汇编语言程序设计

李兆凤　等编

中央广播电视大学出版社

前　言

计算机技术的迅速发展，特别是微型计算机的广泛使用，使计算机的应用日益普及。要想更好地应用和开发微型计算机，必须学习计算机的语言。汇编语言就是作为计算机应用的重要语言之一。

汇编语言是面向机器的语言，使用中有较好的效率，因而在实时和过程控制中特别受到重视。

作为大专院校计算机专业的学生，学习《汇编语言程序设计》课程是必须的。本书即为此目的而编写的。

考虑到现有学校的实验条件和鉴于 Z-80 系统在我国的广泛使用，本书仍然"靠在"Z-80 系统上进行讲述。这样可使学习者学有所用。如果学习了 Z-80 系统的程序设计，我们认为学习其它的汇编语言也会有了基础。

考虑本书能为更多的读者服务，所以在编写中力求由浅入深，循序渐进，讲清道理，多举实例。以使读者通过自学也能学会程序设计。

本书共分十一章。系统完整地介绍了 Z-80 汇编语言的特点及使用。

本书第一章到第三章中，介绍了 Z-80 的指令系统和时序；在第四章到第九章中，分别介绍了各种类型的程序设计。特别突出了非数值处理、输入/输出及中断程序设计。它们是微计算机应用中遇到最多的问题。为使读者更深入一步地了解程序设计方法和程序调试，在本书的第十章和第十一章介绍了程序设计的方法和调试。

为了使读者能较快地掌握汇编语言并运用它较熟练地编制汇编语言程序；本书例举了很多典型的例题并配有一定量的习题，供读者练习。

汇编语言程序设计的能力，只有通过大量实验和亲自实践才能得到提高。为此，我们还组织编写了《汇编语言程序设计实验指导书》作为本书的配套教材。读者可根据其要求作一定量的实验，以加深程序设计的能力。

本书第一章、第四章到第九章由李兆凤同志编写；第十章和第十一章由孟庆昌同志编写；第二章和第三章由丛宏彬同志编写。全书由北京邮电学院教授朱祥华同志主审。

本书可作为高等院校计算机专业学生的教材，也可以作为其它专业学习计算机知识的教学参考书。

由于时间仓促，编者水平有限，书中一定存在不少的错误及不妥之处，请广大读者批评指正。

编　者

1987 年 10 月

目　录

第一章 概 论

随着科学技术的发展，电子计算机已经在各个领域得到广泛的应用。现在可以说，几乎每一个地方都在使用着计算机。它对社会的生活、发展和进步起着巨大的影响。可以预计，在不远的将来，计算机将深入到每个家庭，和每一个人都将发生关系。

本章介绍微型计算机、微型计算机系统的基本概念，以及汇编语言及其特点。目的在于对微型计算机做一整体介绍，以为后面各章学习做些必要的准备。

第一节 引 言

一、微处理机和微型计算机及其系统

自 1946 年诞生第一台电子计算机以来，电子计算机主要经历了从电子管、晶体管、中小规模集成电路到大规模集成电路等四个时代。其体积变得越来越小，计算速度变得越来越高，价格变得越来越低。随着大规模集成电路(LSI)及超大规模集成电路(VLSI)技术的发展，已经可以把计算机的中央处理单元即控制器、运算器制作在一个芯片上，人们称其为微处理机(或称微处理器)，有时就简称为CPU(central processing Vnit)。这就开始了微型计算机的时代。如果将内存储器、输入/输出接口电路与中央处理单元制作在同一芯心上，就是所谓单片微型计算机。

下面给出微处理机、微型计算机及微型计算机系统的含义：

微处理机(Microprocessor)：

它是将运算器和控制器制作在一块半导体芯片上的集成电路器件，是构成微型计算机的核心。也称微处理器。

微型计算机(Microcomputer)：

由微处理机配上存储器、输入/输出接口电路等芯片，由总线联接构成的设备，称为微型计算机。

微型计算机系统(Microcomputer system)：

由微型计算机配上外部设备、电源和软件，则构成一个可以独立工作的完整的计算机整体。我们称这个完整的计算机整体为微型计算机系统。但人们通常所称呼的微型计算机，就是指微型计算机系统。即微型计算机已经是带有外设和软件，并能正常工作的设备了。但严格说起来，还是称为微型计算机系统更为恰当。

二、软件

前面介绍的微型计算机系统，除软件之外，其它部分可称为硬件(Hardware)或硬设备。一个计算机，只有硬设备，仍然是不能工作的。为了使其正常运行，必须配备各种程序。我们称为

使计算机正常运行和能完成运行操作的各种程序为软件(software)。因为它们不像硬件那样可以看得见模得着。软件通常存贮在外部介质(如磁盘、磁带、卡片)上，或运行时存贮在内存储器上。

