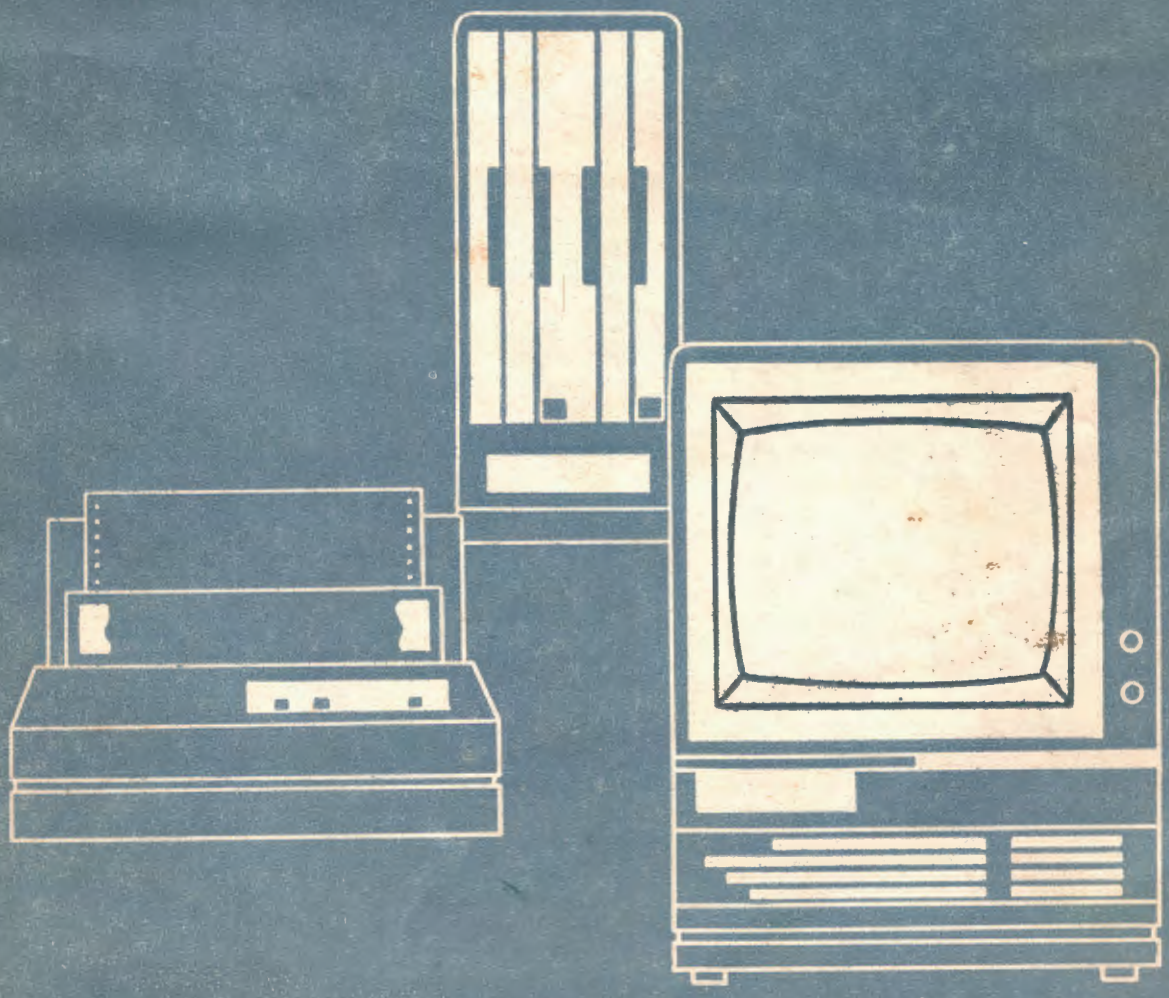
朱仲英　编

微型计算机原理与应用

上海交通大学出版社



封面设计：徐培毅

书　号：13324·0009
定　价：　7.10 元

微型计算机原理与应用

朱仲英 编

上海交通大学出版社

内 容 简 介

本书着重从应用的角度，系统阐述了电子计算机的基础知识和微型计算机的原理与应用。全书共分十七章，其中包括：电子计算机的基本概念和基本原理；典型微处理器的结构原理、指令系统和微处理器的操作时序；微型计算机的汇编语言及其程序设计的基本方法；半导体存贮器及其接口；微型计算机的输入输出、中断系统；可编程序并行、串行输入/输出接口电路；微型计算机输入输出设备及其接口技术；微型计算机系统的组成及微型计算机应用系统设计中的一些共性问题等。

本书可作为大专院校有关专业的教学用书，也可作为广大科技人员的自学教材和参考书。

微型计算机原理与应用

朱 仲 英 编

*

上海交通大学出版社出版
（上海淮海中路 1984 弄 19 号）
常熟文化印刷厂排版
上海群众印刷厂印刷
新华书店上海发行所发行

*

开本：787×1092 毫米 1/16 印张：38.5 字数：951,600
1985 年 2 月第一版 1985 年 2 月第一次印刷
印数：1—65,000

统一书号：13324·0009 科技新书目：82-151

定价：7.10 元

微型计算机原理与应用

朱仲英 编

上海交通大学出版社

内 容 简 介

本书着重从应用的角度，系统阐述了电子计算机的基础知识和微型计算机的原理与应用。全书共分十七章，其中包括：电子计算机的基本概念和基本原理；典型微处理器的结构原理、指令系统和微处理器的操作时序；微型计算机的汇编语言及其程序设计的基本方法；半导体存贮器及其接口；微型计算机的输入输出、中断系统；可编程序并行、串行输入/输出接口电路；微型计算机输入输出设备及其接口技术；微型计算机系统的组成及微型计算机应用系统设计中的一些共性问题等。

本书可作为大专院校有关专业的教学用书，也可作为广大科技人员的自学教材和参考书。

微型计算机原理与应用

朱 仲 英 编

*

上海交通大学出版社出版

（上海淮海中路 1984 弄 19 号）

常熟文化印刷厂排版

上海群众印刷厂印刷

新华书店上海发行所发行

*

开本：787×1092 毫米 1/16 印张：38.5 字数：951,600

1985 年 2 月第一版 1985 年 2 月第一次印刷

印数：1—65,000

统一书号：13324·0009 科技新书目：82-151

定价：7.10 元

前　言

微型计算机(又称微型电脑)是电子计算机(又称电脑)的一个重要分支，它是二十世纪七十年代崛起的一项新兴的科学技术。它的出现为电子计算机的广泛应用，开拓了极其广阔的前景，并且正在人类社会的各个领域中引起一场新的技术革命，其深远意义并不亚于当年蒸汽机的诞生所迎来的第一次技术革命。如果说，以蒸汽机为标志的第一次技术革命，是用机器来代替人类繁重体力劳动的动力革命，那末以电子计算机为标志的新的技术革命，将是用电脑，特别是微型电脑来代替人类部分脑力劳动的信息革命。今天，电脑化、自动化、信息化已成为一个国家现代化的主要标志。

本书着重从应用的角度，系统阐述电子计算机的基础知识和微型计算机的原理与应用。全书共分十七章：第一至第四章系统阐述电子计算机的基本概念、基本原理等基础知识，着重介绍电子计算机的运算基础知识和工作过程，也适当介绍一些必要的数字逻辑电路基础知识；第五章至第七章介绍典型微处理器的结构原理、指令系统和微处理器的操作时序；第八、九两章介绍汇编语言及其程序设计的基本方法；第十章介绍半导体存贮器及其接口；第十一至第十五章介绍微型计算机的输入/输出方式、接口技术、典型的输入/输出设备和接口芯片的功能与应用，着重介绍中断和可编程序接口技术；第十六、十七两章以 Z 80 STARTER SYSTEM KIT 和 SDK-85 单板微型计算机系统为例，扼要地介绍微型计算机系统的组成和微型计算机应用系统设计中的一些共性问题，着重剖析 Z 80 STARTER SYSTEM KIT 的监控程序，使读者了解并掌握程序设计的基本方法和技巧。最后，附录部分收集、整理了微型计算机应用中的一些常用资料，以供读者查用。本书的重点是电子计算机的基础、微处理器的结构原理、指令系统和汇编语言程序设计、输入/输出接口技术等四个部分。

在编写本书过程中，编者力求全面、系统地阐述电子计算机的基本概念、基本原理，贯彻理论联系实际的原则，加强应用部分。在选材上，主要考虑到当前在微型计算机应用领域中是以 8 位微型计算机为主的现实，故选用当前 8 位机中应用广泛而又比较先进的 Intel 8080 A/8085 A、Z 80 及 MC 6800 等机型为实例，横向地组织教材，而不限于纵向介绍某一单个机型。同时，本书内容力求反映当前计算机，特别是微型计算机发展的最新技术。在叙述方法上，力求做到由浅入深、循序渐进，以适应读者自学的需要。

本书原稿曾用作编者在上海交通大学自动控制专业讲授《微型计算机原理与应用》课程的讲义，也曾用作相近专业和各类科技人员培训班的教材。为了推广微型计算机的应用，满足大专院校学生学习微型计算机应用基础知识的需要，帮助科技人员学习和应用微型计算机，在上海交通大学电工及计算机科学系学术委员会推荐和支持下，编者将原有讲义加以充实、整理而编成此书。本书讲授周期约 90 学时，讲授时，可侧重 Intel 8080 A/8085 A 或 Intel 8080 A、Z 80 微型计算机系统，其余部分让读者自学。若读者已具备一定的数字逻辑电路基础知识，则可越过第三章数字逻辑电路部分，直接学习后面章节内容。

本书原稿承蒙上海交通大学电工及计算机科学系陈铁年副教授和白英彩同志审核，提出了许多宝贵意见和建议。在编写过程中，得到了中国科学院学部委员张钟俊教授和支秉彝博士

的悉心指导，上海交通大学、华东师范大学的谢志良、汪燮华、范懋基副教授和谢剑英、袁长奎、陈应麟等同志也给予很大帮助。张文娟同志在整个编写出版过程中协助做了大量工作。在此，谨表示衷心感谢。

由于编者水平有限，缺少经验，书中缺点和错误在所难免，欢迎读者批评指正。

朱仲英于上海交通大学

一九八四年四月

目　录

第一章 概 论

§1-1 电子计算机的发展概况

电子计算机(Electronic Computer)是一种能够自动地、高速地、精确地进行信息处理的现代化的电子设备。根据所处理的信息是数字量或是模拟量，电子计算机又可分为三类：数字计算机、模拟计算机和混合计算机。由于当前广泛应用的是数字计算机，因此，通常把电子数字计算机(Electronic Digital Computer)简称为电子计算机。

电子计算机最初是作为一种现代化的计算工具而问世的。它是人类在长期的生产和科研实践中，为减轻繁重的劳动和加快计算过程而努力奋斗的结果。在电子计算机出现以前，人类曾发明创造了各种各样的计算工具。早在唐朝末叶，我国人民就发明创造了算盘，这是世界上最早的计算工具。后来，在国外又相继出现了许多计算工具。1642 年在法国制成了在世界上第一台机械计算器，本世纪初又出现了电动计算器、卡片计算器等。1931 年美国 V·Bush 为解线性微分方程而设计的微分分析器是世界上第一台电子模拟计算机。

二十世纪四十年代中期，一方面由于导弹、火箭、原子弹等现代科学技术的发展，需要解一些极其复杂的数学问题，原有的计算工具已满足不了要求；另一方面由于电子学和自动控制技术的迅速发展，为研制电子计算机提供了物质技术条件。

1946 年在美国宾夕法尼亚大学由 J·W·Mauchley 和 J·P·Eckert 领导的为弹道设计服务而制成的 ENIAC (Electronic Numerical Integrator And Calculator) 是世界上第一台由程序控制的电子数字计算机。它的字长 12 位，使用 18,800 只电子管，1,500 多个继电器，耗电 150 瓩，占地面积 150 平方米，重量达 30 吨，投资近百万美元，每秒钟只能完成 5000 次加法运算。这就是第一代电子计算机，其特点是体积大、功耗大，但是它为发展电子计算机奠定了技术基础，如程序存贮、数字编码等。自此后三十多年，电子计算机的发展日新月异，近年来，大约每隔几年就有一次重大发展，称之为换代。今天，采用大规模集成(LSI——Large Scale Integration)电路技术，可以把具有 ENIAC 功能的计算机集成到面积仅数平方毫米的硅片上，制成单片微型计算机，芯片价格只有几个美元。

纵观三十多年来电子计算机的发展史，大致经历了三代的演变，目前正全面地向第四代过渡，并开始研制第五代计算机，见表 1-1。

第一代(1946～1958 年) 是电子管数字计算机。计算机的逻辑元件采用电子管；主存贮器先用汞延迟线，后用磁鼓、磁芯，外存贮器已开始采用磁带；软件主要采用机器语言，后期才用汇编语言；应用以科学计算为主。其特点是体积大、耗电大、可靠性差、价格昂贵、维修复杂，但它奠定了以后电子计算机高速发展的技术基础。

第二代(1958～1964 年) 是晶体管数字计算机。计算机的逻辑元件采用晶体管；主存贮器采用磁芯，外存贮器已开始使用更先进的磁盘；软件有了很大发展，出现了各种各样的高级语言及其编译程序；应用以各种事务数据处理为主，并开始用于工业控制。其特点是体积小、耗电少、可靠性较高，性能比第一代计算机提高了一个数量级。

表 1-1　电子计算机的发展简史表

特点 \ 代 / 项目			第一代 1946年～1958年	第二代 1958年～1964年	第三代 1964年～1971年	第四代 1971年以后	第五代 1982年以后
特点	硬件	元件	电子管	晶体管	中小规模集成电路(SSI, MSI)	大，甚大规模集成电路(LSI, VLSI)	超大规模集成电路(SLSI)
		存贮器	磁鼓 水银延时电路 磁芯 磁带	磁芯 磁盘	磁芯 磁盘	半导体存贮器 磁盘	半导体存贮器 磁泡存贮器 磁盘 光盘
		典型机器	IBM-701 (1953、4) IBM-650 (1954、11)	IBM-7090 (1959、11) IBM-7094 (1962、9)	IBM-370(大) (1971年) IBM-360(中) (1964年) PDP-11(小)	巨型机： ILLIAC-IV(由64个处理部件组成，速度1.5～2亿次/秒，1973年) CRAY-1(浮点速度8000万次/秒 1976年) 大型机： IBM3033, 3081 日立M-200, M-280 超小型机〔注〕： VAX-11 微型机： Intel 8080, 8085, MC6800, Z80 MC6809 Intel 8086, Z8000, MC68000 Intel iAPX286; Intel iAPX 432, NS的16032	正在研制中。
	软件		只有机器语言和汇编语言	1. 批量处理操作系统(1960～1965) 2. FORTRAN程序语言(1954～1960) 3. COBOL程序语言(1960～1965) 4. ALGOL60、PL/1等	1. 分时操作系统如：CTSS, IBM360的OS(1965～1970) 2. 会话式语言如：BASIC程序语言(1964) 3. 结构程序设计	1. 软件工程 2. 程序设计自动化 3. 程序设计理论 4. 程序正确性证明 5. 计算复杂性理论 6. 数据库	1. 智识库 2. 自然语言处理 3. 软件工程
应用			1. 科学计算 2. 成批数据处理	1. 科学计算 2. 数据处理如企业管理、商业处理 3. 工业控制	1. 大型科学计算 2. 系统模拟 3. 系统设计 4. 分时操作	1. 尖端科学和军事工程计算 2. 大型事务处理 3. 计算机网络 4. 微型机渗透到各个技术领域 5. 智能模拟	1. 人工智能 2. 计算机专家系统 3. 计算机网络

注：小规模集成（SSI——Small Scale Integration）电路指在单片硅片上集成10个以下门电路或100个以下晶体管的集成电路。

中规模集成（MSI——Medium Scale Integration）电路指在单片硅片上集成10～100个门电路或100～1000个晶体管的集成电路。

大规模集成（LSI——Large Scale Integration）电路指在单片硅片上集成100～1000个门电路或1000～10000～20000个晶体管的集成电路。

甚大规模集成（VLSI——Very Large Scale Integration）电路指在单片硅片上集成1000～10000个门电路或10000～100000个晶体管的集成电路。

超大规模集成(SLSI——Super Large Scale Integration)电路指在单片硅片上集成10000个门电路以上或100000个以上晶体管的集成电路。超小型机通常是指高档的小型机，字长32位，功能比一般的小型机(标准字长16位)更强。

第三代(1964～1971 年)是集成电路数字计算机。计算机的逻辑元件采用小、中规模集成(SSI、MSI)电路；主存贮器仍以磁芯为主；软件逐渐完善，分时操作系统、会话式语言等多种高级语言都有新的进展。在发展大型机的同时，小型机也蓬勃地发展起来；应用领域日益扩大。其特点是小型化，耗电很少，可靠性高，性能比第二代计算机又提高了一个数量级。

第四代 (1971 年以后)是大规模集成电路计算机。计算机的逻辑元件和主存贮器都采用大规模集成电路(LSI)。所谓大规模集成电路是指在单片硅片上集成 1000～20000 个晶体管的集成电路，其集成度比中、小规模集成电路提高了 1～2 个数量级。其特点是微型化、耗电极少、可靠性很高，这些优异的特点正适合于当时原子能利用、电子计算机和空间技术蓬勃发展的需要，这就有力地促进了计算机工业的空前大发展。

第四代计算机无论从硬件或软件方面来看，技术都日臻完善，平均速度在每秒一千万次以上的巨型机已成批生产，每秒一亿五千万次的巨型机已投入运行。为了进一步提高计算机的性能，计算机结构也开始以分布式处理来组织系统，如美国 ILLIAC-IV 巨型机就是由 64 个处理部件组成的，平均运算速度为每秒一亿五千万次。同时，大型机、超小型机、计算机网络、智能模拟、软件工程等都有新的进展。应用范围也广泛深入到社会生活的各个领域。

随着大规模集成电路技术的迅速发展，七十年代以后，计算机除了向巨型机方向发展外，还朝着微型机方向飞跃前进。1971 年末，世界上第一台微处理器和微型计算机在美国旧金山南部的硅谷应运而生，从而开创了微型计算机的新时代。随之而来各种各样的微处理器和微型计算机如雨后春笋般地研制出来，潮水般地涌向市场，成为当年首屈一指的畅销品。这种势头直至今天仍然方兴未艾。

现在美国、日本等许多国家正在加紧研制第五代计算机。可以预见，这将是以超大规模集成电路(SLSI)和人工智能为主要特征的完全崭新的一代计算机。

§1-2 微处理器和微型计算机的发展概况

微处理器 (Microprocessor)，简称 μP 或 MP，是指由一片或几片大规模集成电路组成的具有运算器和控制器功能的中央处理器部件(CPU——Central Processing Unit)，又称微处理机。微处理器本身并不等于微型计算机，它仅仅是微型计算机的中央处理器。有时为了区别大、中、小型中央处理器(CPU)与微处理器，而称后者为 MPU (Microprocessing Unit)。

微型计算机(Microcomputer)，简称 μC 或 MC，是指以微处理器为核心，配上由大规模集成电路制作的存贮器、输入/输出接口电路及系统总线所组成的计算机(简称微型机，又称微型电脑)。有的微型计算机把 CPU、存贮器和输入/输出接口电路都集成在单片芯片上，称之为单片微型计算机。

微型计算机系统(Microcomputer System)，简称 μCS 或 MCS，是指以微型计算机为中心，配以相应的外围设备、电源和辅助电路(统称硬件)以及指挥微型计算机工作的系统软件，就构成了微型计算机系统。

从历史的角度看，二十世纪七十年代微处理器和微型计算机的产生和发展，有其必然性。一方面是由于当时军事工业、空间技术和工业自动化技术的迅速发展，日益要求生产体积微小、可靠性高和功率低的计算机，这种社会的直接需要是促成微处理器和微型计算机产生和发展的强大动力。另一方面是由于大规模集成电路技术和计算机技术的飞跃发展，1970 年已

经可以生产1K位的存贮器和通用异步收发器(UART)等LSI产品；并且计算机的设计日益完善，功能越来越强，如总线结构、模块结构、堆栈结构、微程序结构、有效的中断系统及灵活的寻址方式等等，这些技术为研制微处理器和微型计算机打下了物质基础和技术基础。

因而，自从1971年微处理器和微型计算机问世以来，得到了异乎寻常的发展，大约每隔2～4年就更新换代一次，至今，已经历了三代的演变，现已进入第四代，见表1-2。

微型计算机的换代，通常是按其CPU字长位数和功能来划分的。

第一代（1971～1973年）是4位、低档8位微处理器和微型机。代表产品是美国Intel公司首先制成的4004微处理器及由它组成的MCS-4微型计算机(集成度1200晶体管/片)，随后又制成8008微处理器及由它组成的MCS-8微型计算机。其特点是：采用PMOS工艺，速度较慢，基本指令执行时间约为10～20μs，字长4位或8位，指令系统比较简单，运算功能较差，系统结构未超出台式计算器[注]范围，软件主要采用机器语言或简单的汇编语言，但是它们的价格低廉。

第二代(1974～1978年)是中档8位微处理器和微型机。其间又分两个阶段：1973～1975年为典型的第二代，以美国Intel公司的8080和Motorola公司的MC 6800为代表，集成度提高1～2倍，(Intel 8080集成度4900管/片)，运算速度提高一个数量级；1976～1978年为高档的8位微型计算机和8位单片微型计算机阶段，称之为二代半。高档8位微处理器，以美国Zilog公司的Z 80和Intel公司的8085为代表，集成度和运算速度都比典型的第二代提高了一倍以上(Intel 8085集成度9000管/片)；1979年Motorola公司制成MC 6809，其性能之优异完全可以与Z 80媲美。8位单片微型机以Intel 8048/8748（集成度9000管/片）、MC 6801、Mostek F 8/3870、Z-8等为代表，它们主要是用于工业控制和智能仪器，由于性能/价格比很高，因此，销路很广。近年来，这类单片微型机的发展和性能更新非常快，现在Intel公司已制成更高档的单片微型机Intel 8049/8749和Intel 8051/8751，其中Intel 8751单片微型机上包括中央处理器(CPU)、4 KB EPROM、256字节RAM、4个8位I/O端口、两个16位定时器/计数器、高速全双工串行I/O口和两级外部中断，基本指令执行时间为1μs，这是相当引人注目的。此期间制成的Intel 8022单片微型机，除其本身已是完整的微型机外，还包括两个8位A/D转换器，这更展现出单片微型机发展的广阔前景。总的来说，第二代微型机的特点是采用NMOS工艺，集成度提高1～4倍，运算速度提高10～15倍，基本指令执行时间约为1～2μs，指令系统比较完善，已具有典型的计算机体系结构以及中断、DMA等控制功能，寻址能力也有所增强，软件除采用汇编语言外，还配有BASIC、FORTRAN、PL/M等高级语言及其相应的解释程序和编译程序，并在后期开始配上操作系统，如CP/M(Control Program/Monitor)操作系统便是目前世界上流行的一种微型机磁盘操作系统，它适用于以8080 A/8085 A/Z 80为CPU、带有磁盘和各种外设的微型计算机系统。就目前而言，8位微型机仍然是微型机应用中的主流。

第三代（1978～1981年）是16位微处理器和微型机。代表产品是Intel 8086（集成度29000管/片），Z 8000（集成度17500管/片）和MC 68000（集成度68000管/片）。其特点是采用HMOS工艺，基本指令时间约为0.5μs，从各个性能指标评价，都比第二代微型机提高了一个数量级，已经达到或超过中、低档小型机(如PDP 11/45)的水平。这类16位微型机通常

【注】台式计算器(desk calculator)是一种置于台上操作的小型电子计算装置。通常它仅能完成算术运算和少量逻辑操作并显示其结果，功能简单。

表 1 2　微处理器、微计算机的发展简史

特点 / 项目		第一代 1971年～1973年	第二代 1974年～1975年	第二代半 1976年～1977年	第三代 1978年～1981年	第四代 1981年以后
硬件	元件	PMOS LSI 1200～2000 晶体管/片	NMOS LSI 5000 管/片 左右	NMOS LSI 9000管/片 左右	HMOS LSI, VLSI 20000～68000 管/片	HMOS/CMOS SLSI 10万管/片以上
	字长	4/8 位	8 位	8 位	16位	32位
	引线	16，24 条	40 条	40 条	40，48，64，68 条	64条
	基本指令执行时间(μs)	10～20	2	1.3～1 2.5～10(单片微型计算机)	<1	<0.125
	典型产品	Intel 4004 Intel 8008	Intel 8080 Motorola 6800	Intel 8085 Zilog Z 80 Motorola 6809 R 6500 Intel 8048/8748 Motorola 6801 Z-8	Intel 8086、8088 Zilog Z 8000 Motorola 68000 LSI-11/23, LSI-11/24 Intel iAPX 186 iAPX 286	Intel iAPX432 NS16032 Bell MAC-32 HP 公司的 32 位 μP Intel iACX-96
软件		采用机器语言，汇编语言(简单的)	采用汇编语言，有交叉，驻留汇编程序。配有高级语言如 BASIC, FORTRAN, PL/M 等非驻留的解释程序和编译程序，一般不配操作系统	采用汇编语言，有交叉、驻留汇编程序。配有高级语言，有驻留的解释程序和编译程序，配有操作系统	采用汇编语言、高级语言。 软件均为驻留的汇编程序，解释程序和编译程序，配有操作系统	操作系统，高级语言软件硬化
特点		1. 运算功能较差 2. 系统结构未超出台式计算器范围 3. 价格极低	1. 运算速度比第一代提高 10 倍 2. 集成度比第一代提高 1 倍 3. 具有典型计算机结构如中断、DMA 功能 4. 价格低	1. 运算速度比第二代又提高 1～2 倍 2. 集成度比第二代提高 1 倍，出现单片微型计算机 3. 中断功能大为加强 4. 价格低	1. 性能比第二代提高1～2个数量级 2. 集成度比第二代半又提高1～2倍 3. 具有小型机的体系结构，性能已达到或超过七十年代初的中档小型机(如 PDP11/45)。 4. 价格中等	1. 性能比第三代又提高1～2个数量级 2. 从单元集成过渡到系统集成组成，甚至包括应用软件在内的系统 3. 组成多微机紧耦合系统
应用		面向消费： 1. 家用电器 2. 计算器 3. 简单控制器	面向工业应用： 1. 智能终端和仪器仪表 2. 工业控制 3. 教学和实验	面向工业应用： 1. 智能终端和仪器仪表 2. 工业控制 3. 教学和实验 4. 数据处理	实时控制和实时数据处理： 1. 实时控制系统 2. 数据库 3. 大型事务处理 4. 科学计算 5. 分布式多微处理机系统 6. 局部网络	实时数据处理，实时多任务、多道程序： 1. 大型事务处理 2. 多用户数据处理 3. 科学计算 4. 分布式多微机系统 5. 局部网络

都具有丰富的指令系统、采用多级中断系统、多重寻址方式、多种数据处理形式、段式寄存器结构、乘除运算硬件，电路功能大为增强，并都配备了强有力的软件系统。近年来高档16位微型机的发展很快，Intel公司在8086基础上又制成了iAPX 186(80186)和iAPX 286 (80286)等性能更为优越的16位微型机，其特点是从单元集成过渡到系统集成，以获得尽可能高的性能/价格比，如iAPX 186和iAPX 286在综合性能上分别是iAPX 86(8086)的2倍和5倍。与此同时，Intel公司又制成了iAPX 88(8088)，其内部为16位CPU，而对外部的数据总线为8位，从而使其比其它高档的8位微型机具有更优异的性能，如从执行程序的速度来看，iAPX 88较Z 80 A和MC 6809快2～3倍，这就进一步开拓了8位微型机的应用前景。此外，发展16位微型机的另一途径是将过去已经流行的小型计算机微型化。例如，美国DEC公司以PDP-11系列机为背景，开发了LSI-11系列。目前有LSI-11、11/2、11/23和11/24等4种微处理器。1983年出现的Intel iACX-96 16位单片微型机系列，带有4～8个10位A/D转换器，具有很强的功能。总之，高档16位微型机的迅速发展，将弥补现在8位微型机由于字长和速度的局限性而造成的缺陷，从而为微型机在实时数据处理和实时控制领域中的广泛应用开辟广阔的前途。

第四代（1981年以后）是32位微处理器和微型机。1980年10月，美国国家半导体公司制成了相当于IBM 370-138 CPU功能的单片微处理器NS 16032。1981年初，Intel公司推出了iAPX 432微处理器。此后，HP公司、贝尔研究所又先后研制成新的32位微处理器。

从目前32位微处理器的产品来看，大致可分为两种类型。一种是现有的16位微处理器的扩充，即外部数据总线为16位，而内部寄存器、ALU总线宽度等都可扩充为32位。NS 16032、iAPX 432即属此类。另一种是外部数据总线也为32位的微处理器，如HP公司及贝尔研究所的MP 32微处理器等即属此类。如果按照数据总线的位数来分类，只有HP、Bell Lab和IBM等几家公司的微处理器，才真正称得上32位微处理器。

HP公司的32位微处理器芯片集成了45万个晶体管，时钟频率为18 MHz，微周期为55 ns。贝尔研究所的MAC 32 μP芯片集成了15万个晶体管，微周期为31 ns。由于集成度很高，因此，系统的速度和性能大为提高，可靠性增加，成本降低。现在，32位微型机的功能，已足以同高档的小型机相匹敌，大有取代中、小型机之势。

§1-3 电子计算机的发展趋势

当前电子计算机发展趋势的主要特点，可概括为四个字，即："微"（指微型机）、"巨"（指巨型机）、"网"（指计算机网络）、"智"（指计算机智能模拟），以及相应的软件工程的开发。

下面分别作一简要说明：

一、微型机

如前所述，与大、中、小型机相比，微型机具有一系列显著的优点，使之成为当今世界上发展最迅速、应用最广泛的新技术之一。预计到八十年代末，微型机的性能将成百倍地提高，而价格将下降到现在的1/10～1/20。那时，微型机将成为人们日常生活中所不可缺少的工具，人们将会象离不开电那样离不开微型计算机。

当前，微型机的发展趋势，有以下六个特点：

1．加速发展 LSI/VLSI 技术，不断提高微型机的性能。

随着 LSI/VLSI 技术的发展，目前已出现更高档的微处理器：iAPX 432 和 NS 16000系列。Intel 公司的 iAPX 432 是由三块 LSI 芯片组成的 μP，含有 25 万个晶体管，即 43201 (11 万个管/片)处理指令译码，43202 处理指令执行，43203 处理 I/O 接口。其特点是系统结构组成多微处理机紧耦合系统，并实现包括多处理操作系统和高级语言在内的系统集成。其性能比第三代 μP 提高 2～7 倍。今后高档微型机将以"标准单元"进行设计，其集成度更高、结构更复杂、分系统更多、功能更强、直接寻址范围更大、运算速度更快、I/O 处理能力更强。目前正向多位并行微处理器方向发展，即由 8 位到 16 位乃至 32 位。预计不久将出现功能极强的 64 位微处理器。

2．从 CPU 集成发展到微型计算机系统集成，即发展带有软件硬化的微型计算机系统。

现在集成化技术处于单一CPU 集成化阶段，下一阶段将是高速浮点运算处理机、I/O 通道处理机、存贮器或总线集成化等，使 CPU 功能扩充并强化外围组件的集成化。更进一步将是操作系统和高级语言等软件的固化，甚至包括应用系统软件的固化。

3．加速发展专用化的微型计算机，以满足各种控制、电子产品和家用电器等方面的需要。这种专用化微型机制成标准单元，它包括 CPU、RAM、ROM/EPROM 模块，模拟接口(A/D、D/A 转换电路)，定时器/计数器及其它 I/O 分系统。如美国 Intel 2920 就是带 A/D、D/A 转换电路的单片微型机，它可以自动地对输入的模拟信号采样并进行 A/D 转换，然后经 CPU 数字处理和 D/A 转换后输出，可用于实时处理模拟信号。

4．发展分布式多微处理机系统(Multimicro-Processor System) 和局部网络 (Local Area Network)，将是人们特别关注的发展方向。

所谓分布式是相对于集中式而言的，即指把系统的功能或智能分散到各个子系统去完成，以构成各种各样的分布式多微处理机系统，这类系统具有高速运算能力、高可靠性和高可利用率等显著优点，使之成为微型机发展的重要方向之一。过去集中式的数据处理方式将被分布式处理方式所代替。局部网络也是近年来崛起的一门新兴技术，它是微型机、分布式处理与短程通信技术相结合的产物。一般是指在一个建筑物或同一地区内的不同类型的微型机所组成的高速通信系统，其地理范围在 10 公里之内。由于它在实现资源共享、分布式处理方面正日益发挥着重要作用，因此近年来世界各国竞相研制局部网络并已形成一股热潮。

5．CMOS 工艺将会迅速发展

由于 CMOS 具有功耗低，抗干扰能力强，抗温度变化，抗电源起伏和速度潜力大等特点，因而有广阔的发展前途。它将使整台微型计算机全部用电池供电，做到真正的便携式。

过去 CMOS 发展慢的主要原因，是 CMOS 的开关速度仅为 NMOS 的 1/6，但最近 NS 公司用硅栅代替金属栅工艺，使 CMOS 的速度赶上了 NMOS。近年来，由于集成技术的突破，加上 CMOS 可以把数字和模拟电路制作在同一芯片上，因此，CMOS 工艺有了突飞猛进的发展。人们普遍认为，CMOS 工艺将成为今后最重要的工艺之一。

CMOS 新机种的研制工作进展很快。近年来，Bell 实验室用 CMOS 制成了真正的 32 位 μP Bell MAC-32。

6．采用新技术，提高 RAS 性

RAS 是可靠性(Reliability)、有效性(Availability)、和可维护性(Serviceability)等技术的总称。通过采用各种新技术，如微诊断，提高芯片集成度，减少部件，简化接口等措施来提高微

型计算机系统的RAS性，也是当前国外微型计算机发展的趋势之一。

二、巨型机

巨型机是指高速度、大容量和高性能的巨型计算机。现在平均运算速度为每秒五千万次的巨型机已成批生产，每秒一亿五千万次的巨型机也已投入运行，每秒十亿次甚至百亿次的巨型计算机正在研制中。如美国克雷公司的CRAY-2 S系统，浮点运算速度约10亿次/秒，主存容量400～3200万字等。

目前国际上对巨型机的发展有两种观点：一是巨型机的体系设计；二是用微型机群组成巨型机。巨型机的发展集中地体现了计算机科学的研究水平，因而作为一种发展方向仍将是肯定的。

由于本书重点论述微型机，故对巨型、大型、中型、小型机等就不一一介绍了。

三、计算机网络

计算机网络是指用通信线路把多个分布在不同地点的计算机连接起来的网络系统。其目的是使用户能共享网络中的能源，包括硬件、软件和数据等，均衡负荷，提高可靠性和系统的效率。用户可在不同地点，分时地使用同一个计算机网络。目前世界上最大的和较完善的计算机网络是由美国国防部高级研究局建造的ARPA(Advanced Research Projects Agency)网络。它是用高速传输线(50千位/秒)把不同地点的计算机连接起来构成的。这个网络遍布美国全国以及英国、挪威，通过卫星信道实现数据传输。该计算机网络目前已拥有45台主计算机和35台接口机，主要用于信息检索。随着无线电遥测、遥控、遥信等技术与微型机的日益结合，以及数据传输、光纤通信、电视系统和激光技术等方面的发展，计算机网络将会有更大的发展。

四、计算机智能模拟

指用计算机模拟人的感觉和思维过程的部分智能，以进行识别图象，听懂语言，适应环境，接受启发等智能活动。这是一门探索和模拟人的感觉和思维过程规律的新兴学科。它是建立在控制论、计算机、仿生学、心理学等基础上的边缘学科。目前，美国、日本、英国和德国等国家都在加紧研制智能机器人(Intelligence Robot)，并已制成能"看"、能"听"、能"走路"的机器人。日本已制成能用于装配工序的工业机器人。但这只是计算机智能模拟发展的初级阶段，估计短期内还不会有重大的技术突破。

展望未来，计算机科学将会有许多新的突破。可以预测，未来的计算机将是半导体技术、超导技术、激光技术、电子仿生技术相结合的新型计算机。八十年代，采用超大规模集成电路的计算机已开始出现或正在研制中，速度可能达到每秒运算10～100亿次，即比目前最快的计算机快10～100倍。九十年代以后，集成光路、超导器件(约瑟夫器件)将会有所突破，电子仿生技术也将用于计算机，从而可能诞生现代计算机所望尘莫及的光学计算机、超导计算机和人工智能计算机，计算机科学将被推向一个崭新的阶段。

我国电子计算机工业是在解放后才兴办起来的。1958年试制成我国第一台电子管数字计算机M3(103型)，填补了我国电子计算机的空白。1959年又研制成DJS-2(104型)大型电子管数字计算机。自1964年起，陆续研制成多种晶体管电子计算机，如DJS-6、DJS-8等型号。1971年试制成功每秒运算10多万次的TQ-16型集成电路电子计算机。1973年又研制成

功每秒运算100万次的DJS-11大型集成电路电子计算机。1974年研制成功DJS-130小型多功能集成电路电子计算机。1977年研制成功每秒运算200万次的大型集成电路电子计算机和DJS-183小型多功能计算机。1979年研制成功中规模集成电路的DJS-140机，与此同时DJS-200系列机也相继问世。1983年是我国计算机发展史上获得辉煌成果的一年。这一年，不仅先后研制成功高性能的DJS-153、DJS-142小型机和我国第一台大型向量电子计算机——"757"工程，运算速度达1000万次/秒，而且还研制成功我国第一台超高速的巨型电子计算机——"银河"，运算速度达1亿次/秒，这标志着我国进入了世界研制巨型计算机的行列。

我国从1974年开始研制微型机，1976年研制成功DJS-050系列微处理器（与Intel 8080兼容），以后又研制成功DJS-060系列微处理器（与MC 6800兼容）。现在DJS-050系列和DJS-060系列的微型机都已批量生产。研制和开发16位微处理器的工作也在加紧进行中，1981年骊山微电子公司已研制成LS77 II型微型计算机系列。我国微型机应用是近几年才开始推广的，发展非常迅速，而且已经在实践中开始显示出强大的生命力，当务之急是迅速推广微型机的应用，形成工业生产能力。目前除主要发展8位和4位微型机外，还应积极开发16位微型机，分布式多微处理机系统及局部网络。所有这些都将对各行各业产生深刻的影响。

§1-4 电子计算机的应用

现代电子计算机特别是微型计算机，已经广泛渗透到工业、农业、企业管理、交通运输、商业、国防、科研、文教、通信、生物医学、日常生活等各个领域，日益显示出其强大的生命力。电脑化已经成为衡量一个国家现代化水平的重要标志。

计算机的应用范围主要有以下几个方面：

一、数值计算

数值计算是计算机最原始的应用领域。过去，很多工程设计和科研课题由于计算工作量庞大而无法进行或只能用粗略近似的算法。采用电子计算机后，由于它具有快速、精确的特点，过去人工计算需要几个月、甚至几年的数值计算，现在仅需几天，几小时，甚至几分钟就解决了，而且精确度高，这就大大缩短了研制、设计的周期，把人们从繁琐而重复的计算中解放出来。例如，十八世纪英国数学家商克斯用"手算"花费了二十年光阴，把圆周率计算到707位小数，其中第527位数还有误。而今天，在高速电子计算机上，仅需6.8小时就已突破800万位小数大关。

二、数据处理与自动化管理

数据处理是指计算机对外围设备送来的大量数据，及时地进行采集、加工、合并、分类、传递、存贮、检索等综合分析工作。数据处理应用领域十分广泛。如企业管理、情报检索、气象预报、飞机订票、防空警戒等，已普遍应用计算机进行数据处理。据统计，目前在计算机应用中，数据处理所占的比重最大。目前，世界上工业发达国家都在实现电脑自动化管理，微型计算机的出现促使现代化电脑管理进入以文献信息为主的多种事务管理领域。尤其在办公室自动化（OA——Office Automation）领域中，微型计算机将大显身手。在处理这类事务中，需要人的分析、判断和决策，而且是非重复性的操作，仅用传统的计算机是难以实现的。八十年代以来，由于OA技术的发展，如办公室计算机、个人计算机、文字处理机、计算机终端、智能复印机、仿真等，在软件方面开发了分散性用户语言；同时，连接各种OA装置的局部网络和以卫星

通信为中心的数值交换技术的发展，使OA系统结构成为人、电脑、通信(Human factor、Computer、Communication)之间有机结合的系统，这对于实现现代化管理有着重大的意义。

目前，在我国实现现代化电脑管理的关键，是使微型计算机具有汉字信息处理能力，即具有编码、输入、存贮、处理、打印、显示等一系列技术性能。这些技术有的已基本解决，有的正在解决之中，可望不久将会出现更多实用而功能齐全的汉字微型计算机系统。

三、自动控制

电子计算机借助于传感器、A/D、D/A转换器和执行机构，已进入了模拟信息的工业控制领域。计算机在工业过程控制中的应用，除了巡回检测、自动记录、统计制表、监视报警和自动启停外，还可以直接调节和控制生产过程，以实现工厂自动化(FA——Factory Automation)。

六十年代中期和七十年代初期，在计算机过程控制系统方面用得最多的一种是直接数字控制系统，即DDC系统(Direct Digital Control System)。它是用小型机取代模拟调节器，可以实现几十个乃至几百个回路的PID控制，并且只需变更程序，就很容易实现其它新型的控制规律，(如前馈控制、最佳控制等)，这就有效地克服了模拟式仪表的局限性。但是，由于DDC系统是"集中型"的控制系统，一旦系统中的某个环节出现故障，就会影响整个系统的正常工作，致使控制系统的可靠性下降。另外，集中型控制系统，对于小规模控制系统来说，利用率较低，加之计算机本身价格昂贵，这是造成过去计算机过程控制得不到广泛应用的重要原因。自从微型计算机问世并被广泛用于工业控制领域后，计算机过程控制就进入了一个崭新的阶段。目前，出现了以微型计算机为中心的"分散型控制系统"或称"综合仪表控制系统"。在这种系统中，其控制功能分散给若干台微型计算机处理，而操作管理则高度集中到一台高性能的计算机控制。因此，这种系统又称为集散系统MTDCS(Microprocessor Total Distributed Control System)。它吸收了集中和分散控制的优点，提高了系统的可靠性和可利用率，现已成为计算机控制的重要发展方向。

在仪器仪表应用方面，微处理器可作为一个控制元件安装在仪器仪表设备中，即所谓仪器仪表智能化。例如，在调节仪表中，可安装微处理器进行PID运算；在电子仪器中，微处理器被用于计算输入信号的平均值、极小值和某些统计量等；在计算机系统中，可在外围控制器中安装微处理器构成智能终端(Intelligent terminal)等。在实现家庭自动化(HA——Home Automation)方面，微型计算机将为家庭提供各种信息项目；在家用电器中，微处理器可用作洗衣机控制器、微波电灶控制器、电视机自动选台、录音机自动选曲等控制元件。

§1-5 电子计算机系统的组成

电子计算机作为一种能思维的机器，长期以来曾被一种神秘的浓雾笼罩着，一旦人们拨开迷雾，便可清晰地看到它毕竟是人们长期生产实践和科学研究的产物，它所具有的神奇功能，正是人类赋于的。

一、电子计算机的组成

为了说明电子计算机的组成和工作过程，让我们先考察一下人运用算盘来算题的过程。

假设用纸、笔和算盘来计算 $17+14\times2=?$ 这样一道简单的题目，其计算步骤大致如下：

第一步：根据给定题目，确定计算步骤和方法，并把计算公式、解题步骤和原始数据写在纸上。

第二步：按先乘后加的原则，在算盘上进行计算。先求取中间结果 $14\times2=28$，写在纸上。然后在算盘上做加法，求得最后结果 $17+28=45$。

第三步：把最后结果 45 写在纸上，计算完毕。

从上述计算过程可知，为完成这一道算题，必须具备：

1．运算装置用来进行运算，这里是算盘。

2．记忆装置用来存放题目、计算步骤、原始数据、中间结果和最后结果。这里是纸和人的记忆力。

3．控制装置用来控制整个计算过程，这里是人的大脑。

4．输入输出装置用来输入原始数据的信息，输出运算的结果，这里是眼、手和笔。

电子计算机就是模拟上述解题过程的现代化的电子设备。因此，它也应具备与上述功能相应的几个组成部分，即：

1．运算器(Arithmetical Unit)用来快速地进行各种算术运算和逻辑运算。

2．存贮器(Memory；Storage)用来存贮、记忆大量的程序和数据。

3．控制器(Control Unit)在程序控制下，用来统一指挥整个计算机自动地有节奏的操作。

4．输入输出设备(Input/Output equipment，简称 I/O 设备)用来输入程序和原始数据，输出运算的结果。

通常，我们把组成计算机的上述五个基本部件，统称为硬件(Hardware)。其中，把运算器、控制器和主存贮器合称为主机 (Main frame)；把运算器与控制器合称为中央处理器 CPU (Central Processing Unit)；把除主机以外的各种输入输出设备和外存贮器统称为外部设备 (External equipment)。

由运算器、控制器、存贮器、输入输出设备以及沟通主机与外部设备之间的通道 (Channel) 就组成了一台完整的电子计算机硬件系统，其组成原理框图如图 1-1 所示。计算机硬件系统除上述五个基本组成部件外，还必须有电源装置、控制台或控制面板等。

倘若要实现生产过程的控制，还应包含过程输入输出通道。通常认为，在计算机系统中，除主机以外的其它围绕主机而设置的各种设备，即除包含 I/O 设备和外存贮器外，还包括过程

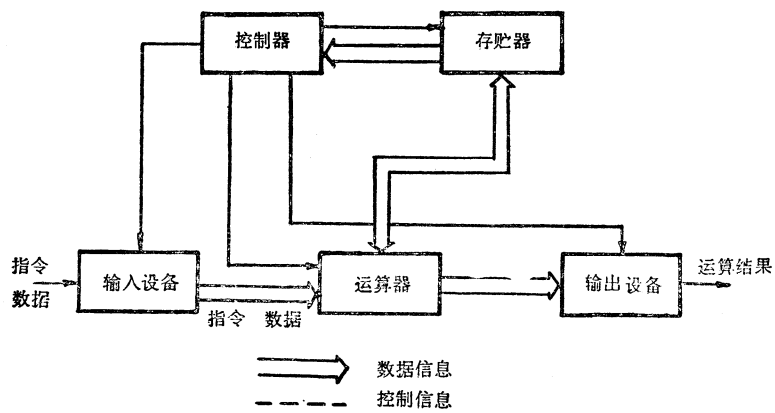


图 1-1 电子计算机组成原理框图

输入输出通道等，统称为外围设备(Peripheral equipment)，简称外设。

二、程序存贮和程序控制原理

假如计算机只有硬件，那末，它还只是具备了计算的可能性。欲把这种可能性变成现实，还必须依靠软件的配合。这就是说，为了让计算机能按照人们的意图进行运算，就必须事先把计算方法和解题步骤翻译成机器能够理解的语言，即二进制代码形式的机器语言。人们使用机器语言来编制解题步骤，这就是编制程序的过程。接着，把编好的程序连同原始数据，通过输入设备存入计算机的存贮器中。然后，启动计算机，计算机便在程序控制下，按人的意图自动地进行操作，直至全部计算完毕后，通过输出设备送出结果。以上就是迄今为止，电子数字计算机所共同遵循的程序存贮和程序控制的原理。这种原理是1945年冯·诺依曼提出的，故又称为冯·诺依曼(John Von Neumann)型计算机原理。

相对于硬件而言，我们把各种各样的程序(Program)，统称为软件(Software)。计算机只有同时具备硬件和软件，才能自动地、快速地工作，从而完成人们所要求的任务。

现在，我们仍以 $17+14\times2$ 这道题为例，说明计算机的工作过程：

第一步：由输入设备将事先编制好的计算步骤(即计算程序)，原始数据17、14、2输入到存贮器存放起来。

第二步：启动计算机，在控制器的指挥下，计算机按计算程序自动地进行操作。

1．从存贮器取出乘数2，送到运算器中。

2．从存贮器取出被乘数14到运算器，进行 14×2 的乘法操作，在运算器中求得中间结果28。

3．将运算器中的中间结果28送到存贮器暂存。

4．从存贮器中取出被加数17，送到运算器中。

5．从存贮器中取出加数到运算器，进行 $17+28$ 的加法操作，在运算器中求得加法结果45。

6．将运算器中的最后结果45存入存贮器。

第三步：由输出设备将最后结果45打印在纸上。

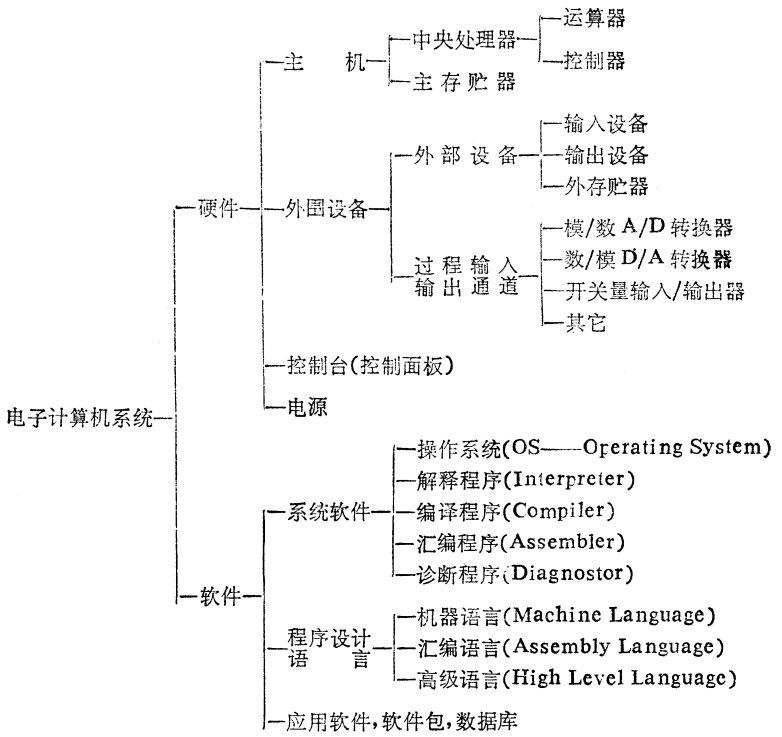
第四步：停机。

三、电子计算机系统的组成

前已述及，一个完整的电子计算机系统包括硬件和软件两大部分。其中，硬件是指组成计算机的任何机械的、磁性的、电子的装置和部件，又称机器系统，它们是组成计算机的物质基础；而软件是指为了方便用户和充分发挥计算机效能的各种程序的总称，又称程序系统，它是属于信息性的东西，是组成计算机的上层建筑。现将一个完整的电子计算机系统的组成归纳如第13页所示。

§1-6 微型计算机系统的组成

微型计算机是在小型计算机的基础上，借助于大规模集成电路技术而发展起来的。它在结构原理上同一般电子计算机有许多共性，但还有其特殊性。



微型计算机也是由硬件和软件两大部分组成的。与大、中、小型机相比，在微型计算机中，硬件和软件更加密不可分。发展带有软件硬化的微型计算机系统已成为一个重要的发展方向。所谓软件硬化，就是把软件功能固化于硬件中，故又称为固件(firmware)。

