十进制，只不过是逢 2 进 1、借 1 当 2，表 2-1 是二进制运算规则。图 2-1 采用 4 位二进制数，示例了二进制数的加减乘除运算，注意加减法会出现进位或借位，乘积和被除数是双倍长的数据，除法有商和余数两部分。

表 2-1 二进制运算规则

加法运算	减法运算	乘法运算
1+0=1	1−0=1	1×0 =0
1+1=0（进位 1）	1−1=0	1×1 =1
0+0=0	0−0=0	0×0=0
0+1=1	0−1=1（借位 1）	0×1=0

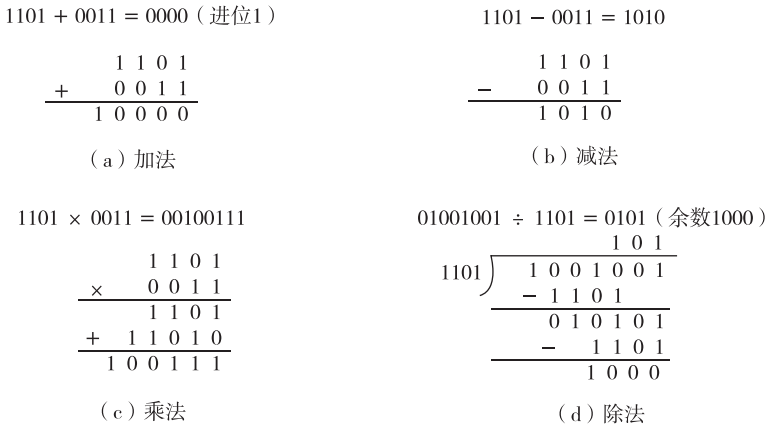


图 2-1　二进制数的算术运算

2. 十六进制

由于二进制数书写较长、难以辨认，因此常用易于与之转换的十六进制数来描述二进制数。十六进制数的基数是 16，共有 16 个数码：0、1、2、3、4、5、6、7、8、9 和 A、B、C、D、E、F(也可以使用小写字母 a～f，依次表示十进制的 10～15)，逢 16 进位，各个位的位权为 16^k。

十六进制数：$a_n a_{n-1}\cdots a_1 a_0 \cdot b_1 b_2 \cdots b_m =$

$a_n \times 16^n + a_{n-1} \times 16^{n-1} + \cdots + a_1 \times 16^1 + a_0 \times 16^0 + b_1 \times 16^{-1} + b_2 \times 16^{-2} + \cdots + b_m \times 16^{-m}$，其中 a_i、b_j 为 0～9 及 A～F 中的一个数码。

十六进制数的加减运算也类似十进制，但注意逢 16 进位 1，借 1 当 16。

例如：

23D9H＋94BEH＝B987H，A59FH－62B8H＝42E7H

这里的后缀字母 H(或小写 h)表示十六进制形式表达的数据。涉及计算机学科知识的文献中，常使用十六进制数表达地址、数据、指令代码等，所以应该熟悉十六进制数的加减运算。

3. 数制之间的转换

(1)二进制数、十六进制数转换为十进制数需按权展开。

例如：

0011.1010B＝$1\times2^1+1\times2^0+1\times2^{-1}+0\times2^{-2}+1\times2^{-3}$＝3.625

1.2H＝$1\times16^0+2\times16^{-1}$＝1.125

这里的后缀字母 B(或小写 b)表示二进制形式表达的数据。

(2)十进制数的整数部分转换为二进制和十六进制数可用除法，把要转换的十进制数的整数部分不断除以二进制数和十六进制数的基数 2 或 16，并记下余数，直到商为 0 为止。由最后一个余数起逆向取各个余数，则为该十进制数整数部分转换成的二进制数和十六进制数。