软件根据其所完成的功能，大致分为两类：

1. 系统软件

为了使计算机正常运行，专门由机器设计者编写了管理计算机工作的程序，这就是系统软件，是计算机赖以工作所必须的。如操作系统、监控程序、调试程序、诊断程序和各种语言的编译程序。系统软件一般是随着购置的计算机设备一起配置的。当计算机工作时，它就担负起对计算机的运行管理工作，以使计算机有条理地进行操作。

2. 应用软件

用户为了应用计算机解决自己特定的实际问题，就要编写应用程序，或者利用已经标准化的应用程序(通称软件包，已商品化，用户可根据自己应用需要而购置)，这些程序称为应用软件。

目前，在我国已有大量作为信息管理使用的商品化的应用软件，且大部分已汉字化(即数据的输入及结果的显示或输出都使用汉字)。了解已有的应用软件的功能，对于计算机的使用者是很必要的。如果有现成的应用软件可用，就不必再花费精力编写应用软件去做别人已经做过的事情。可以直接使用"现成"的应用软件。

第二节　汇编语言程序设计

我们知道，构成计算机的数字电路(逻辑电路)只识别"1"、"0"两个电平状态，即只识别二进制代码。因此机器指令码是用二进制编码实现的。这种能直接为计算机所接受的机器指令码的集合，我们称之为机器语言(Machine Language)。它是与计算机本身结构有关的。不同结构的计算机，有不同的机器指令。这种二进制编码的机器语言，既不便于记忆又难以掌握，使用起来很不方便，所以人们很少使用它。但对有控制面板的计算机，为了进行硬件维护，可以直接通过面板输入机器指令，检查计算机运行情况，仍然是有必要和方便的。现在市场上出售的各种单板计算机，也因为受内存容量的限制，大多使用机器指令代码作为输入语言。故机器语言仍然在一定的限度内被使用。但对大多数计算机工作者，由于其不方便，不再专门使用机器语言编写程序了。

一、汇编语言(Assembly Language)

为了解决机器语言使用上的不方便，人们想出一种利用助记符(Mnemonic)来代替操作码，用符号(synbol)来代替地址的方法。助记符使用与操作功能含义相应的缩写英文字符来表示，符号地址由用户根据需要自己定义，这样就把不容易记忆的机器指令变为易于记忆的助记符指令了，人们称这种改造后的语言为汇编语言。称助记符指令为汇编语言的指令语句。显然，汇编语言比机器语言容易理解、记忆和便于交流。这一改造使汇编语言的广泛使用成为可能。然而，人们使用汇编语言编写的程序，必须经过一次翻译，将其翻译为机器语言才能为机器所接受，因

为计算机并不能识别助记符及符号地址等文字符号。我们将完成这一翻译任务的程序叫做汇编程序，它是由系统预先提供的系统软件之一，是专门完成把汇编语言程序翻译为机器语言程序任务的。汇编程序的操作过程通常称为汇编。平常我们说把汇编语言程序进行汇编，就是把汇编语言程序在汇编程序管理下，翻译为机器语言程序的过程。

汇编语言的指令语句与机器指令是一一对应的，它仍然是依赖于计算机的。本质上说，仍是一种面向机器的语言。它要求使用者对计算机本身要有一定的了解，对机器的指令系统要熟悉，这就给学习汇编语言带来一定的困难。

特别值得指出的是，汇编语言的指令语句与程序设计人员要求计算机所要完成的任务有很大的差别。由于指令语句能做的都是一些简单的操作，如累加器的内容移位，数据在寄存器、存储器间传送，两个寄存器内容相加等，它远非程序设计人员所要求的操作。例如我们要检查在给定数据中大于某个值的数据有多少，并把它们在打印机上打印出来的任务。或者是在满足一定条件的情况下，向某个设备发出一个停机命令要求等。这样一些要求是不能直接用汇编指令语句完成的。必须由程序员把这些要求转换为一系列汇编语言的指令语句。做这种转换工作是有一定困难并且较费时间的。从这个角度来看，学习使用汇编语言编程也是有一定难度的。这也就是为什么后来会发展出一种更乐意为人们所接受的高级语言的原因。

汇编语言程序(PROGRAM)是指用汇编语言由用户所编写出来的程序。从广义上讲，程序是使计算机执行特定任务的一系列指令语句的集合。对汇编语言来说，不仅包括指令语句，还应包括计算机完成规定任务所需要的数据和结果的存储器地址。例如要作加法运算，显然要有相加的两个数和存放结果的地方。计算机程序就是要确定数据和结果的存放地址以及要完成规定的运算指令语句的集合。

二、高级语言(High Level Language)