微型计算机系统硬件的组成框图如图 1-2 所示。它是由微处理器(CPU)，存贮器和输入/

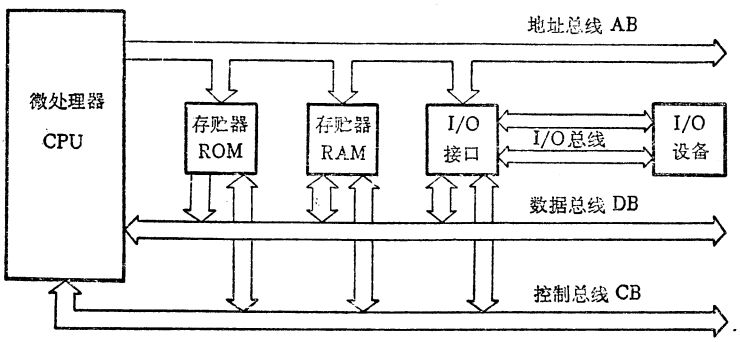
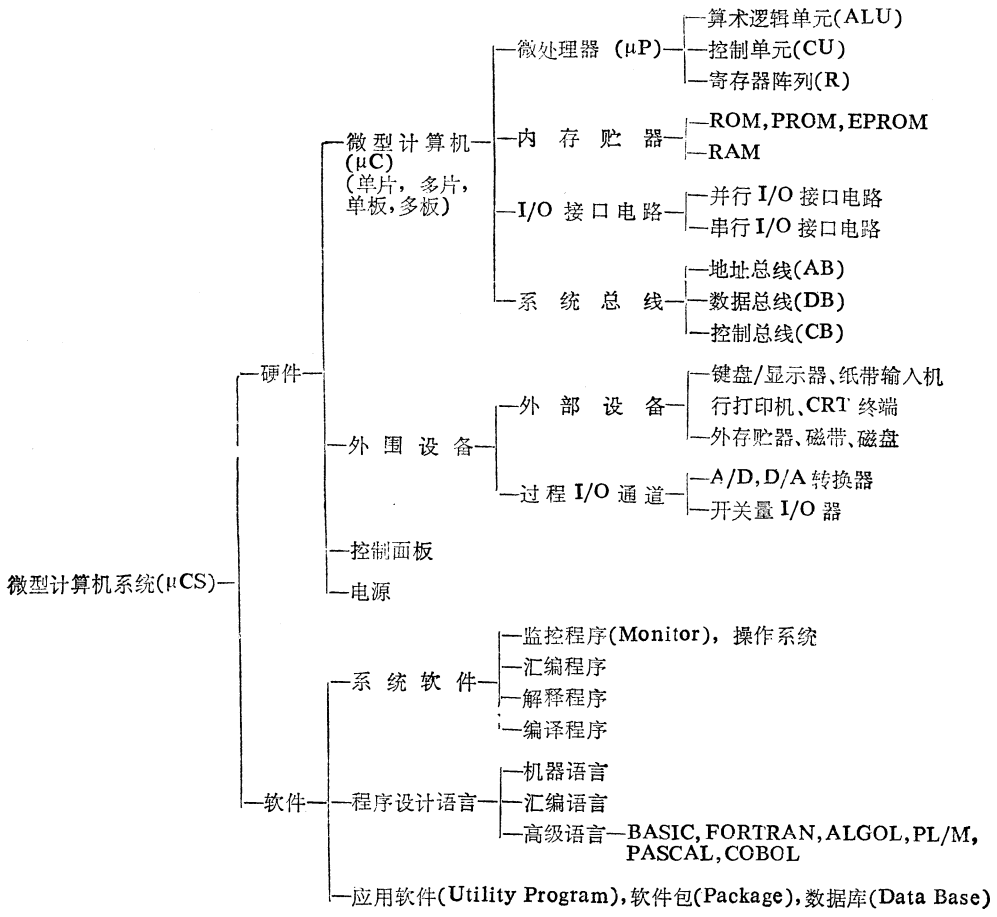


图 1-2　微型计算机系统硬件组成框图

输出接口电路(I/O 接口电路)组成，通过 I/O 接口电路再与外部设备 (I/O 设备) 相连接。它们相互之间通过三组总线 (BUS)——地址总线(Address Bus)，数据总线(Data Bus) 和控制总线(Control Bus)来连接。

存贮器分只读存贮器 (ROM——Read Only Memory) 和随机存取存贮器或读写存贮器 (RAM——Random Access Memory)两大类。

一个完整的微型计算机系统组成如下：



§1-7 计算机的软件

从广义角度来说，软件包括各种程序设计语言、系统软件、应用软件和数据库等。

一、程序设计语言(Programming Language)

程序设计语言是指用来编写程序的语言。通常分为机器语言、汇编语言和高级语言三类。

1．机器语言

机器语言是一种用二进制代码"0"或"1"形式来表示的、能够被计算机识别和执行的语言。这种机器语言对各种不同的计算机来说一般是不相同的。每个计算机都有自己的指令系统(Instruction sets)，即计算机所能识别和执行的指令的集合。所谓指令是一种规定中央处理器执行某种特定操作的命令，通常一条指令对应着一种基本操作。每台计算机的指令系统就是该机器的机器语言。显然，用这种二进制代码形式表示的机器语言来编写程序，直观性差、容易出错，而且烦琐费时。

2．汇编语言(初级语言)

汇编语言是一种用助记符来表示的面向机器的程序设计语言。这种语言比较直观，而且容易记忆和检查。但是，计算机还不能直接识别用汇编语言编写的程序——源程序 (Source

Program)。源程序要经过汇编程序(Assembler)的加工和翻译,才能变成机器语言表示的目标程序(Object Program)。

由于汇编语言的语句与机器指令是一一对应的,对于不同的计算机,针对同一问题所编的汇编语言源程序是互不通用的。因此,汇编语言虽比机器语言有所进步,但仍然比较烦琐费时。

3. 高级语言(又称为算法语言或编译语言)

为了从根本上克服初级语言的弱点,使程序设计语言适合于描述各种算法(计算机的早期应用是数值计算),而不依赖具体计算机的结构和指令系统,以便于推广计算机的应用,于是五十年代中期以 FORTRAN 语言为代表的各种计算机高级语言就应运而生。这里把常用的几种简述如下:

(1) FORTRAN 是 FORmula TRANslator 的缩写。它是美国 IBM 公司为计算机研制的第一种高级语言,这是一种成熟的适合于科学计算的公式翻译语言,特别是它的标准程序库十分丰富,至今它仍是国际上最流行的数值计算语言。FORTRAN 语言经过不断修改,发展,现在 FORTRAN-IV 已成为国际标准,1978 年又修改成为 FORTRAN-77 新标准。

(2) COBOL(Common Business Oriented Language)是面向商业的通用语言。它起源于美国,现已被广泛用于数据处理、情报检索等计算机联机系统,几乎遍及商业、银行、交通等行业。

(3) BASIC(Beginners All-Purpose Symbolic Instruction Code) 是一种小型通用的交互式会话语言。它的特点是简单易学,功能较强,在小型机、微型机上也易于实现,因此得到了广泛应用。它使用解释程序,比编译程序简单,占用的存贮空间较小。如为微型机而设计的微 BASIC,其解释程序占用的内存容量可以小到 2K 字节。扩充的微 BASIC 可超过 12K 字节,一般都以 ROM 芯片形式销售。

(4) PASCAL 是 1970 年研制完成的以著名数学家 Blaise Pascal 命名的高级语言,它吸取了六十年代有关算法语言的优点,增加了较丰富的构造数据结构的方法,故又称为结构程序设计语言。

(5) PL/1(Programming Language/one)是 1964～1969 年期间发展起来的第二代高级语言,它是一种汇集型的多用途的通用语言,兼有 FORTRAN-IV、COBOL 和 ALGOL 60 的特色,还包括了实时控制的功能等,表达能力强,内容丰富。目前在国外已广泛用于科学计算、过程控制、数据处理等领域。Intel 公司专为 8080 μC 研制的微 PL/1-PL/M, 以及 1978 年又为 8086 研制的 PL/M-86, 都是 PL/1 的偏子集。

(6) ALGOL(ALGOrithmic Language) 是一种适合于描述数值计算过程的高级语言。具有代表性的有 ALGOL-60 与 ALGOL-68。它们曾在我国得到广泛的应用。

除了以上几种国内外常用的计算机高级语言外,近年来在国外开始流行的 LISP、FORTH 以及 C 语言也正在引起人们的关注。LISP 是一种适用于表处理和人工智能的语言,它与其它大多数程序设计语言的风格迥然不同,是一种具有自编译能力的函数构造式的语言。FORTH 的英语词意是"向前",即"向着应用目标前进"的意思,它既是一种高级语言,又是一种汇编语言,很适合于过程控制、企业管理等场合,是一种富有生命力的语言。

C 语言是七十年代初期由贝尔实验室的 D. M.RITCHIE 博士作为 UNIX 操作系统的一部分提出来的。它适用于描述操作系统、编译程序等软件。C 语言是一种面向结构的程序设计语言,在时间、空间和效率上它都能和汇编语言程序媲美。自 1979 年以来,C 语言就开始在

带软盘的微型机中使用。最近，它已逐渐成为微型机的主要语言。

现在还处于探索中的高级语言 Ada 语言，被称为八十年代的计算机语言。它是在PASCAL语言的基础上发展起来的，作为编制实时系统的语言时比 PASCAL 更胜一筹。它有两个突出的特征：一是实现了程序设计抽象化，使用的全是抽象的数据类型。二是通过使用多任务和保护命令而实现了实时并行处理。目前它代表了软件工程技术的最新水平。

综上所述，汇编语言和高级语言各有所长，也各有所短，因此，我们在应用中应当根据具体的场合，扬长避短，选择适当的程序设计语言。

二、系统软件(System Software)

系统软件是指为了方便用户和充分发挥计算机效能，向用户提供的一系列软件，包括监控程序、操作系统、汇编程序、解释程序、编译程序、诊断程序及程序库等。

1．监控程序(Monitor)，又称管理程序(Supervisor)。它是为充分发挥计算机的效能、合理使用资源、方便用户而设计的一套程序。其主要功能有：对主机和外部设备的操作进行合理的安排；按轻重缓急处理各种中断；接受分析各种命令；实现人机联系；控制源程序的编译、编辑、装配、装入、启动等等。目前在单板微型机中，一般都配有 1～2 K 字节以上的监控程序，如 SDK-85、Z 80 Starter System Kit 等单板微型机的监控程序为 2 K 字节，SDK-86 单板微型机的监控程序为8 K 字节。

2．操作系统

操作系统是在管理程序的基础上，进一步扩充许多控制程序所组成的大型程序系统。其功能主要有：组织整个计算机的工作流程，管理和调度各种软硬件资源，检查程序和机器的故障，实现计算机资源供多个用户共享等等。从广义角度来说，操作系统应包括引导程序、监控程序、输入/输出驱动程序、连接程序、编辑程序、汇编程序、解释程序、编译程序等等。分时系统(Time-Sharing System)是操作系统的一种类型，它能使一台计算机几乎同时地为许多终端用户服务。由于对每个用户都保证有足够快的响应时间，因而可以实现多用户并行工作。一个多用户分时操作系统能够支持多达 32 个用户终端，从而有效地实现计算机资源的多用户共享。操作系统通常都存放在软磁盘中，容量为十几 K 到几十 K 字节。

目前比较著名的微型机操作系统有：CP/M、ISIS、CDOS、UNIX 等。

3．汇编程序

汇编程序能把用汇编语言写成的源程序翻译成为机器语言的目标程序。它又分为两类：

(1) 自汇编程序。这是指在计算机上能直接把汇编语言的源程序翻译成为机器语言的目标程序，也称为驻留汇编程序。具有驻留汇编能力的微型机，需要足够的存贮容量来存放自汇编程序。

(2) 交叉汇编程序。这是指利用一台外围设备比较完善、存贮容量比较大、功能比较强的计算机作为主机，用主机常用的程序设计语言如 FORTRAN 语言或汇编语言来编写目标计算机的汇编程序，这样，用户使用目标计算机的汇编语言来编写的源程序，就可以借助于主机的汇编程序，翻译成为目标计算机的目标程序，然后再送入目标计算机执行。

4．解释程序

解释程序能把用某种程序设计语言写的源程序 (如 BASIC)，翻译成为机器语言的目标程序，并且每翻译一句，就立即执行一句，翻译完毕，程序也执行完毕。

5．编译程序

编译程序能把用高级语言编写的源程序，编译成某种中间语言(如汇编语言)或者机器语言的目标程序。

6．编辑程序(Editor)

编辑程序的功能是把多个模块程序连接成一个完整的程序，它可以增加、删除或替换程序中的某些段落。

7．诊断程序

诊断程序的功能是检查程序的错误和计算机的故障，并指示出错点等。

8．程序库(Routine library)

把常用的各种标准子程序如初等函数、数制转换程序、典型的计算程序等汇集在一起就构成了程序库，供解题程序用。程序库通常存放在磁带、磁盘中。

三、应用程序

应用程序是专门为解决某个应用领域里的具体任务，如进行生产过程控制或数据处理问题而编制的程序。随着计算机的广泛应用和普及，现已编制出许多应用程序，这些应用程序可依其功能组成不同的程序包(routine package)，这样便可减少大量重复劳动。

四、数据库与数据库管理系统

在数据处理系统中，需要处理大量的数据、检索和建立各种各样的表格，这些数据和表格按一定的形式和规律加以组织，建立数据模型，实行集中管理，这就建立了数据库。对数据库中的数据进行组织和管理的软件称之为数据库管理系统(Data Base Management System)。七十年代以来数据库技术发展十分迅速，并在信息处理，情报检索和各种管理系统中日益得到广泛的应用。

以上各种各样形式的程序统称为软件，丰富的软件是对计算机硬件功能的强有力的扩充，经过扩充以后的计算机系统如虎添翼，性能更强，可靠性更高，使用更方便。

§1-8 电子计算机的特点

一、电子计算机的特点

如前所述，电子计算机的特点可归纳如下：

1．运算速度快

最初的计算机运算速度是每秒几千次，而现代计算机已经达到每秒上亿次的运算速度。计算机的运算速度，通常用每秒钟能运算的次数来表示。由于执行不同的指令所需要的时间不同，因此如何计算运算速度就有不同的方法。一种是以最短指令的执行时间(如加法运算)为标准来计算的，例如 DJS-130 机的运算速度为 50 万次/秒，就是指做定点加法而言的；第二种是根据不同类型指令出现的频繁程度，乘上不同的系数，求得统计平均值，这时所指的速度是平均运算速度，这种方法称吉布森(Gibson)法；第三种是直接给出每条指令的执行时间和机器的主频。

2．精确度高

由于计算机采用二进制数字表示方法，从理论上来说，有效位数越多，其精确度也就越高。

在计算机中，一组二进制数码作为一个整体来处理或运算的，称为一个计算机字，简称字(Word)。计算机的每个字所包含的位数称为字长(Word Length)，它表示计算机的计算精度。巨型机或大型机字长一般为32～64位；中型机字长多为32位；小型机字长一般为16位～32位；微型机字长一般为4～16位，也有32位，目前8位的占多数。通常计算机能进行双倍字长或多倍字长运算。

3．具有"记忆"能力

计算机的存贮器具有存贮、记忆大量信息的功能。存贮容量(Memory capacity)是评价计算机性能的一个重要指标，它是指存贮器可以容纳的二进制数的信息量。存贮容量可以按字长为单位，或按字节(Byte)为单位来计算。在以字节为单位时，约定以8位二进制数为一个字节。每1024个字节(或字)，简称为1K字节(或字)。大型机或巨型机主存贮器容量一般为2～3兆字节；中、小型机主存贮器容量一般为几十K字到一兆字；8位微型机主存贮器容量通常为64K字节，16位微型机主存贮器容量可超过1兆字节。

4．具有逻辑判断能力

计算机借助于数理逻辑和布尔代数，可以进行逻辑推理和判断，从而使计算机"智能化"。

5．程序控制的自动化操作，如前所述

6．通用性强

计算机能广泛地满足各种应用场合。如科学计算、数据处理、自动控制、仪器仪表智能化等等。

二、微型计算机的特点

微型计算机是在小型机的基础上发展起来的。它与普通计算机并没有本质上的区别，所不同的仅在于处理数据的能力，运算速度及系统规模等方面。但是，微型机不仅是小型机体积上简单地缩小，而且在逻辑结构、电路设计技巧以及工艺水平等方面都有新的发展，它除了具有电子计算机的一般特点以外，还具有下列主要特点：

1．体积小、功耗低

采用大规模集成电路的微处理器其大小仅几个mm^2，组成微型计算机后，也只有一块插件板大小。它与功能相近的小型机相比，体积小1～2个数量级。它的功耗一般仅需几瓦至十几瓦，而且散热、冷却问题也易于解决。这个特点使以往无法装备计算机的小型设备、终端设备和家用电器设备等也可装上微型机了，从而大大开拓了计算机的应用范围。

2．价格低、产量大

微型机的价格比相应的小型机低1～2个数量级，而且批量越大，价格就越便宜。一般8位微处理器的价格仅几个美元，组成单板微型机后的价格也不过几百美元，这个特点使得一些中、小型和廉价的设备都能用上微处理器和微型机。

3．可靠性高

由于微型机采用了大规模集成电路技术，使系统内的组件数大幅度下降，相应的焊接点数也比采用中、小规模集成电路的小型机减少两个数量级以上。因此，整机的可靠性显著地提高了。此外，由于大规模集成电路工艺的进展，目前LSI芯片的损坏率仅为0.0005/千小时，因而，通常微型机都可工作数千小时不发生故障。

4．灵活性、适应性强

由于微型机体系结构采用总线结构形式，因而其结构非常机动灵活，易于构成形式多样的系统，而且可以根据需要进一步扩展。同时，由于构成微型机的基本部件大多数已系列化、标准化，这更增强了微型机的通用性。更为重要的是微型机具有可编程序的特点，这使得一个标准的微型机系统，仅通过改变程序就能执行不同的任务，而不必对硬件重新进行设计。这一特点不仅大大增强了微型机的通用性，而且使其研制周期大为缩短。

就目前广泛应用的8位微型机而言，还存在一些不足之处，如字长较短、速度较慢、存贮容量也较小、外围设备不够齐全、软件也不如小型机成熟、丰富等，随着性能更强的第三代微型机的出现，上述这些缺陷已经逐步得到克服。

§1-9 微型计算机的分类

微型计算机可以从各种角度进行分类：

一、按微型计算机CPU的字长分类

从微型计算机的发展史来看，其字长是随着大规模集成电路工艺的发展而增加的。目前已研制成的微型计算机按其字长可分为五种类型：

1．4位并行的微型计算机

1971年末美国Intel公司为台式计算器而设计的MCS-4是世界上第一台专用的4位并行微型机。它的功能简单，能方便地处理二-十进制(BCD)数字，而且具有用户可编程序的特点，适用于台式计算器、简单控制器及家庭电器等。近年来新出现的4位微型机多为单片或4位位片式的，其应用面广、价格低，因而需求量也随之上升。

2．8位并行的微型计算机

由于4位微型机运算速度慢、功能简单，限制了它的应用范围，因此在1972年Intel公司又为生产CRT终端而研制成功8位并行的MCS-8微型机，引入了中断功能，基本上具备了计算机的结构形式。它能方便地处理字符(如ASCII码)数据，并能容易地与各种I/O设备连接。但是，由于8位并行的MCS-8微型机采用PMOS工艺，速度太慢，仍然限制了它的应用范围。为此，Intel公司于1973年末又研制成MCS-80微型机，采用NMOS工艺，运算速度提高一个数量级。此后，各家公司都竞相制造8位微型机，使之成为当前微型机应用中的主流。目前，8位微型机已广泛地应用于工业控制、智能终端、仪器仪表及数据处理等场合。

3．12位并行的微型计算机

国外生产的12位微型机寥寥无几，典型的是美国Intersil公司生产的IM 6100和日本东芝公司的TLCS-12 A系列(CPU T 3100)。主要用于填补8位微型机和16位微型机之间的空白，其应用面较窄，生产量也较少。

4．16位并行的微型计算机

高档的16位微型机，采用短沟道HMOS工艺，运算速度大幅度提高，其性能已达到或超过流行的中、低档小型机。目前，这类微型机主要用于实时数据处理、实时控制等场合。也可以进一步联机操作，组成多微处理机系统和局部网络。

5．位片式微型计算机

位片式微型机采用双极型工艺，处理速度极高，比一般MOS芯片要高出1～2个数量级。

常见的位片式微型机有1位、2位、4位等，用户可以根据需要组成一定字长的微型机。如2位位片的Intel 3000系列，4位位片的美国AMD公司的Am 2900系列等，它们主要用于高速的外围控制器(如磁盘控制器)、通信控制器和实时控制系统等。

二、按微型计算机系统的利用形态分类

1．单片微型计算机(见附录5表A 5-2和表A 5-3)

2．微型计算机套件(Kit)

这是由生产厂提供的微处理器、存贮器片、I/O接口电路片、印刷板、必要的阻容元件等配套器件。由用户自己组装成单板微型计算机。如SDK-80、SDK-85、Z 80 STARTER KIT等。

3．单板微型计算机

这是在一块印刷电路板上，把微处理器、一定容量的存贮器片和I/O接口电路片等LSI器件组装成的微型计算机。通常在其ROM或EPROM芯片中，还固化有容量不大的监控程序，并且一般还配备有键盘和显示器装置(见表17-7)。

4．微型计算机装置

这是把单板微型计算机、若干块存贮器、I/O接口插件板、控制面板、电源等组装在一个机箱内的完整的微型计算机系统。一般还配备有盒式磁带机(Cartridge tape unit)、软磁盘(Floppy Disk)驱动器(Drive)、键盘显示终端(CRT)、行式打印机(Line Printer)等外部设备，并且有丰富的软件支持。如美国的CROMEMCO System 3，其CPU为Z 80 A，在主机中有64 KB的RAM。配备的外设有2～4个8″(或5″)的软磁盘驱动器、CRT终端和点阵式行打印机等。在软件方面，配有CDOS磁盘操作系统，并配有Z 80浮动宏汇编、扩展BASIC、FORTRAN-IV等高级语言。

随着八十年代办公室自动化(OA)技术的发展，个人计算机(Personal Computer)也获得蓬勃的发展。典型的产品如美国的IBM-PC1和IBM-PCXT等。它们的CPU采用先进的Intel-8088(16位)。

三、按微型计算机制造工艺分类

从微型计算机的发展史来看，它是与半导体集成电路工艺技术的发展分不开的。到目前为止，制造微型机电路的器件可以分为两大类：一类是由MOS型(Metal-Oxide-Semiconductor)即金属氧化物半导体场效应晶体管构成的LSI；另一类是由双极型晶体管构成的LSI。目前，大多数微型机是采用MOS工艺。

1．MOS型器件

MOS型工艺主要有三种：

(1) PMOS型(P-channel MOS)

PMOS工艺的特点是集成度高、工艺成熟、价格低，曾被广泛地应用于第一代微处理器的制造工艺中。但是其速度慢，而且难于有效地与TTL(晶体管-晶体管逻辑)I/O接口电路相兼容，因而很快地被NMOS工艺所取代。

(2) NMOS型(N-channel MOS)

NMOS工艺的特点是集成度高、速度较快(比PMOS快5～10倍)、价格较低，而且信号电平可与TTL电路兼容，因此使它成为目前微型机制造工艺中应用最广的工艺技术。第二代微

处理器主要采用 NMOS 工艺制造。

1978 年以来研制成功的高性能的 16 位微处理器即第三代微处理器，采用高密度短沟道的 NMOS (即 HMOS)工艺，如 MC 68000 采用 HMOS 工艺，在一块 6.25×7.14 平方毫米的电路芯片上集成了大约 68000 个晶体管，最高时钟频率达 12 MHz。目前 HMOS 工艺已普遍受到重视。

(3) CMOS 型(互补型 MOS)

CMOS 工艺的特点已如前述，不再重复。近年来 CMOS 工艺发展迅速，应用日益广泛，大有取代 NMOS 优势的趋势。

2．双极型器件

双极型工艺的特点是速度快(其时钟频率可高达 10～20 MHz)，但功耗大，集成度低，价格较贵，主要用于位片式微处理器中。

双极型器件主要有三种：

(1) TTL(Transistor-Transistor Logic)晶体管-晶体管逻辑电路

为了消除饱和延迟对 TTL 开关速度的不利影响，现在普遍采用肖特基钳位电路 STTL (Schottky TTL)。AMD 2900、Intel 3000 系列的微处理器都采用 STTL 工艺。

(2) ECL(Emitter-Coupled Logic) 电流开关逻辑或发射极耦合逻辑电路

这是目前速度最快的一种工艺，但是其功耗大，价格昂贵，主要用于超高速的应用场合。Motorola 公司的 MECL 10800 位片式微处理器就是采用 ECL 工艺制造的。

(3) I^2L(Integrated Injection Logic) 集成注入逻辑电路

I^2L 工艺的特点是工艺简单、集成度高、功耗较低、速度低于 TTL，但作为一种新工艺，目前尚不够成熟。采用 I^2L 工艺制作的产品有 TEXAS INSTRUMENTS 公司的 SBP 0400 (4 位)和 SBP 9900 (16 位)。近来又出现了 I^3L(Isoplanar Integrated Injection Logic)等平面集成注入逻辑电路，速度比 I^2L 快，但工艺复杂。

综上所述，PMOS 工艺成熟，但其性能在 MOS 技术中最差，只是工艺相对简单些。目前，NMOS 工艺在通用微型机中占绝对优势，高档产品几乎都采用 NMOS 工艺。近来，特别是高密度短沟道 NMOS(即 HMOS) 工艺，受到普遍关注。

CMOS 以功耗低，抗干扰能力强为特征，近来由于集成技术的突破，CMOS 已在集成度和速度上向 NMOS 逼近，大有后来居上的趋势。

I^3L 以高速为特点，但因其工艺比 I^2L 复杂，故尚未普遍应用。

ECL 速度最高，但功耗最大，价格昂贵。

总之，MOS 技术的应用日益广泛，而双极技术日趋缩小。NMOS 仍是主流，HMOS 倍受重视，CMOS 后来居上，新工艺日新月异。

表 1-3 列举了国外典型的微处理器及单片微型机的性能比较表。附录 5 部分还列出了目前国外各种新型微处理器的性能比较表。

处理器主要采用 NMOS 工艺制造。

1978 年以来研制成功的高性能的 16 位微处理器即第三代微处理器，采用高密度短沟道的 NMOS (即 HMOS)工艺，如 MC 68000 采用 HMOS 工艺，在一块 6.25×7.14 平方毫米的电路芯片上集成了大约 68000 个晶体管，最高时钟频率达 12 MHz。目前 HMOS 工艺已普遍受到重视。

(3) CMOS 型(互补型 MOS)

CMOS 工艺的特点已如前述，不再重复。近年来 CMOS 工艺发展迅速，应用日益广泛，大有取代 NMOS 优势的趋势。

2．双极型器件

双极型工艺的特点是速度快(其时钟频率可高达 10～20 MHz)，但功耗大，集成度低，价格较贵，主要用于位片式微处理器中。

双极型器件主要有三种：

(1) TTL(Transistor-Transistor Logic)晶体管-晶体管逻辑电路

为了消除饱和延迟对 TTL 开关速度的不利影响，现在普遍采用肖特基钳位电路 STTL (Schottky TTL)。AMD 2900、Intel 3000 系列的微处理器都采用 STTL 工艺。

(2) ECL(Emitter-Coupled Logic) 电流开关逻辑或发射极耦合逻辑电路

这是目前速度最快的一种工艺，但是其功耗大，价格昂贵，主要用于超高速的应用场合。Motorola 公司的 MECL 10800 位片式微处理器就是采用 ECL 工艺制造的。

(3) I^2L(Integrated Injection Logic) 集成注入逻辑电路

I^2L 工艺的特点是工艺简单、集成度高、功耗较低、速度低于 TTL，但作为一种新工艺，目前尚不够成熟。采用 I^2L 工艺制作的产品有 TEXAS INSTRUMENTS 公司的 SBP 0400 (4 位)和 SBP 9900 (16 位)。近来又出现了 I^3L(Isoplanar Integrated Injection Logic)等平面集成注入逻辑电路，速度比 I^2L 快，但工艺复杂。

综上所述，PMOS 工艺成熟，但其性能在 MOS 技术中最差，只是工艺相对简单些。目前，NMOS 工艺在通用微型机中占绝对优势，高档产品几乎都采用 NMOS 工艺。近来，特别是高密度短沟道 NMOS(即 HMOS) 工艺，受到普遍关注。

CMOS 以功耗低，抗干扰能力强为特征，近来由于集成技术的突破，CMOS 已在集成度和速度上向 NMOS 逼近，大有后来居上的趋势。

I^3L 以高速为特点，但因其工艺比 I^2L 复杂，故尚未普遍应用。

ECL 速度最高，但功耗最大，价格昂贵。

总之，MOS 技术的应用日益广泛，而双极技术日趋缩小。NMOS 仍是主流，HMOS 倍受重视，CMOS 后来居上，新工艺日新月异。

表 1-3 列举了国外典型的微处理器及单片微型机的性能比较表。附录 5 部分还列出了目前国外各种新型微处理器的性能比较表。

表 1-3 典型微处理器及单片微型机的性能比较表

处理位数		产品名称	工艺	引线数	指令数	基本指令执行时间(μs)	时钟频率(MHz)	发表时间(年)
4 位		Intel 4004	PMOS	16	46	10.8	0.75	1971
		Intel 4040	PMOS	24	60	10.8	0.75	1974
		Rockwell PPS-4	PMOS	42	50	5	0.2	1974
		TI TMS-1000	PMOS	28	43	12	0.5	1975
8 位		Intel 8008, 8008-1	PMOS	18	48	20, 12.5	0.5, 0.8	1972
		Intel 8080A, 8080A-1, A-2	NMOS	40	72	2, 1.5, 1.3	2, 2.6, 3	1974
		Motorola M6800, A, B	NMOS	40	72	2	1, 1.5, 2	1974
		Rockwell PPS-8	PMOS	42	109	4	0.25	1975
		Fairchild F-8/3870	NMOS	40	74	2	2	1976
		Zilog Z80, Z80A	NMOS	40	150	1.6, 1	2.5, 4	1976
		Intel 8085A, 8085A-2	NMOS	40	74	1.3, 1	3, 4	1977
		Rockwell 6502	NMOS	40	56	2	1, 2	1977
		Intel 8048/8748	NMOS	40	90	2.5	2	1977
		Motorola 6801	NMOS	40	82	2	2	1978
		Zilog Z-8	NMOS	40	96	1.5	4	1978
		Motorola M6809, A, B(E)	NMOS	40	59	2, 1	1, 1.5, 2	1979
12 位		Intelsil IM6100	CMOS	40	80	5, 2.5	4, 8	1975
		东芝 TLCS 12A(CPU T3190)	PMOS	36	108	6.6	1.2	1975
16 位		Texas Inst TMS9900	NMOS	64	69	2.7	3.3, 4	1976
		General Instrument CP1600	NMOS	40	87	2.4	2, 4	1978
		Intel 8086, 8086-2, 8086-1	HMOS	40	133	0.6, 0.3	5, 8, 10	1978
		Zilog Z8000	HMOS	40/48	110	0.67	4	1978
		Motorola MC68000	HMOS	64	56	0.5, 0.4	8, , 10, 12	1979
位片	2 位	Intel 3000	STLL	28	40	0.125	6.6, 10	1973
	4 位	Am 2900	STTL	40/48	16/25	0.1	10, 20	1975
		TI SBP 0400	I^2L	40	45	0.53, 0.11		1975
		Motorola MC10800	ECL	48	100	0.075, 0.04	18, 20	1976

第二章　计算机的运算基础

电子计算机是一种能对数字进行运算和处理的电子设备。通常，在计算机中，数字是以一串“0”或“1”的二进制代码来表示的，这是一种计算机唯一能够识别的机器语言。换言之，所有需要计算机加以处理的数、字母、符号都必须采用二进制编码来表示，它有一整套独特的运算规则。本章主要介绍二进制数包括其它与计算机密切相关的数制以及各种数制的运算规则。

§2-1　数　　制

按进位的方法进行计数，称为进位计数制。在日常生活中，我们最熟悉最常用的是十进制数，还有十二进制、十六进制、六十进制等。在计算机中，常用的是二进制、八进制、十六进制。下面分别作简要介绍。

一、十进制数制(Decimal number system)

其主要特点是：

1．它有十个不同的数字符号(又称数码)，即 0、1、2、3、4、5、6、7、8、9。

2．它是逢“十”进位的。同一数码处于不同的数位（称为“权”），其代表的意义是不同的。例如 555.55 这个数中，小数点左边的第一位“5”代表个位，其值为 5×10^0；左边第二位“5”代表十位，其值为 5×10^1；左边第三位“5”代表百位，其值为 5×10^2；小数点右边的第一位“5”，其值为 5×10^{-1}；右边第二位“5”，其值为 5×10^{-2}。因此这个数可写成：

$$555.55=5\times10^2+5\times10^1+5\times10^0+5\times10^{-1}+5\times10^{-2}$$

一般地说，任意一个十进制数 N 都可以表示为：

$$N=\pm(K_{n-1}\times10^{n-1}+K_{n-2}\times10^{n-2}+\cdots+K_1\times10^1+K_0\times10^0+K_{-1}\times10^{-1}$$

$$+K_{-2}\times10^{-2}+\cdots+K_{-m}\times10^{-m})=\pm\sum_{i=-m}^{n-1}(K_i\times10^i)$$

其中，K_i 表示第 i 位的数码，可以是 0～9 十个数字符号中的任何一个，由具体的数 N 来确定；m、n 为正整数，n 为小数点左边的位数，m 为小数点右边的位数。10^{n-1}、$\cdots10^1$、10^0、10^{-1}、$\cdots10^{-m}$ 称为十进制数的“权”，而“10”为十进制数的基数（即在该计数制中可能用到的数码的个数），它表示“逢十进一”这种计数制即为十进制。

二、二进制数制(Binary number system)

其主要特点是：

1．它只有两个不同的数码，即“0”和“1”。

2．它是逢“二”进位的。如对于十进制数 1＋1＝2，而对于二进制数 1＋1＝10，这里逢二进一变成了“10”。例如，二进制数$(1111.11)_2$可以化成如下十进制数：

$$(1111.11)_2 = 1\times2^3+1\times2^2+1\times2^1+1\times2^0+1\times2^{-1}+1\times2^{-2} = 8+4+2+1+0.5+0.25$$
$$= (15.75)_{10}$$

一般地说，任意一个二进制数 N，都可以表示为：

$$N = \pm(K_{n-1}\times2^{n-1}+K_{n-2}\times2^{n-2}+\cdots+K_1\times2^1+K_0\times2^0+K_{-1}\times2^{-1}+K_{-2}\times2^{-2}$$
$$+\cdots+K_{-m}\times2^{-m}) = \pm\sum_{i=-m}^{n-1}(K_i\times2^i)$$

其中，K_i 只能取 1 或 0，由具体的数 N 确定。m、n 为正整数。"2"是二进制的基数，它表示"逢二进一"，故称为二进制，见表 2-1。

表 2-1 各种数制的对照表

十进制数	十六进制数	八进制数	二进制数	十进制数	十六进制数	八进制数	二进制数
0	0	0	0000	9	9	11	1001
1	1	1	0001	10	A	12	1010
2	2	2	0010	11	B	13	1011
3	3	3	0011	12	C	14	1100
4	4	4	0100	13	D	15	1101
5	5	5	0101	14	E	16	1110
6	6	6	0110	15	F	17	1111
7	7	7	0111	16	10	20	10000
8	8	10	1000				

对于任意进位计数制，基数可用正整数 R 来表示，这时数 N 可表示为：

$$N = \pm\sum_{i=-m}^{n-1}(K_iR^i)$$

其中，m、n 为正整数；K_i 可以是 0、1、…、(R−1)中的任何一个；R 是基数。

三、八进制数制(Octal number system)

其主要特点是：

1．它有八个不同的数码，即 0、1、2、3、4、5、6、7。

2．它是逢"八"进位的。

如上所述，任意一个八进制数 N，可表示为：

$$N = \pm(K_{n-1}\times8^{n-1}+K_{n-2}\times8^{n-2}+\cdots+K_1\times8^1+K_0\times8^0+K_{-1}\times8^{-1}+\cdots+K_{-m}\times8^{-m})$$
$$= \pm\sum_{i=-m}^{n-1}(K_i\times8^i)$$

其中，K_i 可以是 0～7 中的任何一个，取决于数 N；n、m 为正整数；8 为基数，故称为八进制。

由于数 8 与数 2 之间的关系为：

$$8^1 = 2^3$$

因此 1 位八进制数相当于 3 位二进制数，它们之间的关系是对应的，见表 2-1。

根据这种对应关系，二进制与八进制之间的转换十分简单。例如，有一个二进制数(10100101.01011101)，只需从小数点开始分别向左和向右分为每 3 位一组用 1 位八进制数表示即可。

(0)10	100	101	·	010	111	01(0)
↓	↓	↓		↓	↓	↓
2	4	5	·	2	7	2

其中：对小数点左边不足 3 位的应在左边加 0；小数点右边不足 3 位的应在右边加 0，以凑成 3 位一组。

假如，要把八进制数转换为二进制数，只需用 3 位二进制数代换每 1 位八进制数即可。

例如，八进制数$(367.505)_8$可转换为二进制数$(011\ 110\ 111.101\ 000\ 101)_2$

四、十六进制数制(Hexadecimal number system)

其主要特点是：

1．它有 16 个不同的数码，即 0～9、A、B、C、D、E、F。它与十、二、八进制数之间的关系见表 2-1。

2．它是逢“十六”进位的。

如上所述，任意一个十六进制数 N，可表示为：

$$N=\pm(K_{n-1}\times16^{n-1}+K_{n-2}\times16^{n-2}+\cdots+K_1\times16^1+K_0\times16^0+K_{-1}\times16^{-1}+\cdots+K_{-m}\times16^{-m}$$

$$=\pm\sum_{i=-m}^{n-1}(K_i\times16^i)$$

其中，K_i可以是 0～F 之间的任意一个，取决于数 N；n、m 为正整数；“16”为基数，故称为十六进制。

由于数 16 与数 2 之间的关系为

$$16^1=2^4$$

因此 1 位十六进制数相当于 4 位二进制数，只要我们熟悉它们之间的这种关系，十六进制与二进制之间的转换也十分简单。例如，二进制数$(1111111000111.100101011)_2$可用以下方法转换为十六进制数：

(000)1,	1111,	1100,	0111	·	1001,	0101,	1(000)
↓	↓	↓	↓		↓	↓	↓
1	F	C	7	·	9	5	8

结果：$(1111111000111.100101011)_2=(1FC7.958)_{16}$

转换时应注意：最后一组如不足 4 位时需加“0”补齐。上例中最后一组为 1，若忘记补 0，可能视为十六进制数 1，而补 0 后，则成为十六进制数 8。

又如，十六进制数$(3AB.4A)_{16}$可转换为

3	A	B	·	4	A
↓	↓	↓		↓	↓
0011	1010	1011	·	0100	1010

结果：$(3AB.4A)_{16}=(001110101011.01001010)_2=(1110101011.0100101)_2$

目前，在微型计算机中普遍采用十六进制数表示，其原因有两个：

(1) 十六进制数与二进制数之间的转换十分方便；

(2) 目前微型计算机应用以 8 位机为主，其字长为 8 位，可用两位十六进制数表示，书写较短，便于阅读。

为了区别数制可在数的右下角注明数制，或者在数字后面加一字母。如 B(Binary) 表示二进制数制；Q(Octal)表示八进制数制；D(Decimal) 或不带字母表示十进制数制；H(Hexadecimal)表示十六进制数制。

§2-2 二进制的特点

从上一节可知：同一个数用二进制表示要比用十进制表示位数多得多。粗略估计前者约为后者的3倍左右。既然人们习惯于用十进制数，书写又方便，而二进制数书写起来位数太长，又不便于阅读，那么，为什么电子计算机中要采用二进制数？这是由二进制数制本身的特点决定的。因为计算机唯一能识别的是二进制数，这是问题的实质。八进制、十六进制数引入计算机，主要是为了书写方便，仅仅是一种手段而已。

二进制数制与其它数制相比有以下特点：

1．数的状态简单，容易表示

二进制只有“0”、“1”两种状态，它的物质基础就是具有两种稳态的元件。如晶体管的导通或截止，磁芯两个不同方向的磁化等。在计算机中，通常采用电平的“高”、“低”或脉冲的“有”、“无”来分别表示“1”和“0”。这种简单的状态工作可靠，抗干扰能力强。

2．运算规则简单

二进制运算的规则极为简单，故在计算机中实现二进制运算的线路也大大简化了。具体的运算规则以后再介绍。

3．可以节省设备

如果采用十进制数制表示0～9之间的数，需要1位，这1位共需十个设备状态。若采用二进制数制表示，需要4位，每位只需2个状态，总共8个设备状态。而且，这8个设备状态所能表示的数的范围可达0000～1111，即0～15。这说明，采用二进制数制可以节省设备。

4．可以使用逻辑代数这一数学工具对计算机逻辑线路进行分析和综合，便于机器结构的简化

尽管在电子计算机内部采用二进制进行操作。但是，对于人们来说，使用二进制并不方便，如书写冗长，阅读不便。为此，通常用八进制或十六进制的缩写方式。此外，人们还是习惯于用十进制，这就需要解决不同数制之间的相互转换问题。

§2-3 数制之间的转换

一、二进制数和十进制数之间的相互转换

1．二进制数转换为十进制数（即二换十）

这种转换十分简单，只要将二进制数按“权”展开相加即可。

【例】 $(11001.1001)_2 = 1\times2^4+1\times2^3+0\times2^2+0\times2^1+1\times2^0+1\times2^{-1}+0\times2^{-2}$

$+0\times2^{-3}+1\times2^{-4}=16+8+1+0.5+0.0625=(25.5625)_{10}$

二换十的规则就是要算出二进制数每一位为“1”时分别代表的十进制数，然后将所有转换的十进制数相加，即按“权”相加。最好能记住表2-2所列出的二进制数与十进制数之间的对应关系。

2．十进制数转换为二进制数（即十换二）

十进制数转换为二进制数，要把整数部分和小数部分分别换算，然后再相加即可。

（1）整数的十换二（又称除2取余法或辗转相除法）

表 2-2　二进制与十进制数的对照表

整数		小数	
二进制数	十进制数	二进制数	十进制数
1	$2^0=1$	0	0
10	$2\times2^0=2^1=2$	0.1	$2^{-1}=\frac{1}{2}=0.5$
100	$2\times2^1=2^2=4$	0.01	$2^{-2}=\frac{1}{4}=0.25$
1000	$2\times2^2=2^3=8$	0.001	$2^{-3}=\frac{1}{8}=0.125$
10000	$2\times2^3=2^4=16$	0.0001	$2^{-4}=\frac{1}{16}=0.0625$
100000	$2\times2^4=2^5=32$	0.00001	$2^{-5}=\frac{1}{32}=0.03125$
1000000	$2\times2^5=2^6=64$	0.000001	$2^{-6}=\frac{1}{64}=0.015625$
10000000	$2\times2^6=2^7=128$	0.0000001	$2^{-7}=\frac{1}{128}=0.0078125$
100000000	$2\times2^7=2^8=256$		
1000000000	$2\times2^8=2^9=512$		
10000000000	$2\times2^9=2^{10}=1024$		
100000000000	$2\times2^{10}=2^{11}=2048$		
1000000000000	$2\times2^{11}=2^{12}=4096$		
⋮			
10……00（n）	2^n，n是"1"后面"0"的个数	0.00……01（m）	2^{-m}，m是"1"前面"0"的个数（包括小数点前一位的"0"）

除 2 取余法即用 2 不断地去除要转换的十进制数，直至商为 0 为止。将所得各次余数，以最后余数为最高数位，依次排列，即得到所转换的二进制数。

【例】 求$(215)_{10}$的二进制数

解： 设$(215)_{10}=(K_{n-1}\cdots K_2K_1K_0)_2$

这里的关键在于决定$K_{n-1}\cdots K_2$、K_1、K_0的数值。

$$\because\ (215)_{10}=(K_{n-1}\cdots K_2K_1K_0)_2=K_{n-1}\times2^{n-1}+\cdots+K_2\times2^2+K_1\times2^1+K_0\times2^0$$

$$=2(K_{n-1}\times2^{n-2}+\cdots+K_2\times2^1+K_1\times2^0)+K_0 \qquad (2\text{-}1)$$

等式两边同时除以 2 得

$$\frac{215}{2}=(K_{n-1}\times2^{n-2}+\cdots+K_2\times2^1+K_1\times2^0)+\frac{K_0}{2}$$

或

$$107+\frac{1}{2}=(K_{n-1}\times2^{n-2}+\cdots+K_2\times2^1+K_1\times2^0)+\frac{K_0}{2} \qquad (2\text{-}2)$$

因为等式两边整数与小数必须对应相等，所以

$$107=K_{n-1}\times2^{n-2}+\cdots+K_2\times2^1+K_1 \qquad (2\text{-}3)$$

$$\frac{1}{2}=\frac{K_0}{2}\quad 或\quad K_0=1 \qquad (2\text{-}4)$$

由此可知，这一个十进制数除以基数 2，所得的余数即为二进制数的最低位K_0。

再将(2-3)式除以 2，所得的余数，即为二进制数的下一位K_1，以此类推则得K_{n-1}、$\cdots K_2$、K_1、K_0的数值。

其整个计算过程如下式表示：

$$
\begin{array}{r|l}
2 & 215 \\
2 & 107 \cdots\cdots \text{余 } 1 = K_0 \quad \text{最低位} \\
2 & 53 \cdots\cdots \text{余 } 1 = K_1 \\
2 & 26 \cdots\cdots \text{余 } 1 = K_2 \\
2 & 13 \cdots\cdots \text{余 } 0 = K_3 \\
2 & 6 \cdots\cdots \text{余 } 1 = K_4 \\
2 & 3 \cdots\cdots \text{余 } 0 = K_5 \\
2 & 1 \cdots\cdots \text{余 } 1 = K_6 \\
 & 0 \cdots\cdots \text{余 } 1 = K_7 \quad \text{最高位}
\end{array}
$$

$\therefore (215)_{10} = K_7K_6K_5K_4K_3K_2K_1K_0 = (11010111)_2$

如果你对十-二之间的关系很熟悉，也可用试减法进行十换二。即找一个与所要转换的十进制数最接近的 2 的乘方不断地进行试减，直至减到等于或小于 1 为止。

【例】 求$(92)_{10}$的二进制数

解： 用试减法

$$
\begin{array}{r}
92 \\
-64 \quad \rightarrow 2^6 \\
\hline
28 \\
-16 \quad \rightarrow 2^4 \\
\hline
12 \\
-8 \quad \rightarrow 2^3 \\
\hline
4 \\
-4 \quad \rightarrow 2^2 \\
\hline
0
\end{array}
$$

$$\therefore (92)_{10} = 64+16+8+4 = 1\times2^6+0\times2^5+1\times2^4+1\times2^3+1\times2^2+0\times2^1+0\times2^0$$
$$= (1011100)_2$$

(2) 小数的十换二(即乘 2 取整法)

乘 2 取整法即用 2 不断地去乘所要转换的十进制数，直至满足所要求的精确度或小数部分等于零为止。把每次乘积的整数部分，以最初整数为最高数位，依次排列，即得到所转换的二进制数。

【例】 求$(0.6875)_{10}$的二进制数

解： 设$(0.6875)_{10} = (0.K_{-1}K_{-2}\cdots K_{-m})_2 = K_{-1}\times2^{-1}+K_{-2}\times2^{-2}+\cdots+K_{-m}\times2^{-m}$ (2-5)

同样，关键在于求出 K_{-1}、K_{-2}、$\cdots K_{-m}$ 的数值。

将(2-5)式两边同时乘 2，则得

$$1.375 = K_{-1}+(K_{-2}\times2^{-1}+\cdots+K_{-m}\times2^{-m+1}) \tag{2-6}$$

因为等式两边整数与小数必须对应相等，所以

$$K_{-1} = 1$$

$$0.375 = K_{-2}\times2^{-1}+\cdots+K_{-m}\times2^{-m+1} \tag{2-7}$$

由此可知乘积的整数部分 1 即为 K_{-1}。

若将(2-7)式两边再乘以 2，则

$$0.75 = K_{-2}+K_{-3}\times2^{-1}+\cdots+K_{-m}\times2^{-m+2}$$

由此得出：

$$K_{-2} = 0$$

$$0.75 = K_{-3} \times 2^{-1} + \cdots + K_{-m} \times 2^{-m+2}$$

依此类推可逐个求得 K_{-1}、K_{-2}、K_{-3}、…、K_{-m} 的数值，其整个转换过程如下：

```
  0.6875
×      2
---------
  1.3750 ……整数部分 =1=K-1  最高位
  0.3750
×      2
---------
  0.7500 ……整数部分 =0=K-2
×      2
---------
  1.5000 ……整数部分 =1=K-3
  0.5000
×      2
---------
  1.0000 ……整数部分 =1=K-4  最低位
```

∴ $(0.6875)_{10} = (0.K_{-1}K_{-2}K_{-3}K_{-4})_2 = (0.1011)_2$

有的数在十换二时，整个计算过程会无限制地进行下去，这时可以根据精度的要求，选取适当的位数。

对于具有整数和小数部分的十进制数，只要分别将整数部分和小数部分转换为二进制数，然后合并起来即可得到结果。

二、八进制数和十进制数之间的相互转换

一般地说，任意进制数与十进制数之间的转换原理和方法，同二进制与十进制之间的转换相类似。区别仅在于把基数 2 换成相应数制的基数(如 8、16 等)。

1．八换十

与二换十相类似，即将八进制数按“权”相加即可。

【例】 $(51.6)_8 = 5 \times 8^1 + 1 \times 8^0 + 6 \times 8^{-1} = 40 + 1 + 0.75 = (41.75)_{10} = 41.75$

2．十换八

【例】 将十进制数$(75.6875)_{10}$转换为八进制数

解：

(1) 整数部分采用除 8 取余法

```
8|75      余数
 8|9 ……3=K0   (最低位)
 8|1 ……1=K1
   0 ……1=K2   (最高位)
```

∴ $(75)_{10} = (113)_8$

(2) 小数部分采用乘 3 取整法

```
  0.6875
×      8      整数
---------
  5.5000      5=K-1  (最高位)
  0.5000
×      8
---------
  4.0000      4=K-2  (最低位)
```

∴ $(0.6875)_{10} = (0.54)_8$

最后结果为 $(75.6875)_{10} = (113.54)_8$

三、十六进制数与十进制数之间的相互转换

1．十六换十

同二换十相类似，即将十六进制数按“权”展开相加即可。

【例】 $(F3D)_{16}=F3DH=(15\times16^2)+(3\times16^1)+(13\times10^0)=3840+48+13$

$=(3901)_{10}=3901D=3901$

2．十换十六

(1) 整数的十换十六(除16取余法)

【例】

16|3901

16|243 ……余 13 = D = K_0 (最低位)

16|15 ……余 3 = K_1

0 ……余 15 = F = K_2 (最高位)

∴ $(3901)_{10}=K_2K_1K_0=(F3D)_{16}=F3DH$

(2) 小数的十换十六(乘16取整法)

【例】 求$(0.9032)_{10}$的16进制数

解：

0.9032

× 16 整数部分

14.4512 ……14 = E = K_{-1} (最高位)

0.4512

× 16

7.2192 ……7 = K_{-2}

0.2192

× 16

3.5072 ……3 = K_{-3}

0.5072

× 16

8.1152 ……8 = K_{-4} (最低位)