如图 2-2 所示为转换 126 的过程，结果是：126＝01111110B，126＝7EH。

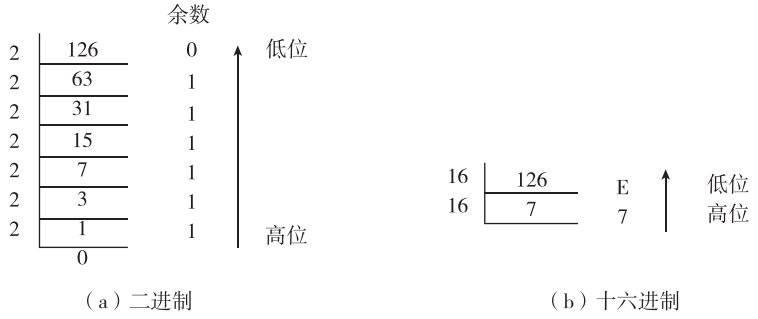


图 2-2　十进制整数的转换

(3)十进制数的小数部分转换为二进制数和十六进制数则分别乘以各自的基数，记录整数部分，直到小数部分为 0 为止。

如图 2-3 所示为转换 0.8125 的过程，结果是：0.8125＝0.1101B，0.8125＝0.DH。

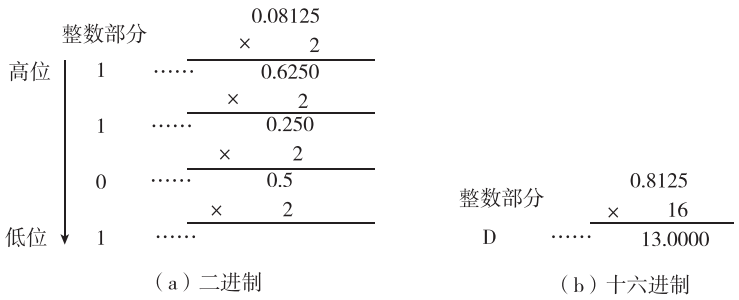


图 2-3　十进制小数的转换

小数部分的转换会发生总是无法乘到为 0 的情况，这时可选取一定位数(精度)，当然这将产生无法避免的转换误差。

(4)二进制数和十六进制数之间具有对应关系：以小数点为基准，整数从右向左(从低位到高位)、小数从左向右(从高位到低位)每 4 个二进制位对应一个十六进制位(表 2-2)，所以相互转换非常简单。表 2-2 还给出了 BCD 码以及常用的二进制位权值。

表 2-2　不同进制间(含 BCD 码)的对应关系

十进制	二进制	十六进制	BCD 码
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	8	8
9	1001	9	9
10	1010	A	—
11	1011	B	—
12	1100	C	—
13	1101	D	—
14	1110	E	—
15	1111	F	—

常用二进制位权
$2^{-3}=0.125$
$2^{-2}=0.25$
$2^{-1}=0.5$
$2^{0}=1$
$2^{1}=2$
$2^{2}=4$
$2^{3}=8$
$2^{4}=16$
$2^{5}=32$
$2^{6}=64$
$2^{7}=128$
$2^{8}=256$
$2^{9}=512$
$2^{10}=1024$
$2^{15}=32768$
$2^{16}=65536$

例如：

00111010B＝3AH，F2H＝11110010B

二、数值的编码

编码是用文字、符号或者数码来表示某种信息(数值、语言、操作指令、状态等)的过程。组合 0 和 1 数码就是二进制编码。用 0 和 1 数码的组合在计算机中表达的数值被称为机器数；对应地，现实中真实的数值被称为真值。对数值来说，主要有两种编码方式：定点格式和浮点格式。

1. 定点整数

定点格式固定小数点的位置表达数值，计算机中通常将数值表达成纯整数或纯小数，这种机器数称为定点数。整数可以将小数点固定在机器数的最右侧，实际上并不用表达出来，这就是整数处理器支持的定点整数，如图 2-4 所示。如果将小数点固定在机器数的最左侧就是定点小数。