高级语言是一种脱离具体计算机，面向过程，更符合人们思维和更易为人们所理解、学习的语言。如经常使用的 BASIC、FORTRAN、PASCAL、COBOL、C 语言等。它们的语句和逻辑结构与人们的思维基本上是一致的。所以用高级语言编程就变得容易而方便了。通常用高级语言编写程序要比用汇编语言编写程序快 10 倍。而在调试程序方面，高级语言会更简单而节省时间。现在使用的大部分应用软件均是用高级语言编写的。它不依赖于计算机的型号，已经有各种国际标准语言版本供参照使用。用某种高级语言所编写的程序，一般在不同计算机上都可以运行(只要该机具有这种语言的编译程序就行)。

三、汇编语言与高级语言比较

概括地说，高级语言有下面一些优点：

(1) 使用高级语言可以更容易，更快速地编写程序。编写出的程序也容易为它人阅读，即可读性好。

(2) 使用高级语言编写的程序在调试中容易查错，使调试工作变得比较简单。

(3) 一般高级语言都带有文件操作功能，可以将有关数据以文件方式存于外存储器中。

(4) 高级语言不涉及具体的计算机体系结构及其指令系统。因此程序设计人员可以不必了

解计算机的任何特性，而把精力集中在他们自己所要完成的任务本身工作上。

(5) 高级语言编写的程序可移植性好。它们可以被移植到有该语言的编译程序的任何计算机上使用(因编译系统的不同，可能有部分语句需要进行修改)。

(6) 通常在一个计算机系统上，都有用高级语言编写的子程序库，供用户使用。

按照上面的说明，似乎高级语言最好，对使用者又容易又方便，为什么还要学习汇编语言呢？事物总是一分为二的。对计算机语言也不例外，有其长处必有其不利之处。

用高级语言编写的程序，计算机不能直接执行，必须先把用高级语言编写的源程序，也要通过翻译，将其翻译为机器语言，我们把完成这一任务的翻译程序称为编译程序。由它将高级语言程序翻译成以机器指令所表示的目标程序，目标程序才能为计算机所执行。编译程序本身长度约为 1 KB 到上百 KB(字节)。而汇编程序长度约为 2 K 到几十 KB。所以使用编译程序额外地占有很大的存储器空间，这就加大了计算机系统的成本。其次，由于高级语言设计时，是以面向过程而设计的，主要考虑怎样更容易编写程序，以解决应用问题。而且编译过程是机械地自动地进行的。它要综合各种可能性而给出结果。这样在将高级语言程序编译为机器语言的目标程序时，就不可避免地会出现不够精练、冗长等问题，以至加大了目标程序的长度和增加了运行时间。换句话说，将要占有较大的存储空间和运行较长的时间。无论从哪方面讲，都是不经济的。所以，高级语言并不能产生有效的机器语言程序。而汇编语言，则恰好能弥补高级语言的不足。它的指令语句是与机器指令一一对应的，由其编写出的程序较为精炼，一般来说会有较高的效率。所以在某些方面，如实时控制等，汇编语言仍然是受到人们青睐的一种语言。

可以这样地说，在信息管理和科学计算方面使用高级语言较为适宜，在实时控制和专用微机系统中，使用汇编语言较为适宜。希望读者在实际工作中，应根据任务的需要，能够正确的选用语言。顺便说一句，高级语言目前常用的已有十几种，它们均有各自使用的特点，应区别应用的不同而选用不同的高级语言。

第三节　如何学习汇编语言程序设计

汇编语言程序的特点已如上述。它介于机器语言和高级语言之间。它仍然要依赖于计算机。可以说，不同厂家设计出的计算机，均有自己的汇编语言。

考虑到 Z-80 CPU 是一种在世界上使用较广的八位微处理机，它在我国也得到广泛的应用。尽管现在市场上出现一批准 16 位或 16 位微型计算机，但从利用计算机芯片组成计算机控制系统来看，八位机仍不失是一种很好的应用系统。围绕 Z-80 CPU，有大量配套芯片供使用，从微型计算机应用的角度来看，掌握 Z-80汇编语言仍然是使用计算机的一个主要方面。因此本书将以 Z-80 汇编语言为基础，讲解汇编语言的程序设计。了解 Z-80 汇编语言程序设计后，学习其他汇编语言也就比较容易了。

本书较详尽地介绍利用汇编语言进行各种程序设计的方法，应用例子，以使读者较好地掌握汇编语言程序设计。在应用例中，注意了实用性，以便在实际工作中可以借用。

下面通过一个具体程序例，来看如何用汇编语言设计程序。

如已知数存于 M 单元及 N 单元中，求其乘积，将结果赋值给 X 单元中。

这件事在使用高级语言时是很容易实现的，只要一个语句就可完成任务。

$$X=M*N$$

上面表达式的写法，对不同高级语言可能有所不同，但其表达方式大致相似。这同我们用数学的表示方法是基本一致的。它显得直观、易懂、易记。但如用 Z-80 汇编语言编程，则完全不能用上面的表达方式。必须通过多次加法运算来表示一个乘法运算，这是因为 Z-80 CPU 只有加、减法指令，而没有乘、除法指令的原因。