∴ $(0.9032)_{10}=(0.E738)_{16}=0.E738H$

§2-4 二进制数的运算

二进制数只有0、1两个数码，它的加、减、乘、除等算术运算规则要比十进制数的运算规则简单得多。

一、加法规则

1．0+0=0

2．0+1=1+0=1

3．1+1=10 =进位+0

4．1+1+1=11 =进位+1

【例】 将两个二进制数1111与1011相加，其加法过程如下：

```
  1 1 1 1      进位
    1 1 1 1    被加数
+   1 0 1 1    加数
-------------
  1 1 0 1 0    和
```

二、减法规则

1. 0－0＝0
2. 1－0＝1
3. 1－1＝0
4. 0－1＝1　向相邻高位借位1当作2

【例】 求二进制数10100减去1001的结果

```
  1 0 1 0 0    被减数
- 1 0 0 1      减数
-------------
  1 0 1 1      差
```

三、乘法规则

1. 0×0＝0
2. 0×1＝0
3. 1×0＝0
4. 1×1＝1

除了1×1＝1外，其它情况乘积均为0，不需要象十进制乘法运算时那样背“九·九”乘法口诀。

【例】

```
      1 1 0 1      被乘数
  ×   1 0 1 0      乘数
-----------------
      0 0 0 0   ┐
    1 1 0 1     │
  0 0 0 0       │ 部分积
+ 1 1 0 1       ┘
-----------------
1 0 0 0 0 0 1 0    乘积
```

从上例可知：在乘法运算时，若乘数为1，则把被乘数照抄一遍，只是它的最后一位与相应的乘数位对齐；若乘数为0，则无作用；当所有的乘数位都乘过之后，再把各部分积相加，便得到最后乘积。因而二进制数的乘法实质上是由“加”（即加被乘数）和“移位”两种操作实现的。

四、除法规则

除法是乘法的逆运算。与十进制相类似，可从被除数的最高位开始检查，并定出需要超过除数的位数。找到这个位时，商记为1。并且将选定的被除数去减除数。然后，将被除数的下一位下移到余数上。若余数不够减，则商为0。这时可把被除数的再下一位移到余数上；若余数够减则商为1，余数去减除数，这样反复进行，直至全部被除数的位都下移完为止。

【例】

```
                0 0 0 1 1 0 1    商
除数 1 1 0 ) 1 0 0 1 1 1 0      被除数
             1 1 0 ↓ | | |
           -------------
               1 1 1 | | |
               1 1 0 ↓ ↓ |
             -----------
                   1 1 0 |
                   1 1 0 ↓
                 -------
                       0
```

综上所述，二进制的加、减、乘、除等算术运算，可以归结为加、减、移位三种操作。实际上，在计算机中为了简化设备，往往只设置加法器，而无减法器。此时需要将减法运算转化为加法运算，这将在§2-6节中进行讨论。这样，在计算机中二进制的四则运算就可以归结为加法和移位两种操作。

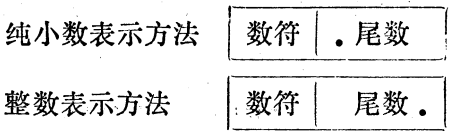
§2-5 计算机中数的定点与浮点表示

在电子计算机中，一个数的小数点是怎样表示的呢？通常有两种表示方法：定点表示法和浮点表示法。采用定点表示法的计算机叫定点机，采用浮点表示法的计算机叫浮点机。

所谓定点与浮点，是指计算机中数的小数点的位置是固定的还是浮动的。下面分别予以说明。

一、定点表示法(fixed-point representation of a number)

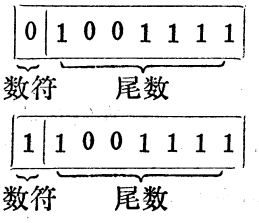
在定点表示法中，小数点的位置是固定不变的。通常把小数点固定在数值部分的最高位之前，或是把小数点固定在数值部分的最后面。前者将数表示为纯小数，后者将数表示成整数。因此，定点数在计算机中有两种表示方法：



其中数符用来表示数的正负，正数为"0"，负数为"1"。通常符号位设在数的最高位之前。尾数(mantissa)是指某数的本身数值部分。

采用纯小数定点表示时，计算机所能表示的数总是小于1的；但实际数可能大于1，这就需要在计算机解题之前，选择适当的比例因子，使全部参加运算的数都缩小若干倍，化成小于1的数，而计算的结果又必须用相应的比例增大若干倍使其恢复为真实值。而且还必须对运算结果加以估计，应该小于1，否则，计算机将产生"溢出"。

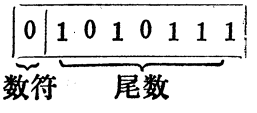
例如，二进制数(1) +0.1001111 和(2) −0.1001111，在定点机中的表示形式为



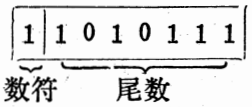
采用整数定点表示时，计算机只能表示大于1的整数，与上述表示法并无原则区别。

假设一个单元可以存放一个8位二进制数(通常称作字长8位)。其中左边第一位表示符号，余下7位表示数值部分。

例如，二进制数(1) +1010111 和(2) −1010111在定点机中的表示形式为



(2)



小数点“.”在计算机中实际上是不表示出来的，需要程序员事先约定它的位置。在本书中，如不加声明，则均约定采用整数定点表示法。

二、浮点表示法(floating-point representation of a number)

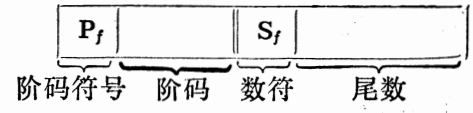
在浮点表示法中，小数点的位置不是固定的，而是浮动的。一般地说，任何一个二进制数N可以表示成下式：

$$N = 2^P \times S$$

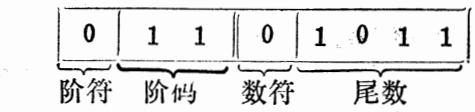
式中：S——数N的尾数，表示数N的有效数值。用S_f表示尾数的符号，$S_f=0$表示正数；$S_f=1$表示负数。

P——数N的阶码，表示小数点的位置。用P_f表示阶码的符号位，$P_f=0$表示阶码为正数；$P_f=1$表示阶码为负数。

因此，浮点数分为阶码和尾数两个部分，在数的表示中都有各自的符号位，其形式为



例如，设尾数为4位，阶码为2位，则二进制数$N=2^{+11}\times 1011$，在浮点机中的表示形式为



三、定点表示和浮点表示的比较

1．对于相同位数的数，浮点数表示的范围比定点数表示的范围大。

设计算机中，用32位二进制来表示数。对定点机，用1位表示数符，31位表示尾数。如果用定点小数法，则它能表示的数的范围为

$$+(1-2^{-31}) \sim -(1-2^{-31})$$

其近似值为

$$+1 \sim -1$$

如果用定点整数法，则它能表示的数的范围为

$$+(2^{31}-1) \sim -(2^{31}-1)$$

对浮点机，若也具有32位，其中8位表示阶码(包括阶符)，24位表示尾数(包括数符)，它能表示的数的范围为

$$+2^{(2^7-1)} \times (1-2^{-23}) \sim -2^{(2^7-1)} \times (1-2^{-23})$$

其近似值为

$$+2^{127} \sim -2^{127}$$

或

$$+2^{(2^8-1)} \times (2^{23}-1) \sim -2^{(2^8-1)} \times (2^{23}-1)$$

$$+2^{127} \times (2^{23}-1) \sim -2^{127} \times (2^{23}-1)$$

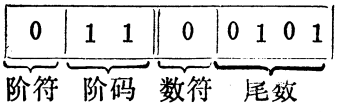
显然，对于相同位数的数，浮点数的表示范围比定点数大得多。而且阶码位数适当多一些，所能表示的数的范围会更大些，但尾数有效数字却减少了，会影响运算精度。因此，阶码位

数与尾数位数如何分配，必须根据具体的使用要求来确定。

在定点小数法中，为了使计算机运算过程中不丢失有效数字，提高运算精度，通常都采用二进制浮点规格化数。所谓规格化数即尾数的最高位是有效数字 1 而不是 0。因此，当尾数满足$\frac{1}{2}\leqslant|S|<1$时，即为规格化数。

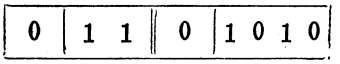
欲使浮点数规格化，只要移动尾数的小数点即可实现。

例如，二进制数 $N_1=2^{11}\times0.0101$，在浮点机中的表示形式为



是非规格化数。

要使浮点数规格化，只要调整阶码的数值即可。若将 N_1 改写为：$N_2=2^{10}\times0.1010$，则 N_1 即为规格化数了。其表示形式为



2．浮点数的运算规则比定点数复杂。

设有两个浮点数

$$N_1=2^{P_1}\times S_1,\quad N_2=2^{P_2}\times S_2$$

若 $P_1=P_2$，则

$$N_1+N_2=2^{P_1}\times S_1+2^{P_2}\times S_2=2^{P_1}(S_1+S_2)$$

换言之，阶码相同的两个浮点数求和，只要尾数相加作为数的尾数，阶码仍为原来的阶码。

但若 $P_1\neq P_2$，则需先考虑“对阶”，即将相加两数的阶码化为相同的数值，然后才能进行尾数相加。

例如，

$$N_1=2^{11}\times0.1001$$
$$N_2=2^{01}\times0.1100$$

“对阶”的方法是使小阶向大阶看齐，本例即将 N_2 尾数右移两位，阶码加 2，得到

$$N_2=2^{11}\times0.0011$$

然后两数相加，

$$\begin{array}{r}2^{11}\times0.1001\\ +)\,2^{11}\times0.0011\\ \hline 2^{11}\times0.1100\end{array}$$

当两数相减时，同样需要对阶。

两个浮点数的乘除法运算如下：

$$N_1\times N_2=(2^{P_1}\times S_1)\times(2^{P_2}\times S_2)=(2^{P_1}\times 2^{P_2})\times(S_1\times S_2)=2^{(P_1+P_2)}\times(S_1\times S_2)$$
$$N_1/N_2=(2^{P_1}\times S_1)\div(2^{P_2}\times S_2)=2^{(P_1-P_2)}\times(S_1\div S_2)$$

即，两个浮点数相乘时，只要阶码相加，尾数相乘；两个浮点数相除时，只要阶码相减，尾数相除。

浮点运算方法复杂，操作控制线路复杂，成本高。一般小型机、微型机多采用定点形式；大、中型机常采用浮点形式或兼有定点、浮点形式。

§2-6 原码、补码和反码

本节主要介绍，在计算机中带符号数的表示法。

一、机器数与真值

通常，数的正、负是用符号“+”、“-”来表示的。在电子计算机中，数的正、负是如何表示的呢？

在计算机中，数是存放在由寄存单元组成的寄存器中，二进制数码 1 和 0 是由寄存器单元的两种不同的状态(比如高、低电平)来表示的。对于正号“+”或负号“-”，也只能用这两种不同的状态来区别。因此，数的符号在计算机中也数码化了。通常规定在数的前面增设一位符号位，正数符号位用“0”表示，负数符号位用“1”表示。

若以 8 位字长的存数单元为例，设有：

$$N_1 = +1001100$$

$$N_2 = -1001100$$

则 N_1 和 N_2 在计算机中表示为

$$N_1:01001100$$

$$N_2:11001100$$

为了区别原来的数与它在计算机中的表示形式，我们将已经数码化了的带符号数称为机器数，而把原来的数称为机器数的真值。上面例子提到的 $N_1 = +1001100$、$N_2 = -10011.0$ 为真值，其在计算机中的表示 01001100 和 11001100 为机器数。

为了运算方便，在计算机中，机器数常用三种表示法，即原码、补码和反码。下面分别进行讨论。

二、原码(true form)

在用二进制原码表示的数中，符号位为 0 表示正数，符号位为 1 表示负数，其余各位表示尾数本身，这种表示法称为原码表示法。

1．对于正数

【例】 $X = +1101001$ (+105)

$[X]_{原} = 0\ 1101001$

符号位 尾数

2．对于负数

【例】 $X = -1101001$ (-105)

$[X]_{原} = 1\ 1101001$

符号位 尾数

3．对于 0

可以认为它是(+0)，也可认为是(-0)，这样，0 在原码中有下列两种表示法：

$$[+0]_{原} = 0\ 00\cdots0$$

$$[-0]_{原} = 1\ 00\cdots0$$

计算机遇到这两种情况都当作 0 来处理。

对于 8 位二进制原码，其表示的数值范围为 $+127 \sim -127$ 即：

$$(+127)_{原} = 01111111$$

$$(-127)_{原} = 11111111$$

由上述原码表示形式可以推出：

当真值 $X>0$ 时，如 $X = +1101001$，

则 $$[X]_{原} = 0\,1101001$$

即 $$[X]_{原} = X$$

当真值 $X<0$ 时，如 $X = -1101001$，

则 $$[X]_{原} = 11101001 = 10000000 + 1101001 = 10000000 - (-1101001) = 2^{8-1} - X$$

由此可知，字长为 n 位的定点数(包括符号位)的原码表示法可定义为：

$$[X]_{原} = \begin{cases} X & (0 \leqslant X < 2^{n-1}) \\ 2^{n-1} - X & (-2^{n-1} < X \leqslant 0) \end{cases}$$

式中：$[X]_{原}$ 为机器数，X 为真值。

若设 $$X = \pm x_1 x_2 \cdots x_{n-1}$$

则 $$[X]_{原} = x_0 x_1 x_2 \cdots x_{n-1}$$

其中，x_0 为原码机器数的符号位，它满足

$$x_0 = \begin{cases} 0 & (X \geqslant 0) \\ 1 & (X \leqslant 0) \end{cases}$$

因此，X 的原码又可定义为

$$[X]_{原} = x_0 + |X|$$

以上介绍的是定点整数的原码表示法。若是定点小数，小数点固定在尾数的最高位的左边。此时，真值 $|X|<1$。与上述方法类似，其原码表示法可定义为：

$$[X]_{原} = \begin{cases} X & (0 \leqslant X < 1) \\ 1 - X & (-1 < X \leqslant 0) \end{cases}$$

采用原码表示法，可以方便地实现乘除法运算，其运算方法是：对尾数进行乘除运算，而积或商的符号为两个数的符号的异或运算，即：$0 \oplus 0 = 0$；$1 \oplus 1 = 0$；$0 \oplus 1 = 1 \oplus 0 = 1$(其中 $\oplus$ 表示异或运算)。但是，遇到两个数相加时，如果是同号，则数值相加，符号不变；如果是异号，数值部分实际上是相减，而且必须先比较出两数绝对值的大小，然后从绝对值较大的数中减去绝对值较小的数。差值的符号与绝对值较大的数的符号一致。这就意味着计算机控制线路复杂化了，并且进行加减法运算的速度也降低了，而加减法运算又是计算机中最常用的运算。因此，为了简化判断手续，提高运算速度，节省机器设备，人们在实践中找到了更适合计算机进行加减运算的机器数表示法——补码表示法。

三、补码(two's complement)

补码表示法可以把负数转化为正数，使减法转换为加法，从而使正负数的加减运算转化为单纯的正数相加的运算，这样就解决了原码表示法在加减法运算中所遇到的矛盾。

1．模为 12 的补码

为了理解补码的意义，现以一个钟表对时的例子来说明，如图 2-1 所示。假定现在标准时间是 3 点整，而一只钟却指在 6 点整。为了校准时钟，可以把时针从 6 倒拨 3 格，即 $6-3=3$，

表明时针最后停在3点上。但是，有的钟表不允许倒拨时针，只允许顺时针转动，这时可以把时针从6顺拨9格到达3点整。这是因为时针顺拨时，当到达12点就从0重新开始，相当于自动丢失一个数字12。因此，把时针从6点顺拨9格时有

$$6+9=12(\text{自动丢失})+3=3$$

这个自动丢失的数(12)就叫做“模”(modulo，简写为mod)。上述加法可称为“按模12的加法”，用数学符号可表示为

$$6+9=3(\bmod 12)$$

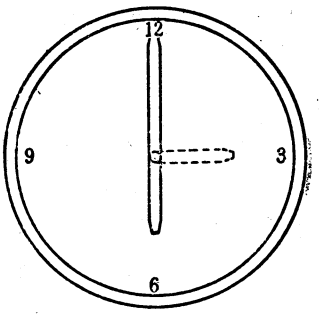


图2-1 钟表对时示意图

由于时针转一圈会自动丢失一个数12，因此我们可以把6－3＝3这一减法运算转化为加法运算，首先把模与减数(－3)相加，即12＋(－3)＝9，然后再把6和9相加得

$$6+[12+(-3)]=6+9=12+3=3$$

（12下方标注：自动丢失）

以上说明，当以12为模时，6－3与6＋9是等价的，于是我们将此处的9和3的关系说成为：顺拨时针的9是倒拨时针的3对12成补数的关系。同时，还可看出两个数(9及3)绝对值(不考虑数字前边的正负号)之和是12，此数字12则称为模数，用符号(mod 12)表示之。模和某个数X(例如－3)相加所得的数[12＋(－3)]＝9就称为该数(－3)对模12的补数，用$[X]_{补}$表示，即

$$[X]_{补}=\text{模}+X$$

引进补数概念后，就可以将原来的减法6－3＝3转化为加法6＋9＝12(自动丢失)＋3＝3。但是为了求出(－3)的补数，仍然用了(12－3)的减法运算，以后我们将会说明，在计算机中求二进制数的补码可以用其它简单的方法而不采用减法。

2．模为2的补码

通过上例的演示，我们就不难理解计算机中负数的补码表示法了。假定在定点机中，字长为n位，那么它的模就是2^n。它是一个n＋1位的数100…0，由于计算机只能表示n位数，因此数2^n(即100…0)在计算机中仅能以n个0来表示，而该数最左边的数字1就自动丢失了。由此可知，若以2^n为模，则2^n和0在计算机中的表示形式是完全相同的。这里以2^n为模也称为以2为模。

倘若将n位字长的存数单元最左边一位用作符号位，则以2^n为模的补码(又称为对2的补码)的表示形式为：

(1) 对于正数，即当$X=+x_1x_2\cdots x_{n-1}$时

$$[X]_{补}=\text{模}+X=2^n+X=2^n+x_1x_2\cdots x_{n-1}=0+x_1x_2\cdots x_{n-1}=X \quad (\bmod 2^n)$$

即对于正数，$[X]_{补}=X$，即正数的补码就是该正数本身。

【例】 $X=+1101001$ (+105)

$$[X]_{补}=\text{模}+X=2^8+1101001=0\,1101001$$

（2^8下方标注：自动丢失）

(2) 对于负数，即当$X=-x_1x_2\cdots x_{n-1}$时

$$[X]_{补}=\text{模}+X=2^n+X=2^n-x_1x_2\cdots x_{n-1} \quad (\bmod 2^n)$$

即对于负数，

$$[X]_{补}=2^n+X \quad (\bmod 2^n)$$

【例】 $X=-1101001 \quad (-105)$

$$[X]_{补}=模+X=2^8-1101001=100000000-1101001=10010111$$

(3) 对于0，即当 $X=00\cdots0$ 时

$$[+0]_{补}=+0=00\cdots0$$

$$[-0]_{补}=2^n-00\cdots0=00\cdots0 \quad (\mathrm{mod}\ 2^n)$$

因此，对于补码而言，$[+0]_{补}=[-0]_{补}=00\cdots0$，即"0"只有一种表示方法。

(4) 当 $X=-2^{n-1}$ 时，仍可用 $[X]_{补}=2^n+X$ 求得 $[-2^{n-1}]_{补}=2^n-2^{n-1}=2^{n-1}$，这样就将真值 X 的取值范围扩大为 $-2^{n-1}\leqslant X<2^{n-1}$，而真值与其补码之间仍保持一一对应关系。

【例】 在字长为8位的计算机中，最高位为符号位，其补码所能表示的数值范围为：

$$+127\sim-128$$

即

$$(+127)_{补}=01111111$$

$$(-127)_{补}=模+(-127)=2^8+(-1111111)=10000001$$

$$(-128)_{补}=模+(-128)=2^8+(-2^7)=2^7=10000000$$

综上所述，可对补码定义如下：

$$[X]_{补}=\begin{cases}X & (0\leqslant X<2^{n-1})\\ 2^n+X & (-2^{n-1}\leqslant X<0,\mathrm{mod}\ 2^n)\end{cases}$$

以上介绍的是定点整数的补码表示法。若是定点小数，则其补码表示法可定义为：

$$[X]_{补}=\begin{cases}X & (0\leqslant X<1)\\ 2+X & (-1\leqslant X<0)\end{cases}$$

从以上讨论来看，引进补码表示法后，解决了把负数转化为正数、使减法转化为加法的问题，但是求补码时还得用减法运算，这仍然是不方便的，于是人们又继续研究出第三种表示机器数的反码表示法。

四、反码(one's complement)

反码的定义如下：

1．对于正数，它的反码表示与原码相同。

即

$$[X]_{反}=[X]_{原}$$

【例】 $X=+1101001 \quad (+105)$

$$[X]_{反}=0\ 1101001$$

2．对于负数，则除符号位仍为"1"外，其余各位"1"换成"0"，"0"换成"1"，即得到反码 $[X]_{反}$。

【例】 $X=-1101001$

$$[X]_{反}=1\ 0010110$$

3．对于0，它的反码有 $[+0]_{反}$ 和 $[-0]_{反}$ 两种表示法。

$$[+0]_{反}=000\cdots0$$

$$[-0]_{反}=111\cdots1$$

因此在反码表示中，"0"的表示不是唯一的。

综上所述可用下列公式来定义反码：

设 $X=+x_1x_2\cdots x_{n-1}$ 或 $X=-x_1x_2\cdots x_{n-1}$，

并设 x_i 变反后用 $\overline{x_i}$ 表示，

则有

$$[X]_{反}=\begin{cases}X & (0\leqslant X<2^{n-1})\\ (2^n-1)+X & (-2^{n-1}<X\leqslant 0)\end{cases}$$

上式中，当 $0\leqslant X<2^{n-1}$ 时，$[X]_{反}=X$ 是由反码的定义确定的，其含意是显而易见的。下面对 $-2^{n-1}<X\leqslant 0$ 时，$[X]_{反}=(2^n-1)+X$ 稍加说明。例如：

原给定数 $X=-x_1x_2\cdots x_{n-1}$

其反码 $[X]_{反}=1\,\overline{x_1}\,\overline{x_2}\cdots\overline{x_{n-1}}$

现将数的绝对值与反码按位(同一位的上下两数)相加为 11…1 的情况表述如下：

$$\begin{array}{ll} x_1\ x_2\cdots x_{n-1} & \longleftarrow\text{数的绝对值}\\ +1\ \overline{x_1}\ \overline{x_2}\cdots\overline{x_{n-1}} & \longleftarrow\text{数 x 的反码}\\ \underbrace{1\ 1\ 1\cdots 1}_{n\text{位}} & \longleftarrow\text{每一位上皆为 1}\end{array}$$

上面的算式用数学关系式表示为：

$$|X|+[X]_{反}=\underbrace{111\cdots1}_{n\text{位}}=2^n-1$$

则可得 $[X]_{反}=(2^n-1)-|X|=(2^n-1)+X$

现将$[X]_{反}$表达式与$[X]_{补}$表达式作一比较：

$$[X]_{补}=\begin{cases}X & (0\leqslant X<2^{n-1})\\ 2^n+X & (-2^{n-1}\leqslant X<0,\ \mathrm{mod}\ 2^n)\end{cases}$$

$$[X]_{反}=\begin{cases}X & (0\leqslant X<2^{n-1})\\ (2^n-1)+X & (-2^{n-1}<X<0)\end{cases}$$

从比较可知：当X为正数时，$[X]_{补}=[X]_{反}=[X]_{原}=X$；当X为负数时，$[X]_{补}=2^n+X=(2^n-1)+X+1=[X]_{反}+1$

由此可知，对于满足 $-2^{n-1}\leqslant X<0$ 的负数，$[X]_{补}$等于$[X]_{原}$对应的各位(除符号位外)求反，然后在最低位上加 1。简称为除符号位外求反加 1。这样，在计算机中求补码时，就避免了减法运算。

反码也可以看做以 2^n-1 为模的补码，因此也称为对 1 的补码。

8 位二进制反码所能表示的数值范围为 +127～−127。表 2-3 列出 8 位二进制数表示的无符号数、带符号数(原码、补码、反码)的对照表。

顺便指出，当已知$[X]_{补}=1\,x_1x_2\cdots x_{n-1}$ 时，若把$[X]_{补}$除符号位外求反加 1 就得到$[X]_{原}$，即

$$[[X]_{补}]_{求补}=[X]_{原}$$

【例】

$$\begin{aligned}X&=-1101001\\ [X]_{原}&=\ \ 11101001\\ [X]_{补}&=\ \ 10010111\\ [[X]_{补}]_{求补}&=\ \ 11101001=[X]_{原}\end{aligned}$$

此外，求负数对 2 的补码还有一种方法。从负数X的最低位开始向左检查，当遇到第一个“1”以前，包括第一个“1”，重写每一位，即原来是“0”仍写为“0”，遇到第一个“1”仍写“1”，此后使其余位均变反，但符号位不变。

表 2-3　8 位二进制数的表示法

二进制数码表示	无符号二进制数	原　码	补　码	反　码
00000000	0	+0	+0	+0
00000001	1	+1	+1	+1
00000010	2	+2	+2	+2
⋮	⋮	⋮	⋮	⋮
01111101	125	+125	+125	+125
01111110	126	+126	+126	+126
01111111	127	+127	+127	+127
10000000	128	−0	−128	−127
10000001	129	−1	−127	−126
10000010	130	−2	−126	−125
⋮	⋮	⋮	⋮	⋮
11111101	253	−125	−3	−2
11111110	254	−126	−2	−1
11111111	255	−127	−1	−0

【例】 $X=-1010011$　　$[X]_{原}=1\ 1010011$

符号位不变　取反　第一个 1 不变

$[X]_{补}=1\ 0101101$

以上介绍的机器数的三种表示形式：原码、补码和反码都是定点表示，不过这些方法也适用于浮点表示的原码、补码和反码。在浮点表示中，一个数分为两个部分：阶码和尾数，只要对这两部分分别利用相应的规则即可得到浮点数的原码、补码和反码的表示形式。表 2-4 表示出两个小于 1 的正负数和两个大于 1 的正负数的变换规律。

表 2-4　原码、补码、反码之间的变换规则

十进制数		$+\frac{13}{128}$	$-\frac{13}{128}$	+2.75	−5.5
二进制数		+0.0001101	−0.0001101	+10.11	−101.1
定点表示法	真　值	+0.0001101	−0.0001101	+0.001011 注 1	−0.010110 注 1
	原　码	0 | 00011010	1 | 00011010	0 | 001011	1 | 010110
	补　码	0 | 00011010	1 | 11100110	0 | 001011	1 | 101010
	反　码	0 | 00011010	1 | 11100101	0 | 001011	1 | 101001
浮点表示法	真值(规格化)	$2^{-11}\times(+0.1101)$	$2^{-11}\times(-0.1101)$	$2^{10}\times(+0.1011)$	$2^{11}\times(-0.1011)$
	原　码	1 | 011 ‖ 0 | 11010000	1 | 011 ‖ 1 | 11010000	0 | 010 ‖ 0 | 101100	0 | 011 ‖ 1 | 101100
	补　码	1 | 101 ‖ 0 | 11010000	1 | 101 ‖ 1 | 00110000	0 | 010 ‖ 0 | 101100	0 | 011 ‖ 1 | 010100
	反　码	1 | 100 ‖ 0 | 11010000	1 | 100 ‖ 1 | 00101111	0 | 010 ‖ 0 | 101100	0 | 011 ‖ 1 | 010011

注：1. 取比例因子为 2^4；
2. 尾数部分长短应根据机器的字长而定，这里取八位(或六位)；
3. 阶码部分变换规则与尾数部分相同；
4. 采用定点小数表示法。

五、补码的加减法运算

在计算机中，凡是带符号的数一律用补码表示，其运算结果也是用补码来表示的。若结果的符号为"0"表示正数，得到的是原码(正数的补码即为原码)；若结果的符号为"1"表示负数，得到的是补码。下面分加、减两种情况进行分析。

1．定点补码的加法运算

下面将证明无论X和Y是正数还是负数，下列公式都是成立的。

$$[X]_{补}+[Y]_{补}=[X+Y]_{补} \quad (\mathrm{mod}\ 2^n)$$

式中：$-2^{n-1}\leqslant X<2^{n-1}$；$-2^{n-1}\leqslant Y<2^{n-1}$；$-2^{n-1}\leqslant(X+Y)<2^{n-1}$。

现根据X、Y的符号及绝对值大小分4种情况证明。

(1) X、Y均为正数，则其和也为正。因为正数的补码就是该正数本身，故有

$$[X]_{补}+[Y]_{补}=X+Y=[X+Y]_{补} \quad (\mathrm{mod}\ 2^n)$$

(2) X为正数，Y为负数，且$|X|\geqslant|Y|$，即$0\leqslant(X+Y)<2^{n-1}$时。

由补码定义可知：

$$[X]_{补}=X$$
$$[Y]_{补}=2^n+Y$$
$$[X+Y]_{补}=X+Y$$

则有 $[X]_{补}+[Y]_{补}=X+2^n+Y=X+Y=[X+Y]_{补} \quad (\mathrm{mod}\ 2^n)$

由于$0\leqslant(X+Y)<2^{n-1}$，而2^n作为符号位的进位而自动丢失，结果为正数，其补码即为该数本身。

【例】 已知X = +0010011，Y = −0000111，进行补码加法运算。

$$\begin{array}{rl} & [X]_{补}=00010011 \quad (+19\text{的补码}) \\ +) & [Y]_{补}=11111001 \quad (-7\text{的补码}) \\ \hline [X+Y]_{补}= & \boxed{1}\,00001100 \quad (+12\text{的补码}) \end{array}$$

丢失 ← 1 ；00001100 ← 符号位

结果：X + Y = +0001100

(3) X为正数，Y为负数，且$|Y|>|X|$，即$-2^{n-1}\leqslant(X+Y)<0$时。

由补码定义可知：

$$[X]_{补}=X$$
$$[Y]_{补}=2^n+Y$$
$$[X+Y]_{补}=2^n+(X+Y)$$

则有 $[X]_{补}+[Y]_{补}=X+2^n+Y=2^n+(X+Y)=[X+Y]_{补} \quad (\mathrm{mod}\ 2^n)$

由于$-2^{n-1}\leqslant(X+Y)<0$，故求得的结果为一个负数的补码形式。

【例】 已知X = +0000111，Y = −0010011，进行补码加法运算。

$$\begin{array}{rl} & [X]_{补}=00000111 \quad (+7\text{的补码}) \\ +) & [Y]_{补}=11101101 \quad (-19\text{的补码}) \\ \hline [X+Y]_{补}= & 11110100 \quad (-12\text{的补码}) \end{array}$$

11110100 ← 符号位

符号位为1表明结果为负数，它以补码形式出现，为求得原码表示的结果，必须将补码变换为原码，即对补码(11110100)再求一次“补”：

$$11110100 \xrightarrow{\text{取反}} 10001011 \xrightarrow{\text{末位加 1}} 10001100$$

符号位　符号位不变

结果：$X+Y=-0001100 \quad (-12)$

(4) X、Y均为负数，则其和必为负。由补码定义可知：

$$[X]_{补}=2^n+X$$

$$[Y]_{补}=2^n+Y$$

$$[X+Y]_{补}=2^n+(X+Y)$$

则有 $[X]_{补}+[Y]_{补}=2^n+X+2^n+Y=2^n+2^n+(X+Y)=2^n+[X+Y]_{补}=[X+Y]_{补} \pmod{2^n}$

由于 $-2^{n-1}\leqslant(X+Y)<0$，故求得的结果也为一负数的补码形式。

【例】 已知 $X=-0011001$，$Y=-0000110$，进行补码加法运算。

$$[X]_{补}=11100111 \quad (-25\text{ 的补码})$$

$$+)\ [Y]_{补}=11111010 \quad (-6\text{ 的补码})$$

$$[X+Y]_{补}=\boxed{1}\,11100001 \quad (-31\text{ 的补码})$$

丢失　符号位

符号位为1表明结果为负数，符号位之前的1是由符号位进位上来的，它将自动丢失。如果将 $[X+Y]_{补}=1\ 1100001$ 再次求补便得到 $[X+Y]_{原}$，

即 $[X+Y]_{原}=1\ 0011111$

结果：$X+Y=-0011111 \quad (-31)$

综上所述，我们可以归纳出一条定理，即：当 $-2^{n-1}\leqslant X<2^{n-1}$，$-2^{n-1}\leqslant Y<2^{n-1}$，且 $-2^{n-1}\leqslant(X+Y)<2^{n-1}$ 时，如下补码运算公式总是成立的：

$$[X]_{补}+[Y]_{补}=[X+Y]_{补} \pmod{2^n}$$

此公式表明：在模 2^n 的意义下(即模 2^n 自然丢失)，任意两个数的补码之和等于该两数和的补码，并且数的符号位也可以当做数的一部分参加运算。如果从符号位有进位发生的，则将该进位1丢掉。

有了这条定理，在计算机中进行两个带符号数的加法运算，就方便得多了。此时，只要将给定的真值用补码表示，就可以直接进行加法运算。在运算过程中不必判断加数和被加数的正负，一律做加法，最后将结果转换为真值即可。

2．定点补码的减法运算

对于定点补码的减法运算，由于存在：

$$X-Y=X+(-Y)$$

$$[X-Y]_{补}=[X+(-Y)]_{补}=[X]_{补}+[-Y]_{补} \pmod{2^n}$$

因此，在进行定点补码减法运算时，只要将减数Y转换为 $[-Y]_{补}$，减法运算便转换为加法运算了。这里的关键在于求 $[-Y]_{补}$，下面分两种情况说明。

(1) 当 $0\leqslant Y<2^{n-1}$ 时

$$[Y]_{补}=0y_1y_2\cdots y_{n-1}=Y$$

$$-Y=-y_1y_2\cdots y_{n-1}$$

$$[-Y]_{原}=1y_1y_2\cdots y_{n-1}$$
$$[-Y]_{补}=1\overline{y}_1\overline{y}_2\cdots\overline{y}_{n-1}+1$$

(2) 当 $-2^{n-1}\leqslant Y<0$ 时

$$[Y]_{补}=1y'_1y'_2\cdots y'_{n-1}$$
$$[Y]_{原}=1\overline{y'_1}\overline{y'_2}\cdots\overline{y'_{n-1}}+1$$
$$[-Y]_{原}=0\overline{y'_1}\overline{y'_2}\cdots\overline{y'_{n-1}}+1$$
$$[-Y]_{补}=0\overline{y'_1}\overline{y'_2}\cdots\overline{y'_{n-1}}+1$$

比较$[Y]_{补}$和$[-Y]_{补}$，可以发现，只要将$[Y]_{补}$连同符号位一起求反加1，便可得到$[-Y]_{补}$。我们把连同符号位一起求反加1的过程，称为变补，即

$$[-Y]_{补}=[[Y]_{补}]_{变补}$$

以区别于前面所述的$[X]_{原}=[[X]_{补}]_{求补}$(即除符号位外的求反加1)。

因此，补码减法运算，可归结为对减数变补，然后做加法。

【例1】 已知 $X=+1100000$ $(+96)$

$Y=+0010011$ $(+19)$

用补码进行减法运算，

$$[X]_{补}=[X]_{原}=01100000$$
$$[Y]_{补}=[Y]_{原}=00010011$$
$$[-Y]_{补}=11101101$$

则

$$\begin{array}{rl} & [X]_{补}=01100000 \quad (+96\text{的补码}) \\ +) & [-Y]_{补}=11101101 \quad (-19\text{的补码}) \\ \hline [X-Y]_{补}= & \boxed{1}\,01001101 \quad (+77\text{的补码}) \end{array}$$

丢失 ← 1 → 符号位

其结果的符号位为"0"，表示正数。

结果：$X-Y=+1001101$

【例2】 已知 $X=-0111000$ (-56)

$Y=-0010001$ (-17)

用补码进行减法运算，即 $X-Y=(-56)-(-17)=?$

$$[X]_{原}=10111000$$
$$[X]_{补}=11001000$$
$$[Y]_{原}=10010001$$
$$[Y]_{补}=11101111$$
$$[-Y]_{补}=00010001$$

则

$$\begin{array}{rl} & [X]_{补}=1\ 1001000 \quad (-56\text{的补码}) \\ +) & [-Y]_{补}=0\ 0010001 \quad (+17\text{的补码}) \\ \hline [X-Y]_{补}= & \boxed{1}\,1011001 \quad (-39\text{的补码}) \end{array}$$

符号位 ←

其结果的符号位为"1"，表明结果为负，为求得原码表示的结果，只要对补码结果再次求补即

可，即 $[X-Y]_{原}=1\ 0100111$

结果： $X-Y=-0100111$

3．定点反码的加减法运算

与补码类似，对于反码，我们可以证明：

$$[X]_{反}+[Y]_{反}=[X+Y]_{反}$$

$$[X]_{反}+[-Y]_{反}=[X-Y]_{反}$$

用反码表示法也可以把减法运算转化为加法运算，不过加得的结果还要再加一个循环进位才能得出正确的答案。所谓循环进位就是指当符号位出现进位时，把该进位加到最低位上去的进位方式。

【例】 已知 $X=+0010001, Y=-0001100$，进行反码加法运算。

$$\begin{array}{rl} [X]_{反}= & 00010001 \quad (+17\text{ 的反码}) \\ +)\ [Y]_{反}= & 11110011 \quad (-12\text{ 的反码}) \\ \hline [X+Y]_{反}= & \boxed{1}\ 00000100 \leftarrow \text{未修正的和} \\ & \downarrow \\ +) & 1 \leftarrow \text{循环进位} \\ \hline & 00000101 \quad (+5\text{ 的反码}) \end{array}$$

结果： $X+Y=+0000101$

由于反码加法需要将循环进位加到本数的最低位上去，这将使线路复杂化，因此，通常不用反码进行加减运算，而常把它作为求补过程的中间形式。

以上介绍的是定点补码加减法运算，浮点补码加减法在运算过程中要多两个步骤，即对阶和规格化，其它的与定点补码运算相类似。

六、不带符号数的运算

不带符号数(或称为无符号数) 实际上是指参加运算的操作数 X、Y 均为正数，且整个字长全部用于表示数值部分。一个 n 位字长的数据字可用来表示 2^n 个正整数，例如一个 8 位数据字可表示的数值范围为 00000000B～11111111B，即 0～255，共 256 个数值。

当两个 n 位不带符号数相加时，由于 X、Y 都是正数，因此 X+Y 的和数也必为正数。当和数 X+Y 超过其字长 n 位所允许的范围时，就向更高位(即进位位)进位。例如 n=8 时，若 X+Y 的和数大于 255(即 2^8-1)，就向更高位进位，即必须采用多字节来表示。

两个不带符号数相减，其差值的符号取决于两数绝对值的大小。在计算机中，对于减法运算一律采用补码运算，即用减数变补相加来代替两数相减。减数变补就是指将整个减数变反加 1，或者说用模 2^n 减去减数。即

$$X-Y=X+(-Y)$$

$$[X-Y]_{补}=[X]_{补}+[-Y]_{补}$$

$$[-Y]_{补}=2^n+(-Y)=2^n-Y$$

所以 $$[X-Y]_{补}=[X]_{补}+[-Y]_{补}=2^n+(X-Y)=2^n-(Y-X)$$

由此可得到以下两点结论：

1．若 X>Y，则 X 减 Y 时无借位，差值为正；$[X-Y]_{补}=[X]_{补}+[-Y]_{补}$ 之和必大于 2^n，最

高位有进位，其所得到的和即是(X－Y)的原码。

【例】 已知 X＝＋01000000，Y＝＋00001010，进行补码减法运算。

$$[X]_{补}=[X]_{原}=01000000$$
$$[Y]_{补}=[Y]_{原}=00001010$$
$$[-Y]_{补}=11110110$$

则

$$[X]_{补}=01000000 \quad (+64\text{ 的补码})$$
$$+)\ [-Y]_{补}=11110110 \quad (-10\text{ 的补码})$$
$$[X-Y]_{补}=\boxed{1}\,00110110 \quad (+54\text{ 的补码})$$

自动丢失，有进位表示无借位结果为正数。

两个补码相加时，最高位有进位表示 X－Y 时无借位，结果为正数。

结果：$X-Y=+00110110$

此例亦可用另一种方法求得结果：

$$X-Y=64-10=64+(-10)=64+(256-10)=256+54=2^8+54=54$$

自动丢失

2.若 X＜Y，则X减 Y 时有借位，差值为负；$[X-Y]_{补}=[X]_{补}+[-Y]_{补}$必小于 2^n，最高位无进位，其所得到的和是$(X-Y)=-(Y-X)$的补码。若令 $Z=[X-Y]_{补}$

当 Z 的结果为负时，根据 Z 求(X－Y)的方法为：先求出 Z 的补码$[Z]_{补}$，然后再加一负号，即为(X－Y)；亦即$(X-Y)=-[Z]_{补}$。证明如下：

根据补码的定义 $[X-Y]_{补}=2^n+(X-Y)$

即 $Z=2^n+(X-Y)$

则有 $X-Y=-2^n+Z=-(2^n-Z)=-\{[(2^n-1)-Z]+1\}=-[Z]_{补}$

【例】 已知 X＝＋00001010，Y＝＋01000000，进行补码减法运算。

$$[X]_{补}=[X]_{原}=00001010$$
$$[Y]_{补}=[Y]_{原}=01000000$$
$$[-Y]_{补}=11000000$$

则

$$[X]_{补}=00001010 \quad (+10\text{ 的补码})$$
$$+)\ [-Y]_{补}=11000000 \quad (-64\text{ 的补码})$$
$$Z=[X-Y]_{补}=\boxed{0}\,11001010 \quad (-54\text{ 的补码})$$

无进位表示有借位，结果为负数

结果：$X-Y=-[Z]_{补}=-00110110$

两个补码数相加时最高位无进位表示结果为负，此时应求出上列结果的补码，再加一负号，即为最后的结果。

综上所述，无论参加运算的两个 n 位二进制数是带符号或不带符号的补码数，对计算机来说，其处理的方法都是相同的，即用减数变补相加来代替两数相减，所不同的仅在于，对结果的符号位采用不同的判别方法。若参加运算的两数为带符号的补码形式，则结果的符号位以最高位(S－Sign)来判别，S＝0 表示正数，S＝1 表示负数；若参加运算的两数为不带符号的数，则

结果的符号位以最高位有无进位来判别：最高位相加时有进位(即两数相减时无借位)表示正数；最高位相加时无进位(即两数相减时有借位)表示负数。

七、溢出(overflow)

从广义上说，在计算机中，如果在运算过程产生的数超过计算机所能允许表示的范围，就称为“溢出”。任何运算都不允许发生溢出，除非是利用溢出作为判断而不使用其结果。因而当发生溢出时，计算机必须能够立即发现，并停止运算，称为溢出停机，或转入检查程序以找出产生溢出的原因，并作相应处理，称为溢出处理。

为了避免可能产生的混淆，本书中的“溢出”，仅限于对带符号数的运算范围。如字长为 n 位时，除去一位符号位外，还有 n－1 位表示数值，它所能表示的补码的范围为 $-2^{n-1}\sim 2^{n-1}-1$。如果运算结果超过此范围，就会产生溢出，从而导致运算错误。

例如，n＝8，最高位为符号位，其余 7 位表示数值，它所能表示的补码的范围为 －128～＋127。如果运算结果超过此范围，就会产生溢出。

现设下列两数相加：

$$\begin{array}{rll} & 0\ 1101001 & (+105) \\ +) & 0\ 0110010 & (+50) \\ \hline & 1\ 0011011 & \end{array}$$

符号位

如果把相加结果视为无符号数，则该数为＋155，这个运算结果是正确的。如果将此结果视为带符号数，其符号位为负，则按补码的规定该数应理解为－101，这显然是错误的。其原因是和数＋155 大于 8 位符号数所能表示的最大值＋127，使数值部分占据了符号位的位置，从而导致运算错误。

又设下列两数相加：

$$\begin{array}{rll} & 1\ 0010111 & [-105]_{补} \\ +) & 1\ 1001110 & [-50]_{补} \\ \hline & 1\ 0\ 1100101 & \end{array}$$

进位丢失　　符号位

两个负数相加，和应为负数，但结果的符号位却为正，这显然也是错误的。其原因是和数 －155 小于 8 位符号数所能表示的最小值－128。

在这里，应当指出溢出与进位及补码运算中的进位丢失之间的区别。

第一，溢出与进位是两个性质不同的概念。

进位是指运算结果的最高位要向更高位进位或借位。设立进位位的目的就在于处理超过一个字长的数字的算术进位或借位，以用于多字节的算术运算。通常多字节运算是以不带符号的字节数作为低位部分，只有最高字节有符号位。由此可知，进位主要用于对无符号数运

算，这与溢出主要用于对带符号数运算是有区别的。

第二，溢出与补码运算中的进位丢失也应加以区别。

现设下列两数相加：

$$\begin{array}{rll} & \boxed{1}\ 1001110 & [-50]_{补} \\ +) & \boxed{1}\ 1111011 & [-5]_{补} \\ \hline & \boxed{1}\ \boxed{1}\ 1001001 & [-55]_{补} \end{array}$$

进位丢失—┘ └—符号位

该两负数相加结果为－55是正确的。这里，虽出现了补码运算中产生的符号位的进位丢失，但由于和数并未超出－128～＋127的范围，因此，不会产生溢出。

下面介绍三种判断溢出的方法：

1．根据参加运算的两个数的符号及运算结果的符号判断溢出

由于字长为n位时，数值部分为n－1位，数值部分的模为2^{n-1}。如果参加运算的两个数X和Y的绝对值都小于2^{n-1}，则(＋X)＋(－Y)或(－X)＋(＋Y)的绝对值都不会超过2^{n-1}，但是，(＋X)＋(＋Y)或(－X)＋(－Y)的绝对值却可能超过2^{n-1}。因而前面两种情况无需讨论，只要讨论后两种情况即可。对于(＋X)＋(＋Y)的运算，由于X和Y的符号位X_0和Y_0均为“0”，故结果的符号Z_0也应为“0”。若发生溢出，则溢出的“1”进入符号位，使Z_0变为“1”，于是出现正数加正数等于负数的错误。同理，(－X)＋(－Y)，结果应为负数，和的符号Z_0应为“1”。若此时发生溢出，则Z_0变为“0”，于是出现负数加负数等于正数的错误。

综上所述，求和运算溢出条件表达式为：

$$溢出 = x_0y_0\overline{z_0} + \overline{x_0}\,\overline{y_0}z_0$$

2．利用变形码判断溢出

所谓变形码就是在原符号位的相邻高位再加一位符号位，实质上就是用两位二进制数来表示数值的正负。

变形码也分为变形原码、变形补码和变形反码，这里着重介绍常用的变形补码。

变形补码与上述补码的区别在于其符号位为两位，即用00表示正；用11表示负。

例如：

$$X = +101101$$

$$[X]_{补} = \boxed{0}\ 101101$$

$$[X]_{变形补} = \boxed{0\ 0}\ 101101$$

$$X = -101101$$

$$[X]_{补} = \boxed{1}\ 010011$$

$$[X]_{变形补} = \boxed{1\ 1}\ 010011$$

由此可知，当$X>0$时，$X=[X]_{补}=[X]_{变形补}$。

当$X<0$时，

$$[X]_{补} = 1x_1'x_2'\cdots x_{n-1}' = 2^n + X$$

$$[X]_{变形补} = 11x_1'x_2'\cdots x_{n-1}' = 2^n + 1x_1'x_2'\cdots x_{n-1}' = 2^n + 2^n + X = 2^{n+1} + X$$

因而变形补码的定义为：

$$[X]_{变形补} = \begin{cases} X & (0 \leqslant X < 2^{n-1}) \\ 2^{n+1} + X & (-2^{n-1} \leqslant X < 0, \bmod 2^{n+1}) \end{cases}$$

对于变形补码，模 2^n 补码的所有性质，只要作些成式上的修改，都将依然成立。如：

$$[X]_{变形补} + [Y]_{变形补} = [X + Y]_{变形补}$$

利用变形码的双符号位的状态，可以很容易地判断加减法运算是否有“溢出”发生。两个符号位都作为数值一起参加运算。当计算结果的两个符号位相同时表示没有溢出；当两个符号位相异时表示溢出，运算产生错误。这时判断溢出的逻辑表达式为：

$$溢出 = z_0'\overline{z_0} + \overline{z_0'}z_0 = z_0' \oplus z_0$$

式中 z_0' 为第二符号位，z_0 为第一符号位，其中 z_0 仍表示结果的正确符号。

3．利用双进位位的状态来判断溢出

在微型计算机中，常用双进位位的状态来判断溢出。这种方法是用字长的最高位(即符号位)与次高位(数值部分的最高位)的进位状态来判断运算结果是否发生溢出。

为了说明这种方法，先引进两个符号 C_S 和 C_{S+1}：

C_S 用来表示数值部分的最高位(即字长的次高位)向符号位进位时的状态，有进位，则 $C_S = 1$，否则 $C_S = 0$。

C_{S+1} 用来表示符号位(即字长的最高位)向相邻高位进位时的状态，有进位，则 $C_{S+1} = 1$，否则 $C_{S+1} = 0$。前已述及，若参加运算的两个数 X 和 Y 的绝对值小于 2^{n-1}，则只有当两数同时为正或同时为负、并且其和 X + Y 超过所允许表示的范围时，才会发生溢出。下面分别予以说明。

(1) X 和 Y 均为正数

对于两个正数相加，其和有两种情况：

① 当 $X + Y < 2^{n-1}$ 时，不会发生溢出。

【例】 已知 X = +0101101，Y = +0101110，试进行加法运算。

```
                C_{S+1} C_S                    进位位
                [0  0]  1 0 1 1                进位
                   0  0 1 0 1 1 0 1   (+45)
        +)         0  0 1 0 1 1 1 0   (+46)
        ------------------------------------
                   0  1 0 1 1 0 1 1   (+91)
                   |  | |  |  |  |
                   |  | |  |  |  └─数值部分次高位
                   |  | |  |  └────数值部分最高位
                   |  | |  └───────
                   |  | └──────────
                   |  └────────────
                   └─符号位
                   正确结果(和小于 +127)
```

由于 $X + Y < 2^{n-1}$，故 $C_S = 0$；又 X 和 Y 均为正，它们的符号位均为 0，故 $C_{S+1} = 0$。由此可知，当 $C_{S+1} = 0$、$C_S = 0$ 时，不会发生溢出。

② 当 $X + Y \geqslant 2^{n-1}$ 时，发生溢出。

【例】 已知 X = +1011010，Y = +1101011，试进行加法运算。

C_{S+1} C_S　　进位位

正溢出　0　1　11101　　进位

$[X]_{补}=$　0　1011010　　$[+90]_{补}$

$+)\ [Y]_{补}=$　0　1101011　　$[+107]_{补}$

$[X+Y]_{补}=$　1　1000101　　$[-59]_{补}$

符号位

正溢出错误(和大于 +127)

由于 $X+Y\geqslant 2^{n-1}$，故 $C_S=1$；又X和Y均为正，它们的符号位均为0，故 $C_{S+1}=0$。由此可知，当 $C_{S+1}=0$、$C_S=1$ 时，就会发生溢出。上例中，两个正数相加的结果，符号位上为1，变成了负数，这显然是错误的。

(2) X 和 Y 均为负数。

对于两个负数相加，也有两种情况：

① 当 $-2^{n-1}\leqslant X+Y<0$ 时，不会发生溢出。

【例】 已知 X = −0000101，Y = −0000010，用补码进行加法运算。

C_{S+1} C_S　　进位位

　1　1　111110　　进位

$[X]_{补}=$　1　1111011　　$[-5]_{补}$

$+)\ [Y]_{补}=$　1　1111110　　$[-2]_{补}$

$[X+Y]_{补}=$　1　1111001　　$[-7]_{补}$

正确结果(和大于 −128)