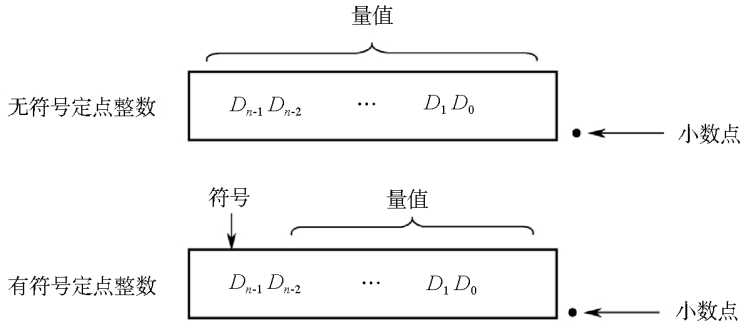


图 2-4　定点整数格式

定点整数如果不考虑正负，只表达 0 和正整数，就是无符号整数(简称无符号数)。在上面的数值转换和运算中，就默认采用无符号整数。8 位二进制有 256 个编码，依次是：00000000、00000001、00000010、……、11111110、11111111，使用十六进制形式是：00、01、02、……、FE、FF，对应表达无符号整数真值为：0、1、2、……、254、255。N 位二进制共有 2^N 个编码，表达真值：0～2^N-1。所以，16 位和 32 位二进制所能表示的无符号整数范围分别是 0～$2^{16}-1$ 和 0～$2^{32}-1$。

如果要表达数值正负，需要占用一个位，通常用机器数的最高位(故称为符号位)，用 0 表示正数、1 表示负数，这就是有符号整数(简称有符号数或带符号数)。

2. 补码

有符号整数有多种表达形式，计算机中默认采用补码。因为采用补码，减法运算可以变换成加法运算，硬件电路只需设计加法器。

补码中最高位表示符号：正数用 0，负数用 1；正数补码同无符号数，直接表示数值大小；负数补码是将对应正数补码取反(将 0 变为 1，1 变为 0)，然后加 1 形成。

例如，正整数 105，用 8 位补码表示：

$[105]_{补码}=01101001B$

负整数－105，用 8 位补码表示：

$[-105]_{补码}=[01101001B]_{取反}+1=10010110B+1=10010111B$

一个负数真值在用机器数补码表示时，需要一个“取反加 1”的过程。同样，将一个最高位为 1 的补码(真值为负数)转换成真值时，也需要一个“取反加 1”的过程。

例如，补码：11100000B，真值：$-([11100000]_{取反}+1)=-(00011111+1)=-00100000=-2^5=-32$。

进行负数求补运算，在数学上等效于用带借位的 0 作减法(下面等式中用中括号表达借位)。

例如：

真值：－8，补码：$[-8]_{补码}=[1]0-8=[1]00000000-00001000=11111000$

补码：11111000，真值：－([1]00000000－11111000)＝－00001000＝－8

注意，求补只针对负数进行，正数不需要求补。另外，十六进制更便于表达，上述运算过程可以直接使用十六进制数。

由于符号要占用一个数位，8 位二进制补码中只有 7 个数位表达数值，其所能表示的数值范围是：－128～－1、0～＋127，对应补码是：10000000～11111111、000000000～011111111，若用十六进制表达是：80～FF、00～7F。16 位和 32 位二进制补码所能表示的数值范围分别是：-2^{15}～$+2^{15}-1$ 和 -2^{31}～$+2^{31}-1$。用 N 位二进制编码有符号整数，仍共有 2^N 个编码，但表达的真值范围是：-2^N-1～$+2^N-1-1$。

使用补码表达有符号整数，和无符号整数表达的数值个数一样，但范围不同。

三、字符的编码

在计算机中，各种字符需要用若干位的二进制码的组合表示，即字符的二进制编码。由于字节为计算机的基本存储单位，因此常用 8 个二进制位为单位表达字符。

1. BCD 码

一个十进制数位在计算机中用 4 位二进制编码来表示，这就是所谓的二进制编码的十进制数(Binary Coded Decimal，BCD)。常用的 BCD 码是 8421 BCD 码，它用 4 位二进制编码的低 10 个编码表示 0～9 这 10 个数字。