下面我们分别给出实现上述乘法运算的流程图及程序。

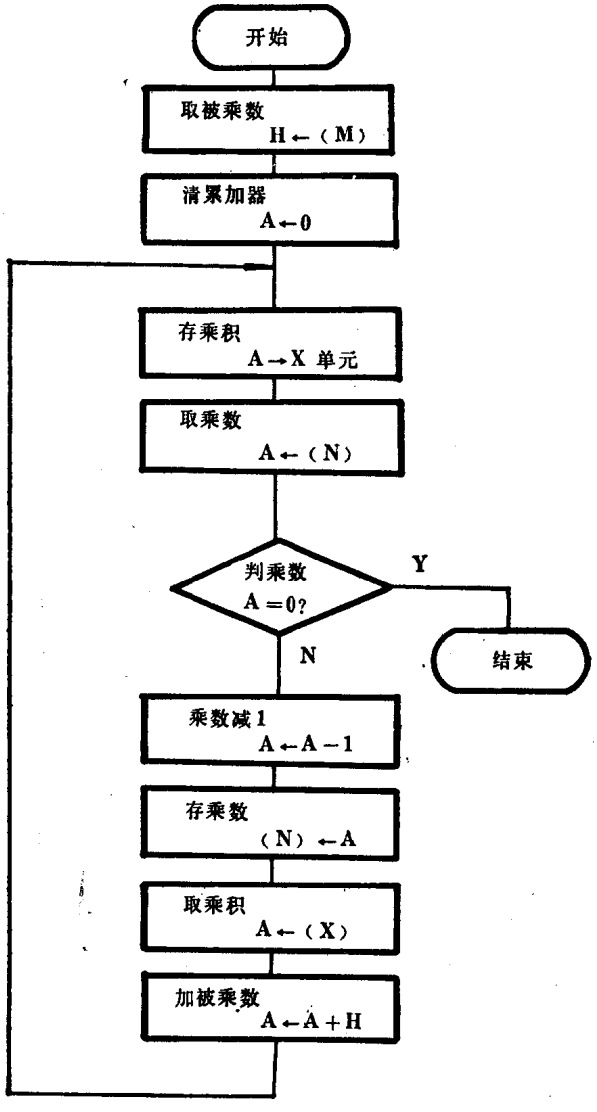


图 1.1

源程序:

```
标    号:操作码    操作数        注释
START:LD        A,(M)   ; 取被乘数送累加器A
      LD        H,A     ; 转送H寄存器暂存
      XOR       A       ; 清累加器A
LOOP: LD        (X),A   ; 存乘积到X单元
      LD        A,(N)   ; 取乘数到累加器A
      OR        A       ; 为判断结果,置标志位Z
      JP        Z,DONE; 结果为零,转DONE执行
      DEC       A       ; 不为零,乘数减1
      LD        (N),A   ; 送乘数到N单元
      LD        A,(X)   ; 取乘积到累加器A
      ADD       A,H     ; 乘积加被乘数
      JP        LOOP    ; 转LOOP,继续下一次加法操作
DONE: HALT                ; 停
      END
```

在程序中,我们首先准备3个单元(它们均在存储器中),放被乘数于单元M,放乘数于单元N,单元X用来存放乘积,即运算结果。A累加器和H寄存器是Z-80 CPU的工作寄存器,本处用来存放运算数。因为Z-80 CPU没有乘法指令,只好利用被乘数不断相加,由乘数值规定相加的次数(变乘法为加法),从而得到乘积。流程图中的减1操作,表示每加一次被乘数后,乘数减去1,这样不断循环相加,直到乘数被减到零为止。由于加法运算指令均要通过累加器A进行,所以程序中出现累加器A的内容不断地与乘数和与乘积单元内容交换操作,这是迫不得已的事。这些"繁锁"的操作就是汇编语言程序设计的特点。

源程序段给出的是用汇编语言指令语句编写的程序。指令语句只具有加、减、移位、传递、转移等功能。

在比较高级语言和汇编语言的编程情况之间,可发现两者有着完全不同的思路。从某种意义上讲,汇编语言更讲究编程技巧。同一问题,由不同的人编写,可以写出完全不同的程序,而其结果是完全相同的。