由于$[X]_{补}$和$[Y]_{补}$的符号位均为1，相加后为10，故和的符号位必有进位，即 $C_{S+1}=1$。又由于$[X+Y]_{补}=[X]_{补}+[Y]_{补}$是负数，其符号位必为1，它是从数值的最高位进位而来的，即 $C_S=1$。由此可知，当 $C_{S+1}=1$、$C_S=1$ 时，不会发生溢出。

② 当 $X+Y<-2^{n-1}$ 时，发生溢出。

【例】 已知 X = −1101001，Y = −1011011，用补码进行加法运算。

C_{S+1} C_S　　进位位

负溢出—　1　0　000111　　进位

$[X]_{补}=$　1　0010111　　$[-105]_{补}$

$+)\ [Y]_{补}=$　1　0100101　　$[-91]_{补}$

$[X+Y]_{补}=$　0　0111100　　$[+60]_{补}$

负溢出错误(和小于 −128)

由于$[X]_{补}$和$[Y]_{补}$的符号位均为1，相加后为10，必有进位，即 $C_{S+1}=1$；但是$(X+Y)_{补}$的符号位为0，表示数值的最高位相加时无进位，即 $C_S=0$。由此可知，当 $C_S=0$、$C_{S+1}=1$ 时发生溢出。

综上所述，我们可以得到双进位位判别溢出的重要结论：即当运算结果的两个进位位 C_S 和 C_{S+1} 出现 00 或 11 时，亦即 C_S 和 C_{S+1} 相同时不产生溢出；如出现 01 或 10 时，亦即 C_S 和 C_{S+1} 相异时则产生溢出。换言之，溢出是符号 C_S 和 C_{S+1} 的异或运算。用逻辑式表示为：

$$\text{溢出} = C_S \oplus C_{S+1}$$

这个结论是设计补码加法器或减法器的基础。设计时只需设置相应的异或逻辑即可，任何溢出都可由 C_S 和 C_{S+1} 通过异或逻辑形成。

§2-7 二进制编码的十进制数

如前所述，计算机只能识别二进制数，因此，在计算机中，所有的数字、符号等等都要以特定的二进制码来表示，这就是二进制编码。尽管计算机内部采用二进制数进行运算，但是，因为它不直观，所以在计算机输入输出时，通常还习惯于用十进制数表示，这就要实行二进制和十进制间的转换，其转换步骤如图 2-2 所示。

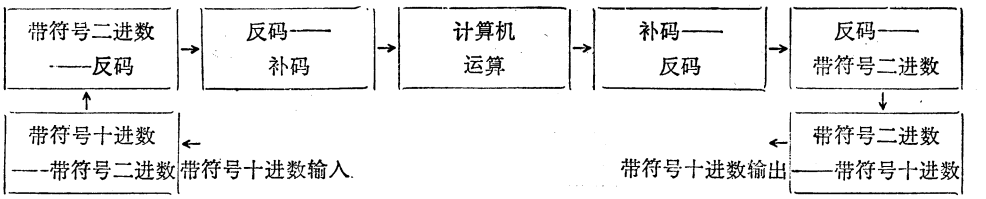


图 2-2 计算机中数的转换过程

显然，这种转换过程要花费很多时间，如果计算的工作量不大，则数制转换所花的时间将会远远超过计算所需的时间。除非计算工作量很大，采用上述转换并用二进制补码计算才是合理的。如果计算工作量不大，为了简化电路和节省计算时间，可采用二进制数对每一位十进制数字编码。这种数制称为二-十进制，简称 BCD(Binary Coded Decimal)。

一、二-十进制码(BCD 码)

这种编码方式的特点是保留了十进制的权，而数字则用二进数码即 0 和 1 的组合来表示。常见的 BCD 码有 8421 码、2421 码、余—3 码、格雷 (Gray) 码等，这里介绍最常用的 8421 BCD 码。

表 2-5 8421 BCD 编码表

十进制数	8421 BCD 码		十进制数	8421 BCD 码	
0		0000	10	0001	0000
1		0001	11	0001	0001
2		0010	12	0001	0010
3		0011	13	0001	0011
4		0100	14	0001	0100
5		0101	15	0001	0101
6		0110	16	0001	0110
7		0111	17	0001	0111
8		1000	18	0001	1000
9		1001	19	0001	1001

在这一编码系统中，用4个二进制数组成十种组合来表示十进制中的10个数字，见表2-5。

从表2-5可知，8421 BCD码是用4位二进制数来表示1位十进制数的，但它是逢"十"进位的，故称二-十进制数。上表中二进制数的每一位都有特定的权，从左到右各位的"权"依次为$2^3=8$、$2^2=4$、$2^1=2$、$2^0=1$，故又称为8421 BCD码，简称8421码。

例如：已知二进制数0101，其对应的十进制数为：

$$0101=(0\times2^3)+(1\times2^2)+(0\times2^1)+(1\times2^0)=5$$

又如：利用8421码，十进制数45可表示为

4	5	十进制数
0100	0101	BCD码

但要注意二进制数01000101B并非十进制数45。因为十进制数4（二进制数0100）的权是十而不是二进制时的$2^4=16$，二-十进制数01000101的等值二进制数为0100乘以"权"10（十进制，10=8+2），再加上0101乘以权1，即此等值二进制数为下列各项之和：

0 1 0 1 × 1 =	0 1 0 1	
0 1 0 0 × 2 =	1 0 0 0	（二进制数乘2，相当于左移1位）
+) 0 1 0 0 × 8 =	1 0 0 0 0 0	（二进制数乘8，相当于左移3位）
45的等值二进制数	1 0 1 1 0 1	

计算机可通过上述方法把二-十进制数转换为等值的二进制数。

此外，4位二进制数可表示16种状态，而十进制数只有0～9共10种状态，即0000～1001。余下6种状态1010～1111在BCD编码中为非法码或称冗余码。倘若在BCD码运算中出现了冗余码，则需作某些修正，才能得到正确的结果。

二、二-十进制加法

前已述及，BCD码低位与高位之间是逢"十"进一的，而4位二进制数是逢十六进一的，因此当两个BCD码相加时，若各位的和都在0～9之间，则其加法运算规则完全同二进制数加法的规则一样；若相加后的低4位（或高4位）二进制数大于9，或大于15（即低4位或高4位的最高位有进位），则应对低4位（或高4位）加6修正。

【例1】 X=21，Y=24，试进行加法运算。

解： 按照二进制数运算规则，有

高4位	低4位
0	0……进位
0010	0001=21
+) 0010	0100=24
0100	0101=45 和数

由于低4位和高4位都无进位，也不超过9，因此不需要进行修正。

【例2】 X=48，Y=69，试进行加法运算。

解： 按照二进制数运算规则，有

	高4位	低4位
	0	1 ……进位
	0100	1000 = 48
+)	0110	1001 = 69
	1011	0001 = B1H(BCD的非法码)
+)	0110	0110 = 66(修正数)
	1 0001	0111 = 117 和数

自动进位

由于高4位的和1011 = 11B大于9，低4位的和10001B大于15，有进位，因此都要加6进行修正，修正后的结果为117。对8位微型计算机来说，高4位的进位1将进入进位位CY，该进位标志对程序可能有用处。通常在微型计算机中，都设置有二-十进制的调整电路，通过一条十进制调整指令DAA，自动地进行上述修正。

三、二-十进制减法

当两个BCD码相减时，若本位的被减数大于减数，即4位二进制数的最高位不发生借位，则不必修正；若该位被减数小于减数，则相减时就会产生借位，借位时，按十进制运算规则，向高位借1作为10，而按二进制运算规则，则借1作为16，为此必须进行减6修正。

【例1】 X = 27，Y = 12，试进行减法运算。

解： 按照二进制数的运算规则，有

	十位	个位	
	0010	0111	= 27
−)	0001	0010	= 12
	0001	0101	= 15

由于4位二进制数的最高位不发生借位，因此不必进行修正。

【例2】 X = 28，Y = 19，试进行减法运算。

解： 按照二进制数的运算规则，有

	十位	个位	
		借位(借1作为16)	
	0010	1000	= 28
−)	0001	1001	= 19
	0000	1111	= 0FH(BCD的非法码)
−)		0110	= 06(修正数)
	0000	1001	= 9 差数

由于个位数向十位数有借位，即二进制数的第三位向第四位(即D_3向D_4)借位时，借1作为16，而BCD中的个位向十位借位时，借1作为10，因此要进行减6修正。

在计算机中，二-十进制数作减法运算时，也是用补码进行运算，减6修正实质上也是按加

$[-6]_{补}$的方式进行的。现以下面实例加以说明。

【例 3】 X = 76, Y = 22, 试进行减法运算。

解： 按照二进制数的运算规则，有

$$
\begin{array}{rlll}
 & \text{十位} & \text{个位} & \\
 & 1 & 1 & \text{进位} \\
[X]_{补} = & 0111 & 0110 & = 76 \\
+)\ [-Y]_{补} = & 1101 & 1110 & = [-22]_{补} \\
\hline
[X-Y]_{补} = & \boxed{1}\ 0101 & 0100 & = 54\ \text{差数}
\end{array}
$$

自动丢失

上式中，二进制数的第 3 位(个位数的最高位)和第 7 位(十位数的最高位)都有进位 1，它们表示两数的个位数和十位数相减时没有借位，故不必进行修正，且相减的结果为正数。

【例 4】 X = 51, Y = 28, 试进行减法运算。

解： 按照二进制数的运算规则，有

$$
\begin{array}{rlll}
 & \text{十位} & \text{个位} & \\
 & 1 & 0 & \text{进位} \\
[X]_{补} = & 0101 & 0001 & = 51 \\
+)\ [-Y]_{补} = & 1101 & 1000 & = [-28]_{补} \\
\hline
[X+Y]_{补} = & \boxed{1}\ 0010 & 1001 & = Z_1\ \text{中间结果}
\end{array}
$$

无进位，表示个位有借位

自动丢失，有进位，表示十位无借位，并表示计算结果为正

上式中，十位数无借位，不必修正；个位数有借位，故需进行减 6 修正，即加$[-6]_{补}$修正。$[-6]_{补}$求法如下：

$$
\begin{array}{ll}
0000 & 0110 \quad = \text{修正数 } 06 \\
1111 & 1010 \quad = -06\ \text{的补码}
\end{array}
$$

现对 Z_1 进行加$[-6]_{补}$修正：

$$
\begin{array}{rll}
 & 0010 & 1001 \quad = Z_1\ \text{中间结果} \\
+) & 1111 & 1010 \quad = [-06]_{补} \\
\hline
 & \boxed{1}\ 0010 & 0011 \quad = Z = 23\ \text{差数}
\end{array}
$$

自动丢失

由于计算 Z_1 时，第 7 位有进位，表示计算结果为正。经修正后得到的 Z = 23 即为最终结果。

【例 5】 X = 21, Y = 76, 试进行减法运算。

解： 按照二进制数运算规则，有

$$
\begin{array}{rcccl}
 & \text{十位} & \text{个位} & & \\
 & 0 & 0 & & \text{进位} \\
[X]= & 0010 & 0001 & = & 21 \\
+)\ [-Y]_{补}= & 1000 & 1010 & = & [-76]_{补} \\
\hline
 & 0\ 1010 & 1011 & = & Z_1\ \text{中间结果}
\end{array}
$$

无进位表示结果为负数

为求得最后结果，可用下列两种方法：

(1) $X+[-Y]_{补}$时，高 4 位和低 4 位的最高位都无进位，表示发生借位，应进行 -66 修正，即加$[-66]_{补}$修正。

$[-66]_{补}$求法如下：

$$
\begin{array}{rcl}
0110\ \ 0110 & = & \text{修正数 66} \\
1001\ \ 1010 & = & -66\ \text{的补码}
\end{array}
$$

现对 Z_1 进行加$[-66]_{补}$修正：

$$
\begin{array}{rcl}
1010\ \ 1011 & = & Z_1\ \text{中间结果} \\
+)\ 1001\ \ 1010 & = & [-66]_{补} \\
\hline
\boxed{1}\ 0100\ \ 0101 & = & Z=45=[X-Y]_{10补}
\end{array}
$$

自动丢失

本例中，由于高 4 位的最高位无进位，故结果为负数，并且是$[X-Y]_{10补}$(对 10^2 的补码)。为求得原码应对其再求补，得

结果：$X-Y=-[100-45]=-55$

(2)对本例还可直接求 Z_1 的补码，再加一负号即为最后结果。Z_1 的补码$[Z_1]_{补}$为 01010101 $=55$，最后结果为$-[Z_1]_{补}=-55$。

上述方法可以用下列方法证明：

$$
\begin{aligned}
X-Y &= 21-76=0010\ 0001-0111\ 0110 \\
&= 0010\ 0001+(1\ 0000\ 0000-0111\ 0110)-1\ 0000\ 0000 \\
&= 0010\ 0001+(1000\ 1010)-1\ 0000\ 0000=1010\ 1011-1\ 0000\ 0000 \\
&\quad\ \uparrow \qquad\qquad \uparrow \qquad\qquad\qquad\qquad \uparrow \\
&\quad (21) \qquad [-76]_{补} \qquad\qquad Z_1=21+[-76]_{补} \\
&= -(1\ 0000\ 0000-1010\ 1011)=-(0101\ 0101)=-[Z_1]_{补}
\end{aligned}
$$

通过上述各例的演示可以看出：当两个正数相减时，即 $X-Y$ 时，可先求出$[-Y]_{补}$，然后再求 $Z=X+[-Y]_{补}$。

若运算中，最高位(第 7 位)有进位，则表示计算结果为正。这时又分两种情况：如第 3 位向第 4 位有进位，则不必修正；如无进位，则应对低 4 位进行减 06 修正。

若运算中，最高位(第 7 位)无进位，则表示计算结果为负。这时，应对高 4 位进行减 6 修正。然后，如第 3 位向第 4 位有进位，则不必修正；如无进位，则应对低 4 位进行减 6 修正[注]。最后对修正后的结果求补，再加一负号，即为最终结果。

【注】 在二-十进制数减法运算中，若减数的低 4 位为0，则变补后相加，必无借位，不必修正。

§2-8 字符的编码

在计算机中，除了处理数字信息外，还必须处理用来组织、控制或表示数据的字母(如英文 26 个字母等)。计算机与外设之间通信，还需识别许多特殊的符号。这些字母和符号统称为字符，它们也必须按特定的规则用二进制编码才能在计算机中表示。

目前在微型计算机中，普遍采用 ASC II 码 (American Standard Code for Information Interchange 美国信息交换标准代码)，编码表见表 2-6。

表 2-6 ASCII (美国信息交换标准码)字符表(7 位码)

LSD $b_3b_2b_1b_0$		MSD $b_6b_5b_4$							
		0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0	0000	NUL	DLE	SP	0	@	P	、	p
1	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	DC2	"	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	$	4	D	T	d	t
5	0101	ENQ	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB	'	7	G	W	g	w
8	1000	BS	CAN	(	8	H	X	h	x
9	1001	HT	EM	)	9	I	Y	i	y
A	1010	LF	SUB	*	:	J	Z	j	z
B	1011	VT	ESC	+	;	K	[	k	{
C	1100	FF	FS	,	<	L	\	l	|
D	1101	CR	GS	-	=	M	]	m	}
E	1110	SO	RS	.	>	N	↑	n	~
F	1111	SI	US	/	?	O	←	o	DEL

注：表中二进制代码按顺序 $b_6\ b_5\ b_4\ b_3\ b_2\ b_1\ b_0$ 排列。

NUL 空
SOH 标题开始
STX 正文结束
ETX 本文结束
EOT 传输结果
ENQ 询问
ACK 承认
BEL 报警符
BS 退一格
HT 横向列表
LF 换行
VT 垂直制表
FF 走纸控制
CR 回车
SO 移位输出
SI 移位输入
DLE 数据链换码
DC_1 设备控制 1
DC_2 设备控制 2
DC_3 设备控制 3
DC_4 设备控制 4
NAK 否定
SYN 空转同步
ETB 信息组传送结束
CAN 作废
EM 纸尽
SUB 减
ESC 换码
FS 文字分隔符
GS 组分隔符
RS 记录分隔符
US 单元分隔符
SP 空格
DEL 作废

ASC II 码包括英文 26 个大写字母、26 个小写字母、0～9 共 10 个数字，还有一些专用字符，如：!、% 等以及控制符号如换行、回车等等共计 100 多个字符。ASC II 码用 6 位、7 位或 8 位来表示数字和字符，常用的是 7 位二进制数编码，它可表示 128 个字符。从表中可看到，数字 0～9，用相应的 0110000～0111001 即 30H～39H 来表示，大写英文字母 A～Z，用相应的 41H～5AH 来表示。

如果用 8 位二进制数编码，其中 7 位用来表示 ASCII 码，余下的最高位（即第 7 位），作为奇偶校验（Parity chek）用。例如字符 C 的 7 位编码为

1000011

它共有 3 个“1”。如果用偶校验，则在最左边的奇偶校验位上加一个“1”，变成偶数个“1”，这时字符 C 的编码变为

1 1000011

如果采用奇校验，则在最左边的奇偶校验位上加一个“0”，变成奇数个“1”，这时字符 C 的编码变为

0 1000011

因此奇偶校验码的编码规则可以用一个简单的数学关系式来表述。设码长度为 n、组成的元素依次为 x_{n-1}、x_{n-2}、$\cdots x_1$、x_0，

若令 $$Y = x_{n-1} + x_{n-2} + \cdots + x_1 + x_0 \quad (\text{模 } 2^n)$$

则有

$$\begin{cases} \text{若满足 } Y = 1, \text{ 构成奇校验码} \\ \text{若满足 } Y = 0, \text{ 构成偶校验码} \end{cases}$$

假如采用偶校验，信息从一处发出时为偶数个“1”，即 Y = 0，当这一信息被传送到另一处时，经过奇偶校验如发现变为奇数个“1”，就说明信息在传递过程中发生了错误，就应作相应的出错处理。应当注意的是奇偶校验只能查出单数性错误，而不能查出双重错误，如信息在传递过程中丢失两个“1”，不过发生这种双重错误的可能性是很小的。

第三章　数字逻辑电路

本章的主要内容，是从组成电子数字计算机的角度，讨论能够实现二进制运算的数字逻辑电路的原理和逻辑功能。如果读者已经具有数字逻辑电路的基础知识，可以越过本章，直接阅读第四章的内容。

§3-1　基本逻辑门 (Basic logic gates)

“逻辑学”是一门研究人类思维规律和形式，以便对客观事物作出正确推理和判断的学科。在人们的社会实践中，常常会遇到两种需要通过逻辑推理加以判断的互相对立的命题或逻辑态(状态)。例如，高(H)与低(L)；真(T)与假(F)；是(Y)与否(N)；通(ON)与断(OFF)等等。

如图 3-1 所示，当开关接通时，灯泡两端加有电池组电压 (H)，灯泡“亮”，表示“真”态；当开关断开时，则灯泡上电压为零(L)，灯泡“暗”，表示“假”态。

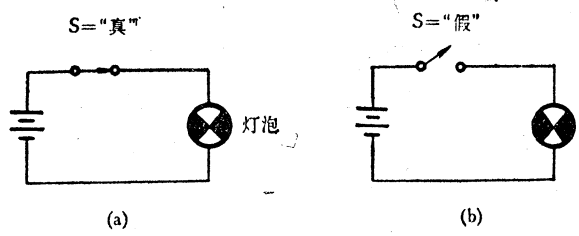


图 3-1　用单刀单掷开关表示“真”态(a)和“假”态(b)

我们可以用数字“1”与“0”，来表示上述逻辑态。如用“1”表示高、真、是、通等，用“0”表示低、假、否、断等。这种只能取“1”与“0”两个数值之一的数学变量称作为逻辑变量。能够对逻辑变量“1”与“0”进行逻辑运算的电路就称为数字逻辑电路。

电子计算机是由成千上万个数字逻辑线路，和一些机械部件组成的复杂的数字逻辑装置。这些逻辑线路，是由若干种最基本的逻辑电路组成的。为了描述逻辑电路(又称门电路或开关电路)和由它们所组成的逻辑线路，选择最佳设计方案，需要用到逻辑代数(或称为开关代数、双值代数)。它是由英国数学家乔治·布尔(G·Boolean)在 1854 年首先提出的，故又称它为布尔代数。这种数学工具不仅能反映单个逻辑元件的两种状态“1”或“0”，而且能反映一个复杂的逻辑线路中各个逻辑元件之间的联系。这种联系反映到数学上就是几种最基本的运算关系：“与”、“或”、“非”运算。人们依据这三种基本的逻辑关系，把许多逻辑线路组合成庞大的电子数字计算机系统。下面介绍几种基本的逻辑门电路的工作原理和逻辑功能。

一、与门 (AND gate)

“与门”是对两个或几个逻辑变量执行逻辑乘法 (“与”运算) 的数字电路。逻辑乘通常称为“与”(AND)，它表示仅当所有参加“与”运算的逻辑变量都取“1”时，逻辑积才为“1”；否则，逻辑积为“0”。常用的两个逻辑变量 A、B 的“与”运算表达式为

$$Y = A \cdot B = AB = A \wedge B \tag{3-1}$$

根据逻辑乘法的定义，对于两个变量、三个变量及其逻辑积的真值表如表 3-1、表 3-2 所示，其中“1”为“真”，“0”为“假”。

表 3-1　Y=A·B 的真值表

A	B	Y=A·B
0	0	0
0	1	0
1	0	0
1	1	1

表 3-2　Y=A·B·C 的真值表

A	B	C	Y=A·B·C
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

从表 3-1、表 3-2 可知，“与”运算的规则和普通代数中乘法相同，故称为逻辑乘法。

利用图 3-2 由开关串联而成的电路，可以实现“与”运算。只有在 A、B 两只开关都接通时，即 A、B 均为“1”时，电池组的电压才能施加在灯泡上。此时，灯泡亮，即 Y 为“1”；否则，灯泡暗，即 Y=0。

现在各种逻辑门采用多种逻辑符号表示，这里介绍两种应用较广的异形和矩形符号。1973 年国际电气与电子工程师学会(IEEE)的 91 号标准 (ANSI-Y32.14-1973) 规定异形和矩形符号可同等地采用，我国仅用矩形符号。图 3-3 所示是两输入端的与门的异形符号(a)和矩形符号(b)。

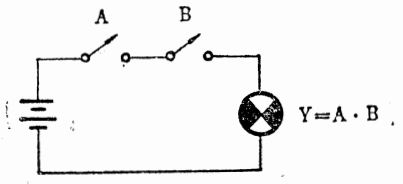
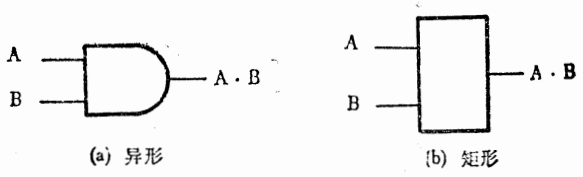


图 3-2　执行 Y=A·B 的电路



(a) 异形　　(b) 矩形

图 3-3　两输入端与门逻辑符号

二、或门 (OR gate)

或门是对两个或两个以上逻辑变量执行逻辑加法(“或”运算)的数字电路。逻辑“加”通常称为“或”(OR)，它表示当任一参加“或”运算的逻辑量取“1”时，逻辑“加”就为“1”。常用的两个逻辑变量 A、B 的“或”运算表达式为

$$Y=A+B=A\vee B \tag{3-2}$$

根据逻辑加法的定义，对于两个变量、三个变量的逻辑加真值表如表 3-3、表 3-4 所示。从表 3-3、表 3-4 可知，逻辑“加”时 1+1=1，这与二进制数的加法运算 1+1=10 是不同的。

利用图 3-4 由开关并联而成的电路，可以实现“或”运算。只要开关 A 和 B 之一或两只开关都接通，电池组电压便使灯泡点亮。两输入端的“或”门的异形和矩形符号如图 3-5 所示。

表 3-3　Y = A + B 的真值表

A	B	Y=A+B
0	0	0
0	1	1
1	0	1
1	1	1

表 3-4　Y = A + B + C 的真值表

A	B	C	Y=A+B+C
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

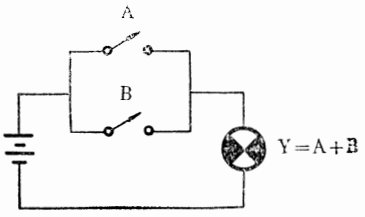


图 3-4　执行 Y=A+B 的电路

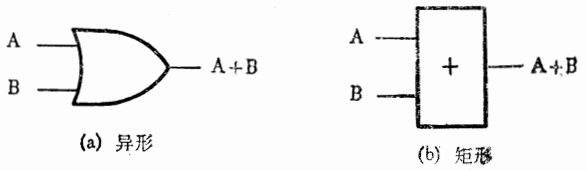


图 3-5　两输入端或门逻辑符号

三、非门（反相器 NOT gate）

非门或反相器是对单一逻辑变量执行逻辑否定（“非”运算）的数字电路。逻辑否定通常称为“非”(NOT)，它表示对变量取反。例如，若 A 为 “1”，A 的“非”运算为非 A，非 A = 0。变量 A 的“非运算”表达式为

$$N = \overline{A} \tag{3-3}$$

式中，$\overline{A}$ 读作 “A 非”或“非 A”。

利用反相器能实现“非”运算。表 3-5 是非门的真值表，其相应的逻辑符号如图 3-6 所示。

表 3-5　非门的真值表

A	$\overline{A}$
0	1
1	0

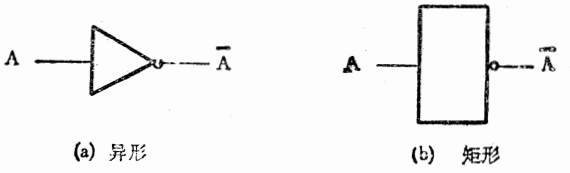


图 3-6　非门的逻辑符号

四、与非门（NAND gate）

与非门是先对两个或两个以上逻辑变量实行“与”运算，然后再对其结果实行“非”运算的数字电路。只要用一个“非”门使“与”门的输出反相，便构成“与非”门。如图 3-7 所示。对变量A、B 的“与非”运算表达式为：

$$Y = \overline{A \cdot B} = \overline{AB} = \overline{A \wedge B} \tag{3-4}$$

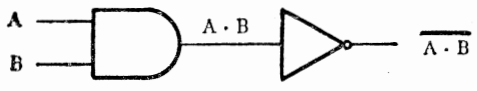


图 3-7　由与门和非门组成的与非门

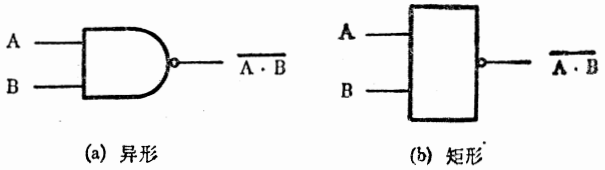


图 3-8　与非门逻辑符号

“与非”门是构成数字集成电路最基本、最常用的逻辑门之一。通常用图 3-8 所示的逻辑符号来简化图 3-7 所示的形式。两个变量和三个变量的“与非”运算真值表列于表 3-6、表 3-7 中。

表 3 6 $Y=\overline{A\cdot B}$

A	B	$Y=\overline{A\cdot B}$
0	0	1
0	1	1
1	0	1
1	1	0

表 3 7 $Y=\overline{A\cdot B\cdot C}$

A	B	C	$Y=\overline{A\cdot B\cdot C}$
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

五、或非门 (NOR gate)

或非门是先对两个或两个以上逻辑变量实行“或”运算，然后再对其结果实行“非”运算的数字电路。只要用一个“非”门使“或”门的输出反相，便构成“或非”门。如图 3-9 所示，变量 A、B 的或非运算表达式为

$$Y=\overline{A+B}=\overline{A\vee B} \tag{3-5}$$

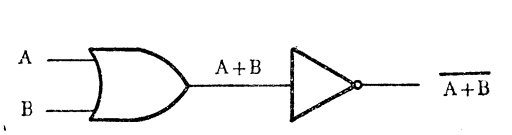


图 3-9 由或门和非门组成的或非门

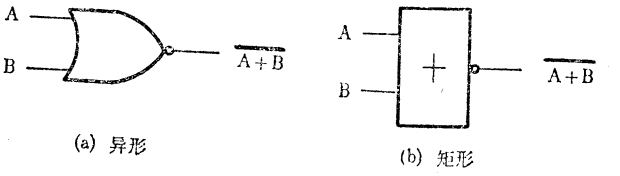


图 3-10 或非门的逻辑符号

“或非”门同样是构成数字集成电路的最基本的逻辑门之一，通常用图 3-10 所示的逻辑符号来简化图 3-9 所示的形式。两个变量和三个变量的或非运算真值表列于表 3-8、表 3-9中。

表 3-8 $Y=\overline{A+B}$

A	B	Y=A+B
0	0	1
0	1	0
1	0	0
1	1	0

表 3-9 $Y=\overline{A+B+C}$

A	B	C	$Y=\overline{A+B+C}$
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

§3-2 布尔代数 (Boolean Algebra)和逻辑网络

如前所述，布尔代数是一种用来分析和综合数字逻辑系统的强有力的数学工具，布尔代数在电子计算机中主要用来解决两个基本问题：

第一，逻辑线路的分析。对已经设计好的线路，应用逻辑表达式来描述它的工作，然后将这个逻辑表达式作形式上的变换，用于分析线路的经济性。也就是说，通过分析的方法来观察

是否可用更简单的具有更少数量的逻辑门，以完成给定的功能。

第二，逻辑线路的综合。对于按给定逻辑功能的逻辑函数(即由逻辑变量按一定规律组织起来的函数关系式)，用适当的方法加以变换，简化为最简的逻辑关系式，以便用最少的逻辑门，来实现同一逻辑功能的逻辑网络。常用的最小实现法有卡诺(M. Karnaugh)作图法等。限于本书篇幅对此不作讨论，读者可参阅有关书籍。

一、基本定理

如§3-1所述，逻辑变量只有三种最基本的运算，即逻辑乘、逻辑加及非运算。它们的运算规则如下：

1. $0\cdot 0=0$ (3-6a) $1+1=1$ (3-6b)
2. $0\cdot 1=1\cdot 0=0$ (3-7a) $0+1=1+0=1$ (3-7b)
3. $1\cdot 1=1$ (3-8a) $0+0=0$ (3-8b)
4. $\overline{1}=0$ (3-9a) $\overline{0}=1$ (3-9b)

研究上列方程可以发现：每一对都存在对偶关系，即把0与1对调，·与+对调，就可以从一个等式变为与之对偶的另一个等式。

根据逻辑变量的取值特点以及三种基本运算规则，可以证明下面十个基本定理：

1. 交换律

$A\cdot B=B\cdot A$ (3-10a) $A+B=B+A$ (3-10b)

2. 结合律

$A\cdot(B\cdot C)=(A\cdot B)\cdot C$ (3-11a) $A+(B+C)=(A+B)+C$ (3-11b)

3. 分配律

$A\cdot(B+C)=(A\cdot B)+(A\cdot C)$ (3-12a) $A+(B\cdot C)=(A+B)\cdot(A+C)$ (3-12b)

4. 恒等律

$A\cdot 1=A$ (3-13a) $A+0=A$ (3-13b)

5. 衡消律

$A\cdot 0=0$ (3-14a) $A+1=1$ (3-14b)

6. 互补律

$A\cdot\overline{A}=0$ (3-15a) $A+\overline{A}=1$ (3-15b)

7. 重叠律

$A\cdot A=A$ (3-16a) $A+A=A$ (3-16b)

8. 吸收律

$A+A\cdot B=A$ (3-17a) $A\cdot(A+B)=A$ (3-17b)

9. 折转律

$$\overline{\overline{A}}=A \tag{3-18}$$

10. 反演律〔狄摩根(Demorgan)定理〕

$$\overline{(A\cdot B\cdot C\cdots)}=\overline{A}+\overline{B}+\overline{C}+\cdots \tag{3-19a}$$

$$\overline{(A+B+C+\cdots)}=\overline{A}\cdot\overline{B}\cdot\overline{C}\cdot\cdots \tag{3-19b}$$

利用上述基本定理可以方便地对逻辑函数进行化简或变换，使所设计的逻辑线路变得简单、合理。

二、组合逻辑网络

组合逻辑网络是指在某一时刻，若网络输出端的状态，只与现时输入端的状态有关的逻辑网络。例如，上述各种门电路都是简单的组合电路。计算机中典型的组合逻辑网络有加法器、减法器、译码器、奇偶校验电路等。如果在网络中存在记忆元件，按给定的时间顺序工作，输出不仅与现时输入端的状态有关，而且也与输入端在此以前的状态有关，则称此电路为时序逻辑网络。计算机中典型的时序逻辑网络有乘法器、除法器、触发器、多谐振荡器、计数器、寄存器、移位电路等。

实践证明，在数字系统设计中所遇到的逻辑函数，都可以用逻辑网络实现，无论多么复杂的逻辑网络，都能用上述的基本逻辑门组合而成。由于组合逻辑网络的输出，只与输入的现时状态有关。因此，只要通过真值表列出输入与输出的对应关系，就可以设计出相应的逻辑网络。

下面以异或门设计为例来说明上述原理：

异或门(Exclusive-OR gate 简写为 XOR)是构成加法器、减法器的基本门电路。它的定义是仅当逻辑变量取异值时其输出才为 1，否则为 0。在二进制加法中，若不考虑进位，则有下列规则：

$$\begin{array}{rrrr} 0 & 0 & 1 & 1 \\ +\ 0 & +\ 1 & +\ 0 & +\ 1 \\ \hline 0 & 1 & 1 & \text{丢失}\leftarrow\boxed{1}\ 0 \end{array}$$

这种不考虑进位的加法，称为按位加。能够实现按位加的逻辑电路即为异或门电路。据此，异或运算的真值表应如表 3-10 所示。从表中可知，仅当 $\overline{A}$ 与 B 或 A 与 $\overline{B}$ 的条件成立时即 A、B 相异时，输出 Y 才为 1；而当 A、B 相同时，输出 Y 为 0。因此异或运算的逻辑表达式为

$$Y=(A\cdot\overline{B})+(\overline{A}\cdot B)=A\oplus B \tag{3-20}$$

表 3-10　异或门真值表

输　入		输　出
A	B	$Y=A\oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

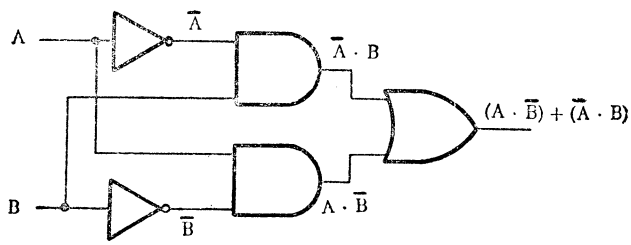


图 3-11　A-O-N 形式的异或门电路

布尔表达式的这种形式，称为积-和式，式中两个逻辑积 $A\cdot\overline{B}$ 和 $\overline{A}\cdot B$ 则称为小项。通常异或运算用符号⊕表示，读作“A 异或 B”。

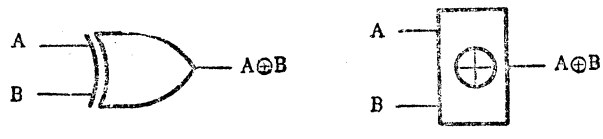
实现上述异或运算的一种“与”-“或”-“非”(A-O-N) 式的异或门电路如图 3-11 所示。异或门逻辑符号如图 3-12 所示。

由异或门的设计推广到一般组合逻辑网络的设计，其步骤大致如下：

1．按照系统提出的要求列出真值表；

2．列出输出为 1 时各输入的逻辑积(与运算，即小项)；

3．取所有输出为 1 的小项的逻辑和(或运算)；



(a) 异形　　(b) 矩形

图 3-12　异或门逻辑符号

4．利用逻辑代数或卡诺图、奎因-麦克劳斯制表法进行化简；

5．根据最简的逻辑表达式绘出逻辑网络图。

三、正逻辑与负逻辑

如果逻辑变量是电信号，就必须按不同的电平来区分逻辑“0”与逻辑“1”。通常规定：

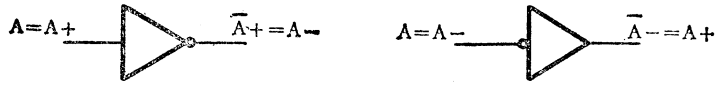
若用高电平表示逻辑“1”，用低电平表示逻辑“0”，这种表示法称为正逻辑。反之，若用低电平表示逻辑“1”，用高电平表示逻辑“0”，这种表示法称为负逻辑。

假设变量 A，用“A_+”表示采用正逻辑，用“A_-”表示负逻辑。根据正、负逻辑的定义可得出：

$$A_+ = \overline{A}_- \tag{3-21a}$$

$$A_- = \overline{A}_+ \tag{3-21b}$$

这种关系可由图 3-13 的非门符号表示。这就是说，“非”门可用来将正逻辑变换为负逻辑，反之亦然。



(a) 正逻辑　　(b) 负逻辑

图 3-13　非门

对于两输入的逻辑门而言，在正、负逻辑情况下的真值表如表 3-11a 和表 3-11b 所示。

表 3-11a　正逻辑真值表

输　入		输　出				
A	B	与　门	或　门	与非门	或非门	异或门
0	0	0	0	1	1	0
0	1	0	1	1	0	1
1	0	0	1	1	0	1
1	1	1	1	0	0	0

表 3-11b　负逻辑真值表

输　入		输　出				
A	B	与　门	或　门	与非门	或非门	异或门
0	0	0	0	1	1	1
0	1	1	0	0	1	0
1	0	1	0	0	1	0
1	1	1	1	0	0	1

对照表 3-11a 和表 3-11b 可以看出两点：第一，把所有正逻辑输入和输出分别取反，即把所有 0 与 1 对换，就得到相应的负逻辑；第二，正逻辑的与门、或门、与非门和或非门分别是负逻辑的或门、与门、或非门和与非门。它们的逻辑符号如图 3-14 所示。

现以正逻辑的或门就是负逻辑的与门为例予以说明。从表 3-11a、3-11b 可知：只要把正逻辑或门的输入、输出两端分别取反就得到负逻辑的与门了。即

$$\overline{A_+ + B_+} = \overline{A}_+ \cdot \overline{B}_+ = A_- \cdot B_-$$

它们的逻辑符号如图 3-15 所示。

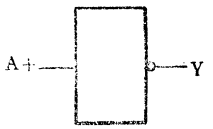
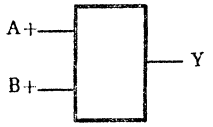
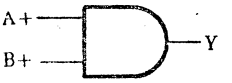
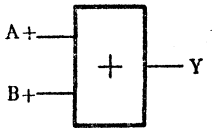
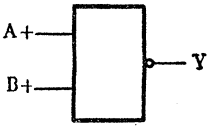
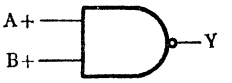
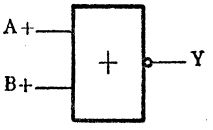
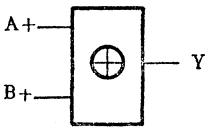
序号	名称	国家标准（正逻辑）	国际标准（正逻辑）	国际标准（负逻辑）	逻辑表达式
1	非门				$Y=\overline{A}$
2	十与门 一或门				$Y=A\cdot B$ $=\overline{(\overline{A}+\overline{B})}$
3	十或门 一与门				$Y=A+B$ $=\overline{\overline{A}\cdot\overline{B}}$
4	正与非门 负与非门				$Y=\overline{A\cdot B}$ $=\overline{A}+\overline{B}$
5	正或非门 负或非门				$Y=\overline{A+B}$ $=\overline{A}\cdot\overline{B}$
6	异或门				$Y=A\oplus B$ $=A\overline{B}+\overline{A}B$

图 3-14　正、负逻辑的逻辑门所对应的逻辑符号

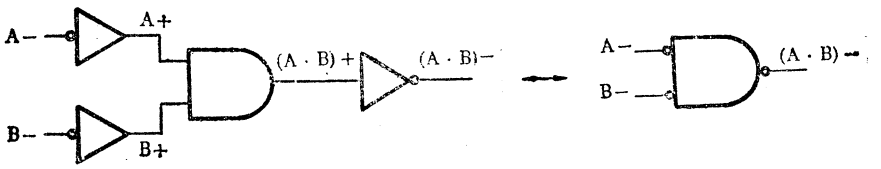


图 3-15　负逻辑“与”门

§ 3-3　基本逻辑部件

计算机的各个组成部件如运算器、控制器等都是由基本逻辑部件构成的。基本逻辑部件是由逻辑门电路和触发器所组成的具有一定逻辑功能的部件。本章从组成电子计算机的角度来介绍常用的基本逻辑部件，如寄存器、计数器、译码器、加法器、运算器以及总线系统等。

一、触发器(Flip-flop)

触发器是计算机中最常用的、能够暂时保存“1”和“0”代码的逻辑电路。与上述的组合逻辑网络不同，它是一种时序逻辑网络，即它的输出值不仅取决于现时的输入值，而且还取决于过去的输入值。因此它具有记忆特性。一个触发器可以存放一位二进制数(bit)的信息。常用的触发器有：

1．门控 R-S 触发器(时钟 R-S 触发器)

门控 R-S (复位-置位) 触发器是组成存贮器的基本单元之一。图 3-16 示出 R-S 触发器的逻辑符号。其中，Q 及 $\overline{Q}$ 表示触发器的输出端；CLK 表示同步时钟输入（也可用 CP、CK、CL 及 T)，用它来控制触发器的状态作同步变化，即门控 R-S 触发器只按此时钟线上发生的脉冲作相应的变化。

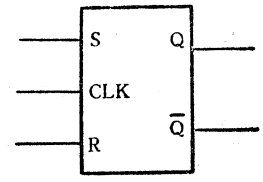


图 3-16 R-S触发器逻辑符号

门控 R-S 触发器是由基本逻辑门构成的，用 4 个与非门即可构成。如图 3-17 所示。其真值表如表 3-12 所示。表中 t_n 表示 CLK 端加脉冲（时钟脉冲或选通脉冲）的时间，t_{n+1} 表示该脉冲结束后的时间。

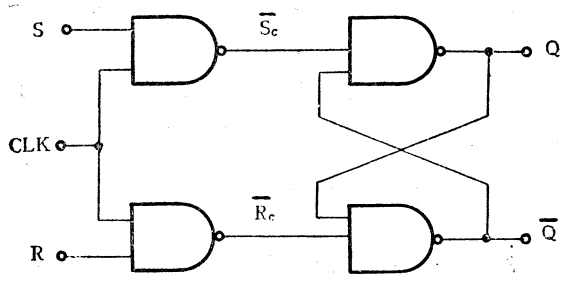


图 3-17 由与非门组成的门控 R-S 触发器

表 3-12 门控 R-S 触发器的真值表

t_n		t_{n+1}	
S	R	Q	$\overline{Q}$
0	0	Q_n	$\overline{Q}_n$
0	1	0	1
1	0	1	0
1	1	不	定

在 CLK 端为逻辑“1”时，触发器处于“允许”状态。此时，若 R-S 有变化，则输出状态就跟随输入而变化。当 CLK 端为逻辑“0”时，触发器的输出状态被封锁，不反映输入的变化。因而，这类触发器又称为时钟锁存器或 R-S 锁存器。这种能使输入对输出起作用的方法称为电平电压启动法，其时间波形图，如图 3-18 所示。在采用这种方法启动时，在时钟输入为逻辑 1 的整个期间，时钟锁存器的状态随着输入值的变化而改变，如果时钟脉冲高电平太宽，R-S 锁存器的状态就会来回翻转多次，这就造成不能可靠触发的空翻现象。

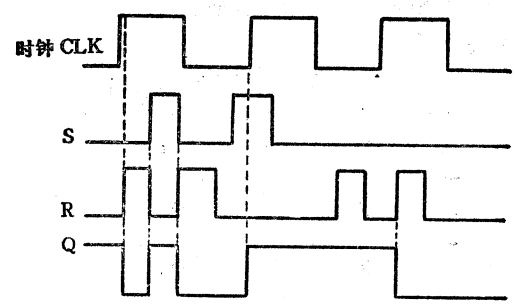


图 3-18 R-S 锁存器的时间波形图

为了使触发器的状态随着时钟脉冲准确地变换一次，常采用边沿触发的启动方法，这种方法的实质，是保证输入信号的生效和输出状态的转换都发生在时钟脉冲上一定的跳沿时间上。为了使输入能在时钟从低到高跳变（正边沿或上升沿触发）或在从高到低跳变(负边沿或下降沿触发）时启动触发器，必须设计适当的输入选通电路。采用边沿触发方式的 R-S 触发器称为

边沿触发 R-S 触发器。负边沿触发的时间波形图如图 3-19 所示，其中仅在时钟脉冲的负边沿到来时，S 和 R 的输入才能影响 Q 的状态作相应的改变；而在此以前，S 和 R 的任何变化都不会影响触发器的输出状态。

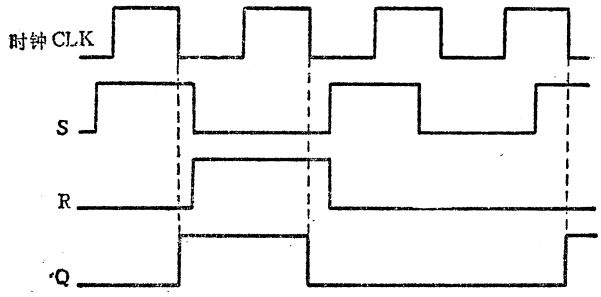


图 3-19　负边沿触发 R-S 触发器的时间波形图

但是边沿触发也有缺点，这就是在触发器改变状态的瞬间，如果刚好发生输入条件变化，就会造成输出状态不确定的结果(竞态情况)。利用主从触发器可以克服上述缺点，其工作过程将在 J-K 触发器中介绍。

2．J-K触发器(J-K flip-flop)

J-K 触发器和门控 R-S 触发器的区别在于不再存在无效的输入条件(即造成输出不定的条件)。J-K 触发器包括两个数据输入端(J 和 K)、一个时钟输入(CLK)以及输出 Q 和 $\overline{Q}$，如图 3-20 所示，其真值表如表 3-13 所示，表中各符号的意义与表 3-12 是相同的。

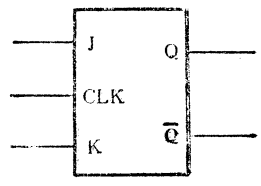


图 3-20　J-K 触发器的逻辑符号

J-K 触发器电路如图 3-21 所示。

比较表 3-12 和表 3-13 可知：在前三种条件下，只要用 J 和 K 分别代替 S 和 R，两种触发器的功能相同。区别仅在于最后一种条件下，若在 t_n 时 J 和 K 均为逻辑 1，则在 t_{n+1} 时 $Q=\overline{Q}_n$，即在 $J=K=1$ 时，每来一个时钟脉冲 CLK，输出就改变(翻转)一次。

表 3-13　J-K 触发器真值表

t_n		t_{n+1}	
J	K	Q	$\overline{Q}$
0	0	Q_n	$\overline{Q}_n$
0	1	0	1
1	0	1	0
1	1	$\overline{Q}_n$	Q_n

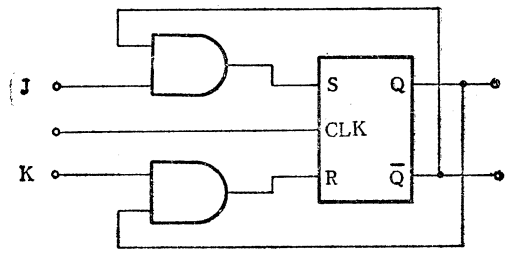


图 3-21　J-K 触发器电路

把 J 和 K 端连接起来形成一个输入端，就构成 T 型触发器。其逻辑符号见图 3-22，真值表见表 3-14。

表 3-14　T 型触发器的真值表

t_n	t_{n+1}
T	Q
0	Q_n
1	$\overline{Q}_n$

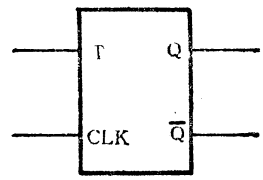


图 3-22　T 型触发器的逻辑符号

T 型触发器是构成计数器的重要单元之一。当 T 为逻辑 "1" 时，触发器的状态将随着每个时钟脉冲而变反。反之，则状态保持不变。这种触发器也称为"套环"(Toggle)触发器，它的输入端也因此而被命名为 "T" 端。

J-K 触发器一般有两类：边沿触发器和主从 J-K 触发器。边沿触发器的特点是在送入输入逻辑值时，它的输出按时钟脉冲的前沿或后沿进行置位。一般多数采用负边沿触发。主从 J-K 触发器的特点是在 CLK 脉冲前沿做准备，而在后沿才真正触发，这样翻转可靠。

主从触发器由主触发器 M 和从触发器 S 两部分组成，触发器的工作仍由时钟脉冲控制，其电路图和时序图如图 3-23 和 3-24 所示，其真值表与表 3-14相同。主从 J-K 触发器的工作过程如下：静态(稳态)时，主从两触发器的状态是一致的，并且由主触发器决定从触发的状态。动态时，CLK 脉冲前沿使主触发器翻转，在 CLK 脉冲的持续时间内从触发器的状态保持不变，只有在 CLK 脉冲结束(后沿产生)后才能变化。对触发器整体来说，相当于 CLK 脉冲前沿作准备，后沿触发翻转。由于 CLK 脉冲分别对主、从两触发器有封锁作用，因此两者互不影响，翻转可靠。如图 3-24 所示，首先由时钟线输入低值启动，这时由于 S 相为逻辑 1，故主触

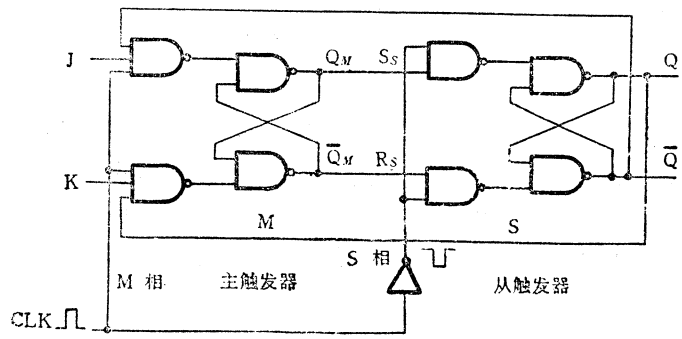


图 3-23 主从式 J-K 触发器

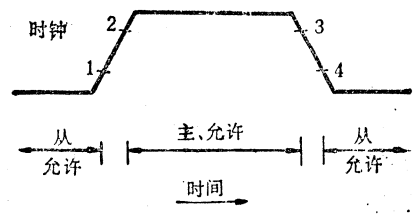


图 3-24 主从 J-K 触发器的主要时序

发器与从触发器相耦合。当时钟脉冲到达 1 处时，从触发器与主触发器退出耦合，数据只进入主触发器；在 2 处，J 和 K 输入启动主触发器，使主触发器置位；到时钟脉 3 处，J 和 K 输入与主触发器退耦，即任何输入变化不影响主触发器输出；最后到 4 处，主从触器再度耦合，此时主触发器的状态便送到从触发器，从而 Q 和 $\overline{Q}$ 得到相应的输出。