BCD 码很容易实现与十进制真值之间的转换。例如：

BCD 码：0100 1001 0111 1000.0001 0100 1001，十进制真值：4978.149

如果将二进制 8 位即字节的高 4 位设置为 0，仅用低 4 位表达一个 BCD 码，被称为非压缩(Unpacked)BCD 码；而通常用 1 字节表达两位 BCD 码，就被称为压缩(Packed)BCD 码。

BCD 码虽然浪费了 6 个编码，但是能够比较直观地表达十进制数，也容易与 ASCII 码相互转换、便于输入输出。另外，它还可以比较精确地表达数据。例如，对于一个简单的数据 0.2，采用浮点格式就无法精确表达。而采用 BCD 码便可以只使用 4 位“0010”来表达。最初的计算机支持十进制运算，8086 处理器中使用调整指令实现十进制运算。

2. ASCII 码

ASCII 码

字母和各种字符也必须按特定的规则用二进制编码才能在计算机中表示。编码方式可以有多种，其中最常用的一种编码是 ASCII 码。现在使用的 ASCII 码源于 20 世纪 50 年代，完成于 1967 年，由美国标准化组织 ANSI 定义在 ANSI X3.4—1986 中。

标准 ASCII 码用 7 位二进制编码，故有 128 个，见表 2-3。计算机存储单位为 8 位，表达 ASCII 码时最高 D7 位通常为 0；通信时，D7 位通常用作奇偶校验位。

表 2-3　标准 ASCII 码及其字符

ASCII	字符	ASCII	字符	ASCII	字符	ASCII	字符
00H	NUL	20H	SP	40H	@	60H	、
01H	SOH	21H	!	41H	A	61H	a
02H	STX	22H	”	42H	B	62H	b
03H	ETX	23H	#	43H	C	63H	c
04H	EOT	24H	$	44H	D	64H	d
05H	ENQ	25H	%	45H	E	65H	e

续表

ASCII	字符	ASCII	字符	ASCII	字符	ASCII	字符
06H	ACK	26H	&	46H	F	66H	f
07H	BEL	27H	'	47H	G	67H	f
08H	BS	28H	(	48H	H	68H	h
09H	HT	29H	)	49H	I	69H	i
0AH	LF	2AH	*	4AH	J	64H	j
0BH	VT	2BH	+	4BH	K	6BH	k
0CH	FF	2CH	,	4CH	L	6CH	l
0DH	CR	2DH	-	4DH	M	6DH	m
0EH	SO	2EH	.	4EH	N	6EH	n
0FH	SI	2FH	/	4FH	O	6FH	o
10H	DLE	30H	0	50H	P	70H	p
11H	DC1	31H	1	51H	Q	71H	q
12H	DC2	32H	2	52H	R	72H	r
13H	DC3	33H	3	53H	S	73H	s
14H	DC4	341H	4	54H	T	74H	t
15H	NAK	35H	5	55H	U	75H	u
16H	SYN	36H	6	56H	V	76H	v
17H	ETB	37H	7	57H	W	77H	w
18H	CAN	38H	8	58H	X	78H	x
19H	EM	39H	9	59H	Y	79H	y
1AH	SUB	3AH	:	5AH	Z	7AH	z
1BH	ESC	3BH	;	5BH	[	7BH	{
1CH	FS	3CH	<	5CH	\	7CH	|
1DH	GS	3DH	=	5DH	]	7DH	}
1EH	RS	3EH	>	5EH	^	7EH	~
1FH	US	3FH	?	5FH	-	7FH	Del

ASCII 码表中的前 32 个和最后一个编码是不可显示的控制字符，用于表示某种操作。并不是所有设备都支持这些控制字符，也不是所有设备都按照同样的功能应用这些控制字符。不过，有些控制字符使用非常广泛，例如：0DH 表示回车 CR(Carriage Return)，控制屏幕光标时就是使光标回到本行首位；0AH 表示换行 LF(Line Feed)，就是使光标进入下一行，但列位置不变；08H 实现退格 BS(Backspace)，7FH 实现删除 DEL(Delete)。另外，07H 表示响铃 BEL(Bell)，1BH(ESC)常对应键盘的 ESC 键(多数人称其为 Escape 键)。ESC(Extra Services Control)字符常与其他字符一起发送给外设(例如打印机)，用于启动一种特殊功能，很多程序中常使用它表示退出操作。