下面谈谈学习汇编语言程序设计中的注意事项。

(1) 汇编语言程序设计与计算机的硬件有密切的关系。因此,为了掌握汇编语言程序设计,必须要较好地了解Z-80 CPU的结构、性能及其有关部件的功能。如果要与外部设备进行数据传输,还要了解有关接口电路的功能。

(2) 使用汇编语言编程时,对指令语句的选择是很重要的。同样的功能,使用不同的指令语句,可以有很大的差别。有经验的编程人员可以编写出占用存储器空间少和执行时间短的程序,这就使得程序有较高的效率。所以在学习指令系统时,就应对每条指令的功能、作用有较深刻的

理解和熟记，以便在编程时能够正确的运用。

(3) 在编写稍微复杂一点的程序时，不要用指令语句直接编写程序。要先画出完成指定任务的程序流程图，经检查无误后，再按流程图编写程序。这样编写出的程序逻辑错误较少。即使发现错误，根据流程图查错也很方便。所以，应该养成利用程序流程图编程的习惯。

(4) 不管多么熟练的程序设计者，也不能保证编写出的程序绝对无错。因此，经过计算机执行检验结果，是检查程序正确性的最好手段。学习计算机语言，要想学到程序设计的技能和技巧，只能多编程，多上机，也就是多实践，才能掌握好和学到手。所以有人说：计算机学的好坏与上机时间成正比。这一点初学者一定要引起重视。

(5) 程序设计是一件繁锁而又需耗费脑力劳动的事。特别是汇编语言程序设计更是如此。因此初学者一定不要怕学习中的困难，要坚持多编程、多上机，逐步地就会进入较自由的境地，那时就会发现，程序设计是件非常有趣的事。

(6) 通过汇编语言程序设计的学习，应该掌握一些经常应用的较成熟的子程序，以便今后编程需要时，加以应用。

(7) 通过汇编语言程序设计和上机操作，应该学会对微型计算机系统的使用。特别是了解操作系统所提供的系统子程序，它们的功能，使用条件，以备需要时调用。

(8) 在汇编语言的学习过程中，应该读懂一些像单板计算机监控程序这种规模的程序。以提高编写大一些程序的工作能力。

总之，我们希望读者在开始学习程序设计时，就能对上面提到的事项给予充分注意。如果能认真做好习题，并能多做一些实验，相信一定会为学习者打下一个汇编语言程序设计的初步基础。祝愿每位读者都能取得好的学习成绩。

第一章习题

1. 试说明微处理机，微计算机，微计算机系统的含义。它们之间是什么关系。

2. 什么叫单板计算机，单片计算机，微型计算机系统。你知道它们间的区别和在应用方面的不同吗？

3. 请说明软件、硬件的含义。你能说明它们之间的关系吗？

4. 说明高级语言与汇编语言的各自特点。应用中根据什么选择它们。

5. 请区别下面术语：汇编语言，汇编语言程序，汇编程序，汇编，编译。

第二章　Z-80 CPU 的结构

本课程讲述的汇编语言的模型机的中央处理器是 Z-80 CPU。Z-80 CPU 是一种 8 位单总线结构的微处理器。通过对 Z-80 CPU 的介绍，使读者对一般 8 位微处理器的基本结构有一定的了解，同时也为了解 16 位微处理器打下基础。

第一节　Z-80 CPU 的结构组成

Z-80 微处理器(或称 Z-80 CPU)由寄存器组、算术逻辑单元、指令寄存器和指令译码器、定时和系统控制信号电路等四部分组成。它们的内部结构和相互关系如图 2-1 所示：