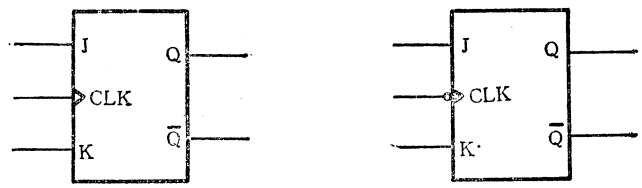


图 3-25 主从 J-K 触发器的逻辑符号

为了与时钟锁存器相区别，有时用一个小△形画在 CLK 时钟线端上，如图 3-25 所示。△表示主从触发器输出值的改变，只允许出现在时钟脉冲的一个边沿上。为了区分正边沿或负边沿，通常对输出变化发生在时钟线的负边沿的主从触发器，在 CLK 端画一个反相小圆圈来表示。

3．D 触发器和 D-E 触发器

D 触发器是门控 R-S 触发器的变形，除保留时钟输入端外，只有一个数据输入端。逻辑符号见图 3-26。真值表见表 3-15。

从表中看出，若 D = 0(1)，则在 t_{n+1} 时 Q = 0(1)，即 D 触发器的状态将随着输入线 D 上出现的逻辑值而变化。故又称为延迟(D)触发器。

D 触发器的电路如图 3-27 所示。

D 触发器一般都是边沿触发的，或主从式的。边沿触发时通常用正边沿触发。

D-E 触发器也称为 D 锁存器，或双稳态锁存器。其逻辑符号见图 3-28，真值表见表 3-16。

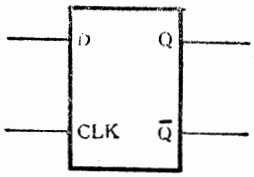


图 3-26　D 触发器的逻辑符号

表 3-15　D 触发器真值表

t_n	t_{n+1}	
D	Q	$\overline{Q}$
0	0	1
1	1	0

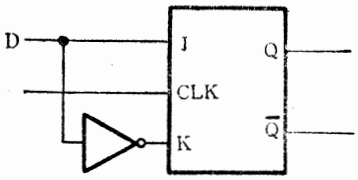


图 3-27　D 触发器电路

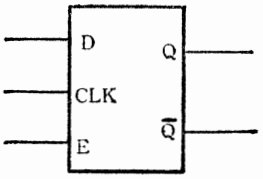


图 3-28　D-E 触发器的逻辑符号

D-E 触发器是在一个一般的D触发器上，通过逻辑门另加一个 E 输入端（允许端或启动端）而构成的。这里，E 输入端用作控制D端的输入。当 E 为逻辑 “0” 时，即使D端输入发生变化，触发器状态仍保持不变；仅当 E 为逻辑 “1” 时，触发器在 CLK 脉冲触发下，输出 Q 才跟随D端的输入值而变化，即 E 允许端起到控制作用。现在，D-E 触发器被广泛用于微型计算机中的总线结构和 I/O 接口电路中。

表 3-16　D-E 触发器真值表

t_n		t_{n+1}	
D	E	Q	$\overline{Q}$
0	0	Q_n	$\overline{Q_n}$
0	1	0	1
1	0	Q_n	$\overline{Q_n}$
1	1	1	0

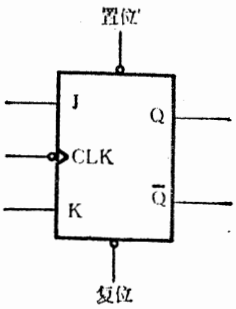


图 3-29　具有置位和复位端的 J-K 触发器

这里还要指出，以上 J-K 触发器、D 触发器和 D-E 触发器通常还带有置位（预置）S 和复位（清除）R 输入端，如图 3-29 所示。当置位和复位端无有效输入时，触发器的状态由一般的输入信号和时钟线来确定。当置位或复位端有有效输入时，触发器便被强制到相应的 “1” 或 “0” 态。

二、寄存器(Register)

计算机的各个部分都需要能暂时存放数码的部件。所有能够存放数码的部件统称为寄存器。它由一组触发器构成，若要寄存 n 位二进制数码，就需要有 n 个触发器。通常按寄存器的逻辑功能，可以分为数码寄存器和移位寄存器两种。

1．数码寄存器

对于数码寄存器而言，应具有存数（接收代码）、取数（发送代码）和清除（清 0）的功能。它可用 R-S 型、D 型或 J-K 型触发器构成。

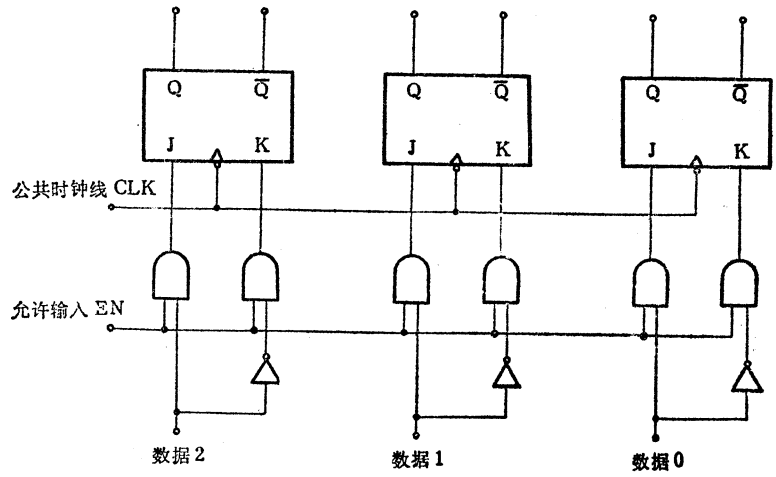


图 3-30　由 J-K 主从触发器组成的数码寄存器

图 3-30 所示的是由 J-K 主从触发器构成的三位数码寄存器。为了与系统中其它部件保持同步，把公共时钟线连接到中央时钟源上去。利用两个“与”门和一个反相器组成一个寄存器输入控制门，以使信息有选择地传送到寄存器中去。从图 3-30 中看出，只有当允许输入线为逻辑“1”时，从数据源来的每个信号才能到达相应的触发器 J 端，并经反相后到达 K 端。这样，寄存器中每个触发器才能在 CLK 脉冲的同步下，寄存相应的数码。

2．移位寄存器

计算机在进行算术运算和逻辑运算时，常常需要把寄存器中的数向左或向右移位。能实现移位功能的寄存器称为移位寄存器。

(1) 串行输入移位寄存器

图 3-31 示出一个由 D 触发器构成的串行输入移位寄存器。在这种移位寄存器中，只从最左位触发器 Q_0 的 D 端输入数码，时钟脉冲 CLK 作为移位信号，逐位将数码移到寄存器的各位中去，而由最右位输出代码，这种方式叫做串行输入、串行输出方式；也可以用并行方式取出，称为串行输入、并行输出，可用于把串行输入转换为并行输出。复位端用于开始工作时清零，即先将各输出端 Q_0～Q_3 置 “0”。

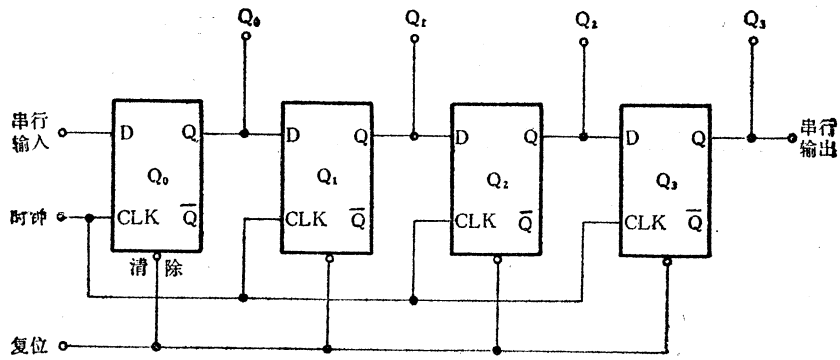


图 3-31　四位串行输入移位寄存器

(2) 并行输入移位寄存器

图 3-32 示出一个由 J-K 触发器构成的四位并行输入移位寄存器。开始工作时，先在复

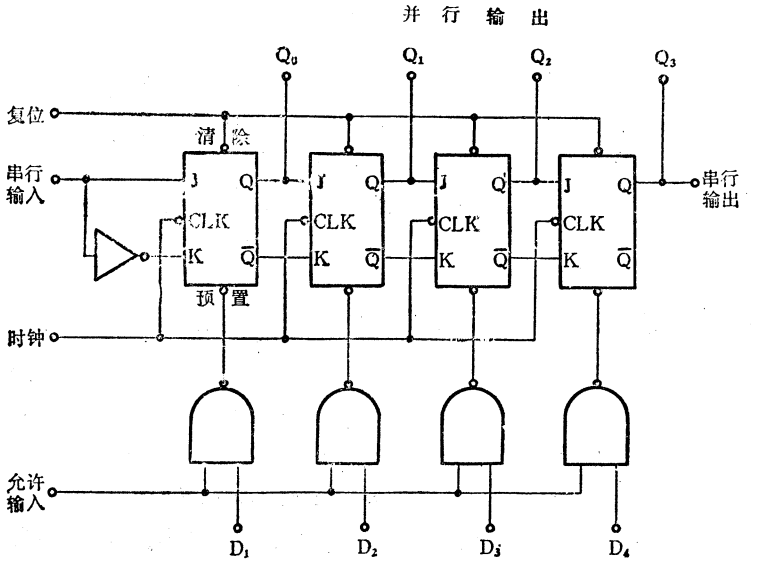


图 3-32 四位并行输入移位寄存器

位端加一个清零脉冲(逻辑 0)，然后把 D_1～D_4 一起加到各输入端上，同时给允许输入线加一个正脉冲(逻辑 1)，于是数据便通过预置端存入所有的触发器。此后，每来一个时钟脉冲，数据便向右移位一位。数据既可以串行读出，也可以并行读出。并行输入、串行输出的移位寄存器也叫并-串行转换器。

以上所述的移位寄存器是单向移动的，实际上常用的是能够双向移动数据的寄存器。为此，必须在电路中设置两个寄存器控制门，分别控制右移或左移。

三、计数器(Counter)

1．二进制计数器

计数器是用来累计并寄存输入脉冲数目的逻辑部件。可用本章前面所述的任何一种触发器来构成二进制计数器，图 3-33 是采用负边沿触发的 T 型主从触发器构成的 4 位二进制计数器电路。它的逻辑符号如图 3-34 所示。由于 4 个触发器不是同时接受时钟脉冲的，因此称为异步计数器。其中每位触发器的 Q 输出端接到下一位触发器的 CLK 计数端，而将所有触发器的 T 端连接到“允许计数”线上，时钟脉冲 CLK 则加于最右面触发器的时钟 CLK 端，从而构成了当

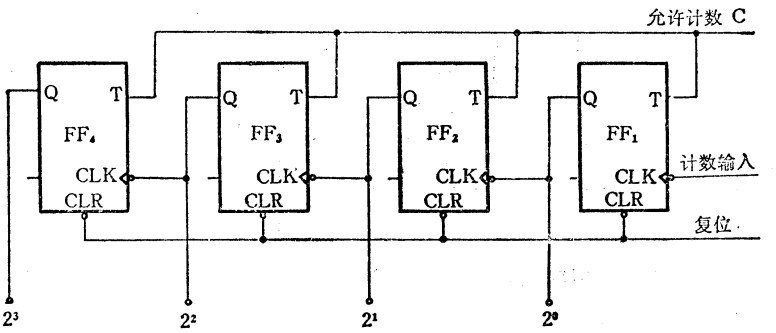


图 3-33 4 位二进制异步计数器电路

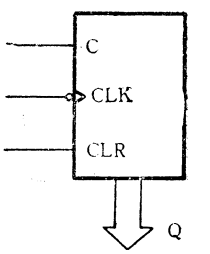


图 3-34 计数器的逻辑符号

允许计数线置逻辑“1”时，在CLK脉冲的每一负边沿处触发器改变一次状态。设最初全部触发器都已清除(置“0”)，当第一个CLK脉冲来到时，图中2°位被置“1”，其余各位仍保持“0”；第二个CLK脉冲来后，使FF_1复位，其输出就是FF_2的输入，作2^1位置“1”而其它位并不改变状态。这样，各触发器的状态的跳变循序自右向左推移连续计数，其时序见图3-35。实际上计数器也是一种分频器。2°位使时钟频率2分频，同样，2^1、2^2和2^3位分别使时钟频率4分频、8分频和16分频。

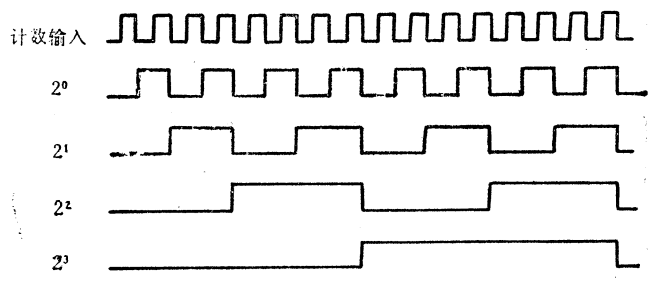


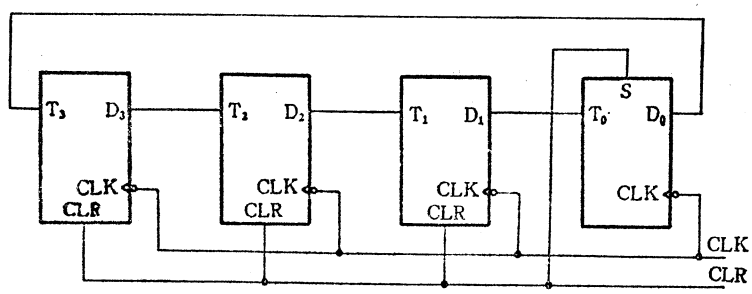
图 3-35　四位二进制异步计数器的时间波形

计数器的种类很多，按计数制来分有二进制计数器和非二进制计数器；按计数功能来分又有加法、减法和加减法都能进行的可逆计数器等等，限于本书篇幅，这些内容不作专门介绍。

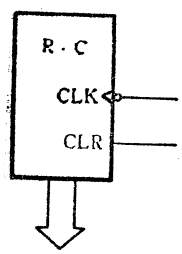
2．环形计数器

以上介绍的是二进制异步计数器，如果把计数脉冲直接加到计数器的各级触发器，使得各位触发器在计数脉冲的作用下同时翻转，这种计数器称为同步计数器。利用移位寄存器可构成移位计数器，它是同步计数器的一种变型，它又分为环形计数器和扭环形计数器。图3-36 (a)、(b)分别示出了环形计数器的电路图和框图。此电路是将一个4位移位寄存器的输出端接到输入端而构成的。其工作过程如下：

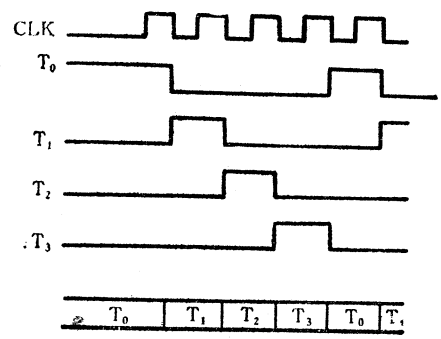
当CLR = 1时，$T_3 = 0$，$T_2 = 0$，$T_1 = 0$，$T_0 = 1$，即T = 0001。当送入时钟脉冲CLK时，每出



(a) 电路



(b) 框图



(C) 时间图

图 3-36　环形计数器

现一次负跳沿，T 依次为 0010→0100→1000→0001→…，如图 3-36(c)所示。由于环形计数器每个输出端的状态是按一定顺序变化的，因此可作为数字系统中的脉冲序列发生器和脉冲分配器。

四、输出缓冲器(Output Buffers)

微型计算机系统中通常用来隔离输出端的缓冲器有两种电路形式：

1．集电极开路门

图 3-37(a) 为标准的 TTL 与非门，(b)为集电极开路的与非门。两者的差别仅在于图(b)中省去了晶体管 T_4 集电极上的元件。

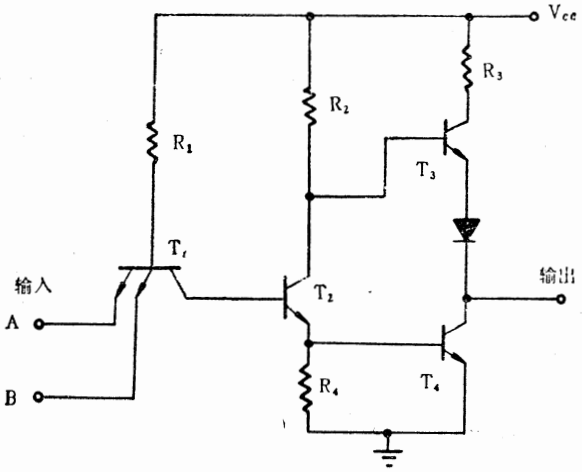


图 3-37(a) 标准的 TTL 与非门

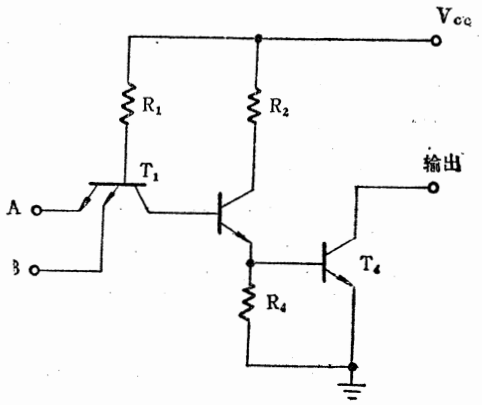


图 3-37(b) 集电极开路的 TTL 与非门

对图 3-37(a) 标准的 TTL 与非门来说，如果把两个门的输出端连在一起，在一个处于低态而另一个处于高态的情况下，从电源 V_{cc} 经过两个并联的 R_3，贯穿低态输出管到地的电流比一个门单独使用时(只有一个 R_3) 大得多，可能超过额定电流，因此不允许两个或更多个的标准逻辑门的输出端连接在一起。

采用图 3-37(b) 的集电极开路门，它们的输出端可以连接在一起。这是因为它们在工作时，在输出管集电极到 V_{cc} 之间外接一个限流电阻(一般为 2kΩ)，所以流过的电流不会超过额定值。

2．三态门 (Tri-State Gates)

在采用总线结构的计算机系统中，许多存贮器芯片、输入/输出接口电路都并联在总线上，在某一瞬时只能允许一个设备的发送门打开发送信息，为了避免挂在总线上的其它器件输出对总线的影响，要求凡是挂到总线上的器件(芯片)的数据线都必须经过三态门输出。所谓三态就是指有三种不同的状态，即除了 "0" 和 "1" 状态外，还有第三态——高阻抗状态。此时，其输出端虽连在总线上，但是在电气上是与总线断开的，因而对总线不会产生任何影响。

图 3-38(a)、(b) 就是 TTL 三态与非门电路及其逻辑符号。它是在标准 TTL 门的基础上增加控制端构成的。当控制输入被允许(置 1)，作用就与普通的 TTL 门相同，输出为低阻抗逻辑 1 和逻辑 0。当控制输入被禁止(置 0)，电路呈现高阻抗状态即第三态，此时，T_4 和 T_5 均截止，不论输入为 1 或 0，输出均为高阻抗状态相当于开路。表 3-17 所示是三态非门的真值表。

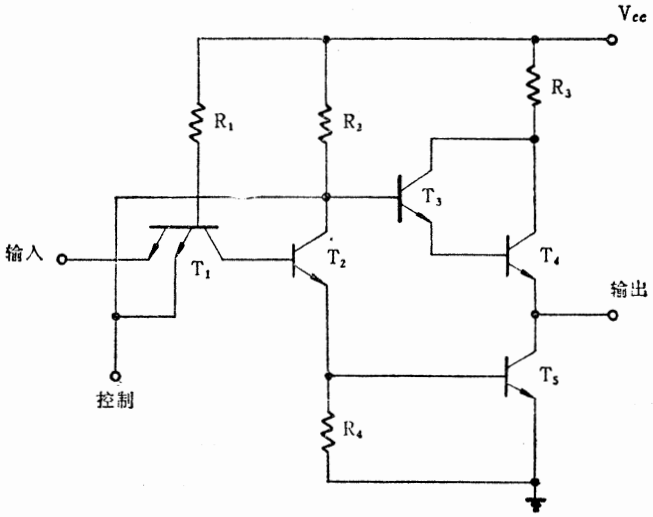


图 3-38(a) TTL 三态"非"门电路

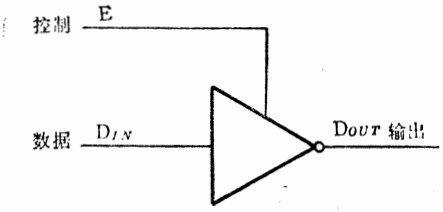


图 3-38 (b)逻辑符号

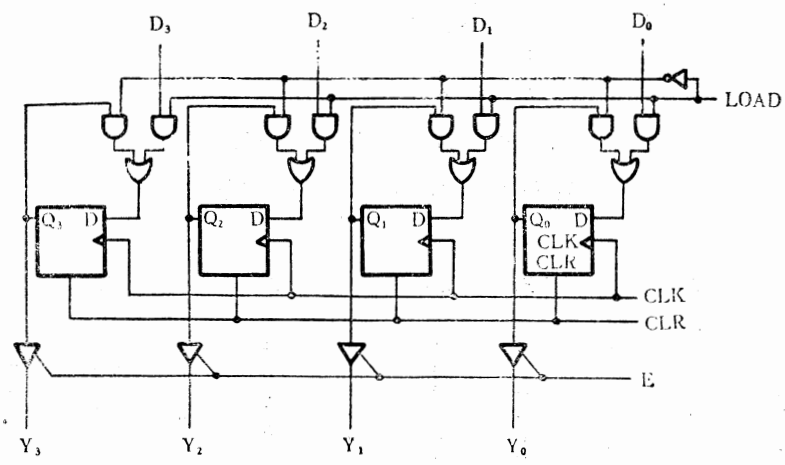
三态门也可以采用 MOS 工艺的逻辑电路构成。

表 3-17 三态"非"门的真值表

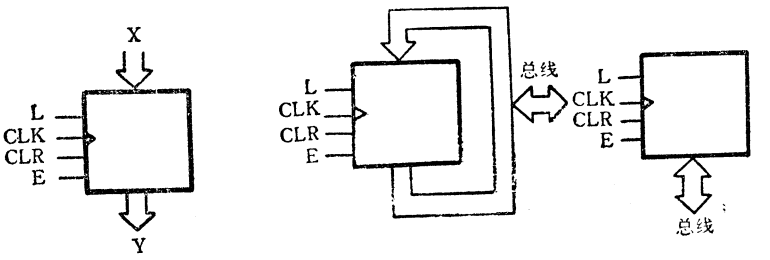
数 据	控 制	输 出
0	0	高 阻 抗
1	0	高 阻 抗
0	1	1
1	1	0

3．三态输出寄存器

利用三态门可以将二态输出的寄存器更改为三态输出的缓冲寄存器，如图 3-39 所示。当 CLR = 1 时，所有触发器被复位，Q = 0。当 LOAD = 1 时，在 CLK 脉冲正跳沿瞬间，将数据 D 装入 Q，即 Q = D。当 LOAD = 0 时，Q 保持不变。这是数据装入时的情况。输出数据时需视允许输出信号 E 来确定。当 E = 1 时，输出 Y = Q；当 E = 0 时，输出 Y = 高阻抗，与各个触发器的 Q 的状态无关。对于具有三态输出的缓冲寄存器，可以将其数据输出线和数据输入线合并在一起，以便挂到总线上去，如图 3-39(c)所示。



(a) 电路



(b) 框图　　(c) 具有数据总线的寄存器

图 3-39　具有三态输出的缓冲寄存器

五、译码器(Decoder)

在计算机中，数码寄存器或计数器中所存放的数码，有时需要把它翻译成另一种代码或一定的控制信号，例如将代表不同地址的代码翻译成相应的片选信号；又如将操作码寄存器中代表不同操作的代码翻译成相应的控制电位等等。能够完成上述代码转换的逻辑部件称为译码器。

2^n 中取 1 译码器是一种基本的译码器。它是由具有 2^n 个输出线和 n 个输入线的复合网络所构成的。每一条输出线对应于 2^n 个可能的输入组合中唯一的一个。图 3-40 所示的是一个 8 中取 1 (即 2^3 中取 1) 译码器，或称为 3-8 译码器，其真值表如表 3-18 所示。

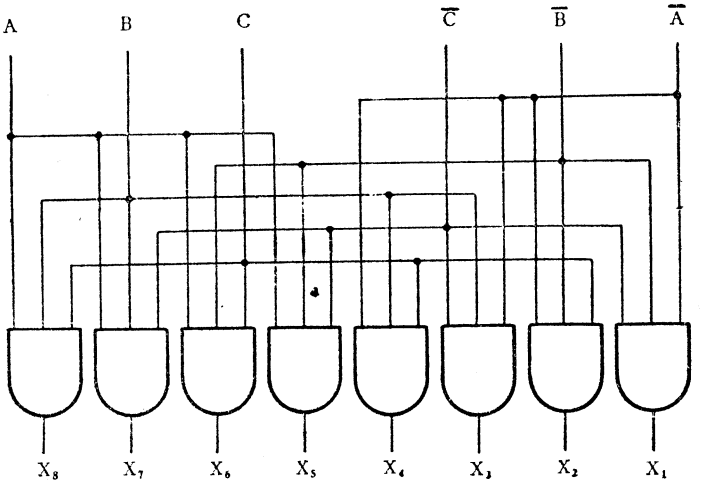


图 3-40　8 中取 1 译码器的逻辑电路

表 3-18　8中取 1 译码器的真值表

输入			输出							
A	B	C	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

从表中看出，3 个输入变量可以形成 8 种组合。对于任一确定的输入来说，8 个输出中只有一个为高电平，其余均为低电平。例如对 X_1 来说，当 3 个输入都是 0 时，X_1 取 1，而其余输出都是 0，此时输入的逻辑积等于 X_1，或 $X_1=\overline{A}\cdot\overline{B}\cdot\overline{C}$，从反演律又可得 $X_1=\overline{A+B+C}$。依此类推，可得如下 8 个逻辑式：

$$X_1=\overline{A}\cdot\overline{B}\cdot\overline{C}=\overline{A+B+C}$$

$$X_2=\overline{A}\cdot\overline{B}\cdot C=\overline{A+B+\overline{C}}$$

$$X_3=\overline{A}\cdot B\cdot\overline{C}=\overline{A+\overline{B}+C}$$

$$X_4=\overline{A}\cdot B\cdot C=\overline{A+\overline{B}+\overline{C}}$$

$$X_5=A\cdot\overline{B}\cdot\overline{C}=\overline{\overline{A}+B+C}$$

$$X_6=A\cdot\overline{B}\cdot C=\overline{\overline{A}+B+\overline{C}}$$

$$X_7=A\cdot B\cdot\overline{C}=\overline{\overline{A}+\overline{B}+C}$$

$$X_8=A\cdot B\cdot C=\overline{\overline{A}+\overline{B}+\overline{C}}$$

从上式可知，可以用 8 个与门或 8 个或非门来实现 8 中取 1 译码器，图 3-40 示出的是采用 8 个与门构成的译码电路。实际的译码器中常设置一条附加输入线——允许线，用来控制全部输出线。仅当允许线置逻辑 1 时才能启动译码器，产生有效的输出信号；反之，当允许线置逻辑 0 时，所有的输出线全被强制到逻辑 0。为了完成这一功能，只要在每个输出与门上多加一条输入线，连接到允许线上即可。

六、运算电路 (Arithmetic circuit)

在第二章中已经指出，二进制数的加、减、乘、除等四则运算，可以归结为加和移位两种操作，移位操作是在移位寄存器中实现的，加法是在加法器中实现的。

1．一位二进制数加法与半加法器 (Half adder)

假设两个一位的二进制数 A 与 B 相加，其真值表如表 3-19 所示。

表 3-19　A 和 B 之和的真值表

A	B	和 S	进位 C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

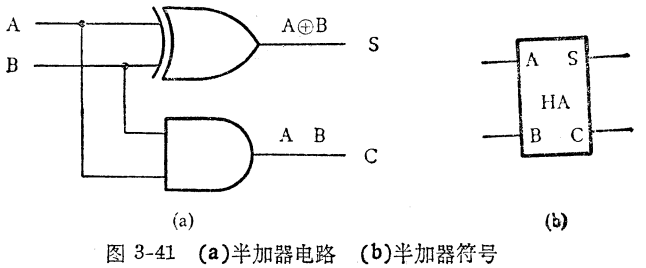


图 3-41　(a)半加器电路　(b)半加器符号

利用第二章的结论，由该表可得：

$$\text{和数}\quad S=(A\cdot\overline{B})+(\overline{A}\cdot B)=A\oplus B \tag{3-22a}$$

$$\text{进位}\quad C=A\cdot B \tag{3-22b}$$

能够实现上述运算功能的电路称为半加器，如图 3-41 所示。

由于半加器没有考虑低位进位，因此只能完成没有进位的加法。对于有进位的加法，应采用带低位进位输入端的加法器，即全加器。

2．多位二进制数加法与全加器 (Full adder)

表 3-20　A、B 与 C_i 之和的真值表

进位输入 C_i	A	B	和 S	进位 C_0
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

考虑有低位进位 C_i 的和数 S 的真值表如表 3-20 所示。

同样，从表 3-20 中可得：

和数　$S=(\overline{C_i}\cdot\overline{A}\cdot B)+(\overline{C_i}\cdot A\cdot\overline{B})+(C_i\cdot\overline{A}\cdot\overline{B})+(C_i\cdot A\cdot B)$　(3-23a)

进位　$C=(\overline{C_i}\cdot A\cdot B)+(C_i\cdot\overline{A}\cdot B)+(C_i\cdot A\cdot\overline{B})+(C_i\cdot A\cdot B)$　(3-23b)

简化后得：

和数　$S=C_i\oplus(A\oplus B)$　(3-24a)

进位　$C_0=(A\cdot B)+[C_i\cdot(A\oplus B)]$　(3-24b)

能实现上述功能的电路称为全加器，见图 3-42。该电路包括两个半加器和一个或门，A、B 分别是被加数和加数，S 是和数，C_i 是低位进位输入，C_0 是进位输出。

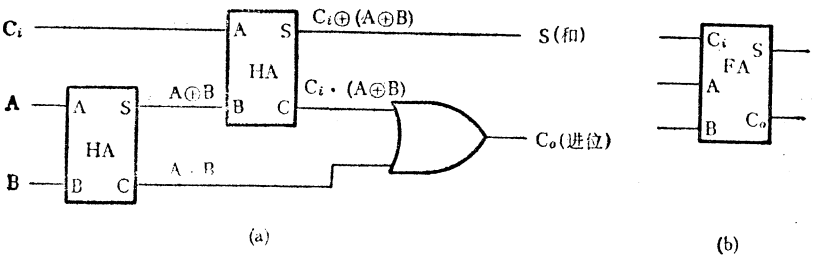


图 3-42　(a)全加器电路　(b)全加器符号

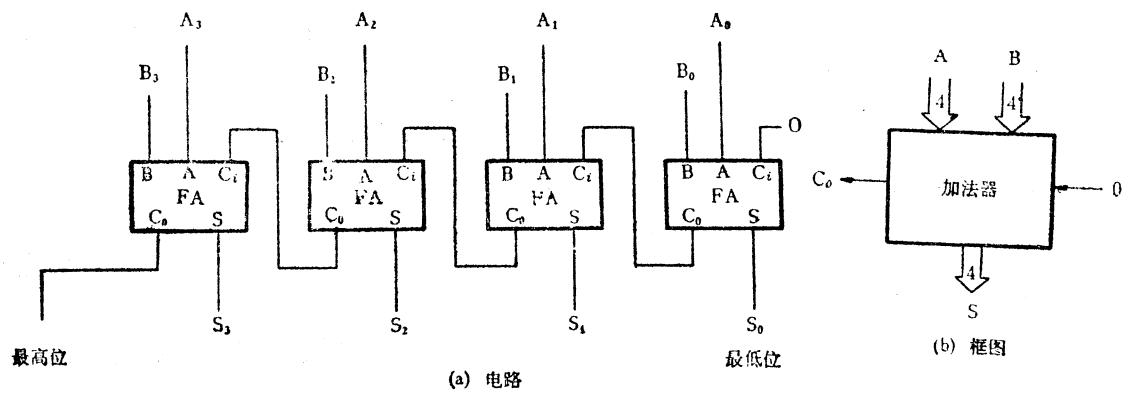


图 3-43　四位加法器

3．加法器

利用 n 个全加器级联起来就可构成一个能完成两个 n 位二进制数的加法运算的加法器，如图 3-43 所示。

4．二进制数补码加法器/减法器

由于采用补码减法运算，可以归结为对减数变补，然后做加法，因此减法器和加法器的布尔表达式基本上是相同的，可用一个逻辑网络来实现加法和减法运算。只要设置一条控制线来选择网络中的减数值 B_i，使之适合于所要求的运算即可，可以指定控制线置逻辑“1”时进行减法运算，置逻辑“0”时进行加法运算。在这里，可以用“异或”门来选择所需要的 B_i。为了说明这个原理，我们来分析如下的布尔表达式：

$$C_{in}\oplus B_i = C_{in}\overline{B_i} + \overline{C_{in}}B_i \tag{3-25}$$

设 $C_{in}=0$，则式中右侧第一项等于 0，第二项则等于 B_i；若 $C_{in}=1$，则右侧第二项为 0，第一项即为 $\overline{B_i}$，因此 $C_{in}\oplus B_i$ 能够实现所期望选择的逻辑。这样，设计时只需设置相应的异或门，就可以在加法器的基础上构成二进制数补码加法器/减法器，如图 3-44 所示。

执行加法时，加/减端和全加器的进位输入端 C_{in} 都是 0，每个异或门有一个输入端为 0，$B_0\sim B_{n-1}$ 通过这些门时不会发生变化(不取反)，因而得到 A＋B 之和，其溢出由带输入端 C_s 和 C_{s+1} 的异或门来决定。

执行减法时，加/减端和进位输入端 C_{in} 均为 1，这时每个异或门有一个输入端为 1，$B_0\sim B_{n-1}$ 通过这些门时都将被取反，又因 $C_{in}=1$ 被加到这个反码上，结果就得到 −B 的补码形式。接着它再与 A 相加。与加法时的情况一样，其溢出也由 C_s 和 C_{s+1} 通过异或门来决定。

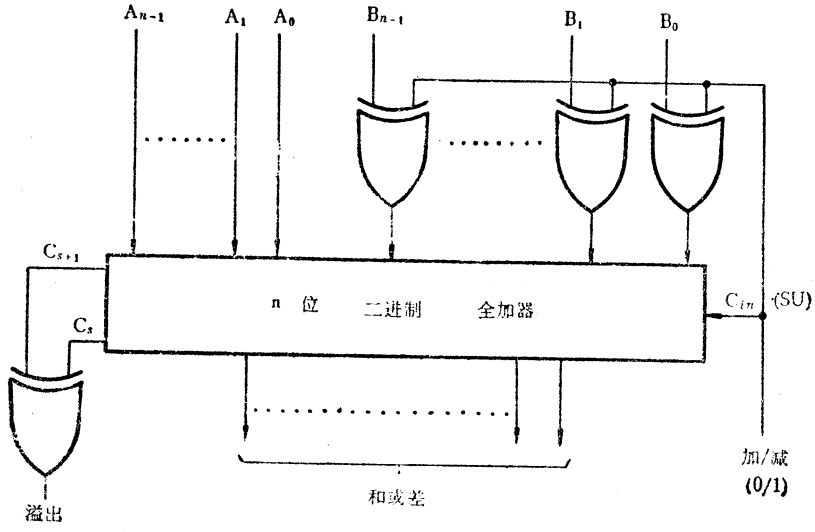


图 3-44 二的补码加法器/减法器

第四章　计算机的基本结构与工作原理

为了便于分析，本章先从一个以实际的计算机结构为基础的经过简化的模型计算机来着手分析计算机的基本结构与工作原理。然后，再扩展引伸到实际的微型计算机结构。

微型计算机的结构框图如图 1-2 所示。为了简化问题，先暂不考虑外部设备及其接口电路，而是先从一个仅由 CPU 和存贮器构成的最简单的模型计算机着手进行分析。

§4-1　模型计算机的结构

模型计算机的结构如图 4-1 所示。它是由微处理器（CPU）和存贮器通过总线连接组成的。下面分别加以说明。

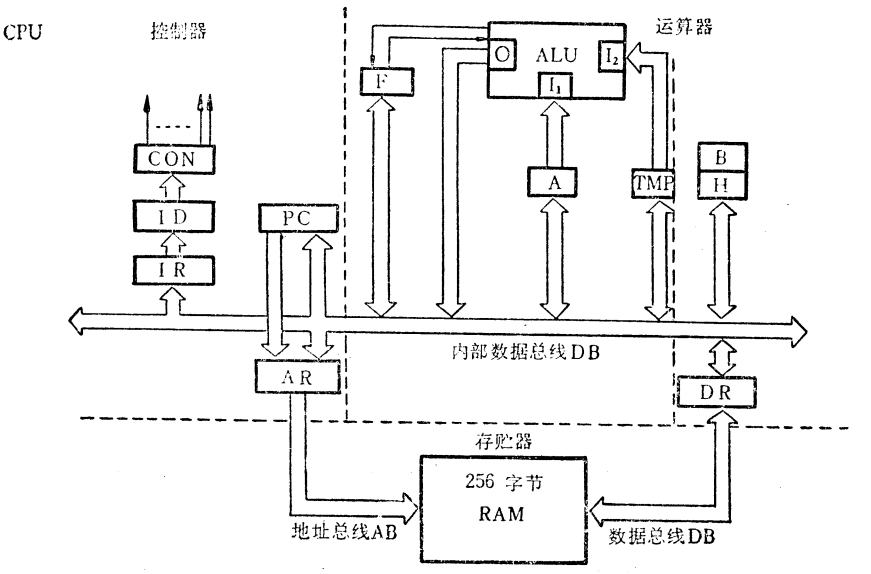


图 4-1　模型计算机的结构图

一、存贮器

1．存贮器的功用和分类

存贮器是存放程序和数据的装置。根据其功能可分为内存贮器（主存贮器）和外存贮器。内存贮器（简称内存）设在主机内部，用来存放当前运算所需要的程序和数据，以便向 CPU 高速地提供信息，相对于外存贮器而言，其容量较小，但速度较快。外存贮器（简称外存）通常设在主机外部，用来存放大量暂时不直接参与运算的程序和数据，在需要时可与内存成批地交换信息，其特点是存贮容量大，但速度较慢。

根据工作方式的不同还可分为：

随机存取存贮器 RAM 和只读存贮器 ROM。RAM可以随机地存入或取出信息；ROM 对

存入的信息只能读出，不能随机地写入。

七十年代以前大、中、小型计算机的内存贮器主要以磁芯存贮器为主。随着大规模集成电路技术的发展，半导体存贮器也得到迅速的发展。由于它具有速度高、体积小、功耗低、成本低、应用方便等优点，近年来已被广泛采用，并正在逐步取代磁芯存贮器。对微型计算机来说，通常都采用半导体存贮器。

2．存贮器的主要技术指标

存贮器主要的技术指标是存贮容量、存取周期和存取时间。

(1) 存贮容量 (Memory Capacity)

存贮容量是指存贮器可以容纳的二进制信息量。通常，表示存贮容量的方式有两种：一种是以存贮器所能记忆的字数乘以字长来表示，即存贮容量 = 存贮单元数 × 字长。例如，存贮器有4096 个存贮单元，字长为 8 位，则说该存贮器的存贮容量为 4K 字节(简称 4KB)。这里，为了书写和阅读的方便，常以符号 1K = 1024 为单位来计算存贮容量。另一种以存贮器所能记忆的全部二进制信息量直接来表示。例如，某存贮器有 4096 个存贮单元，字长为 8 位，则该存贮器的存贮容量可表示为 4K × 8 位 = 32K 位(Bit)。

(2) 存取周期(access cycle)和存取时间(access time)

存取周期是指存贮器进行一次完整的存取操作所需的全部时间，即存贮器进行连续存取操作所允许的最短时间间隔。存取时间是指向存贮器存入一个数或取出一个数所需要的时间。通常，在微型计算机中提到存贮器工作速度时，如果不加以指明，是指它的存取时间。

一般情况下，存取周期约为存取时间的两倍。两者都表示存贮器的工作速度，并都直接影响到计算机的工作速度。

3．存贮器结构

存贮器是由一系列存贮单元组成的。每个存贮单元有一个唯一的编址。它可以按字编址，也可以按字节编址。在按字编址时，每个存贮单元能存贮的二进制信息的长度即为字长；在按字节编址时，每个存贮单元能存贮的二进制信息的长度为 8 位，即一个字节，通常要用几个字节才能组成一个字。在微型计算机中，通常是按字节编址的。

存贮器内部结构通常由存贮单元阵列、地址寄存器、地址译码器和数据缓冲器等四个部分组成。如图 4-2 所示，这里为按字节编址。

从图中看出，当访问指定的存贮单元时，必须将该存贮单元的地址送入地址寄存器。若地

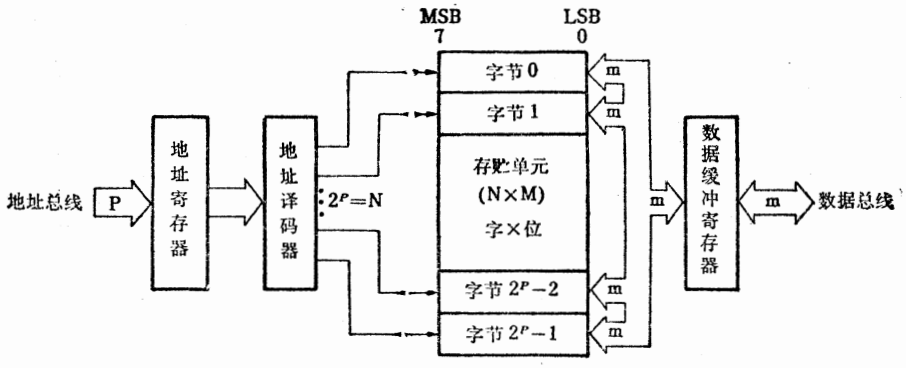


图 4-2　存贮器的结构图

址寄存器宽度为 p（即有 p 位），则通过地址译码器可从 $N=2^p$ 个存贮单元中选择指定的那个存贮单元，并将其所存放的数码（字长 m 位）读出至数据缓冲寄存器再输出。

在模型机中，为了更清楚地说明其工作原理，可将存贮器内部结构简化为如图 4-3 所示。

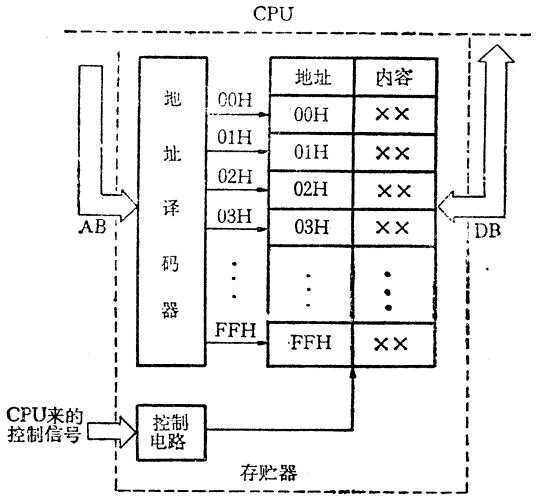


图 4-3　模型机的存贮器结构图

设它由 256 个单元组成，它们的地址是 00H、01H、02H、…FFH 等；每一个存贮单元可存放 8 位二进制信息，即存贮单元的内容。在这里，要特别注意：切不可把每一个存贮单元的地址和这一地址中存放的内容混淆起来。

4．随机存贮器的操作过程

当我们要将一个数码写入某一存贮单元或从某个存贮单元读出时，首先要指出该存贮单元的地址。这是由 CPU 通过地址总线送到存贮器的地址寄存器，经地址译码器来寻址的，即每给定一个地址号，可以从 256 个存贮单元中找到相应于该地址号的存贮单元。其次，CPU 等待从存贮器中取出所需的数码。延迟的时间取决于存贮器的存取时间。然后，由 CPU 通过控制总线向存贮器发出读或写的命令。接着，就可以对存贮单元的内容进行读或写的操作。

（1）读操作

若在地址为 03H 号的存贮单元中存放的指令内容为 10000101B（即 85H），现要求将其读入 CPU 中。其操作过程如下：首先，由 CPU 的程序计数器 PC 将其所指出的地址号 03H，经 CPU 的地址寄存器 AR，通过地址总线送到存贮器的地址寄存器（图中略）；接着，存贮器的地址译码器对地址号进行译码，找到 03H 单元；然后，CPU 发出读命令，于是 03H 单元的内容 85H 就出现在数据总线上，并被送至 CPU 的数据寄存器 DR，如图 4-4 所示。

（2）写操作

如果要把 CPU 数据寄存器中的内容 30H 写入到 16H 号单元。其操作过程如下：首先，由

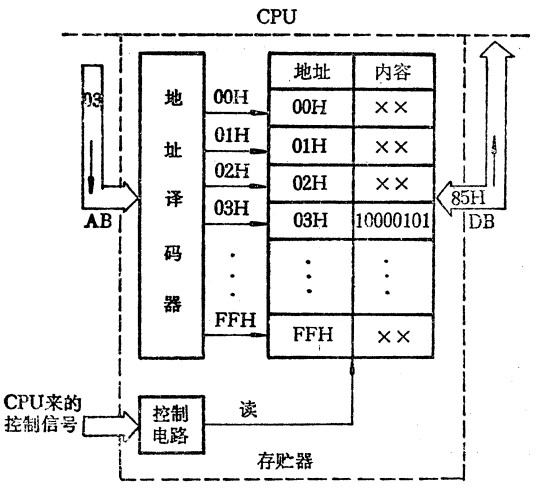


图 4-4　存贮器读操作示意图

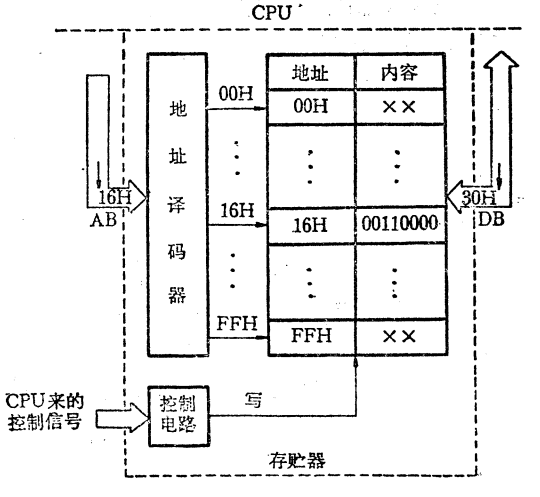


图 4-5　存贮器写操作示意图

CPU 的程序计数器 PC 将其所指出的地址号 16H 经 CPU 的地址寄存器 AR，通过地址总线送到存贮器的地址寄存器(图中略)；接着，存贮器的地址译码器对地址号进行译码，找到 16H 单元；然后，CPU 发出写命令，于是数据寄存器中的内容 30H 就经数据总线而写入 16H 号单元中，如图 4-5 所示。

二、中央处理器 (CPU)

模型计算机的 CPU 结构，如图 4-1 虚线上半部所示，它是由运算器和控制器组成的。

1．运算器

运算器是在控制器控制下，对二进制数进行算术逻辑运算及信息传送的装置。由于任何数学问题最终都可以用加法和移位这两种最基本的操作来实现。因此，运算器的主要功能就是实现加法和移位。运算器由算术逻辑单元 ALU、累加器 A、通用寄存器 B、H、暂存寄存器 TMP、标志寄存器以及其它逻辑电路所组成。

(1) 累加器A或 AC (Accumulator)

累加器是运算器的关键部件之一。它有两种功能：一是作为 ALU 的一个操作数输入端(一般存放被加数)；二是用于存放 ALU 的运算结果。它既是操作数寄存器，又是结果寄存器。从结构上看，累加器实际上是一个有并行输入/输出能力的移位寄存器，其位数等于微型计算机数据字的字长，每一位有一个触发器。有些微处理器有两个以上的累加器，这样可以使 CPU 的功能更加灵活。模型计算机中只设置一个累加器，由 8 位触发器构成。

假设初始时累加器A中的内容为 00H，寄存器B中的内容为 05H，在执行寄存器 B 和累加器A中的内容相加后，所得结果为 05H，这一个数被存入A中。这时，A 中的数由原来的 00H 变为 05H。假如把数 03H 送入寄存器 B，再执行一次加法，则累加器A中的内容变为 (05H + 03H) = 08H。由此得出，累加器中所存的数是各次累加的总和。

除算术运算外，计算机在进行逻辑运算时也常用到累加器。

(2) 通用寄存器(General-Purpose Register)B、H

通用寄存器用来暂时存贮参加运算的操作数、中间结果或地址。它是为高速处理数据而设置的，其数量根据不同种类的 CPU 而异。

(3) 暂存寄存器 TMP (Temporary Register)

暂存寄存器 TMP 用来暂存从数据总线或通用寄存器送来的操作数，并将该数据送入ALU进行运算。同时，它也能将数据送到内部数据总线。

(4) 标志寄存器 F(Flag Register 或 Status Register)

标志寄存器F用来保存 ALU 操作结果的特殊状态，这种状态将作为判断是否控制程序转移的条件。它的详细功用，将在第六章指令系统部分介绍。

(5) 算术逻辑单元 ALU

ALU 是由加法器和其它逻辑电路如移位电路、控制门等组成的。在指令译码后产生的控制信号的控制下，它能完成各种算术和逻辑运算。它以累加器A的内容作为一个操作数，另一个操作数由暂存寄存器 TMP 供给。有时，还包括标志寄存器F送来的进位等。运算结果送累加器A或数据总线。同时，将运算结果的状态送标志寄存器保存。

下面举例说明 ALU 进行加法运算的情况，如图 4-6 所示。

设累加器A中的数为 21H，寄存器B中的数为 43H，当执行加法指令 ADD A,B 时，寄存

器B和累加器A的内容相加。这时，寄存器B中的数43H先送到暂存寄存器TMP中，如图4-6(a)所示。然后，TMP中的数43H和累加器A中的数21H送到ALU中相加，相加得的结果64H经过内部数据总线重新送入累加器A中，如图4-6(b)所示。由两数相加结果所引起的标志寄存器F中各标志位状态的变化情况，将在后面介绍。

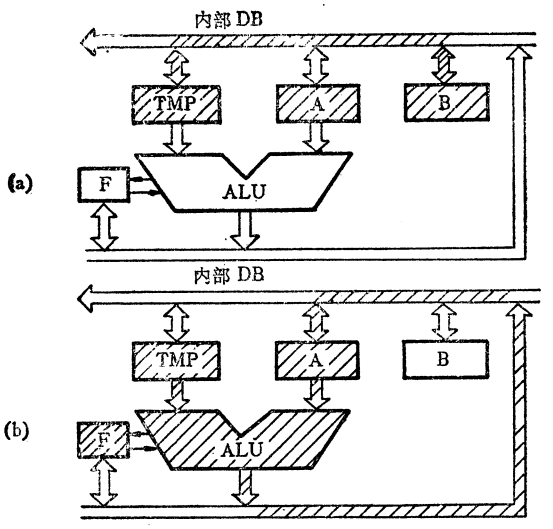


图 4-6　指令 ADD　A,B 的执行情况

ALU究竟是执行算术运算还是执行逻辑运算？这取决于指令的操作码经过译码后所产生的控制信号。这种控制信号通过控制运算器的功能输入端，以决定执行算术运算或逻辑运算；并且，通过控制运算器的功能选择输入端来决定具体的运算功能，如“与”运算或是“或”运算等。