ASCII 码表中从 20H 开始(含 20H)的 95 个编码是可显示和打印的字符，其中包括数码、英文字母、标点符号等。从表中可看到，数码 0～9 的 ASCII 码为 30H～39H，去掉高 4 位(或者说减去

30H)就是 BCD 码。A～Z 的 ASCII 码为 41H～5AH，而 a～z 则是 61H～7AH。大写字母和对应的小写字母相差 20H(32)，所以大小写字母很容易相互转换。ASCII 码中 20H 表示空格。尽管它显示空白，但要占据一个字符的位置；它也是一个字符，表中用 SP(Space)表示。熟悉这些字符的 ASCII 码规律对解决一些应用问题很有帮助，例如英文字符就是按照其 ASCII 码大小进行排序的。

处理器只是按照二进制数操作字符编码，并不区别可显示(打印)字符和非显示(控制)字符，只有外部设备才区别对待，产生不同的作用。例如，ASCII 字符设备总是以 ASCII 形式处理数据，要显示(打印)数字“8”，必须将其 ASCII 码(38H)提供给显示器(打印机)。

另外，PC 还采用扩展 ASCII 码，主要表达各种制表用的符号等。扩展 ASCII 码最高 D7 位为 1，以与标准 ASCII 码区别。

3. Unicode

ASCII 码表达了英文字符，但无法表达世界上所有语言的字符，尤其是像非拉丁语系的语言，例如中文、日文、韩文、阿拉伯文等。为此，各国也都定义了各自的字符集，但相互之间并不兼容。例如，1981 年我国制定了《信息交换用汉字编码字符集基本集》(GB 2312—1980)国家标准(简称国标码)。规定每个汉字使用 16 位二进制编码，即两个字节表达，共计 7445 个汉字和字符。实际应用中，为了保持与标准 ASCII 码兼容、不产生冲突，国标码两个字节的最高位被设置为 1，这称为汉字的机内码。不过，汉字机内码会与扩展 ASCII 码冲突(因它们的最高位都是 1)，所以一些西文制表符有时会显示为莫名其妙的汉字。

为了解决世界范围的信息交流问题，1991 年国际上成立了统一码联盟(Unicode Consortium)，制定了国际信息交换码 Unicode。在其官方网站上对“什么是 Unicode”给出了如下解答：“Unicode 给每个字符提供了一个唯一的数字，不论是什么平台，不论是什么程序，不论是什么语言。”Unicode 使用 16 位编码，能够对世界上所有语言的大多数字符进行编码，并提供了扩展能力。Unicode 作为 ASCII 码的超集，保持了与其兼容。Unicode 的前 256 个字符对应 ASCII 字符，16 位编码的高字节为 0、低字节等于 ASCII 码值。例如，大写字母 A 的 ASCII 码值是 41H，用 Unicode 编码是 0041H。

现在 Unicode 已经越来越被大家认同，很多程序设计语言和计算机系统都支持它。例如，Java 语言和微机 Windows 操作系统的默认字符集就是 Unicode。

第二节 常量表达

学习 C 语言时，也是先讲解数据类型，掌握如何使用常量和变量表达数据。C 语言的基本数据类型有字符 char、整型 int(包括短整型 short int 和长整型 long int)，以及浮点单精度(float)和双精度(double)实数。本章涉及最基本的整数编码，包括整型和字符，实际上字符可以看作 8 位的整数。

那么，汇编语言如何使用常量和变量形式表达整数编码呢？基于前节基本知识，在本节先介绍如何使用常量表示数值和字符，下节将说明怎样将它们保存在存储器中，最后再讲解处理器指令如何访问它们。

常量(Constant)是程序中使用的一个确定数值，在汇编语言中有多种表达形式。

一、常数

这里的常数指由十、十六和二进制形式表达的数值，见表 2-4。各种进制的数据以后缀字母区