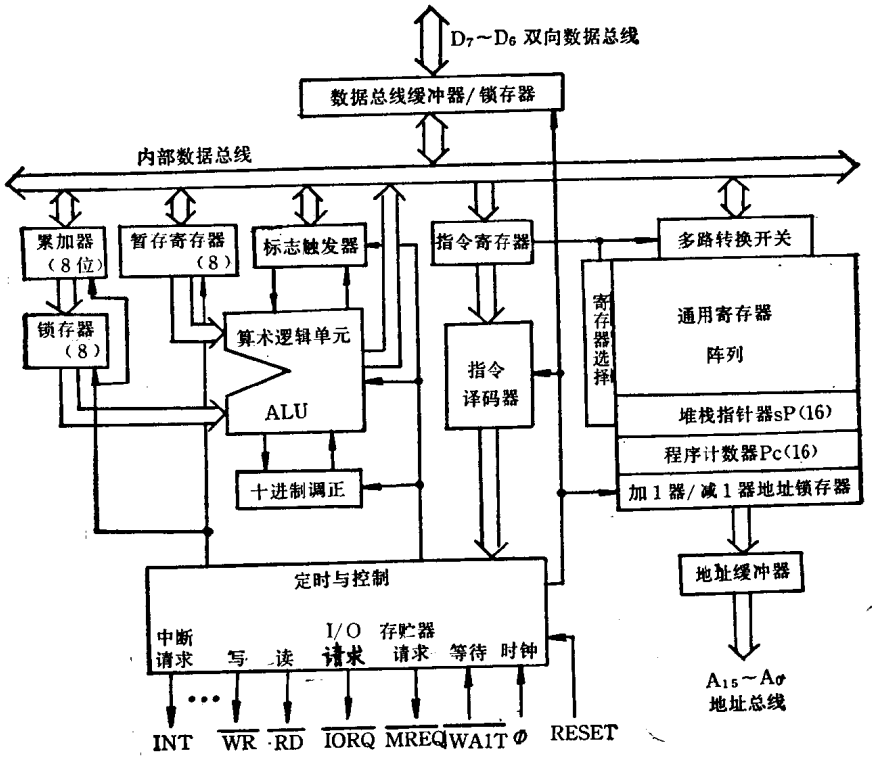


图 2-1　CPU 内部结构方框图

图中列出了 Z-80 CPU 的基本组成部分。从图中可以看出，这几部分都是并接在内部数据总线上的，这样的结构就称为单总线结构。下面分别介绍 Z-80 CPU 内部的各部分的作用。

一、寄存器组

1. 通用寄存器

Z-80 中内部寄存器分为两大类，一类是通用寄存器，另一类是专用寄存器。Z-80 设有 6 个 8 位的通用寄存器，分别用 B、C、D、E、H、L 表示，用来寄存参加运算的操作数和中间结果(也可以是操作数的地址)。对应于这 6 个寄存器，Z-80 内部还设有 6 个备用寄存器 B′、C′、D′、E′、H′、L′，作为前 6 个寄存器的后备寄存器。通常称 B、C、D、E、H、L 为通用寄存器的主寄存器，而称 B′、C′、D′、E′、H′、L′ 为通用寄存器的辅寄存器(或备用寄存器)。

2. 专用寄存器

Z-80 内部的专用寄存器有 PC、IX、IY、I、R、SP、F、A、A′、F′ 等，它们在 Z-80 中分担不同的角色，各有各自的功能，是专门用来存放某种特定意义的数据的。

① 程序计数器 PC(Program Counter)

程序计数器用来存放将要执行的下一条指令的内存地址。它是一个 16 位寄存器。CPU 将程序计数器 PC 中的内容作为地址，按照这个地址从内存中取出一条指令，通过译码之后，执行这条指令。PC 中的内容总是下一条该执行的指令的 16 位地址。因为一段程序是由一系列指令组合而成的，它们顺序地存放在内存中一段连续的区域中，而执行程序实际上就是顺序地执行一条一条的指令。所以，在一般情况下，当取出一条指令或一个指令字节后，PC 就自动加 1，只有在执行转移指令和子程序调用指令(或中断)时，PC 的内容就不再是顺序执行时下一条指令的地址，而是程序要转向的地址。

② 变址寄存器 IX 和 IY(Index Register)

变址寄存器 IX 和 IY 是两个独立的 16 位寄存器。通常它们各包含一个 16 位的基地址(这个地址是根据需要写入 IX 和 IY 的，一般在程序的首部的赋值部分完成)。由基地址加上指令中给出的偏移量，以形成操作数的实际地址。这种寻址方式在数据表格处理方面应用最多，它能使许多类型的程序大大简化。

③ 中断页地址寄存器 I

I 寄存器是一个 8 位寄存器，用来存放中断矢量的高 8 位。当 Z-80 CPU 以中断方式与外围设备(简称外设)交换信息时，外设若有中断请求，而 Z-80 CPU 也允许外设中断，此时，程序要转向中断服务子程序。在 Z-80 中断方式 0 中，由外设提供 RST 00 H—RST 38 H 指令中的任一条，程序就固定地转向 0000 H—0038 H 执行。对于中断方式 2，就需要将中断服务子程序的入口地址存放到由中断矢量决定的内存单元中(有关中断问题将在后面详细介绍)。中断矢量是一个 16 位的地址，中断矢量的高 8 位，事先由指令写入中断矢量寄存器 I 中，中断矢量的低 8 位，由申请中断的外设提供。由这两个 8 位地址联合成一个 16 位的地址，即构成完整的中断矢量，这个矢量就确定了中断服务子程序的入口地址在内存中存放的位置。

④ 存贮器刷新地址寄存器 R(Refresh)

Z-80 的存贮器采用 MOS 动态存贮器，它利用寄生电容来存贮信息。为防止信息消失(由于电容泄漏放电)，必须定期对动态存贮器进行刷新(即对电容定期充电)。Z-80 微处理器为使用动态存贮器(RAM)的用户提供了自动刷新功能，在控制信号线中提供了自动刷新信号 $\overline{\text{RFSH}}$，由 R 寄存器提供刷新地址。这样，在每次取指令之后，Z-80 就进入刷新工作状态。这时，