2．控制器

控制器是发布操作命令的机构，是计算机的指挥中心，犹如人脑的神经中枢一样。计算机程序和原始数据的输入、CPU内部的信息处理、处理结果的输出、外部设备与主机之间的信息交换等，都是在控制器的控制下实现的。如前所述，计算机是遵循程序存贮和程序控制的原理工作的。程序本身由一系列指令所组成，每条指令又由操作码和地址码(或操作数)所组成。当计算机进入自动计算时，控制器的任务是：逐条地取出指令、分析指令、执行指令，并为取下一条指令作好准备。

为了完成上述功能，控制器至少必须具有指令部件、时序部件和微操作控制电路。控制器的组成框图如图4-7所示，下面分别予以说明。

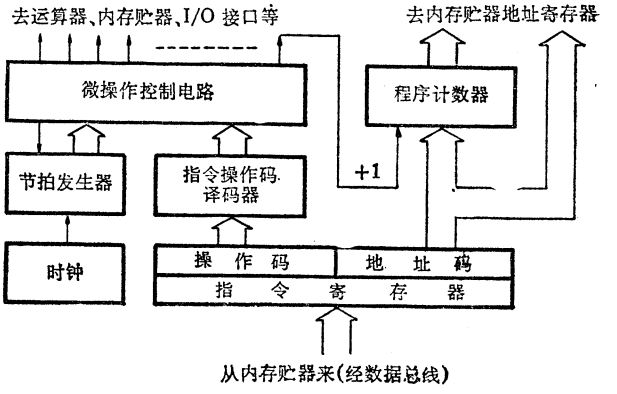


图 4-7　控制器的组成框图

(1) 指令部件

指令部件至少包括程序计数器、指令寄存器和指令译码器。

①程序计数器PC(Program Counter)

程序是指令的有序集合。计算机运行时，通常按顺序执行存放在存贮器中的程序。先由PC指出要执行指令的地址，每当该指令取出后，PC的内容就自动加1(除转移指令外)，指向按顺序排列的下一条指令的地址。在正常情况下，CPU按顺序逐条地执行指令。如果遇到转移指令(JMP)、调用子程序指令(CALL)或返回指令(RET)等时，这些指令就会把下一条指令的地址直接置入PC中。

例如在图4-8中，开始时PC中的内容为2000H，它表示第一条指令在存贮器中的地址为2000H。在执行第一条指令(设为单字节指令)的过程中，PC的内容自动加1，变为2001H。接着执行地址为2001H的指令。当执行到转移指令JMP X时，这一条转移指令表示下一条指令

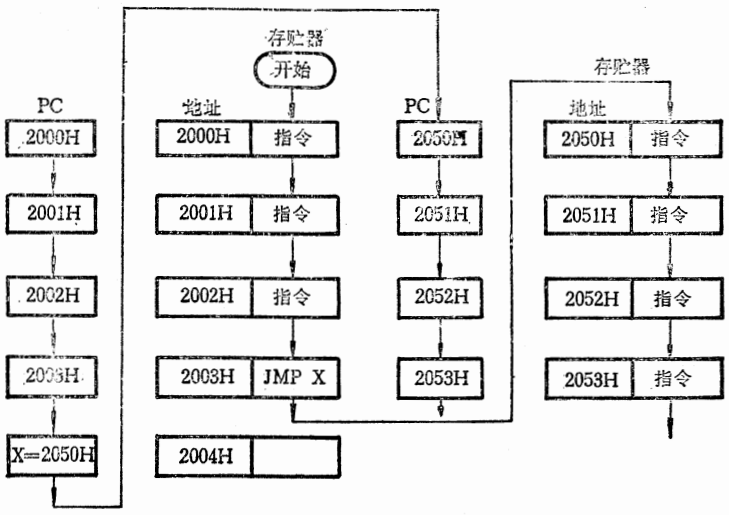


图 4-8　程序的执行和PC内容的变化

的地址不是 2004H，而是转移到地址 X。假设 X = 2050H，于是 PC 的内容由原来的 2003H 变为 2050H，随即执行地址为 2050H 的指令，接着再执行下一条 2051H 的指令。

程序计数器 PC 的位数取决于微处理器所能寻址的存贮空间。现模型计算机中，内存贮器的存贮容量仅有 256 字节、故其 PC 只需要 8 位。

② 指令寄存器 IR (Instruction Register)

它用来存放当前要执行的指令内容，它包括操作码和地址码两部分。操作码送往指令译码器；地址码送至操作数地址形成电路。在 8 位微型计算机中，IR 的宽度通常与微处理器的基本指令字长相同，即亦为 8 位，故仅用于存放所要执行的指令操作码，而指令地址码则通过 CPU 来形成操作数的真正地址。

③ 指令译码器 ID(Instruction Decoder)

指令译码器是分析指令的部件。操作码经过译码后产生相应操作的控制电位。例如，8 位操作码经指令译码器译码后可以转换为 $2^8 = 256$ 种操作控制电位，其中每一种控制电位对应一种特定的操作。

(2) 时序部件

前已述及，计算机的工作是周期性的，取指令，分析指令，执行指令，再取指令等等。这一系列操作的顺序都需要精确的定时。

时序部件是用来产生计算机各部件所需的定时信号的部件。它由时钟系统(包括脉冲源、启停逻辑)、时钟脉冲分配器等部件组成。

① 时钟系统

脉冲源：它用来产生具有一定频率和宽度的脉冲信号(称为主脉冲)。在微型计算机中一般都使用石英晶体振荡器，因为它的频率稳定。

计算机的电源一旦通电，脉冲源立即以给定的频率重复发出矩形脉冲。两个相邻脉冲前沿的时间间隔称为一个时钟周期或 T 时态，它是 CPU 操作的最小时间单位。

通常，不同的计算机，其主频也不一样。小型机和微型机的主频一般为几百 kHz 到几 MHz，甚至更高。

时钟启停逻辑：它用作控制启停主脉冲信号的开关，按指令和控制台的操作要求，可准确地开启或关闭时钟脉冲序列。

② 脉冲分配器 (Pulse distributor)

计算机在执行一条指令时，总是把一条指令分成若干基本动作，由控制器发生一系列节拍和脉冲，每个节拍和脉冲信号指挥机器完成一个微操作。产生这些节拍和脉冲的部件称为脉冲分配器又叫做节拍脉冲发生器。它们用作产生计算机各部分所需要的能按一定顺序逐个出现的节拍电位或节拍脉冲的定时信号，以控制和协调计算机各部分有节奏地动作。

节拍脉冲发生器的基本组成就是 §3-3 中所介绍的环形计数器。

前已提及，时钟周期 T (又称节拍)是 CPU 操作的最小时间单位。一般总是由若干节拍构成一个循环，称为机器周期 (Machine Cycle)。

在微处理器如 intel 8080 A 中，通常由 3～5 个时钟周期组成一个机器周期或称 M 周期。它用来完成一个基本操作，如存贮器读、存贮器写、I/O 读和 I/O 写等。在每个机器周期的第一个时钟周期 T_1 时，CPU 输出一个同步信号，表明一个机器周期开始工作。完成一条指令通常需要 1～5 个机器周期，根据指令的种类而定。一条指令的取出和执行所需的时间称为指令周期(Instruction Cycle)。图 4-9 为指令周期、机器周期、与时钟周期之间的关系示意图。

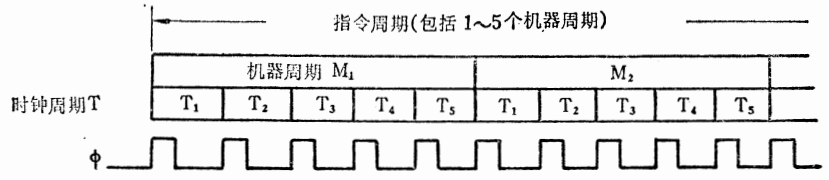


图 4-9　指令周期、机器周期和时钟周期的关系

(3) 微操作控制部件(CON)

微操作控制部件的主要功能，是根据指令产生计算机各部件所需要的控制信号。这些控制信号是由指令译码器的输出电位、节拍脉冲发生器产生的节拍电位、节拍脉冲，以及外部的状态信号等进行组合而产生的。它按一定的时间顺序发出一系列微操作控制信号，以完成指令所规定的全部操作。

微操作控制部件，可采用组合逻辑控制、微程序控制及可编程序逻辑阵列控制的方式来实现。

① 组合逻辑控制

计算机的工作过程是执行指令序列的过程。每条指令又可分解为若干微操作，因而又是执行微操作序列的过程。

计算机中信息的传送，是通过打开和关闭某些控制门实现的。这些“门”的“打开”或“关闭”，是受微操作控制信号控制的。微操作控制信号采用组合逻辑设计方法实现时，称为组合逻辑控制，详细介绍见 §4-3 节。

② 微程序控制(存贮逻辑控制)

微程序控制方式，是利用程序存贮控制的原理，采用微程序完成机器指令系统中每一条指令功能的控制方法。换言之，先把每条指令都看成由若干条微指令组成的微程序，将这些微程序存放在只读存贮器 ROM (又称控制存贮器)中。计算机工作时，逐条地取出微指令，执行一段微程序，便实现了这条机器指令的控制功能，这就是微程序控制的基本含义。

采用微程序控制技术便于计算机的设计和调试，但它的速度不如组合逻辑控制方式快。

③ 可编程序逻辑阵列 PLA(Programmable Logic Array)

PLA 是一种通过程序设计来执行特定逻辑功能的组合逻辑结构，它兼有上面两种控制的优点。

§4-2 程序的编制和执行过程

归根结底，计算机的全部操作都是在程序控制下进行的。本节将简要介绍微型机的指令系统、程序编制和执行过程，以说明计算机的工作原理。

一、指令系统简介

指令 (Instruction) 是一种规定计算机的操作类型、操作数或其地址的命令。计算机所能识别和执行的全部指令的集合，称为该计算机的指令系统。为解决某一问题而编制的一系列指令构成程序 (Program)。指令是程序中的一步，它告诉计算机在程序中某一步应如何操作。

指令包括操作码(Opcode 即 Operation Code)和地址码 (Address Code)。操作码指定计算机执行什么类型的操作；地址码指明操作数所在的地址和运算结果送往何处。每条指令的基本格式为：

操 作 码	地 址 码

计算机指令系统的全部指令，都是用二进制编码的形式来表示的，这就是指令的机器代码 (Machine Code)。机器代码是计算机所能唯一识别的语言，但是如果它来编制程序，就太繁琐了，而且容易出错。因而，人们就采用助记符(Mnemonic Symbol)来表示操作码。通常，助记符用相应的指令功能的英文缩写词来表示。如在 Z80 微处理器中，数的传送用助记符 LD (Load 的缩写)，加法用 ADD，输出用 OUT 等，这就是用符号代码表示的汇编语言。与机器语言(一连串的 0 和 1) 相比，汇编语言比较直观，且易于记忆。当前，在微型机中主要还是应用汇编语言来编写源程序。

在第一章中已指出，计算机中的“字”，就是将一组数码作为一个整体来处理或运算的单位。它的长度用二进制码的位数来表示，称为“字长”。在微型机中，常用的字长为 8 位。有时，用一个字节还不能表示各种操作码和地址码，因而有的指令包含多个字节，如二字节、三字节、四字节等。

二、程序的编制和执行过程

我们以一个简单的例子来说明程序的编制和执行过程。

若要求计算机执行两个数 32 和 16 相加。

首先，要根据解题的要求编制程序。编制程序时，必须用该计算机的合适的指令来完成所要求的操作。例如用表 4-1 所示的三条指令来完成两个数的相加。

用助记符形式表示的程序为：

```
LD    A,  20 H
ADD   A,  10 H
HALT
```

表 4-1 程 序 表

名　　称	助 记 符	机　器　代　码		说　　明
立即数送入累加器	LD　A, n	00111110 B 00100000 B	3EH 20H	这是一条两字节指令，把指令第二字节的立即数 n 送累加器 A。
加 立 即 数	ADD A, n	11000110 B 00010000 B	C6H 10H	这是一条两字节指令，将累加器A中的内容与指令第二字节的立即数相加，其结果保留在A中。
暂　　停	HALT	01110110 B	76H	暂停操作

但是，模型计算机不能直接识别助记符形式的指令，因此必须用机器代码表示。同样，数据也只能用二进制代码表示。

第一条指令　00111110 B 操作码(LD　　A)　　00100000 B 操作数(20H)

第二条指令　11000110 B 操作码(ADD　　A)　　00010000 B 操作数(10H)

第三条指令　01110110 B 操作码(HALT)

总共有三条指令、5 个字节。

如前所述，程序应放在存贮器中，若它们放在以 00H 开始的存贮单元中(如图 4-10 所示)，则总共需要 5 个存贮单元。

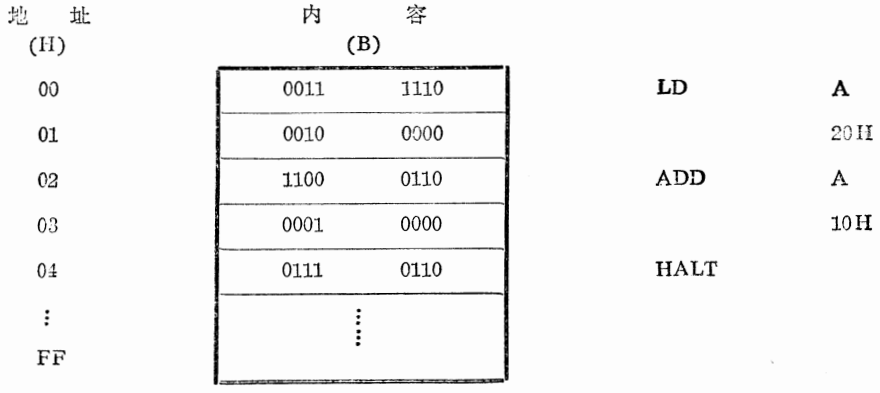


图 4-10　存放在存贮器中的指令

在执行程序时，要先给程序计数器 PC 赋以第一条指令的地址 00H，然后进入第一条指令的取指阶段，具体操作过程如下：

(1) PC 的内容 00H 送至地址寄存器 AR；

(2) 当 PC 的内容送入 AR 后，PC 的内容自动加 1 后变为 01H；

(3) AR 将地址号 00H 通过地址总线送至存贮器，经地址译码器译码，选中 00H 号单元；

(4) CPU 发出读命令；

(5) 所选中的 00H 号单元的内容 3EH 读至数据总线上；

(6) 读出的内容经过数据总线送至数据寄存器 DR；

(7) 因是取指阶段，取出的必为指令，故 DR 将它送至指令寄存器 IR，然后经过指令译码器译码后，发出执行这条指令所需要的各种控制命令，其过程如图 4-11 所示。

接着，转入执行第一条指令的阶段。CPU 经过对操作码的译码后，判定这是一条将操作数送累加器A的指令，而操作数放在指令的第二字节。因而，执行第一条指令，必须取出指令

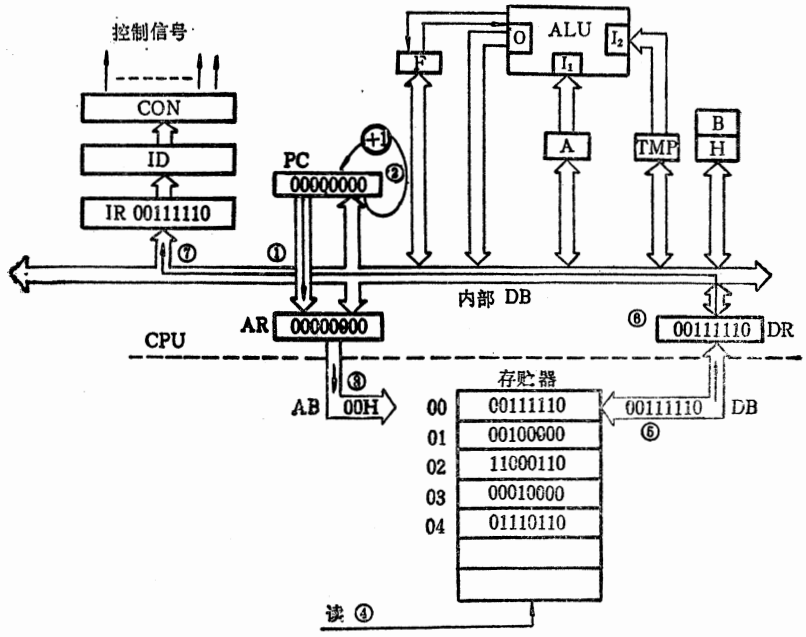


图 4-11　取第一条指令的操作示意图

第二字节中的操作数。

其操作过程如下：

(1) 把 PC 的内容 01H 送至地址寄存器 AR；

(2) 当 PC 的内容送入 AR 后，PC 自动加 1，变为 02H；

(3) AR 把地址号 01H 通过地址总线送至存贮器，经过地址译码器译码选中 01H 单元；

(4) CPU 发出读命令；

(5) 所选中的 01H 单元的内容 20H 读至数据总线上；

(6) 通过数据总线，把读出的内容送至 DR；

(7) 因已知读出的是操作数，且指令要求把它送累加器 A，故由 DR 通过内部数据总线送至 A，如图 4-12 所示。

至此，第一条指令执行完毕，进入第二条指令的取指阶段。

取第二条指令的过程为：

(1) 把 PC 的内容 02H 送至 AR；

(2) 当 PC 的内容送入 AR 后，PC 自动加 1，变为 03H；

(3) AR 通过地址总线把地址号 02H 送至存贮器，经译码后，选中相应的存贮单元；

(4) CPU 发出读命令；

(5) 选中的存贮单元的内容 C6H，读至数据总线上；

(6) 读出的内容通过数据总线送至 DR；

(7) 因是取指阶段，故读出的为指令，DR 把它送至 IR，经过译码发出执行该条指令的各种控制命令。其过程如图 4-13 所示。

CPU 经过对指令译码后，判定这是一条加法指令，以 A 中的内容为一操作数，另一操作数在指令的第二字节中，因此执行第二条指令，必须取出指令的第二字节。

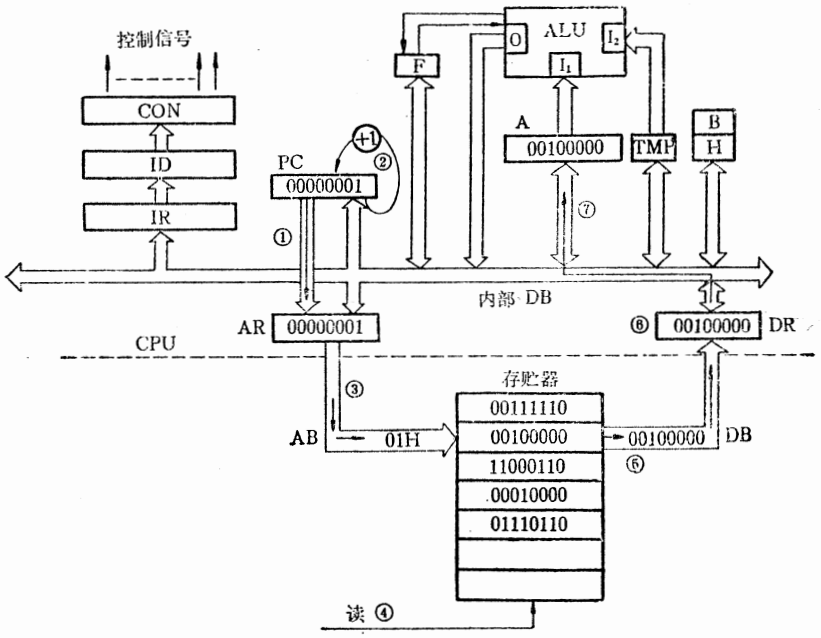


图 4-12　取立即数的操作示意图

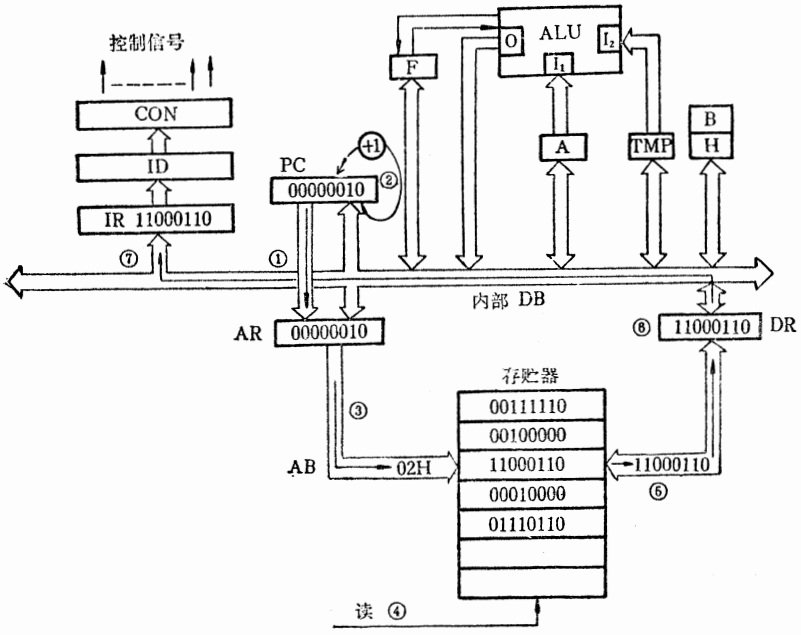


图 4-13　取第二条指令的操作示意图

取第二字节及执行指令的过程为：

(1) 把 PC 的内容 03H 送至 AR；

(2) 当 PC 内容送至 AR 后，PC 自动加 1，变为 04H；

(3) AR 通过地址总线把地址号 03H 送至存贮器，经过译码，选中相应的单元；

(4) CPU 发出读命令；

(5) 选中的存贮单元的内容 10H 读至数据总线上；

(6) 数据通过数据总线送至 DR;

(7) 因由指令译码已知读出的为操作数,且要与A中的内容相加,故数据由 DR 通过内部数据总线送至暂时寄存器 TMP 中;

(8),(9) 此时,由于 ALU 的两个输入端均有操作数,故可执行加法操作;

(10) 相加的结果由 ALU 输出至累加器A中,如图 4-14 所示。

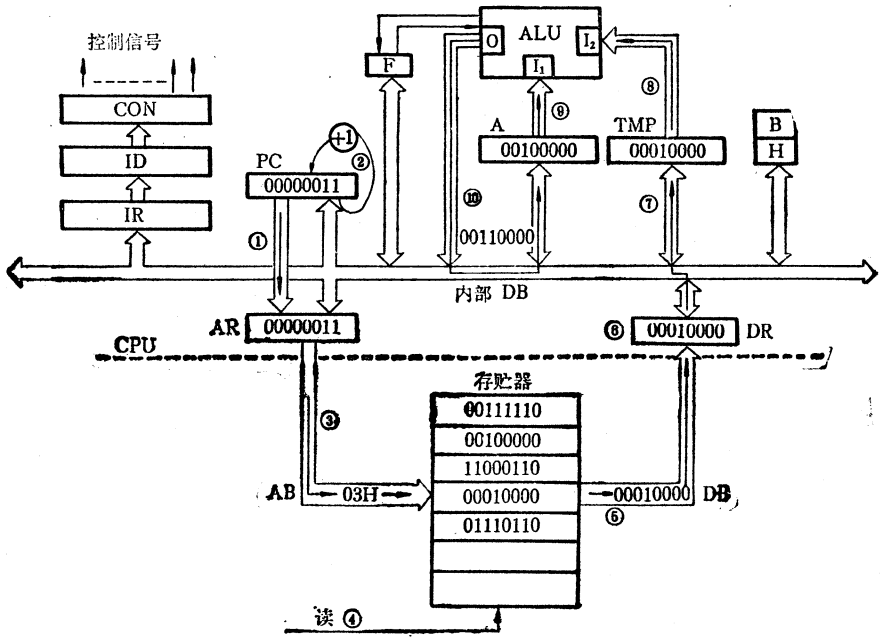


图 4-14 执行第二条指令的操作示意图

至此,第二条指令的执行阶段结束,开始转入第三条指令的取指阶段。

按上述类似的过程取出第三条指令,经译码后暂停操作。

以上通过一个简化的模型计算机执行程序的例子,可以看出:计算机执行程序的过程,是周而复始地取指令、分析指令和执行指令的过程,直至完成该程序的全部指令,实现预期要求的任务为止。

通过上例的演示,我们可以把计算机执行程序的过程归纳如下,并用图 4-15 示意。

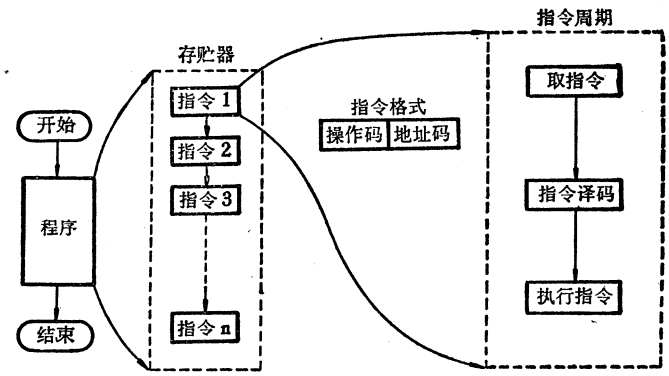


图 4-15 在计算机内执行一个程序的示意图

(1) 编制解题程序

通常在编制程序时,需要先绘制程序流程图。因上例十分简单,故略去这一步骤。其次,要分配存贮单元。通常,程序是顺序执行的,程序中的指令也按顺序存放在某段连续的存贮单元(即程序存贮区)中。操作数则存放在另一些存贮单元(即数据存贮区)中。然后编制程序,如表 4-1 所示。

(2) 程序存贮

程序存贮就是把编制好的解题程序送到计算机的内存贮器中保存起来。在计算机执行程序之前，必须先将程序和原始数据通过开关、键盘、磁带机等输入设备经 I/O 接口，沿着数据总线送入 CPU，然后存入到内存贮器中；或者不经过 CPU 直接存入存贮器。

(3) 执行程序

程序输入到存贮器后，只要通知 CPU 程序的起始地址，然后给它送一个启动命令，CPU 就能按顺序逐条地从存贮器中取出指令加以执行。这里，必须有一个电路能正确地指示和跟踪指令所在的地址，这就是程序计数器 PC。在开始执行时，要给 PC 赋以程序中第一条指令所在的地址。然后，每取出一条指令，PC 的内容自动加 1，以指向下一条指令的地址，这就保证了指令的顺序执行。只有当程序中遇到转移指令、调用子程序或遇到中断时，PC 才转到所需要的程序段地址上去。

为了加深印象，我们将以上计算机执行一条指令的过程归纳如下：

(1) PC 指出当前要执行的指令的地址，并把该指令地址输出到地址总线上；

(2) 按照该指令地址，从存贮器中取出指令，通过数据总线传送到指令寄存器 IR。这是取指阶段；

(3) IR 把指令操作码送到指令译码器进行译码，产生相应操作的控制电位。这是分析指令的阶段；

(4) 在微操作控制电路中，对经指令译码器译码后产生的控制电位和由时钟脉冲发生器产生的定时信号进行组合，从而产生一组执行该指令所需要的控制信号，去控制计算机执行相应的操作；

(5) 指令地址码依据不同的寻址方式，在 CPU 中形成操作数的真正地址，然后从存贮器或寄存器中取出指令所需的操作数；

(6) 算术逻辑部件 ALU 执行指令码所规定的操作(算术运算、逻辑运算或数据传送等)，并提供运算结果的标志信号，作为判断程序是否转移的条件，而其中的进位标志还可以作为 ALU 的输入。至此执行一条指令的工作过程便告结束。

(7) PC 自动加 1，形成下一条要执行的指令地址。当发生转移时，由转移指令决定转移后的程序起始地址，这是为取下面一条指令作好准备。

(8) 在执行下一条指令之前，还要检查有无外来请求信号(如中断请求等)，并作出响应。关于中断的概念，以后再介绍。

综上所述，计算机的工作过程，实质上就是在程序控制下，自动地、逐条地从存贮器中取指令、分析指令、执行指令，再取下一条指令，周而复始地执行指令序列的过程(这种循环过程称为指令周期)，直至该程序所有指令执行完毕为止。这就是冯·诺依曼 (1945 年) 提出的程序存贮和程序控制的计算机的基本工作原理。

了解计算机的典型的工作过程，以及各部件在其中所起的作用，将有助于从整机的观点来理解计算机内各功能部件的工作原理。

§4-3 微操作控制电路的剖析

为了更直观地了解微操作控制电路，本节将对其作进一步的剖析。这里，仅介绍用组合

逻辑控制的方法实现的微操作控制电路。

一、微操作控制电路的设计步骤

用组合逻辑设计方法得到的微操作控制电路，通常是由大量逻辑门电路组成的。根据每条指令的要求，让节拍电位和节拍脉冲有步骤地去控制机器的各有关部分，一个节拍一个节拍地依次执行指令所规定的微操作，从而在一个指令周期里完成一条指令所规定的全部操作。为此，设计微操作控制电路应遵循如下步骤：

第一，分解每条指令，归纳成若干微操作，根据操作的先后顺序画出指令操作流程图。

第二，把指令流程图中的微操作合理地安排到指令周期的相应节拍里，从而形成一个指令操作时间表。

第三，对全部指令的操作时间表进行综合分析，把凡是要执行某一微操作的所有条件（在什么指令，在哪一节拍）都考虑在内，加以归类，列出产生微操作条件的逻辑表达式，然后借助于布尔代数加以简化，得到最合理的逻辑表达式。

第四，根据逻辑表达式画出微操作控制线路，然后，用逻辑门电路的组合逻辑来实现。

二、模型计算机-II 的结构

为了直观地展示微操作控制电路的内部结构，先分析图 4-16 所示的模型计算机 -II。它是由挂在总线上的 ALU、寄存器、存贮器和控制电路构成的。这条总线既传送数据又传送地址。所有挂到总线上的寄存器的输出都必须是三态的。

下面简要介绍各部件的作用。

1．随机存取存贮器 RAM

RAM 的基本组成和工作原理见图 4-3。这里假设 RAM 的容量为 256×8 位，其内容如

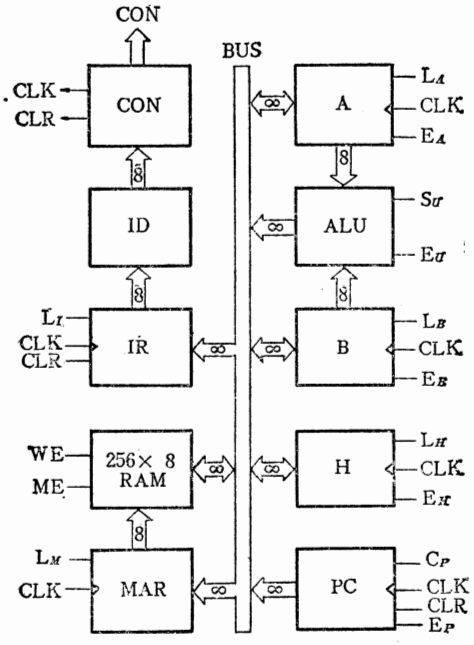


图 4-16　模型计算机-II 的结构

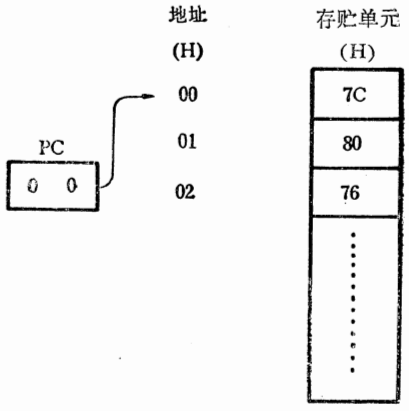


图 4-17　RAM 单元的内容

图 4-17 所示。ME 是 RAM 的三态输出控制端。WE 是 RAM 的写控制端。

2．存贮器地址寄存器 MAR(8 位)

它用来存放被访问的存贮单元的地址。其内容可以来自 PC，也可以来自 CPU 中的寄存器。它的二态输出送往 RAM。

3．累加器 A

A 是一个 8 位寄存器，用于存放操作数或操作结果。它有两个输出：一是送往 ALU 的二态输出，它不受 E_A 信号的控制；另一是送往总线的三态输出。

4．B、H 寄存器

B、H都是 8 位寄存器，用于暂时存放参加运算的操作数。它们挂在总线上的一端是三态输出，分别受 E_B、E_H 控制；其中 B 还有一个送往 ALU 的二态输出端，以供 ALU 进行运算。

5．ALU

ALU 是一个以加法器/减法器为核心的算术逻辑部件（见 § 3-3），S_U 电平的低或高决定它进行加或减。

当 $S_U = 0$ 时，ALU = A + B

当 $S_U = 1$ 时，ALU = A − B

ALU 的两个输入端 A 和 B 均为二态输入，所以 ALU 可随时对 A 和 B 送来的操作数进行运算。当 $E_U = 1$ 时，它的内容才送往总线。

6．程序计数器 PC

本例中，PC 是一个 8 位计数器。计算机复位后，PC 置成 00H。每当 CPU 从存贮器中取出一条指令的一个字节之后，PC 的内容自动增 1，为取下一个字节做好准备。

7．指令寄存器 IR(8 位)

在微型机中，通常指令寄存器仅用来存放现行指令的操作码，并将其送往指令译码器。

8．指令译码器 ID(8 位)

ID 接受 IR 送来的 8 位操作码并进行译码，产生相应操作的控制电位，送往微操作控制电路。

9．微操作控制电路 CON

CON 电路对 ID 送来的控制电位和时钟脉冲发生器送来的定时信号进行组合，从而产生一组控制信号，指挥计算机有条不紊地执行指令所要求的操作。

上述九个部件，可进一步归纳为下列三大部分：

1．运算器——包括 ALU 及 A、B、H 等寄存器。

2．控制器——包括 PC、IR、ID、CON。

3．存贮器——包括 RAM、MAR。

上述三部分是计算机的主要组成部分，统称为主机。

三、指令的执行过程

为了便于说明，现以单字节指令来实现两个数 2 和 3 相加为例，并假定两个数已存放在寄存器 B，H 中，B = 2、H = 3。

根据 § 4-2 的编程方法，我们可以用三条指令编制成一个简单的加法程序，如表 4-2 所示。

表 4-2 程序表

地址 (H)	机器代码 (H)	助记符	操作结果
00	7C	LD A,H	这是一条寄存器传送指令。把寄存器H的内容送至累加器A中，即 A←H=3
01	80	ADD A,B	这是一条加法指令。累加器A中的内容与寄存器B中的内容相加，结果放在A中，即 A←A+B=2+3=5
02	76	HALT	暂停

由于这三条指令都是单字节指令，因此其指令周期仅需要一个机器周期。通常一个机器周期由 3～5 (或 3～6)个时钟周期组成，其中，前三个时钟周期为取指阶段，后两个(或三个)时钟周期为执行阶段。在本例中，相加过程只与 CPU 内的寄存器有关，不必访问存贮器。

下面着重分析指令的执行过程。

任何一条指令的第一个机器周期必然是取指周期，即从存贮器中取出指令，这是在头三个(T_1、T_2、T_3)时钟周期内完成的，取指过程如下：

T_1：PC OUT。开始取指时，先把 PC 的内容经地址总线 AB送到 AR 中。

T_2：PC←PC＋1 和读指令。AB 上的地址送至存贮器，经地址译码器译码选中相应的存贮单元，大约经过几百个毫微秒，被选中的存贮单元的内容便送至与数据总线 DB 相连的存贮器输出接头上；CPU 利用这段时间去完成 PC 增 1 的操作，为执行下一条指令作准备。

T_3：IR←Inst。此间，指令码已从存贮器读出并经数据总线 DB，送至指令寄存器 IR。

接着需要对指令码进行译码并执行指令，这些操作是在 T_4、T_5 两个时钟周期内完成的。即：T_4：ID←IR。在 T_4 期间，ID 对指令码进行译码分析，并把译码所产生的控制电位送往微操作控制电路与定时信号进行组合，从而产生一组执行该指令所需要的控制信号；与此同时 T_4、T_5 期间即可完成其所要求的操作。各条指令的操作时间表如表 4-3 所示。

表 4-3 指令操作时间表

指令	指令周期				
	机器周期 1				
	取指阶段			执行阶段	
	T_1	T_2	T_3	T_4	T_5
LD A,H	MAR←PC L_M, E_P	PC←PC+1 C_P	IR←Inst L_I, ME	A←H L_A, E_H	
ADD A,B	MAR←PC L_M, E_P	PC←PC+1 C_P	IR←Inst L_I, ME	A←A+B L_A, E_U	
HALT	MAR←PC L_M, E_P	PC←PC+1 C_P	IR←Inst L_I, ME	HALT	

四、微操作控制电路的构成

图 4-18 (a)中的虚线框内和图 4-18(b) 为微操作控制电路内部结构的示意图，图中时钟

脉冲发生器用来产生固定频率的节拍脉冲，协调各部件按统一的节拍操作。它可由环形计数器 RC 构成，每当来一个主脉冲时，输流使 T_1、T_2、T_3、T_4、T_5 为高电平。

图中，指令译码器 ID 用来对指令操作码进行译码，如当指令操作码为 10000000 B 时，输出信号ADD 应为高电平，其余输出信号均为低电平，依此类推。

微操作控制电路用来在不同指令的不同节拍内产生不同的控制信号，如表 4-3 所示。以 L_A 为例，在下列两种情况下都需要出现 L_A 信号。

(1) LD A 指令的 T_4 节拍：此时 LD A = 1，T_4 = 1，由此 L_A^1 = 1，L_A = 1。

(2) ADD 指令的 T_4 节拍：同理可得 L_A^2 = 1，L_A = 1。

因此，$L_A = L_A^1 + L_A^2$，如图 4-18(b)所示。

在这里，为了便于说明问题，已经把微操作控制电路化简为最简单的结构。实际上，微操作控制电路应对全部指令的操作时间表进行综合分析。不仅要对每一条指令，在不同节拍内的微操作进行“组合”，而且还要对不同指令中的同一微操作进行“组合”，即把凡是要执行某一微操作的所有条件都加以归类，列出产生微操作条件的逻辑表达式，然后用逻辑门的组合逻辑来实现。

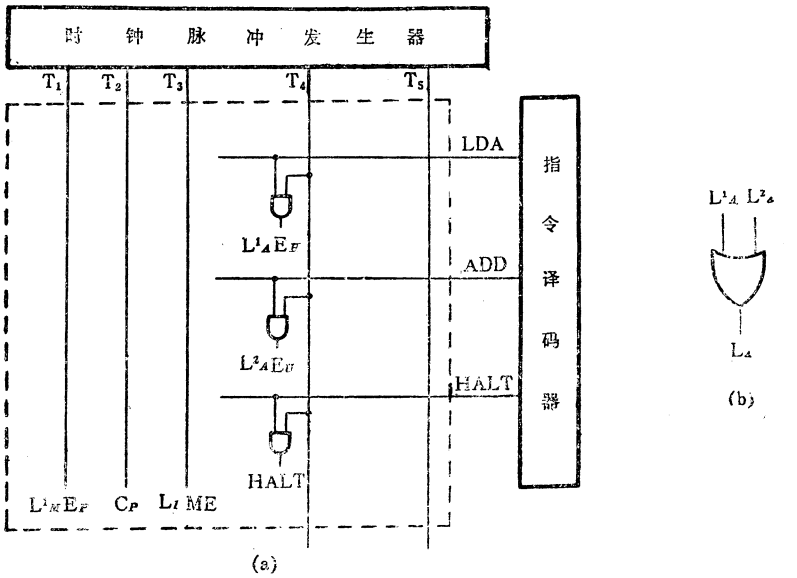


图 4-18 微操作控制电路

§4-4 模型计算机的扩充

以上所述的模型计算机简单、直观，便于了解计算机内部操作的全貌和特点，但是它毕竟有相当的局限性，为了完整地了解电子计算机系统的结构和工作过程，还需要将模型计算机加以扩充，即添上输入输出设备及其接口电路。此外，还需要引入一些重要的概念如堆栈、数据通道、总线和中断等。

现在，我们将图 1-2 和图 4-1 略加展开，画出扩充后的模型计算机的基本结构，如图 4-19 所示。从图中可以看到计算机各组成部件之间的相互关系，以及数据信息和控制信息在计算机内部的流向，这样使我们得以更加全面地了解计算机的工作概貌。在图 4-19 中，CPU 和存贮器的结构及其工作原理已在 § 4-2 介绍过，本节着重说明几个重要概念和输入输出设备及其

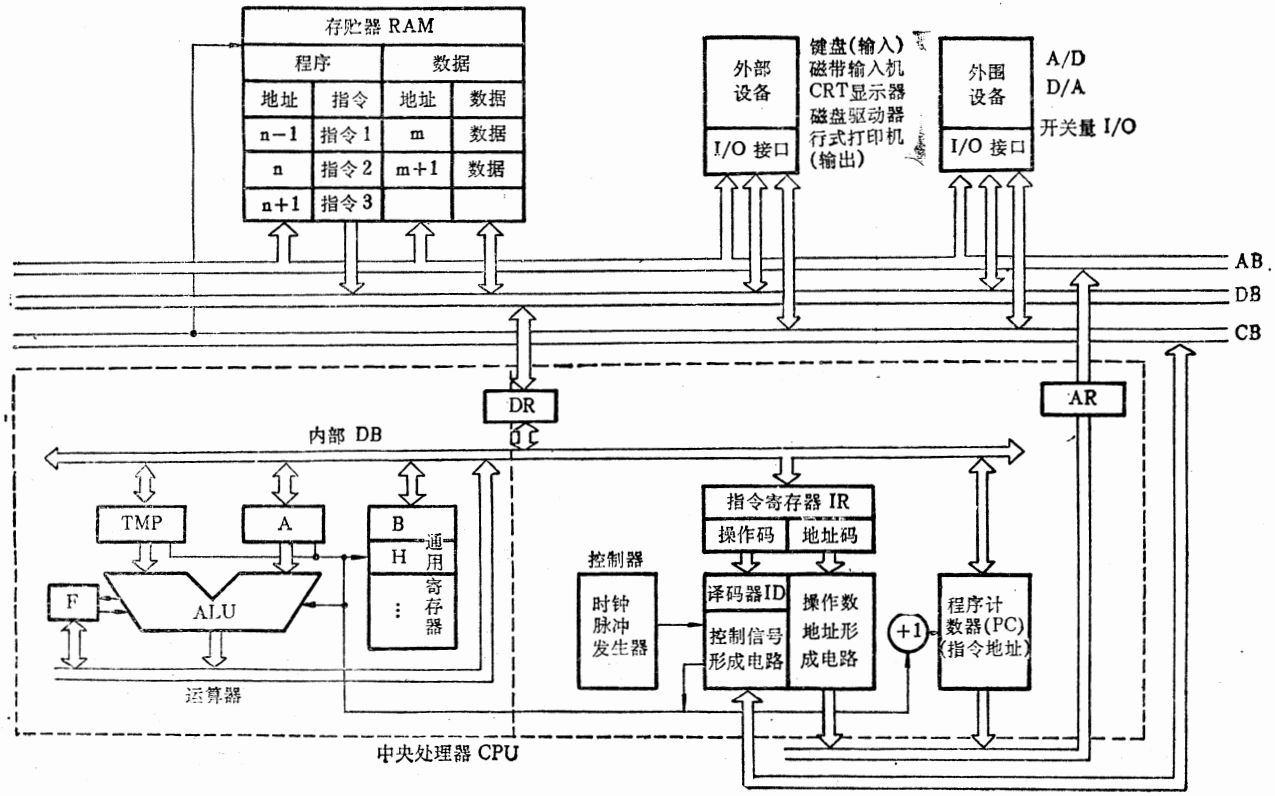


图 4-19　计算机的基本工作原理图

接口电路的基本原理。

一、堆栈(Stack)

现代计算机中，通常都设有堆栈，用来暂时存贮数据，它由一组寄存器或存贮单元构成。堆栈中的数据是以“后进先出”(LIFO——Last In First Out)的方式处理的。例如在计算机中，处理的程序常要转入子程序(Subroutine)，子程序完成后又返回主程序(Main Program)。这就要将主程序的返回地址存放在堆栈中。但子程序在执行过程中，又可能转入另一子程序并返回。因此，可能发生多重嵌套，采用后进先出的堆栈，保存多重返回地址，就可以保证子程序执行完后能有秩序地返回主程序继续执行，如图 4-20 和图 4-21 所示。

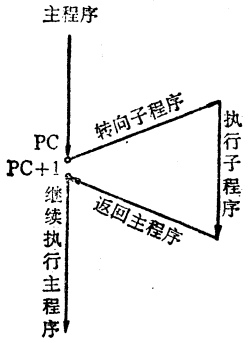


图 4-20　主程序转子程序及返回示意

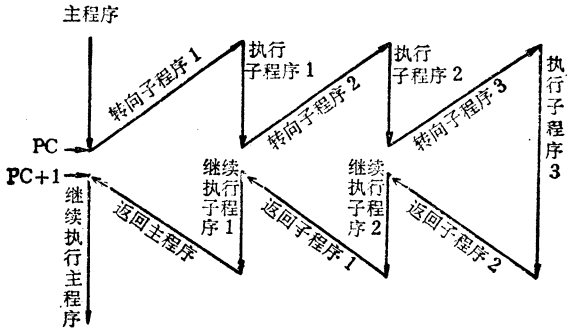


图 4-21　子程序嵌套示意图

实现堆栈的方式有两种：

1．硬件堆栈

它由一组直接位于微处理器芯片内部的寄存器组成，其优点是访问寄存器的速度较快，缺点是寄存器数量有限，即堆栈深度有限。早期的微处理器如 Intel 8008 即采用硬件堆栈，它由 8 个 14 位寄存器组成，若这 8 个寄存器用满，则表示堆栈已满，必须把堆栈内容复制到存贮器中，否则程序将会出错。同时，还必须用一种状态来表明堆栈是“满”或是“空”的。这往往会带来很多不便，因此，目前很少用这种方法。

2．软件堆栈

目前大部分微处理器采用软件堆栈技术，它是在微处理器外部的随机存贮器(RAM)中开辟一个连续存贮单元的区域作为堆栈，而在微处理器内部设置一个堆栈指示器(SP——Stack Pointer)或称堆栈指针，实际上它是一个专门用来指示堆栈的栈顶单元地址的地址寄存器。

通常，对堆栈的写入和读出，是通过堆栈指令来实现的，而栈顶由堆栈指示器自动管理。每当执行一次数据入栈(推入 PUSH)操作之前，堆栈指示器便自动执行栈地址减 1，即 SP-1→SP，使 SP 指向一个新单元，然后再把寄存器的数据推入 SP 所指的单元；反之，每当执行一次数据出栈(弹出 POP)，操作之后，堆栈指示器便自动执行栈地址加 1，即 SP+1→SP (SP 仍指向堆栈的顶部)。这样就保证了堆栈指示器始终指向栈顶地址。SP 的初值可由程序设定，通常总是指向内存 RAM 中最大的可用地址。在管理程序中除设置好堆栈的基址外，还要预先估计好堆栈的深度。下面举例说明入栈与出栈的执行过程。

【例 1】 将 CPU 的内部寄存器 B、C 的内容推入堆栈(即执行 PUSH BC 指令)，设执行前 B = 45 H，C = 24H，SP = 2 FC0H。

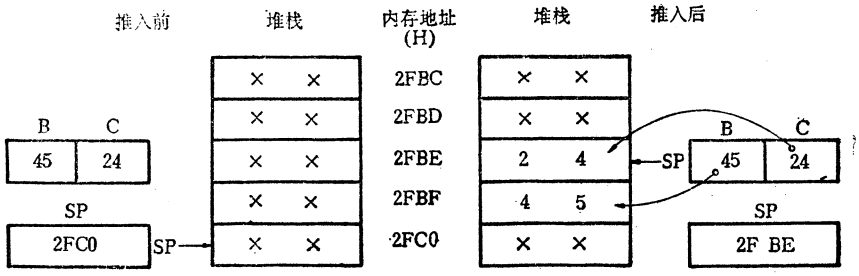


图 4-22　执行 PUSH　BC 指令示意图

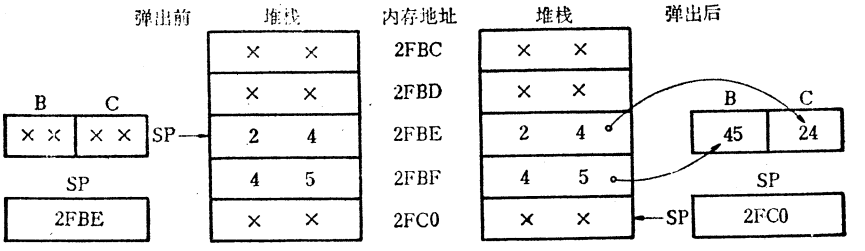


图 4-23　执行 POP BC 指令示意图

图 4-22 表示执行 PCSH BC 指令，必须先令 SP-1→SP，使 SP 指向 2 FBFH 存贮单元，接着把寄存器B(即寄存器对 BC 的高位)的内容 45H 推入 2FBFH 单元；然后再令 SP-1→SP，把寄存器 C (即寄存器对 BC 的低位)的内容 24 H 推入 2FBEH 单元，推入后 SP 就指向 2 FBEH 单元，即 SP 指向堆栈的顶部。

【例 2】 将堆栈中内容弹出到 CPU 内的寄存器 B、C，即执行 POP BC 指令，设执行前 SP = 2 FBEH，存贮单元 (2 FBEH) = 24 H，(2 FBFH) = 45 H。

图 4-23 示出执行 POP BC 指令的示意图，当执行 POP BC 时，先把 SP 所指单元的内容弹出，送至寄存器 C(寄存器对的低位)，然后令 SP + 1→SP；再把 SP 所指的内容弹出送至寄存器 B(寄存器对的高位)，然后令 SP + 1→SP，使 SP 仍然指向堆栈的顶部。

应当注意，堆栈指针有两种指示方法：一种是堆栈指针指出堆栈操作之后的堆栈顶部数据地址，如 Intel 8080 A/8085 A、Z 80 就是采用此种方法，上面两个例子即采用这种方法；另一种是堆栈指针指出堆栈操作之后的堆栈顶部以上一个空单元的地址，如 MC 6800 即采用这种方法。

堆栈主要用于处理调用子程序和中断控制，特别是对于子程序多重嵌套和多级中断控制，堆栈方式提供了很大的方便。

二、数据通道和总线结构

1．数据通道

计算机中传送信息和数据的装置称为通道。它包括两方面的含义：一是指 CPU 与主存和 I/O 设备间的双向数据总线传输的途径，连同暂时存放它的寄存器、锁存器及缓冲器等统称为数据通道(Data channel)；另一则是指 CPU 内部的信息传输途径，为区别起见，有时称后者为数据通路(Data path)。这里，我们泛指所有的信息传输途径，一律统称为通道。

在微型计算机中，数据通道用来连接各部分(包括内部和外部)，以保证数据和指令的流通。

数据字的流通是从主存或外设输入通道，送往 CPU 内的累加器或其它寄存器的。运算结果暂存于累加器，并可从这里送往主存或通过 I/O 总线送到输出设备去。

控制器能根据流通的字的类型来启动不同的数据通道。下面通过例子来说明，在信息传送过程中，控制器是如何启动数据通道的。

图 4-24(a) 给出了 CPU 内各寄存器之间通过总线连接的情况。假设要把信息从寄存器 B 传送到寄存器 C，控制器只需提供能启动寄存器 B 上的与门的逻辑电平，就能把寄存器 B 的内容从或门输出(即总线)。当 B 的内容到达总线后，控制器立即向寄存器 C 的时钟脉冲轮

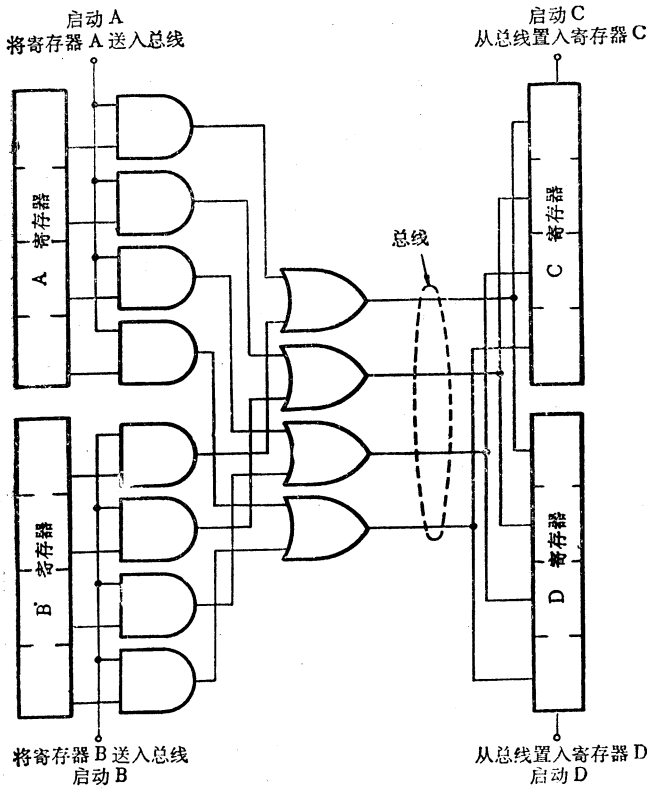


图 4-24(a)　在控制器控制下两个寄存器间的数据传送

入端送一个脉冲，于是总线上的信息就送入寄存器 C。在任何给定时刻都只能允许将一个寄存器的信息送到总线上，而所有脉冲都通过时钟脉冲给予同步化。说明这种信息传送过程的时序如图 4-24 (b)所示。

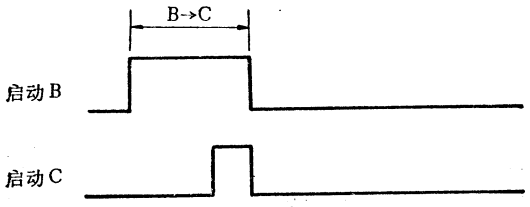


图 4-24(b)　为了将图 (a) 中寄存器 B 的内容传送给寄存器 C，控制器产生的典型定时信号

CPU 与主存及外设之间的信息传送方式也与此类似。

2．总线结构

(1) 总线的概念

总线(BUS)是指连接计算机各部件、或计算机之间的一束公共信息线，它是计算机中传送信息代码的公共途径，如图 4-25 所示。

由图可见，正如水从水管中流过一样，信息代码可以从总线中通过。总线有发送端和接收

端，各端分别设有若干个发送门和接收门。发送门打开，便将信息代码送到总线上去，接收门打开便能接收从总线上传送来的信息代码。

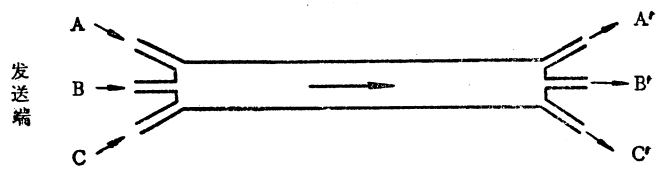


图 4-25　总线的示意图

由于在计算机中，信息代码的传送是用电平的高低来表示的，因此在某一瞬时，不允许几个发送门同时向总线上发送信息代码，换言之，在某一瞬时只能允许一个发送门打开发送信息，不同的发送门需要在不同的时刻发送信息，这种性质称为发送端的分时性。因此，为了避免挂在总线上的其它部件输出对数据的影响，要求凡是挂到总线上的部件（芯片）的数据线，都必须经过三态驱动缓冲电路而输出。

（2）总线结构

采用总线结构可以减少信息传输线的根数，提高系统的可靠性，增加系统的灵活性，便于实现系统积木化，因此现代计算机已普遍采用总线结构。

计算机总线通常有以下三种：

① 芯片总线（Chip BUS），这种总线是微处理器片内的总线，用来连接微处理器内部的各逻辑功能单元。

② 内总线（Internal BUS），又称系统总线（System BUS），就是通常所说的微型计算机总线。这种总线用来连接微型机内的各部件，如 CPU 与内存贮器或 I/O 接口电路之间的连接。如 Intel 8080A/8085A 和 Z80 微型机使用的 S-100 总线[注1]。

③ 外总线（External BUS），又称通信总线（Communication BUS），这种总线用于各微型机系统之间或微型机系统与其它系统（控制系统等）之间的通信。如 IEEE-488总线[注2]。

这里，着重介绍微型机的系统总线和芯片总线。

（3）系统总线

通常微型机的系统总线分为地址总线、数据总线和控制总线，即所谓三总线结构（见图 1-2）。现分述如下：

① 地址总线（AB）

它是微处理器用来输出确定的存贮器或 I/O 部件地址的总线。一般对于 8 位微处理器而言，它有一条 16 位宽的单向地址总线，可允许寻址 $2^{16}=65536$ 个单元。由于 LSI 电路封装的限制（目前一般多为 40～48 条引线），因此有时也用地址总线中的 8 条来传送数据。这时，为了区分传送的是地址还是数据，必须靠定时的时间脉冲来选通。

② 数据总线（DB）

数据总线总是双向的，用来实现 CPU、存贮器和 I/O 部件三者之间的数据交换。数据总线的宽度一般与 CPU 处理数据的字长相同。对于 8 位微处理器而言，它有一条 8 位宽的具有三态控制的双向数据总线，这样便于实现 CPU 的数据总线与系统数据总线的连接或“脱开”，这对于快速传送数据方式，即直接存贮器存取（DMA——Direct Memory Access）是必要的。

【注1】 1976 年美国部分厂家与用户召开会议，确定以此总线作为 8080 为核心的微型机系统内总线标准，该总线共有 100 条线，故称 S-100 总线。

【注2】 IEEE-488 总线是 Institute of Electrical and Electronic Engineers-Standard 488 的简称。它是微型机系统的并行外总线标准。

③ 控制总线 (CB)

控制总线用来传送使微型机内各个部件的动作同步和协调的定时信号和控制信号，从而保证正确地执行各种数据的传送操作。控制总线中由 CPU 发出的信号有：同步脉冲、定时脉冲、读写操作控制及对外界信号的应答信号；由其它部件输入到 CPU 的状态信号有：准备就绪、中断请求、保持请求等信号。

④ 双向数据总线的实现

为了实现数据总线的双向传送，必须采用三态门来控制数据的流向。图 4-26(a) 示出某微型机 4 位宽的典型双向数据总线。它由一个 DE 触发器和一个三态逻辑门组成。DE 触发器是在一般的 D 触发器上另加一个 E 允许端而构成的。这里，E 输入端用以控制 D 的输入。若 E 为 "0"，则即使 D 为 1、也不能输入。三态门的控制端是一个发送端，当它禁止发送时，这个门就如同断开一样，其数据可以直接送入 D 触发器。当发送数据时，接收端 E 禁止 D 端输入，发送端使三态门接通，Q 信号输出。这样，在总线上就可以并行双向传送 4 位数据。

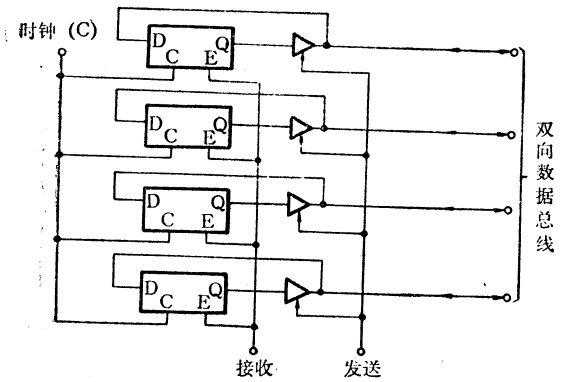


图 4-26 (a)　带有锁存器的 4 位双向数据总线

总线上往往还有双向缓冲驱动器，其结构图如图 4-26(b) 所示。其中，所用的基本电路即为三态门电路，其控制端受外界信号控制。当输出控制端 E_W 为有效("1" 电平)时，数据从 CPU 写到或输出到存贮器或 I/O 部件；反之，当输入控制端 E_R 为有效("1" 电平)时，数据将从存贮器或 I/O 部件读入或输入到 CPU 中来。

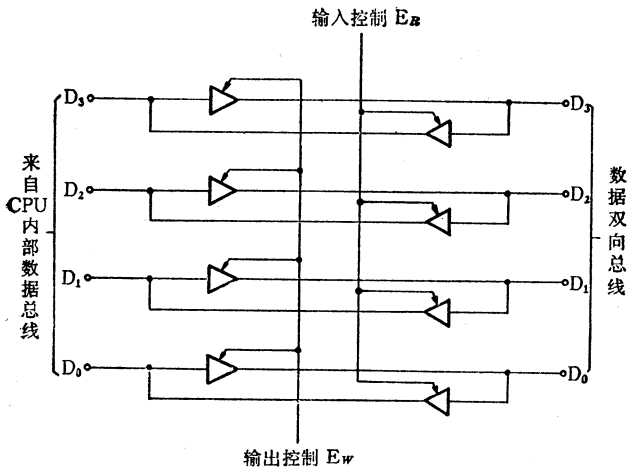


图 4-26(b)　双向缓冲驱动器

(4) 芯片总线

现代微处理器的芯片总线有下列三种：

① 单总线结构

在图 4-27 所示的单总线结构中，所有各部件的输入和输出都公用一条一定宽度的数据总线，其优点是在芯片结构中总线占用面积最小，节省空间，比较经济、简单。缺点是操作速度较慢，如为了把两个操作数送到ALU，需要分两次进行，而且还需要两个缓冲器。例如执行一次加法操作的过程如下：第一个操作数先存入A缓冲器，接着再将第二个操作数存入B缓冲器。只有两个操作数同时出现在ALU的两个输入端，ALU 才执行加法操作。当加法结果出现在单总线上时，由于输入数已暂存于缓冲器中，故不会受到影响，然后，进行第三个操作，以便把加法的结果选通到目的寄存器中。

尽管单总线操作速度较慢，但由于它节省硅片总线面积，而这是目前微处理器设计制造时

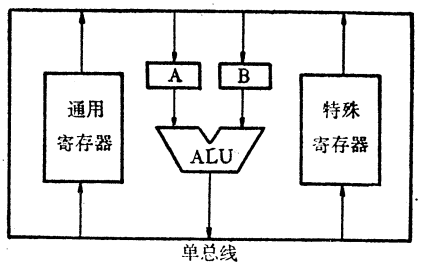


图 4-27　面向 ALU 的单总线结构

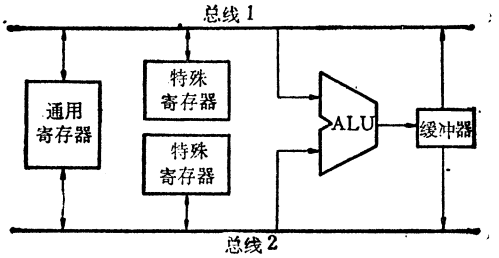


图 4-28　面向 ALU 的双总线结构

需要考虑的主要因素，因此目前大多数微处理器都采用单总线结构。

② 双总线结构

为了提高微处理器的操作速度，也有采用双总线和三总线结构的。双总线结构如图 4-28 所示。

在双总线结构中，两个操作数同时送到 ALU 进行运算，只需要一次操作控制，且可以立即得到运算结果。图中，两条总线各自把其数据送到 ALU 的输入端。ALU 的输出不能直接送到总线上去。这是因为，在形成操作结果的输出时，两条总线都被输入数所占据，故必须在 ALU 的输出端设置缓冲器。因此，操作的控制要分两步完成：第一步，在ALU两输入端输入操作数，形成结果并送入缓冲器；第二步，把结果送入目的寄存器。

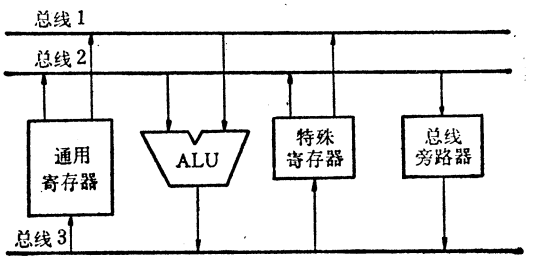


图 4-29　面向 ALU 的三总线结构

③ 三总线结构

三总线结构如图 4-29 所示，在三总线结构中，ALU 的两个输入端分别由两条总线供给，而 ALU的输出则另与第三条总线相连。这样，算术逻辑操作便可在一步的控制内完成。另外，还设置了一个总线旁路器。如果一个操作数不

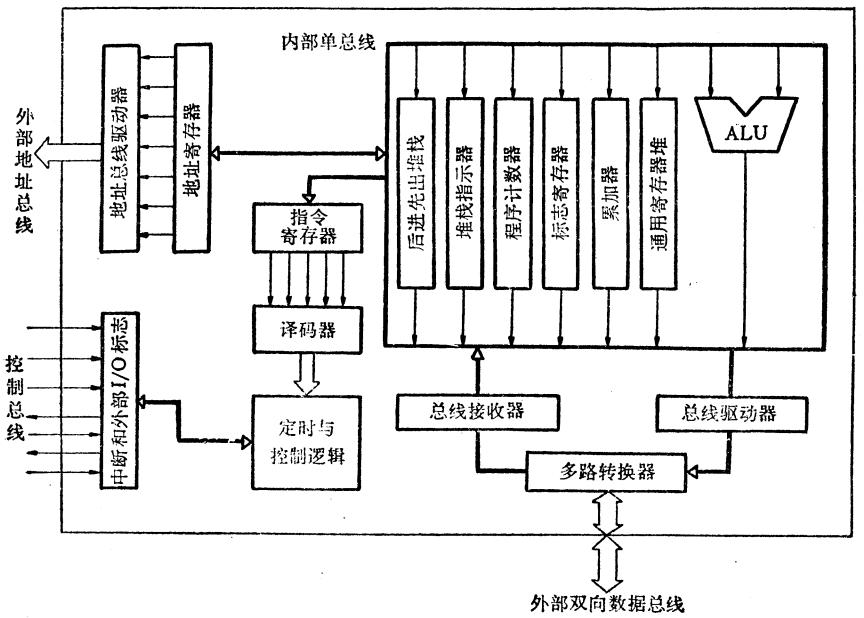


图 4-30　单总线结构的微处理器

需修改，而直接从总线 2 传送到总线 3，则可以通过控制总线旁路器将数据传出。虽然三总线结构的操作速度较快，但由于它在硅片上实现比较困难，因此，目前在微处理器中尚很少采用。

芯片总线用来连接微处理器内部的部件，而系统总线则用来连接微处理器与外面的部件。它们之间又相互连接沟通，从而形成微型机内信息流通的完整的通道。图 4-30 示出一个典型的单总线结构的微处理器框图，从中可以清楚地看出微处理器内各部件之间的关系。

三、输入输出设备

通常输入输出设备可分为两类：

1．外部设备

它是指计算机中除主机以外的其它设备，一般包括输入输出设备和外存贮器。

输入设备的任务是将程序、原始数据以计算机所能识别的形式送入计算机中，供计算机处理。

输出设备的任务是将计算机对信息处理的结果，以人所能识别的各种形式，如数字、字母或符号、图形等表示出来。

微型计算机常用的外部设备有：

(1) 纸带输入机

通常用的光电纸带输入机(photo-electric paper tape reader) 的基本原理，是用光源通过纸带上的孔照射到光敏元件上，从而把纸带上的孔信息转换成计算机所能接受的电信号而送到计算机中去。使用者将需要送入计算机的信息码在纸带上进行穿孔，有孔表示“1”，无孔表示“0”，这称为穿孔纸带。目前常用的是 8 单位纸带和 5 单位纸带。8 单位纸带如图 4-31 (a)所示，每横排有 9 个孔，一般来说，其中 1 个中导孔，1 个特征孔，6 个代码孔和 1 个奇偶校验孔。5 单位纸带每横排有 6 个孔，一般来说，其中 1 个中导孔，1 个特征孔，3 个代码孔和 1 个奇偶校验孔，如图 4-31(b) 所示。

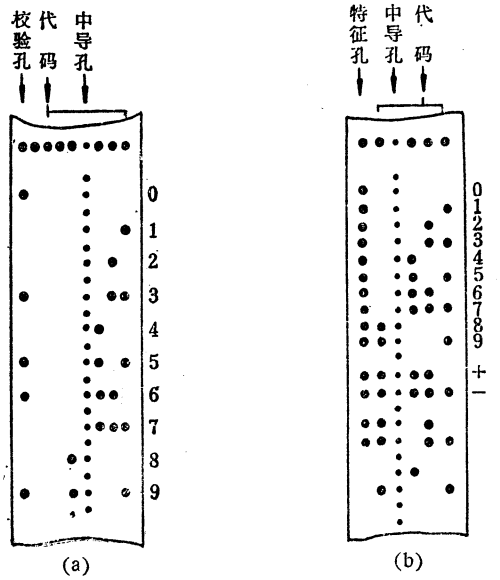


图 4-31　穿孔纸带格式

中导孔也称同步孔，在输入时它产生一连串的脉冲，作为机电设备与电子线路之间的同步脉冲。特征孔用来区别该排孔是数码还是特征码。奇偶校验孔用来及时发现纸带输入时是否出错，事先人为地编排好每排孔：代码孔＋奇偶校验孔＝偶数(或奇数)，若在输入时出现不符合上述编排的情况，则输入机立即执行“奇偶出错”停机。

除光电纸带输入机外，还有电容式纸带输入机(capacity paper tape reader)，其原理和光电纸带输入机大致相仿。

(2) 盒式磁带机 (Cartridge tape unit)

盒式磁带用录音磁带串行记录二进制的数字信息，它可以用来取代纸带输入机和穿孔机，作为计算机的输入设备或外存贮器。

(3) 电传打字机 (teletypewriter)

电传打字机既可以作为输入设备也可以作为输出设备。它主要由三个部分构成：发送、接收和凿孔部分。发送部分具有象打字机一样的键盘，按动字键即发出相应的电码脉冲、经传输线路送到计算机，同时也送到本机的接收部分，打印出字符和数字，或产生所需要的动作，如换行、复位等。计算机的处理结果也可送到其接收部分，进行打印或凿孔输出。

(4) 行式打印机 (Line Printer)

按行打印字符的打印机称为行式打印机。它是计算机的输出设备，计算机根据使用的要求，随时可以输出打印。通常分宽行打印机和窄行打印机两种，其结构基本相似。常用的有15行16字符的窄行打印机，有行宽为80行、120行、160行，字符为64种、96种、128种的宽行打印机。

(5) 阴极射线管(CRT——Cathode-Ray Tube)显示装置

CRT 显示器是一种先进的输入/输出设备。它的特点是速度快、显示的信息量大，直观性强和使用方便等，现已被广泛用于各种类型的计算机中。许多微型机配备专用接口和程序，可用一般民用电视机构成简易型 CRT 显示器。有的 CRT 显示器还可利用光笔在屏幕上修改数据或图形，使用更加方便。

2. 外围设备

外围设备是构通计算机和应用对象之间联系的输入/输出设备。常见的外围设备有：

(1) 模/数转换器 (A/D Converter——Analog/Digital Converter)

在计算机内部，只能处理数字形式的信息，但我们日常所接触到的信息，有很多都是连续变化的模拟量。例如温度、压力、流量、位移、转速等。模/数转换器的任务就是将输入的模拟量转换为数字量，供微型机处理。

(2) 数/模转换器 (D/A Converter——Digital/Analog Converter)

数/模转换器的任务是将微型机处理后的数字量转换为模拟量形式的控制信号。计算机用于生产过程控制，经常需要控制机电执行机构和控制部件以进行连续调节，如利用电动调节阀去控制锅炉的水位等。由于计算机输出的是数字量，必须通过数/模转换器转换成相应的模拟量，才能达到连续调节、控制的目的。

在微型机中，通常把在微型机与外界之间提供通信手段并参与数据处理的设备笼统地称为输入/输出(I/O)设备或外围设备，简称外设。

3. I/O 接口 (Input/Output interface)

I/O 接口是主机与外围设备的连接部分。主机与外围设备之间的数据传送都是通过 I/O 接口实现的，因而，需要相应的 I/O 接口逻辑。I/O 接口仅仅是接口中的一种。通常，接口一词泛指两个不同系统的交接部分，例如计算机之间的通信接口等。这里，我们专指 I/O 接口。

中央处理器按照其本身固有的时钟规范和操作特点进行工作，而种类繁多的外设又都按其各自的特点操作，I/O 接口的作用是在这两者之间进行协调。

通常，I/O 接口的主要功能有地址译码、数据缓冲、信息转换、提供命令译码和状态信息以及定时和控制等。

由于外围设备种类繁多，使得 I/O 接口部分成为计算机系统中最为复杂的一部分。在微型机中，常用的接口逻辑电路大多采用 LSI 芯片，而且为了加强通用性而设计成可编程序 I/O 接口芯片。

至此，经过扩充后的模型计算机已是一台完整的电子计算机系统了。其结构原理如图 4-19 所示。

第五章　微处理器

本章先选择一个具有典型结构和信息的微处理器进行剖析，然后列举若干种常用的微处理器(如 Intel 8080A/8085 A、Z80 和 MC 6800 等)详细讨论它们的内部结构特点及操作原理。

§5-1　典型微处理器的结构

假设典型微处理器操作字的字长为 8 位二进制数，它有一组 8 位双向的数据总线，一组 16 位单向地址总线，可直接寻址 $2^{16}=65536=64KB$ 个存贮单元。控制总线由不同类型的微处理器来确定。

图 5-1 所示为典型微处理器的内部结构。

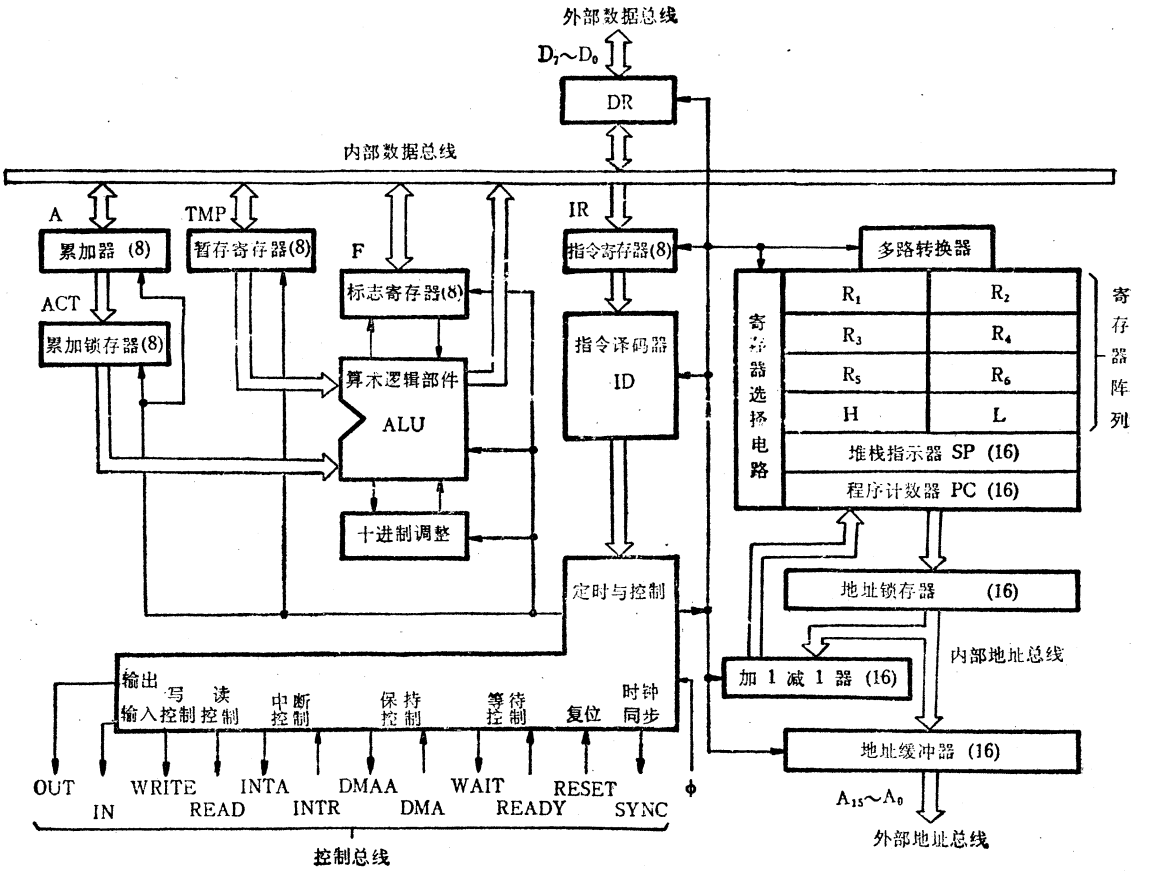


图 5-1　典型微处理器的内部结构框图

通常，典型微处理器的内部结构至少包括下列三个基本功能部件：

1．寄存器阵列，包括通用寄存器和专用寄存器，用作暂时存放数据和地址；

2．算术逻辑部件 ALU，简称运算器；

3．控制与定时部件，简称控制器。

现分述如下，读者可结合第四章中有关内容进行研究。凡是第四章中已经详述过的内容本章将不再重复，仅指出其特点。

一、寄存器阵列

寄存器阵列是微处理器内部的临时存贮单元，它既可以存放数据和地址，也可以控制程序的执行次序。适当增加微处理器内部寄存器的数量，CPU 就可以直接从寄存器阵列得到更多的数据，从而减少访问存贮器的次数，有利于提高 CPU 的运算速度。

微处理器内部各寄存器大致可分为以下四类：

1．存放待处理数据的寄存器。

2．存放地址码的寄存器。

3．存放控制信息的寄存器。

4．在数据传送过程中起缓冲作用的寄存器（缓冲器、锁存器和暂存器等）。

下面分别予以说明：

1．存放待处理数据的寄存器

这类寄存器主要包括通用寄存器及累加器等。

(1) 累加器 A

通常微处理器中至少要有一个累加器，如 Intel 8080A/8085A 微处理器内只有一个累加器。这样，程序在处理累加器进出数据方面，将要花很多时间。现在，大多数小型机和某些微处理器（如 MC 6800）都设置两个以上的累加器，以提高处理数据的效率。

(2) 通用寄存器组 R_1～R_6、H、L

通用寄存器是在数据处理过程中可以被指定为各种不同用途的一组寄存器。

为了高速处理数据，在 CPU 内通常设置通用寄存器组，用来临时存放数据和地址。由于 CPU 可以直接处理这些数据和地址，因此可以减少访问存贮器的次数，从而提高运算速度。对于 8 位微处理器来说，每一个寄存器由 8 位触发器组成，并和内部数据总线进行双向连接，同时由寄存器选择电路来确定哪个寄存器参加操作。它们既可以单独使用，也可成对（16 位）使用。如 Intel 8080 A/8085 A 有 6 个通用寄存器（也可组成 3 个寄存器对 BC, DE, HL），而 Z80 具有两组与 Intel 8080 A相同的通用寄存器组，因而在功能和效率上都大大增强了。而 MC 6800 不设置通用寄存器，以双累加器配合主存贮器来承担通用寄存器的任务。

值得指出的是，在 Intel 8080 A/8085 A 和 Z80 微处理器中，H 和 L 寄存器通常作为保存数据的存贮器地址指针。这就是说，用 HL 寄存器对来存放待访问数据的存贮单元的 16 位地址。HL 寄存器对所指定的存贮单元可以同 CPU 内任何寄存器传送数据。当然，用其它寄存器对 BC（或 DE）也可以作为 16 位数据指针。但是，这时仅能在存贮单元和累加器之间传送数据。这一点应当注意。

2．存放地址码的寄存器

主要有程序计数器（PC），堆栈指示器。

(1) 程序计数器 PC（16 位）

程序计数器用来存放现行指令的 16 位地址，每当取出现行指令后，PC 就自动加1（除转移

指令外)，以指向下一条指令地址。如果指令是多字节的，则每取一个字节，PC 就增 1；取出一条指令的所有字节之后，PC 仍指出下一条指令地址。一般指令是按顺序执行的，故 PC 可以用来控制程序执行的顺序。仅当遇到转移指令时，PC 的内容才被所指定的地址取代，从而改变程序执行的正常次序，实现程序的转移。

(2) 堆栈指示器 SP(16 位)

堆栈指示器是用来指示 RAM 中的堆栈栈顶地址的 16 位寄存器。每次推入或弹出一个数据时，它能自动减 1 或加 1，以保证始终指向栈顶地址。

3．存放控制信息的寄存器

主要有指令寄存器 IR 和标志寄存器 F。

(1) 指令寄存器 IR

在微处理器中，IR 专用于存放指令的操作码，一直保存到被指令译码器 ID 译码完成为止。IR 的字长与微处理器的字长相同。

(2) 标志寄存器 F

标志寄存器用来保存 ALU 操作结果的条件标志，它可以用专门的指令来测试，以判断程序是否需要转移。

大多数算术、逻辑操作都会影响一个至几个条件标志，每个标志都可视为一个触发器。其置位和复位的状态由最后执行的算术逻辑运算的结果决定。条件标志寄存器的状态位，取决于 CPU 所需要检测的运算结果的状态。它一般由 8 位触发器组成。通常状态标志有以下几种：

进位(C 或 CY)标志位。

全零(Z)标志位。

符号(S)标志位。

辅助进位(AC 或 H)标志位。

奇偶校验(P)标志位。

溢出(V)标志位。

加/减(N)标志位。

以上标志位的含义将在第六章指令系统中详细介绍。标志寄存器的内容除了可由专门指令来检测外，还可以由程序员通过指令来设置，这样可以增强微处理器的灵活性。

4．起数据缓冲作用的寄存器

这里包括数据缓冲和地址缓冲寄存器。如

(1) 暂存寄存器 TMP

(2) 累加锁存器 ACT——Accumulator latch

ACT(8 位)用来锁存从累加器送来的操作数并送往 ALU 进行运算。

设置 TMP 和 ACT 的主要目的，是为了避免在单总线结构的微处理器内的总线上产生输入输出数据的冲突，也就是起数据缓冲的作用。

(3) 数据缓冲寄存器 DR

DR 是在 CPU 内部数据总线和外部数据总线之间起缓冲作用的三态双向缓冲器。CPU 送出的数据，先保存在 DR 中，待外部数据总线允许传送时，再把数据送至外部 DB 上；或者当从外部 DB 向 CPU 送入数据时，先把数据送入缓冲寄存器，待 CPU 内部接收该数据的寄存器准备好时，或内部 DB 允许传送时，再把数据送至相应的寄存器，这是为了避免总线冲突而采

取的措施。

(4) 地址锁存器 (16 位)

地址锁存器用来存放 16 位地址码，并送往加 1 减 1 器或地址缓冲器。

(5) 地址缓冲器 (16 位)

地址缓冲器是具有三态控制的单向缓冲器，用于在 CPU 内部地址总线和外部地址总线之间起缓冲作用。

图 5-2 示出了微处理器内 16 位地址信息的传送情况。PC、SP 中的 16 位地址码和 HL、BC、DE中的16位信息都可通过地址锁存器送到加 1 减 1 器中，执行加 1 或减 1 后再送回原处。

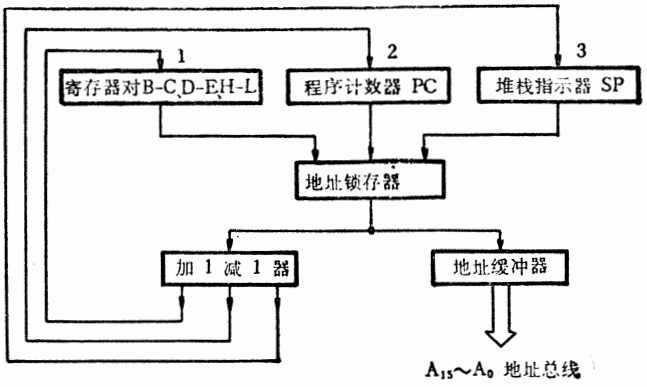


图 5-2 十六位地址信息的传送

二、算术逻辑部件 ALU——运算器

运算器是一个专门执行算术和逻辑运算的部件。如图 5-1 所示，它由累加器 A、累加锁存器 ACT、暂存寄存器 TMP、标志寄存器 F、ALU 以及十进制调整线路等组成。

执行运算时，信息从 ACT、TMP 及 F 的 CY 输入 ALU，运算结果送往数据总线或累加器 A。同时，将运算结果的状态送标志寄存器 F 保存，以作为条件转移指令的转移条件。

ALU 实际上是一个以加法器为核心的算术逻辑部件，这一加法器能按照二进制法则进行加法运算，又能通过二进制补码方法，把减法转化为加法进行运算。一般 8 位微处理器的 ALU 不具备乘除功能，而是通过编制乘除法子程序来实现乘除运算的。现在 16 位微处理器已具有乘除硬件，可直接利用乘除指令实现乘除运算。

三、控制与定时部件——控制器

控制器由指令寄存器、指令译码器和定时控制逻辑电路组成，如图 5-1 所示。它是分析和执行指令的部件，是统一指挥微型计算机按时序协调操作的中心部件。

控制器在程序控制下，从存贮器中读取指令操作码送往指令寄存器 IR，经指令译码器译码后将那些代表各种操作的操作码转换为表示该指令码操作类型的相应的控制电位，送入定时与控制逻辑电路，并在定时脉冲的同步下产生一系列执行该指令所需要的控制信号，使各部件按时序协调操作，从而执行该指令。

图 5-1 中定时与控制电路所接收的时钟、状态线 5 条和发出的控制信号线 8 条，其含义如下：

ϕ——时钟脉冲
RESET——复位
READY——准备就绪
DMA——直接存贮器存取请求
INTR——中断请求
SYNC——同步
WAIT——等待
DMAA——DMA 响应
INTA——中断响应
READ——读控制信号
WRITE——写控制信号
IN——输入
OUT——输出

关于上述信号的详细功能，将结合微处理器的实例进行说明。

§5-2 微处理器的操作

尽管各种类型的微处理器的功能千差万别，但是它们都是通过一些基本操作来实现的。深入理解这些基本操作，是考察一台微处理器的特定操作的必要前提。

通常，微处理器有以下几种基本操作：

1．取指令；
2．存贮器读或写；
3．输入或输出；
4．中断；
5．等待；
6．保持。

现分述如下：

一、定时

微处理器的操作是周期性的，取指令、执行指令、再取指令等等。这些操作发生的顺序都需要精确的定时。因而，微处理器需要有一个自激振荡的时钟发生器，为微处理器的一切动作提供一个基准。

在8位微处理器中，这种时钟信号，有的是由外部时钟电路供给的，并用石英晶体来稳频，如 Intel 8080 A 是采用外部提供的两相非重迭时钟作为微处理器内部或外部系统的定时信号；也有的是由微处理器内部的时钟电路来提供定时信号，如 Intel 8085 A。但是，目前微处理器时钟电路所用到的发振晶体，一般都还制作在微处理器芯片外部。

下面简要复习一下定时中的几个术语：

1．指令周期

一条指令的取出和执行所需的时间称为指令周期。通常一个指令周期包含若干个机器周

期，如 Intel 8080 A/8085 A 的指令周期包含 1～5 个机器周期(M_1～M_5)，Z 80 包含 1～6 个机器周期。而且任一指令周期的第一个机器周期都是用来取该条指令的操作码的，故又称之为取指周期。若指令是由一个以上字节(如二、三字节)所组成，则需要 2～3 个机器周期才能取完一条指令。而指令所规定的操作将在本指令周期的其它机器周期里执行。

2．机器周期

机器周期是指完成某一明确规定的基本操作的时间。不同的微处理器的机器周期的类型也不尽相同。通常一个机器周期包含若干个时钟周期(亦称时态)，如 Intel 8080 A，每个机器周期包含 3～5 个时钟周期(T_1～T_5)；Intel 8085、Z 80，每个机器周期包含 3～6 个时钟周期(T_1～T_6)。

3．时钟周期(时态)

时钟周期是指定时振荡器两个相邻脉冲前沿的时间间隔。它是微处理器完成一个微操作的最小的时间单位(T 时态)。

二、取指令

任一指令周期的第一个机器周期都是取指周期。它的任务就是从存贮器中取出指令的操作码，并将其送入指令寄存器 IR。关于取指过程，第四章已作了详尽介绍，这里不再重复。

三、存贮器读

这是指按给定地址从存贮器中取出指令或数据，并分别送到指令寄存器、数据寄存器或累加器的基本操作。“取指令”只不过是一种特殊的存贮器读操作而已。取出的指令还可能要求从存贮器再读出数据。此时，由 CPU 再发出一次读信号，并把相应的操作数的地址送给存贮器，而把再次读出的数据送往累加器或其它通用寄存器。

四、存贮器写

存贮器写操作除了数据流向不同外，其它和读操作相似。CPU 先给出指定的地址送至存贮器，接着发出一个写控制信号。然后，把要写的数据送入存贮器指定的地址单元。

五、输入/输出

输入/输出操作与存贮器读写操作类似，所不同的是被寻址的对象是某一输入/输出设备，而不是某一存贮单元。开始输入/输出操作前，CPU 先给出所指定的 I/O 设备的地址，接着发出相应的输入或输出控制信号，然后就可以接收数据或是发送数据。数据的输入或输出能够以并行或串行方式进行。所谓并行的输入/输出即指同时传送一个字的所有位，如 8 条线同时传送 8 位数据。串行的输入/输出是指在一条线上顺次传送 8 位数据。

六、中断

在计算机执行正常程序的过程中，一旦出现异常情况和特殊请求，计算机就应尽快地在适当时候暂时停止现行程序的执行，转去执行对异常情况和特殊请求的处理，待处理完毕之后，又返回到被暂时中断了的程序继续执行，这个过程称为中断。例如，当计算机输出数据到打印机时，由于慢速的打印机与快速的计算机之间在处理时间上的矛盾，致使 CPU 等待时间过长。

为此，采用中断技术，当 CPU 给打印机输出一个数据后，仍继续执行主程序。仅当打印机准备好接收下一个数据时，才向 CPU 请求“中断”。一旦 CPU 响应中断，就暂停执行主程序，并自动地转移到一个中断服务程序，输出下一个数据。待该数据输出后，CPU 继续返回执行主程序。这与调用子程序的情况十分相似，不同之处仅在于中断的启动来自外设，而不是来自程序。关于中断技术，将在第十二章详细介绍。

七、等待

“等待”是指微处理器的工作速度要受到存贮器或 I/O 设备数据的存取时间的限制。当 CPU 向存贮器送出一个指定地址时，存贮器需要足够的时间才能作出响应，然后，CPU 才能进行读写操作。大多数的存贮器能够以比 CPU 的系统存取时间快得多的速度作出响应。如果存贮器或 I/O 设备数据存取时间超过 CPU 的系统存取时间，这时可设置一个等待电路，使 CPU 的 READY(准备就绪)线上置一个请求信号，使 CPU 进入等待状态，暂时空转，直到存贮器或 I/O 设备将存取的数据准备就绪后，才释放 READY 线，CPU 由等待状态进入下一步操作，指令周期继续进行下去。

八、保持

“保持”又称为总线请求，它是指外部设备请求 CPU 与地址总线、数据总线断开，即处于高阻抗状态，以便进行 DMA (Direct Memory Access)操作。

在通常的输入输出操作中，CPU 控制着全部的数据传送。如输入设备的数据要存入存贮器中去，必须先将数据取入 CPU 的累加器中，再由累加器存入指定的存贮单元。同样，由存贮器送到输出设备的数据也必须经过 CPU。如果需要传送的数据很多，这种传送方法的速率就受到限制。

当外设与主存贮器之间有大批数据，需要高速传送时，通常采用 DMA 传送方式。这时，由外部 DMA 控制器取代 CPU 来控制地址总线和数据总线，以便外设与主存贮器之间直接传送数据。

任何一种类型的微型计算机系统的工作过程都是由上述几种基本操作来实现的。至于更深入地分析微处理器的操作时序关系，我们将在第七章专题讨论。

§5-3 Intel 8080 A 微处理器

1974 年底美国 Intel 公司研制成功 8 位并行微处理器 Intel 8080，随后又制造出 8080 的改进型产品 Intel 8080 A。它有 16 位地址总线，8 位数据总线，内存寻址范围可达 64 KB，I/O 设备寻址范围为 256 个输入/输出端口，可进行二进制、十进制及双精度运算，具有中断和 DMA 功能。

一、8080 A μP 的结构

8080 A μP 由 4 个功能部件组成：寄存器阵列及地址控制逻辑；算术逻辑运算单元；指令寄存器及控制部件；双向三态数据总线缓冲器。它们是通过内部数据总线联系的，其内部结构框图如图 5-3 所示。

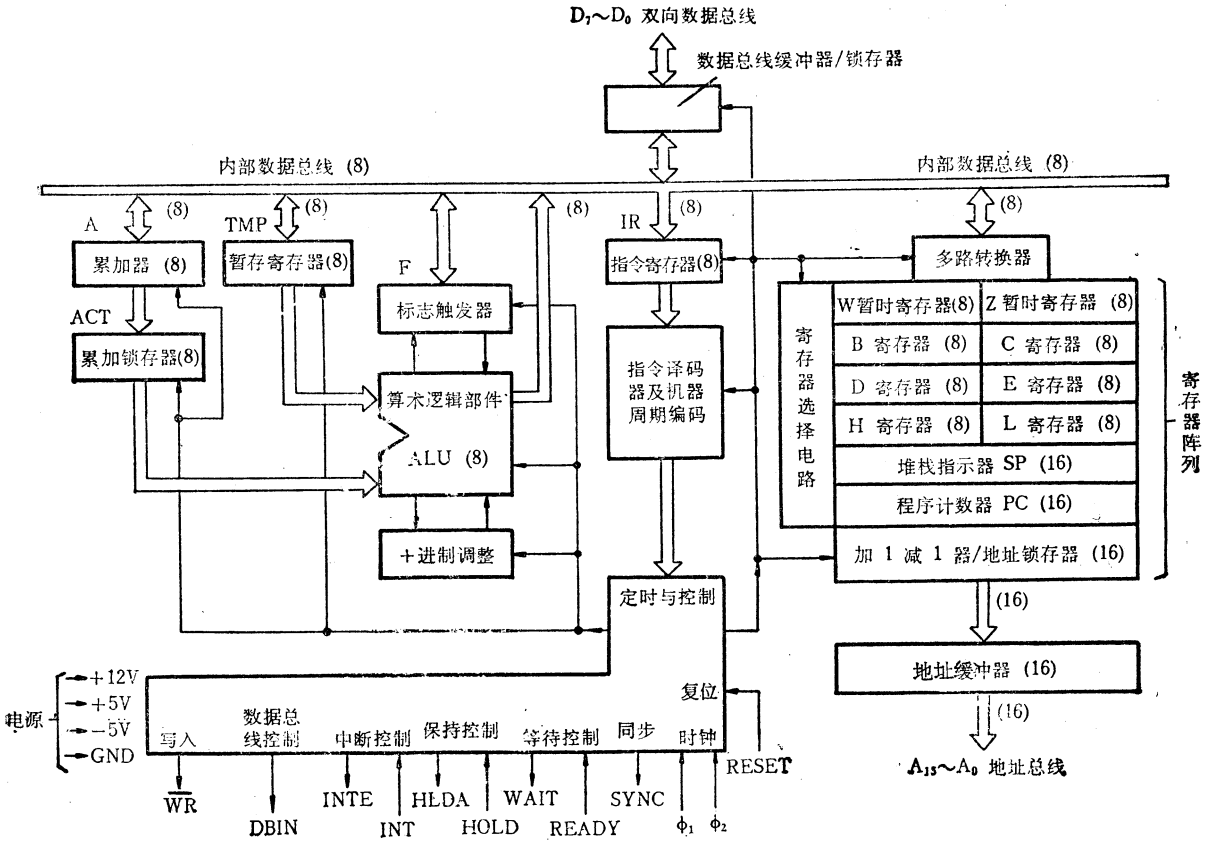


图 5-3 Intel 8080 A 内部结构框图

现分述如下：

1．寄存器阵列与地址控制逻辑

寄存器阵列实际上是一个静态读写存贮器(RAM)阵列，它由 6 个 16 位的寄存器构成。其中包括：

(1) 程序计数器 PC(16 位)。

(2) 堆栈指示器 SP(16 位)。

(3) 6 个通用寄存器 B、C、D、E 及 H、L (均为 8 位)，用来存贮数据或地址。它们既可作为单个寄存器使用，也可作为寄存器对(16 位)使用，即由 BC、DE、HL 组成 3 个寄存器对，分别写作“寄存器对 B”、“寄存器对 D”、“寄存器对 H”。16 位的地址可存于寄存器对中，地址的高 8 位存于 B、D、H 中；低 8 位存于 C、E、L 中。一般常用 HL 寄存器对保存存贮器的地址。

以上各寄存器都是可以编程的。

(4) 两个不可编程的 W、Z 暂时寄存器，仅在执行某些指令时，供 CPU 内部使用。

8 位的数据字节通过寄存器选择电路和多路转换开关，能够在内部数据总线和寄存器阵列之间进行传送。

16 位字能够在寄存器阵列和地址锁存器或加 1 减 1 器之间进行传送(见图 5-2)。地址锁存器从 3 个寄存器对 BC、DE、HL 中的任何一个接收数据并送至 16 位地址输出缓冲器(A_{15}～A_0)或加 1 减 1 器。加 1 减 1 器从地址锁存器接收数据，然后把它送到寄存器阵列。地

址缓冲器用来连接内部地址总线与系统外部地址总线。

2．算术逻辑运算单元 ALU

从广义上说，ALU 包括一个 8 位的累加器 A、一个 8 位的累加锁存器 ACT、一个 8 位暂存寄存器 TMP、一个 8 位的 ALU、一个 8 位条件标志寄存器（只用到其中 5 位：进位位 CY、零位 Z、符号位 S、辅助进位位 AC 及奇偶位 P）等以及十进制调整线路。

算术逻辑运算、循环移位或比较等操作都是在 ALU 中执行的。

3．控制器

它包括下列部件：

指令寄存器 IR(8 位)、指令译码器 ID 和定时与控制电路。

在指令译码器中，一个 8 位操作码能译码出表示 $2^8=256$ 种操作的控制电位，8080 A 实际上只用其中的 244 种操作电位。

4．数据总线缓冲器/锁存器

8 位双向三态数据总线缓冲器，用来隔离微处理器内部数据总线和外部数据总线（D_7～D_0）。在输出方式时，内部数据总线内容被送到一个 8 位锁存器，再送至数据总线的输出缓冲器。在输入方式时，数据从外部数据总线传送到内部数据总线。

二、8080A 引线功能

8080 A 共有 40 条引线，其中电源（$V_{CC}=+5V$，$V_{BB}=-5V$，$V_{DD}=+12V$）和接地端 V_{SS} 占 4 条。另有 36 条引线，其中包括地址总线、数据总线以及控制信号线（见图 5-4）。

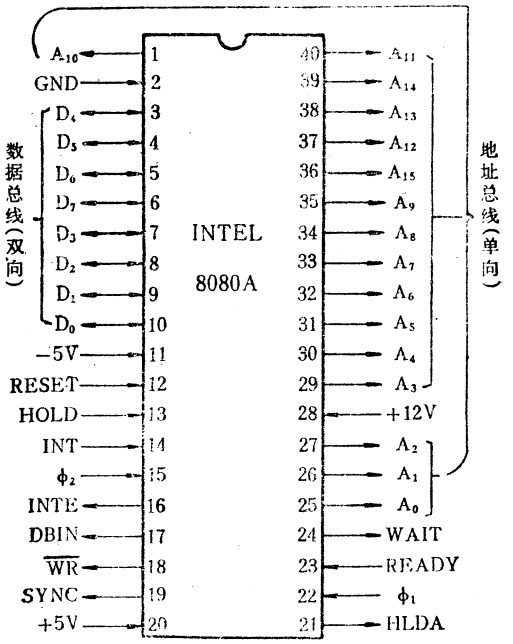


图 5-4　Intel 8080 A 引线图

下面分别介绍各条引线的含义及功能。

1．A_{15}～A_0 地址总线（单向，三态输出，16 条）

它用来发送存贮器和 I/O 设备地址。其中存贮器寻址范围 64 KB；I/O 设备寻址范围为 256 个输入和 256 个输出端口。A_0 为最低位。

2．D_7～D_0 数据总线（双向，三态，8 条）

它提供微处理器与存贮器、输入/输出设备之间进行指令与数据传送的双向通路。在每个机器周期的第一个时态（T_1），8080 A 输出一个字节的周期信息码送数据总线，表示现行机器周期的类型。D_0 是最低位。

3．控制信号线（10 条）

8080 AμP 有10种控制信号，其中，从8080A 送出的有 6 个定时及控制信号（SYNC，DBIN，WAIT，$\overline{WR}$，HLDA 及 INTE），而为 8080 A 接收的有 4 个状态信号（READY，HOLD，INT 及 RESET）。

(1) SYNC(Synchronizing signal)同步信号（输出）

它是 CPU 在每个机器周期开始工作时的同步信号。同时它还表明 CPU 的周期信息码已

送到数据总线(D_{7-0})上。

(2) READY 准备就绪信号(输入，高电平有效)及 WAIT 等待状态信号(输出,高电平有效)

设置这两条控制信号线，是为了使微处理器能与慢速的存贮器或输入/输出设备同步工作。当 CPU 送出一个地址码后,若 READY 输入端为低电平,则 CPU 就进入"等待"时态(T_W)。此时，CPU 停止读、写而输出线 WAIT 变为高电平。仅当 READY 升为高电平时,CPU 才由 T_2 或 T_W 进入 T_3 时态,指令就可以继续执行下去。WAIT 信号用来表明 CPU 是否处于"等待"时态。

(3) HOLD(HOLD Request)保持信号(输入,高电平有效)及 HLDA(HOLD Acknowledge)保持响应信号(输出,高电平有效)

设置这两条控制信号线,是为了使微型机具有直接存贮器存取(DMA)操作的功能。当外部设备向 HOLD 端输出高电平时,即向 CPU 请求"保持"，CPU在现行机器周期结束时即响应此"保持"请求。此时，地址总线和数据总线都处于高阻抗状态。同时,通过 HLDA 端输出高电平,表示 CPU 已响应"保持"请求。在"保持"期间，地址总线和数据总线处于发出请求的那个外部设备控制器的控制之下，直接同存贮器进行数据传送。当外设完成它的数据传送后，HOLD 申请结束,HLDA 变为低电平,CPU 又恢复到继续执行下一个机器周期的工作。

(4) DBIN(DATA BUS IN)数据总线允许输入控制信号(输出,高电平有效)

这个信号可用来控制从存贮器或 I/O 设备输入数据到 CPU 的门控电路。当 DBIN 为高电平时,表明数据总线处于输入工作状态,即数据由存贮器或 I/O 设备流向 CPU。

(5) $\overline{WR}$(WRITE)写入控制信号(输出,低电平有效)