$\overline{RFSH}$ 信号有效，R 寄存器将其 8 位值送到地址总线的低 8 位，形成刷新地址，动态存贮器 RAM 便可自动刷新。每刷新一次，R 寄存器的内容自动加 1。只要在 2 ms 的时间内将所有的动态存贮器 RAM 全部刷新一次，动态存贮器 RAM 就能够正常地工作。值得一提的是，真正有效的刷新地址是由 R 寄存器的低 7 位提供的，每次刷新是刷新内存中一行的单元，并不是内存中的一个单元。

⑤ 堆栈指针 SP(Stack Pointer)

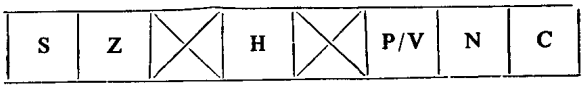
堆栈是 CPU 外部存贮器中的一个按照先进后出原则组织的存贮区域。堆栈指针是一个 16 位的地址，它始终指向栈的顶部。堆栈中的内容和寄存器中内容的交换，必须用堆栈操作指令 PUSH 和 POP 并借助于堆栈指针 SP 来完成。利用堆栈，可以完成多次中断(即中断嵌套)和子程序的嵌套及其他操作。

⑥ 累加器 A(Accumulator)

累加器 A 是一个 8 位寄存器，它与状态寄存器 F 相连。在进行算术和逻辑运算时，累加器 A 中的内容一定是一个操作数，而参加运算的另一个操作数存放在其他的通用寄存器中或在内存的某一个单元中或是一个立即数。运算的结果一定是存放在累加器 A 中的。与累加器 A 对应，还有它的备用寄存器 A′。

⑦ 标志寄存器 F(Flag)

标志寄存器为一8 位寄存器，主要用来表示累加器 A 中当前操作的状态。它的每一位表示的意义如下图(只定义了 6 位)：



这些状态反映了当前操作的特征，如有没有进位，有没有溢出，结果是否为零等等。这些状态都是根据操作由机器自动填入寄存器中的。这 6 位的主要含义如下：

a. 进位位 C(Carry)

进位位是 F 寄存器中的最低位。分有两种情况，一是在进行加法运算时，最高位向更高位有进位时，C＝1，否则，C＝0；二是当进行减法运算时，最高位向更高位有借位时，C＝1，否则，C＝0。当进行移位操作时，右移会将寄存器或存贮单元的最低位 D_0 移入 C 中；左移时，会将寄存器或存贮单元的最高位移入 C 中。

b. 减法标志位 N

表示前一次的操作是加法还是减法。如果是减法，N＝1；否则，N＝0。

c. 奇偶/溢出标志 P/V(Parity/Overflow)

在进行算术运算时，P/V 标志运算的结果有无溢出，如果 P/V＝1，则说明运算发生溢出；如果 P/V＝0，则无溢出。在进行逻辑运算时，P/V 表示运算的结果中含有 1 的个数为一偶数，还是奇数。如果为偶数 P/V＝1；如果为奇数 P/V＝0。

d. 半进位位 H(Half carry)

表示在加法或减法运算时，D_3 有无向 D_4 的进位或借位。当 D_3 向 D_4 有进位或借位时，

H＝1；无进位或借位时，H＝0。该标志主要用于二——十进制运算时，反映8位的低半个字节向高半个字节的进位或借位，以便根据有无进位或借位来校正结果。

e. 零标志位 Z(Zero)

用来表示运算的结果是否为零。如果结果为零，则 Z＝1；如果结果不为零，Z＝0。

f. 符号位 S(Sign)

用来存放累加器 A 中的最高位(D_7)的状态，在进行带符号数的算术运算时(采用二进制补码)，在规定的数的范围内，它表示运算结果是正数还是负数。如果是正数，S＝0；如果是负数，S＝1。一般地，S 是根据运算结果中的最高位 D_7 来决定自己的取值的。D_7＝1 时，S＝1；D_7＝0 时，S＝0。对于无符号数的运算，S 标志无意义。

二、算术逻辑运算单元 ALU(Arithmetic and Logic Unit)

Z-80 CPU 的运算器包含暂存寄存器(8位)，累加器锁存器(8位)，累加器(8位)和算术逻辑运算单元。

算术逻辑运算单元是运算器的核心部件，用来进行算术逻辑运算。在进行算术逻辑运算时，参与运算的一个操作数是累加器中的数，它锁存在锁存器中；另一操作数由暂存器暂存，运算结果送到累加器 A 中或送其它部件。注意暂存器中暂存的操作数是根据指令由内存中某个单元送来的，或者就是指令中包含的一个立即数。

算术逻辑运算单元所能完成的运算有：加、减、逻辑"与"、逻辑"或"、逻辑"异或"、比较、左移、右移或循环、加1运算、减1运算，位操作等等。

三、指令寄存器 IR 和指令译码器 ID

1. 指令寄存器 IR(Instruction Register)

指令寄存器 IR 用来存放从存贮器中取出的将要执行的指令，它从内部数据总线上读入指令，然后送到指令译码器 ID 中进行译码。

2. 指令译码器 ID(Instruction Decoder)

指令译码器 ID 将指令译码。它将指令寄存器中的指令按不同的内容译成不同的电位组合，来控制完成指令中所需完成的动作。它从指令寄存器 IR 中取出将要译码的指令，译码之后，直接联结到微操作控制线路。

四．定时及系统控制信号电路

系统控制信号是 Z-80 各条指令实际操作的电位组合，它们在定时信号下，有组织、有规律地谐调工作，完成各条指令的功能。

Z-80 CPU 的定时信号由外部振荡器产生，然后引入微处理机。

定时和系统控制信号共有 24 个，可分成四组：系统控制、CPU 总线控制、CPU 控制和基本定时信号。

1. 系统控制信号

① $\overline{M}_1$(第一机器周期)信号

$\overline{M}_1$ 信号是低电平有效的输出信号。当该信号为低电平时(信号有效)，表示计算机正在进行

取指令操作，当指令的操作码是两字节的操作码时，则每一取操作码周期都发出$\overline{M_1}$信号。在中断响应周期，$\overline{M_1}$信号及$\overline{IORQ}$信号（见后）同时有效。

② $\overline{MREQ}$（存贮器请求）信号

$\overline{MREQ}$信号是低电平有效的三态输出信号，当该信号有效时（$\overline{MREQ}$电平为低），表示地址总线上的地址码是访问内存贮器的地址，这时，进行的是对存贮器的读/写操作。

③ $\overline{IORQ}$（输出输入请求）信号

$\overline{IORQ}$信号是低电平有效的三态输出信号。当该信号有效时，表示地址总线上的地址码是访问输入/输出（I/O）设备的地址，这时进行的操作是对I/O设备的读/写操作（实际上是输出输入操作）。

若$\overline{M_1}$、$\overline{IORQ}$两信号同时变为有效，表示I/O设备提出的中断请求已被CPU响应，这时申请中断的设备应将一中断矢量送到数据总线上。

④ $\overline{RD}$（读）信号

$\overline{RD}$信号是低电平有效的三态输出信号。当该信号有效时，表示CPU正进行的操作是从内存贮器或I/O设备读取数据的操作。

⑤ $\overline{WR}$（写）信号

$\overline{WR}$信号是低电平有效的三态输出信号。当该信号有效时，表示CPU正在进行的操作是向存贮器或I/O设备写入（即输出）信息的操作。

⑥ $\overline{RSFH}$（刷新）信号

$\overline{RSFH}$信号是低电平有效的三态输出信号。当信号有效时，表示CPU正在进行的操作是对存贮器的刷新操作，此时地址总线的低7位表示动态存贮器的刷新地址。

2. CPU的控制信号

① $\overline{HALT}$（暂停）信号

$\overline{HALT}$信号是低电平有效的输出信号。当该信号有效时，表示CPU正在执行HALT指令，机器处在暂停状态。这时只有CPU收到中断请求信号或重新启动信号时，才能退出暂停状态。暂停期间CPU执行NOP（无操作或空操作）指令，以维持对动态存贮器RAM的刷新动能。

② $\overline{WAIT}$（等待）信号

$\overline{WAIT}$信号为低电平有效的输入信号。这个信号的设立是为了使任意速度的I/O设备或存贮器与高速CPU能同步工作。一般地，存贮器和I/O设备的读/写速度与CPU的速度相比要慢许多，所以当存贮器或I/O设备进行读/写操作时，就向CPU发出一个$\overline{WAIT}$信号，告诉CPU存贮器或I/O设备的数据还没有准备好，此时CPU就进入等待状态。直到数据准备好（即存贮器或I/O设备的读/写操作完成），这时$\overline{WAIT}$信号变为高电平（即无效），然后CPU才进行下面的操作。不过这样的工作方式浪费了CPU的许多宝贵时间。

③ $\overline{INT}$（可屏敝中断请求）信号

$\overline{INT}$信号是低电平有效的输入信号。这个中断请求信号是由I/O设备发出的可以被系统屏敝的中断请求信号。当该信号有效时，当CPU内部的中断允许触发器IFF为开放中断状态（即