它是控制数据总线上的数据写入存贮器或输出至 I/O 设备的定时信号。当 $\overline{WR}$ 为低电平时,允许 CPU 将数据从数据总线写入存贮器或 I/O 设备中去。

(6) INTE(Interrupt Enable)允许中断控制信号(输出,高电平有效)

这条输出线用来表示 CPU 内部允许中断触发器的状态。当 INTE 为高电平时,表示微处理器可以接受中断请求,它可用 EI(Enable Interrupt)指令置为高电平(开放中断)，也可用 DI(Disable Interrupt)指令置为低电平(禁止中断)。当微处理器响应中断后，即将 INTE 置为低电平,以禁止接受另外一个新的中断申请。利用 RESET 信号也可将 INTE 置为低电平。

(7) INT(Interrupt Requst)中断请求信号(输入,高电平有效)

I/O 设备通过 INT 端输入高电平向 CPU 请求中断。CPU 在执行完一条指令后 ，或处于暂停状态时,都可在此输入线上识别中断。当 INTE 为高电平时，则 CPU 能响应中断。如果 CPU 处于 HOLD 状态或 INTE 处于低电平,则不能响应中断请求。

(8) RESET 复位信号(输入,高电平有效)

它是确定微处理器初始状态的信号或称为人工干预机器的控制信号，对其它所有信号都具有优先权。当 RESET 为高电平时，程序计数器 PC 被置"0"，表示程序从零号地址开始执行。但是累加器、数据寄存器、堆栈指示器以及标志寄存器都不清除,这些寄存器的内容保持不确定状态,直至程序明确它们的初始状态为止。INTE 和 HLDA 触发器也被复位,禁止接受外来请求。此时,地址总线和数据总线都处于高阻抗状态。

4. ϕ_1、ϕ_2 时钟信号(输入)

它们是微处理器操作的定时信号。

三、8080 A 中央处理器模块的组成

应当指出，Intel 8080A 系统中的 CPU 模块至少需要 3 片电路组成(见图 5-5)：

(1) 8080 A 微处理器；

(2) 8224 时钟发生器驱动器(外接石英晶体)；

(3) 8228 系统控制器。

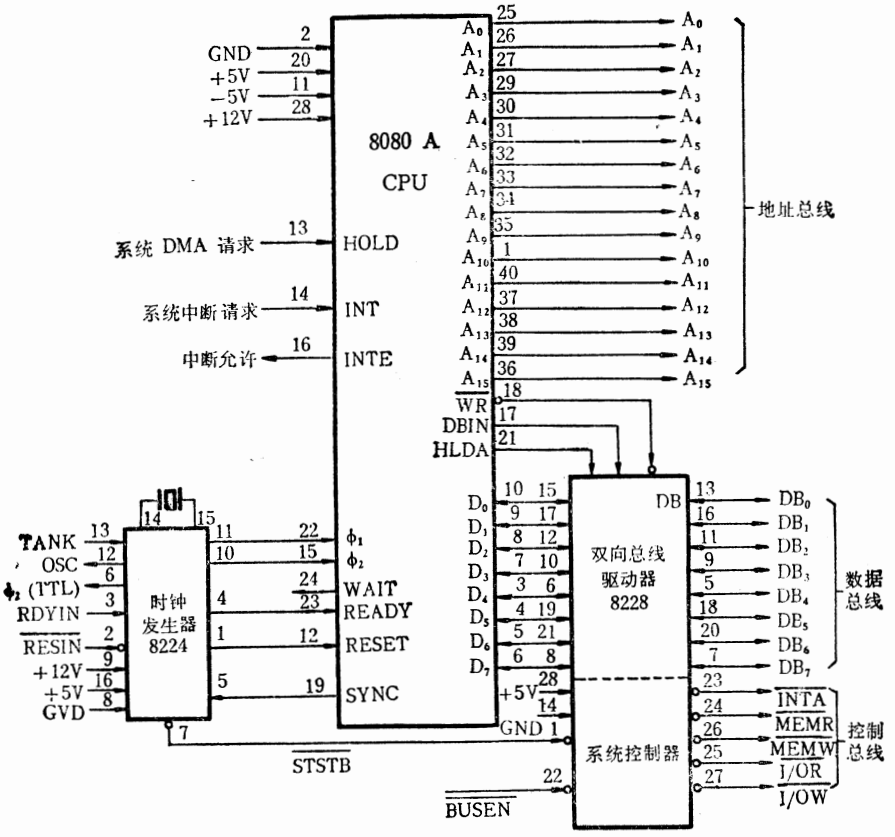


图 5-5　8080 A CPU 模块的连接图

下面分别介绍 8224 和 8228 的电路结构及其功能。

1．时钟发生器驱动器(8224)

8224 是肖特基双极型 TTL 集成电路。它主要用来产生 8080 A CPU 所需要的双相非重迭时钟信号 ϕ_1、ϕ_2，并且使异步的复位(RESET)和准备就绪(READY)信号得到同步。

图 5-6 和图 5-7 分别是 8224 的逻辑框图和引线图，它由一个受晶体控制的振荡器、一个 9 分频的计数器、两个高电平驱动器和若干门电路所组成。

(1) 晶体振荡器(Quartz crystal oscillator)

8224 晶体振荡器的引线 XTAL1 和 XTAL2 外接发振晶体。外部晶体频率的选择取决于 8080 A 的运算速度。对于典型的 8080 CPU，其晶体频率为 18.432 MHz。这一频率经 9 分频变为 2.048 MHz 的两相时钟信号 ϕ_1 和 ϕ_2。ϕ_1、ϕ_2 被送到 8080 A 中去作为定时信号。

TANK 是接谐波型晶体的引线，由于这类晶体的增益较低，因此需要从 TANK 端连接一

个并联 LC 电路，以提供附加的增益。振荡器的输出经过缓冲，在 OSC 端输出晶体频率的定时信号以供系统使用。

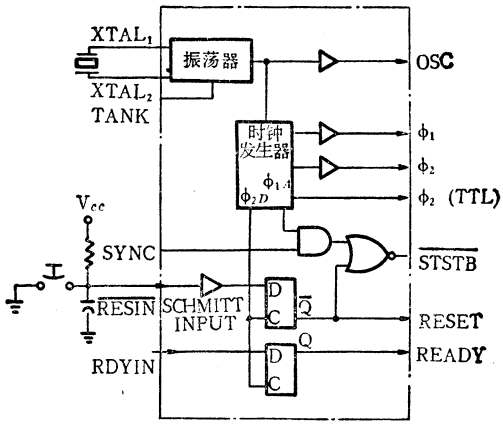


图 5-6　时钟发生器 8224 方框图

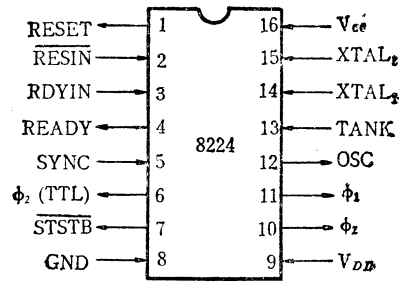
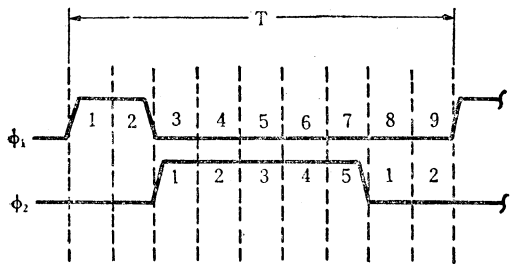


图 5-7　8224 引线图

(2) 时钟发生器(Clock Generator)

时钟发生器包括 9 分频计数器和译码器，它产生 8080 A 两个时钟脉冲 ϕ_1、ϕ_2 以及其它辅助定时信号。其波形图如图 5-8 所示。



$$1UNIT = \frac{1}{OSC\ FREQ}$$

图 5-8　ϕ_1、ϕ_2 定时波形图

时钟发生器的输出 ϕ_1、ϕ_2 通过两个高电平驱动器转换为 8080 A 所需要的 MOS 电平。为了与 TTL 电路匹配，时钟发生器另有一个 TTL 电平输出的 ϕ_2(TTL)，可供外部定时，例如用于 DMA 定时操作。

从 8080 A 来的 SYNC 信号与一个内部定时信号 ϕ_{1A} 取得同步后得到一个周期选通信号 $\overline{STSTB}$(Status Strobe)，送到系统控制器 8228，用来锁存 8080 A 的周期信息。开启电源的复位信号(RESET)也能产生 $\overline{STSTB}$ 信号。

图 5-6 中，连接到 $\overline{RESIN}$ 输入端的是一个外部 RC 网络。电源上升的缓慢变化由内部一个斯密特触发器来监视，当输入电平到达预定值时，这个电路把它的缓慢变化转换成光洁的快速边沿。斯密特触发器的输出连接到 D 型触发器，经与内部定时信号 ϕ_{2D} 同步，产生符合 8080 A 要求的复位信号，送 8080 A CPU。由图 5-6 知，还可用复位按钮将 $\overline{RESIN}$ 瞬时接地的方法使系统人工复位。

从存贮器或 I/O 设备送来的异步“等待请求”信号 RDYIN (Ready In)经与 ϕ_{2D} 同步后成为 READY 信号送到 8080 A CPU。

图 5-9 示出 8224 时钟发生器和 8080 A CPU 的连接图。

2．系统控制器(8228)

系统控制器 8228 是用于 8080 A 系统的单片系统控制器和数据总线驱动器，如图 5-10 所示。它由 8 位双向数据总线驱动器、周期信息锁存器以及门控电路阵列三部分组成。它产生与微型计算机系统相连的 RAM、ROM、I/O 接口所需要的控制信号。

(1) 8 位双向数据总线驱动器

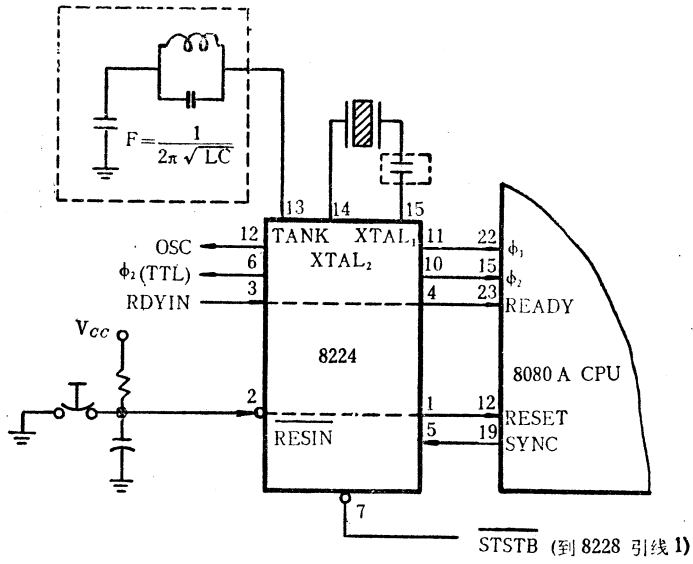


图 5-9　8224 时钟发生器及其与 8080A 的连接图

它具有足够的输出电流(10mA)，可用来增大 Intel 8080 A 数据总线的驱动能力，即它可将原低电平吸收电流 1.9 mA 放大为 10 mA，使之能连接较多的存贮器和输入/输出接口电路。

双向总线驱动器由门控电路阵列的信号来控制。在 DMA 操作时，总线驱动器的输出为高阻状态。

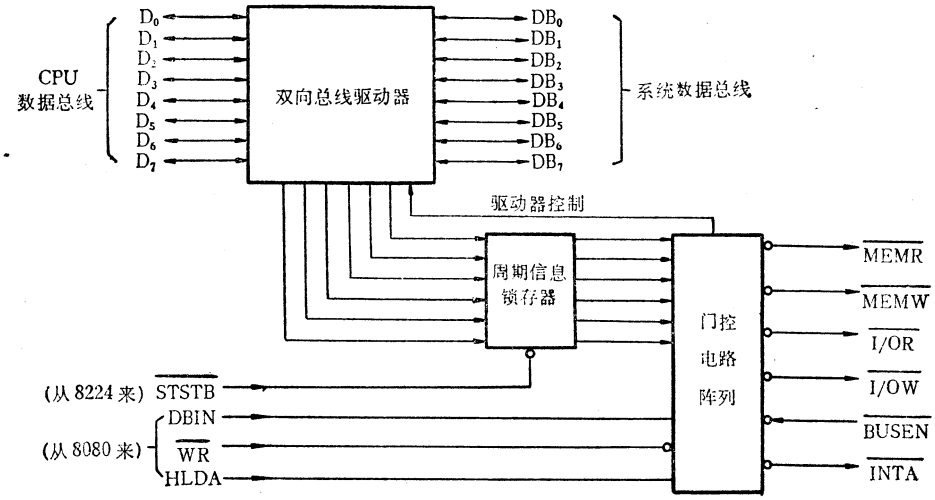


图 5-10　系统控制器 8228 方框图

(2) 周期信息锁存器

当每一机器周期的第一个时钟周期 T_1 时，8080 A 把 8 位周期信息码送到数据总线 D_7～D_0 上，指明本机器周期将执行什么操作。$\overline{STSTB}$ 是由时钟发生器 8224 输出到 8228 的周期选通信号。当 $\overline{STSTB}$ 是低电平时，把周期信息码存入到周期信息锁存器内。周期信息锁存器的输出接至门控电路阵列，作为产生控制信号的部分信息。

“读”控制信号(MEMR 存贮器读、I/O R 输入读、INTA 中断响应)由特定的周期位(一位或几位)和8080 A 的 DBIN 输入逻辑组合而得。

“写”控制信号(MEMW 存贮器写,I/O W 输出写)由特定的周期位(一位或几位) 和 8080 A 的 WR 输入逻辑组合而得。

所有的控制信号都是低电平有效,并直接送至 RAM、ROM 和 I/O 接口电路。

BUSEN (总线允许工作)输入给门控电路阵列,是一种异步输入。当它为“1”电平时,使数据总线的输出缓冲器进入高阻状态。当它为“0”电平时,就进行正常的操作。

§5-4 Intel 8085 A 微处理器

Intel 8085 A 是 Intel 公司 1977 年研制成功的第二代半高性能的 8 位微处理器。它是在 Intel 8080 A 基础上,从工艺上及逻辑上加以改进的结果,它的指令系统与 8080 A 微处理器兼容,但功能更强,速度更快,集成度更高,而且允许由三片集成电路:8085 A(CPU)、8155 (RAM)和 8355/8755(ROM/EPROM)组成最小规模的 MCS-85 微型计算机系统。

一、8085 A CPU 的结构

8085 A CPU 的结构框图如图 5-11 所示。由框图可知 8085 A CPU 内部结构由下列几部分组成:

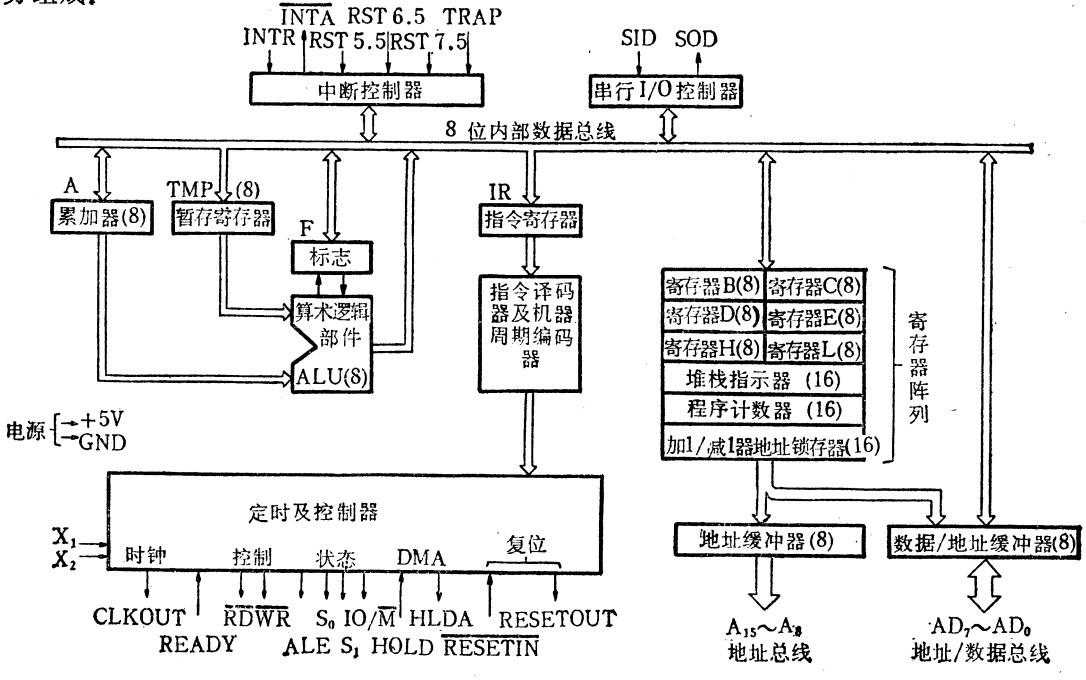


图 5-11 Intel 8085 A CPU 的结构框图

1. 寄存器阵列与地址控制逻辑

寄存器阵列的结构与 8080 A 大致相同,它包括:

(1) 程序计数器 PC(16 位)。

(2) 堆栈指示器 SP(16 位)。

(3) 通用寄存器 B、C、D、E、H、L (均为 8 位)。

(4) 两个不可编程的暂存寄存器(W、Z)。

(5) 地址控制逻辑中还有加 1 减 1 器,地址锁存器。

2. 算术逻辑运算部件(ALU)

算术逻辑运算部件包括下列部件:

(1) 一个 8 位累加器(A)。

(2) 一个 8 位暂存寄存器(TMP)。

(3) 算术逻辑部件(ALU)。

(4) 一个 8 位标志寄存器(F),只用到其中 5 位,与 8080 A 相同。

3. 控制器

它包括下列部件:

(1) 指令寄存器(IR),8 位。

(2) 指令译码及机器周期编码器。

指令译码及机器周期编码器是分析指令的部件。8085 A 实际用的译码器输出有 246 种操作电位。

(3) 定时与控制电路。

在 8085 A CPU 中,定时与控制电路包括以下几个部分:

① 定时与控制电路

8085 A 定时与控制电路包含时钟发生器和系统控制器并带有外部晶体、LC 或 RC 网络。由它产生一系列执行指令所需要的定时信号。

② 中断控制器

中断控制器用于接受外设的中断请求并发出中断响应信号以及处理 CPU 的 5 种中断(INTR, RST5.5, RST6.5, RST7.5 和 TRAP)。

中断控制器还设有中断屏蔽寄存器 (I),供片内可屏蔽中断设置中断屏蔽位使用,这是 8085 A 所特有的。

③ 串行输入/输出控制器

串行 I/O 控制器用来处理串行输入/输出数据。8080A CPU 无此控制器,而 8085 A CPU 能直接与串行输入/输出设备(如 TTY, 盒式磁带等)相接,进行串行数据传送。

4. 数据/地址缓冲器和地址缓冲器

它们用来隔离微处理器内部总线和外部总线,并提供附加的驱动能力。

二、8085A 引线功能

8085 A CPU 的引线图如图 5-12 所示。各条引线的含义和功能如下:

1. AD_7～AD_0 多重地址/数据总线(双向,三态,输入/输出)

多重地址/数据总线,在每个机器周期的第一时钟周期(T_1)时,存贮器地址(或 I/O 地址)的低 8 位出现在总线上,在第二和第三时钟周期时则成为数据总线。在保持(HOLD)和暂停(HALT)方式期间,AD_7～AD_0 处于高阻抗状态。

2. A_{15}～A_8 地址总线(单向三态输出)

它提供高 8 位存贮器地址或 8 位 I/O 设备地址。在保持(HOLD)或暂停 (HALT) 方式期间，A_{15}～A_8 处于高阻抗状态。

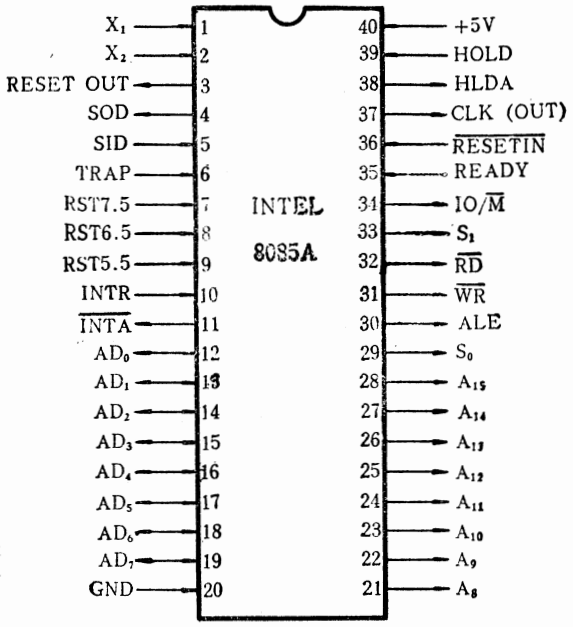


图 5-12　Intel 8085 A 引线图

3．ALE 允许地址锁存信号(Address Latch Enable)(输出)。

它在每个机器周期的第一时钟周期产生，并允许地址锁存到外部电路的地址锁存器中。ALE 的下跳沿应保证建立并保持地址信息。ALE 也能用来选通状态信息。

4．S_0、S_1 和 $IO/\overline{M}$ 机器周期状态(输出)

机器周期的编码状态如表 5-1 所示：

5．$\overline{RD}$ 读控制信号(三态，输出，低电平有效)

若 $\overline{RD}=0$，则表示从所选中的存贮器或 I/O 设备中读出数据并表示数据总线可用于数据传送。在保持(HOLD)、暂停(HALT)和复位(RESET)方式期间，$\overline{RD}$ 线为三态。

6．$\overline{WR}$ 写控制信号(三态，输出，低电平有效)

若 $\overline{WR}=0$，则表示数据总线上的数据写入到所选中的存贮器或 I/O 设备中，并在 $\overline{WR}$ 的后沿设置数据。在 HOLD、HALT 和 RESET 方式期间，$\overline{WR}$ 线为三态。

表 5-1　机器周期的编码状态

$IO/\overline{M}$	S_1	S_0	状　　态
0	0	1	存贮器写
0	1	0	存贮器读
1	0	1	I/O 写
1	1	0	I/O 读
0	1	1	取操作码
三态	0	0	暂停
三态	×	×	保持
三态	×	×	复位

注：×为任意

$IO/\overline{M}$ 表示对于存贮器或 I/O 设备进行读或写。在暂停(HALT)和保持(HOLD)方式期间，$IO/\overline{M}$ 处于三态。

S_1 也能用作扩充的 R/W 状态。

7．READY 准备就绪(输入，高电平有效)

如果在读或写周期时，READY(准备就绪)信号为高电平，则表示存贮器或 I/O 设备已准备好传送或接收数据；如果 READY 信号是低电平，则 CPU 进入等待状态，直至 READY 信号为高电平时才脱离等待状态(即从 $T_w \rightarrow T_3$)。

8．HOLD 保持(输入，高电平有效)

它表示微型机系统中另有外围设备请求使用地址总线和数据总线。CPU 接受保持请求

后，一旦完成现行机器周期，就交出对总线的控制权。这时 CPU 对不需要总线的操作仍能继续操作。只有当 HOLD 信号撤除之后，CPU 方能再度使用总线。当 HOLD 请求被响应时，地址、数据、$\overline{RD}$、$\overline{WR}$ 和 IO/$\overline{M}$ 等信号线都处于三态。

9．HLDA 保持响应(输出，高电平有效)

它表示 CPU 已接受 HOLD 请求，并将在现行机器周期完成后的下一个时钟周期交出对总线的控制权。当 HOLD 请求被撤除之后，HLDA 线变成低电平。再过半个时钟周期后，CPU 重新控制使用总线。

10．INTR 中断请求(输入，高电平有效)

CPU 在执行现行指令的最后一个时钟周期内或在 HALT 状态期间采样 INTR 输入线，以识别中断。如果该信号有效，则完成该指令后，PC 将停止计数，并发出 $\overline{INTA}$ 命令。在此期间插入 RESTART(重新启动)或 CALL(调用)指令，于是程序便转移到中断服务程序。INTR 是否得到允许是由软件来决定的。当复位或已接受一个中断时，INTR 将被禁止。INTR 与 8080 A 的 INT 的功能相同。

11．$\overline{INTA}$ (Interrupt Acknowledge)中断响应(输出，低电平有效)

CPU 在接受中断请求(INTR)之后的指令周期内，$\overline{INTA}$ 用来代替 $\overline{RD}$。它也用于启用 8259 中断控制器或其它一些中断端口。

12．RST 5.5
RST 6.5
RST 7.5 } (RESTART Interrupts)重新启动中断(输入，高电平有效)

这三个输入同 INTR 具有相同时序，不同之处在于它能自动插入一个内部 RESTART 信号。另外，这些中断的优先级比 INTR 高。它们的优先级次序如下：

RST 7.5→高优先级

RST 6.5

RST 5.5→低优先级

13．TRAP(TRAP Interrupt)不可屏蔽中断(或自陷中断)(输入)

它是一个不可屏蔽的重新启动中断，不受任何屏蔽或允许中断信号的影响，并且具有中断的最高优先级。通常用于电源掉电等严重故障情况。TRAP 输入信号在脉冲边沿和电平信号都有效时才起作用。

14．$\overline{RESET\ IN}$ 复位输入(输入，低电平有效)

它把程序计数器置零，并使允许中断和 HLDA 触发器复位。其功能与 8080 A 中的复位信号相同。

15．RESET OUT 复位输出(输出，高电平有效)

它表示 CPU 正处于复位状态，用作系统总清，该信号与 CPU 时钟脉冲同步。

16．X_1、X_2 (输入)

晶体或 RC 网络通过 X_1、X_2 连接到片内时钟脉冲发生器。X_1 也能代替晶体作为外部时钟脉冲的输入，其输入频率是片内工作频率的两倍。

17．CLK(Clock)时钟(输出)

当晶体或 RC 网络作为一个输入脉冲信号输入到 CPU 时，时钟脉冲发生器输出信号 CLK 可用作这一系统的时钟脉冲信号。CLK 的周期为 X_1、X_2 输入周期的两倍。

18．SID(Serial Input)串行输入数据线(输入)

每当执行 RIM 指令时，此线上的数据便被打入到累加器A的第7位。

19．SOD(Serial Output)串行输出数据线(输出)

它按照SIM指令来表明输出SOD是置"1"或置"0"。它可把累加器第7位的信息锁存起来，并送到SOD输出线上去。

20．Vcc +5V 电源。

21．Vss 基准地线。

三、8085 A 的主要特点

与 8080 A 相比，8085 A 有以下特点：

1．8085 A 把 8080 A CPU 模块中许多辅助功能如时钟发生、系统控制和优先中断等功能都集成在 CPU 芯片上，这是 8085 A 微处理器的主要特点。

2．8085 A 基本指令 74 条，包含了 8080 A 基本指令 72 条，而且新增加两条指令(即 RIM 和 SIM)，指令功能增强了，系统的软件与 8080 A 完全兼容。

3．8085 A 采用硅栅 NMOS 工艺，只需用单一 +5V 的电源(8080 A 需要 3 种电源：+5V，−5V 和 +12V)。此外，8085 A 采用单相时钟，工作频率 3 MHz，时钟周期为 330 ns(8080 A 的时钟周期为 500 ns)，因而提高了系统运算速度。

4．8085 A 采用多重总线结构。由于 8085 A 把 8080 A CPU 的 3 片电路：8080 A、8224 和 8228 的功能组合在一起。因此，40 条引线数量就不够用了。为此，在 16 条地址线中，把低 8 位 AD_7～AD_0 作为地址/数据多重总线，根据时间上的差别，来区分多重总线的信息是 8 位数据，还是低 8 位地址。见图 5-13 所示。8085 A 专门设置了一条允许地址锁存线 ALE，用来指明多重总线上的信息是地址而不是数据。

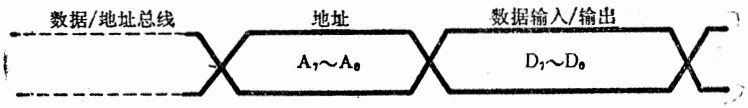


图 5-13 分时多重总线

5．增设中断控制器，加强了中断处理能力。

8085 A 有 5 条中断输入线(INTR，RST5.5，RST6.5，RST7.5 和 TRAP)。其中，INTR 和 8080 A 的 INT 相似，它用来使 CPU 接受一条由外部电路送到数据总线上的 RST 指令，并根据不同的 RST 指令转移到 8 个中断服务程序入口中的一个。这 8 个入口地址为 00 H、08 H、10H、18H、20H、28H、30H 和38H。INTR 中断输入可用允许中断(EI)指令和禁止中断(DI)指令来控制是否允许中断。另一种中断请求是 3 条中断输入线(RST5.5，RST6.5 和 RST7.5)。这类中断请求和 INTR 中断请求的区别在于，它们是用 SIM 指令来设置屏蔽的。SIM 指令是中断屏蔽设置指令。由它决定是允许还是禁止某屏蔽标志位所对应的那个中断请求。这 3 种可屏蔽中断产生 RST 的内部执行信号，把 PC 内容送入堆栈并使程序转移到重新启动地址。第三种中断是 TRAP，它是不可屏蔽的，这是级别最高的中断。在 TRAP 输入信号的上升沿，就触发 CPU 内的中断硬件电路，而不管这时是否执行过允许中断或禁止中断指令，或者是否有屏蔽标志置"1"。5 条中断输入指令的优先权级别及响应中断时转移的地址见表 5-2。

表 5-2　8085 A 中断类型和优先权级别

名　　称	优　先　权	中断后转移地址(1)
TRAP	1	24H
RST 7.5	2	3CH
RST 6.5	3	34H
RST 5.5	4	2CH
INTR	5	(2)

注：(1) 在 TRAP 和 RST 5.5～RST 7.5 的情况下，在转移以前，把 PC 内容推入堆栈。

(2) INTR 转移地址取决于 8085 A CPU 在响应中断时，外部电路或 8259 电路供给其什么指令($RST_{7\sim0}$)。

6．8085 A 增加了一个串行 I/O 控制器

为了减少规模较小的微型计算机系统中所用电路片的数目，8085 A 中增设了两条信号线，串行输入数据线 SID 和串行输出数据线 SOD。SID 和 SOD 线使 8085 A CPU 与串行 I/O 设备之间的接口大为简化。这里，可以用 RIM、SIM 指令控制串行数据的传送。每次执行 RIM 指令，都可使 SID 输入线上的信息读入到累加器中的位 7。同样，每次执行 SIM 指令，可把累加器中位 7 的信息锁存起来，并送到 SOD 输出线上去(假定这时累加器的位 6 是置"1")。由此可见 RIM 和 SIM 指令有双重的作用，它除了上述的可以管理中断的屏蔽标志位以外，还可以用来传送串行数据。

由于 SID 和 SOD 线可以按位传送信息，因此它们除了可用于串行 I/O 接口以外，还分别有下述用途：SID 可以用作通用测试(TEST)输入端；SOD 可以用作一位的控制位输出。

四、8085 A 与 8080 A 信号的对比

8085 A 和 8080 A 信号的对比，见图 5-14 和图 5-15 所示。

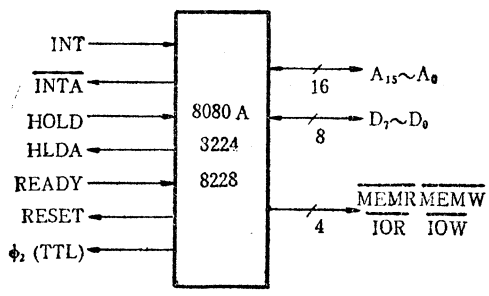


图 5-14　MCS-80 系统总线接口信号

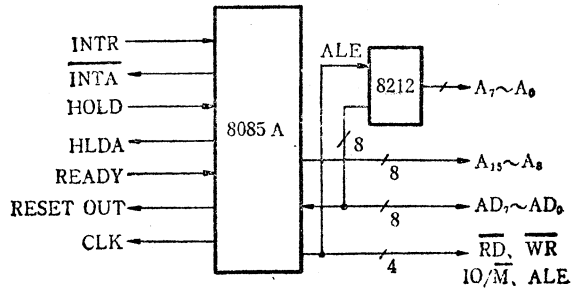


图 5-15　MCS-85 系统总线接口信号

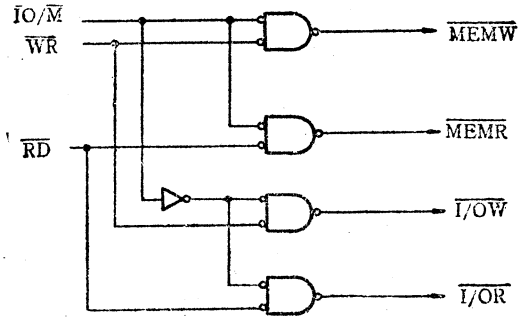


图 5-16　8085 A 产生与 8080 A 相容的存贮器和 I/O 读、写信号

从以上比较可知，MCS-85 只需要用 8212 锁存器就可以产生 MCS-80 型的总线信号。因而这两个系统是相互兼容的。

对于 8085 A，可用图 5-16 所示的电路产生 8080 A 相应的控制信号。

§5-5 Z 80 微处理器

Z 80 微处理器是美国 Zilog 公司于 1976 年在 Intel 8080 A 微处理器基础上研制成功的第二代半高性能的 8 位微处理器，而且在软件上对 8080 A 是向上兼容的。直至目前，人们仍普遍认为 Z 80 微处理器是 8 位微处理器中最好的一种系列。

一、Z 80 CPU 的结构

Z 80 CPU 的内部结构框图如图 5-17 所示。由图可知，Z 80 CPU 内部结构由下列几个部分组成：

1．寄存器阵列与地址控制逻辑

如图 5-18 所示，Z80 CPU 寄存器阵列包括 18 个 8 位寄存器、4 个 16 位寄存器和两个不可编程的 8 位暂存寄存器 W、Z，总共 224 位。所有寄存器都用静态 RAM 实现。两组各 6 个的通用寄存器可以单独用作 8 位的寄存器，也可成对地用作 16 位的寄存器。还有两组累加器和标志寄存器。

现分述如下：

(1) 程序计数器 PC(16 位)。

(2) 堆栈指示器 SP(16 位)。

(3) 两组各 6 个的通用寄存器——主寄存器和辅助寄存器(又称替换寄存器或备用寄存器)：主寄存器有 B、C、D、E、H、L 寄存器；辅助寄存器有 B′、C′、D′、E′、H′、L′ 寄存器。

(4) 两个累加器(A 和 A′)和两个标志寄存器(F 和 F′)。

(5) 两个不可编程的暂存寄存器(W、Z)。

Z80 CPU 内部的主寄存器组和辅助寄存器之间成一一对应关系。程序员只要通过两条交换指令，就可以在这两组寄存器之间相互交换内容。但要特别注意，任何时候辅助寄存器不能直接参与运算，它们必须通过 Z80 CPU 的交换指令与主寄存器对应的寄存器交换内容后，由主寄存器参加运算，运算后的结果，仍可通过交换指令存放到辅助寄存器中。

Z80 设置主寄存器组和辅助寄存器组的明显优点是：增加了 CPU 内部暂存数据的存贮容量。另外，当处理单级中断或子程序时，如需要保存寄存器的状态(也称保护现场)，可不必采用外部堆栈来保护现场，而把现场信息存放在 CPU 内部的辅助寄存器中，从而大大加快中断响应的速度。当然，对于多级中断嵌套而言，仅有两套寄存器组还不能满足其在中断处理时保护现场的要求。

(6) 两个变址寄存器 IX 和 IY(16 位)。

两个独立的变址寄存器用来存放变址寻址方式中用的 16 位基地址。变址指令中还有一个附加的字节，用来指明相对此基址的位移量。规定位移量是对 2 补码的带符号整数。例如，当用变址寻址方式执行一条指令时，两个变址寄存器(IX 和 IY)中的一个提供基地址，这个内容再加上指令中所包含的位移量 d，形成要访问的存贮单元的 16 位地址(IX + d 或 IY + d)。这

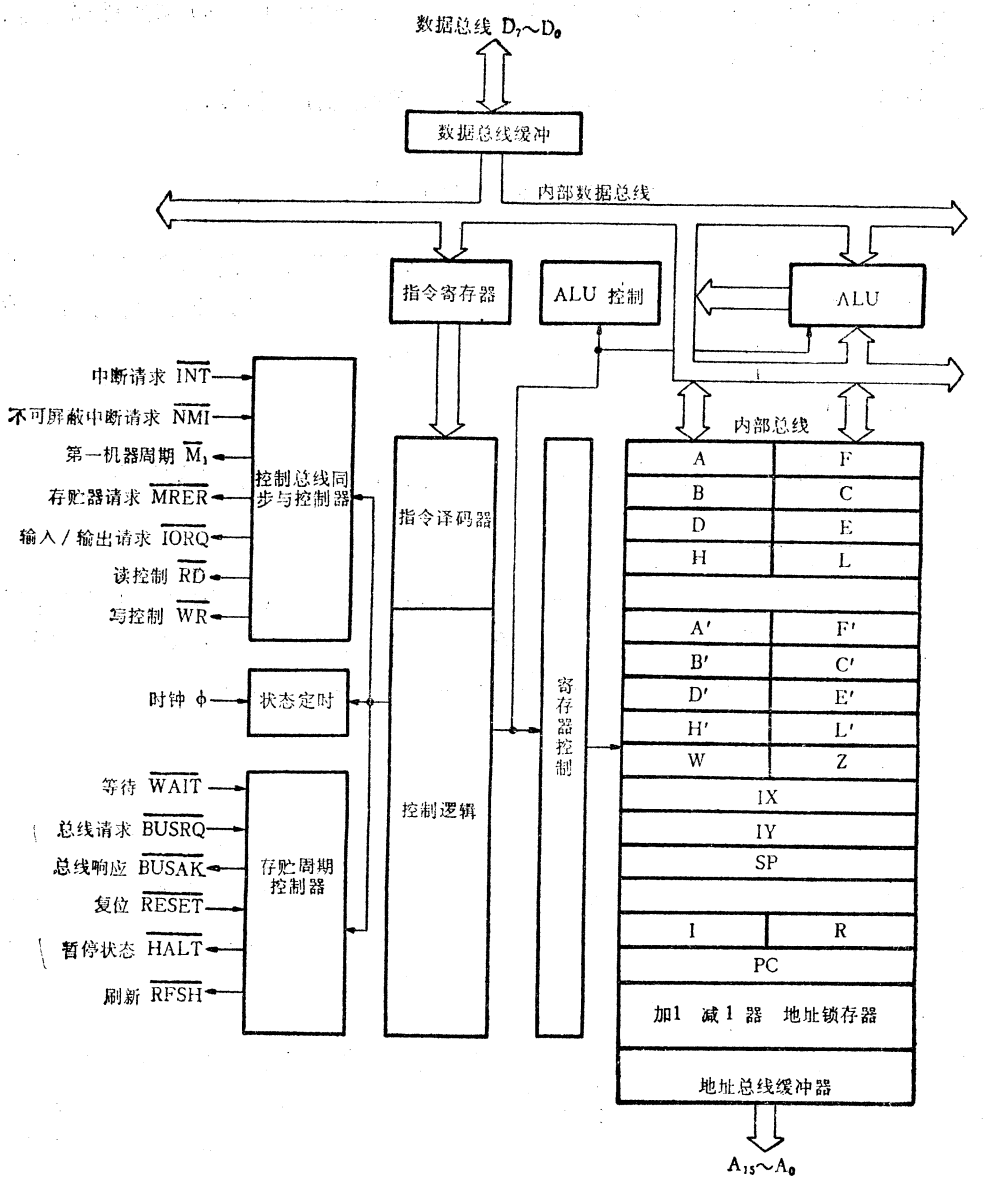


图 5-17 Z80 CPU 内部结构框图

一寻址方式大大简化了程序，特别对于处理数据表格更加方便。

(7) 中断矢量地址寄存器(又称中断页面地址寄存器 I，8 位)。

当 Z80 CPU 以方式 2 响应中断时，I 寄存器用来提供 16 位中断矢量地址指针的高 8 位地址，而指针的低 8 位地址，则由请求中断的外设提供。应当指出，Z80 CPU 中断方式 2 是以间接寻址方式找到中断服务程序的入口地址的。由中断矢量地址指针所指定存贮单元的内容，给出该中断服务程序入口地址的低 8 位；与所指定存贮单元相邻的下一个单元的内容，给出中断服务程序入口地址的高 8 位。关于中断方式 2 的工作过程，将在第十二章中详细介绍。

(8) 动态 RAM 刷新(再生)寄存器(R-Refresh，8 位)。

为了提高半导体存贮器的集成度，在大容量的存贮器中广泛采用 MOS 动态存贮器。它利

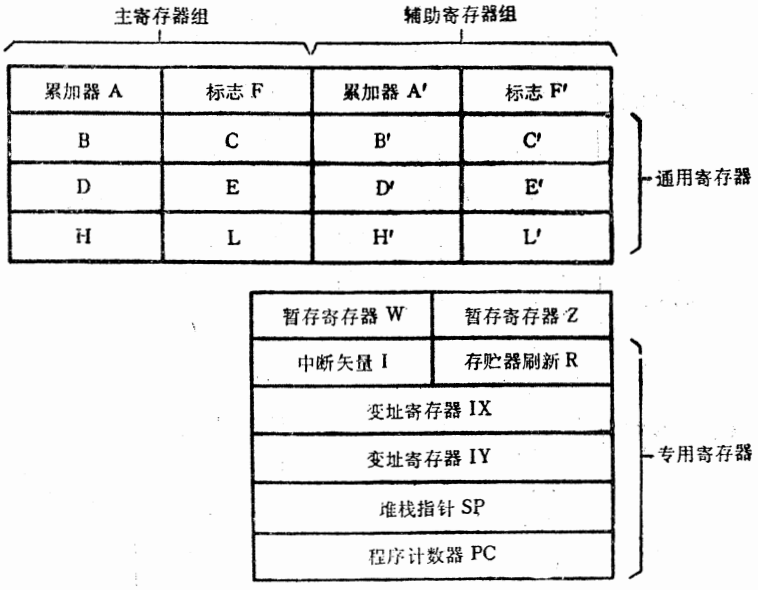


图 5-18　Z80 CPU 的寄存器结构

用寄生电容来存贮信息，为了防止泄漏信息，需要定时(一般为 2 ms)对动态 RAM 进行刷新。Z80 是利用取指周期的后两个 T 时态(此时 CPU 对指令进行译码和执行内部操作，不使用存贮器)来刷新的。因而对 CPU 的正常工作速度没有影响。每一个取指周期对一部分存贮器进行刷新，以保证在 2 ms 内对整个 64 k 内存都刷新一遍。每次刷新内存中的一行，这个行地址就由 R 寄存器(7 位)提供，而在每刷新一行后，R 的内容自动加 1，以指向下一行。

2．运算器(ALU)

运算器包括 ALU 部件、两个独立的 8 位累加器 A、A′、两个 8 位的标志寄存器 F、F′(仅用 6 位)、累加器锁存器 ACT 和暂存寄存器 TMP 等。

CPU 的 8 位运算逻辑指令是在 ALU 中执行的。ALU 通过有缓冲作用的内部总线，完成 CPU 内部寄存器和外部数据总线之间的数据交换。ALU 执行的功能有：加、减、增量、减量、逻辑与、逻辑或、逻辑异或、位置位、位复位、位测试以及算术的和逻辑的左移、右移或循环移位等。

Z80 的标志寄存器比 8080A 多设置一个 N 标志 (减法标志)和 V (溢出标志)。P 与 V 合占一位。

3．控制器

控制器包括指令寄存器、指令译码器和定时与控制电路共三个部分。

每条指令从存贮器取出后便放到指令寄存器中，随即送到指令译码器进行译码，并通过定时与控制电路在规定的时刻，产生和提供相应的控制信号，以执行所需要的操作。

二、Z80 CPU 引线功能

Z80 CPU 用工业标准 40 条引线的双列直插式管壳封装。其引线图如图 5-19 所示。

各引线的含义和功能如下所述：

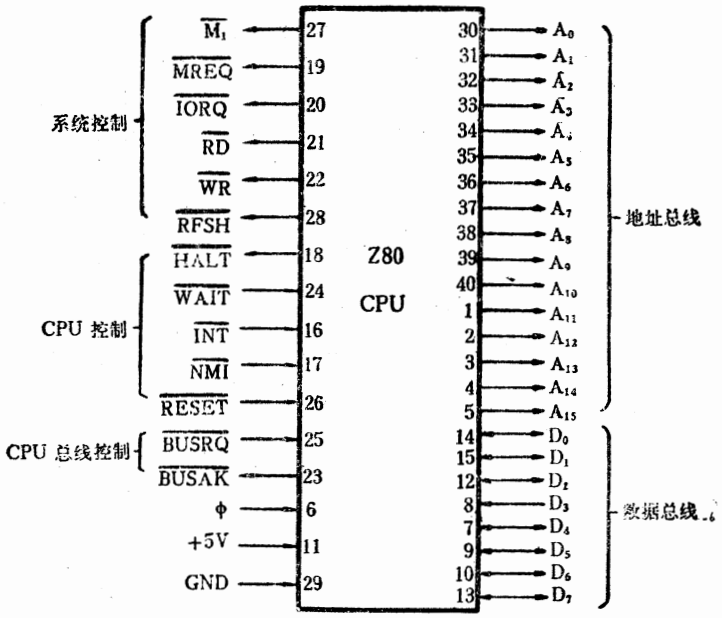


图 5-19　Z80-CPU 引线逻辑图

1．A_{15}～A_0 地址总线(单向,三态,输出)

A_{15}～A_0 组成 16 位地址总线，用于为 CPU 与内存贮器 (高达 64k 字节) 及 I/O 设备之间交换数据提供地址。当 I/O 寻址时，使用地址总线的低 8 位，用户可直接选择 256 个输入和 256 个输出端口。A_0 是最低有效地址位。在刷新期间，低 7 位地址由 R 寄存器提供，给出一个有效的刷新地址。

2．D_7～D_0 数据总线(双向,三态,输入/输出)

D_7～D_0 组成 8 位双向数据总线,用于 CPU 和存贮器或 I/O 设备之间进行数据交换。

3．$\overline{M_1}$ 机器周期 1(输出,低电平有效)

$\overline{M_1}$ 指明现行机器周期是取指令操作码周期。如果操作码有两字节,则每取一个操作码字节就产生一个 $\overline{M_1}$ 信号。这样的两字节操作码的第一字节不外乎是 CBH、DDH、EDH 或 FDH。此外，$\overline{M_1}$ 和 $\overline{IORQ}$ 同时为低电平时,还表示现行周期是中断响应周期。

4．$\overline{MREQ}$(Memory Request)存贮器请求(三态,输出,低电平有效)

这个信号指明地址总线上保持有一个供存贮器读或写操作的有效地址。

5．$\overline{IORQ}$ (I/O Request)输入输出请求(三态,输出,低电平有效)

这个信号指明地址总线的低 8 位保持有一个供 I/O 读或写操作所需的有效的端口地址。当中断得到响应时，$\overline{IORQ}$ 和 $\overline{M_1}$ 就同时产生，通知外设把中断矢量放到数据总线上。在 $\overline{M_1}$ 期间可以发生中断响应操作,但绝不会发生 I/O 操作。

6．$\overline{RD}$ 读控制信号(三态,输出,低电平有效)

$\overline{RD}$ 指明 CPU 要求从存贮器或 I/O 设备读入数据。所寻址的 I/O 设备或存贮器可应用此信号作为门控信号(即作为三态门的选通信号),将数据选通到 CPU 的数据总线上。

7．$\overline{WR}$ 写控制信号(三态,输出,低电平有效)

$\overline{WR}$ 信号指明 CPU 的数据总线上保持有待存入选定的存贮器或 I/O 设备的有效数据。

8. $\overline{RFSH}$(Refresh)再生或刷新(输出,低电平有效)

$\overline{RFSH}$ 信号指明地址总线的低 7 位包含有动态存贮器的刷新地址。这个信号与$\overline{MREQ}$信号一起用于刷新所有动态存贮器。A_7是逻辑 0，地址总线的高 8 位包含I寄存器内容。

9. $\overline{HALT}$暂停状态(输出,低电平有效)

$\overline{HALT}$ 信号指明 CPU 已经执行了 HALT 指令，处于暂停状态，直到 CPU 接受不可屏蔽或可屏蔽中断时,才重新开始操作。在暂停时期,CPU 执行空操作(NOP)指令，以保持存贮器刷新操作照常进行。

10. $\overline{WAIT}$ 等待(输入,低电平有效)

$\overline{WAIT}$ 信号给 CPU 指明所寻址的存贮器或 I/O 设备尚未为数据传送作好准备。这时 CPU 继续插入等待时态，直至此信号变为无效。此信号能使 CPU 与慢速的存贮器和 I/O 设备同步。

11. $\overline{INT}$ 中断请求(输入,低电平有效)

此中断请求信号来自 I/O 设备。如果由内部软件控制的中断允许触发器(IFF)处于允许中断状态，并且总线请求信号 $\overline{BUSRQ}$ 无效时,则在现行指令执行结束时响应此中断请求。当 CPU 接受中断时，在下一指令周期的开始，将发出中断响应信号 ($\overline{M_1}$ 和 $\overline{IORQ}$ 信号同时为低电平)。CPU 可以三种不同方式响应中断,将在第十二章中介绍。

12. $\overline{NMI}$ (Non-Maskable Interrupt)不可屏蔽中断(输入,下降沿触发)

不可屏蔽中断请求线比 $\overline{INT}$ 具有更高的优先权，只要无 $\overline{BUSRQ}$ ($\overline{BUSRQ}$ 优先权高于 $\overline{NMI}$)，不论 IFF 触发器的状态如何，它永远在现行指令执行完毕时得到响应。$\overline{NMI}$ 信号自动迫使 Z80 CPU 转向 0066H 存贮单元开始执行中断服务程序。此时,程序计数器 PC 的内容自动保存到外部堆栈中,以使中断服务结束后能返回到中断前的程序。

13. $\overline{RESET}$ 复位(输入,低电平有效)

它迫使程序计数器和指令寄存器清零,并且使 CPU 初始化。CPU 的初始化包括:

(1) 使中断允许触发器(IFF)复位(即禁止中断);

(2) 置 I 寄存器为 00H;

(3) 置 R 寄存器为 00H;

(4) 置 CPU 于中断方式"0"。

在复位期间,地址总线和数据总线都进入高阻抗状态，且所有控制信号都处于无效状态。此时停止刷新操作。

14. $\overline{BUSRQ}$(Bus Request)总线请求(输入,低电平有效)

这个总线请求信号用于要求 CPU 把地址总线、数据总线和三态的输出控制信号转为高阻状态,以便其它设备能控制这些总线。这主要用于快速的存贮器直接存取方式 (DMA 方式)。当 $\overline{BUSRQ}$ 有效时,CPU 在现行机器周期结束时,将使所有三态总线转入高阻抗状态。

15. $\overline{BUSAK}$ (Bus Acknowledge) 总线响应(输出,低电平有效)

这个总线响应信号向提出总线请求的设备指明 CPU 地址总线、数据总线和三态的控制总线已处于高阻抗状态。此时外部设备可以控制使用这些总线。

16. ϕ 单相 TTL 时钟(输入)

时钟频率:Z80 2.5MHz; Z80A 4MHz; Z80B 6MHz; Z80H 8MHz。

17．电源：$V_{CC}=5V$，V_{SS}=地线。

三、Z80 CPU 的主要特点

1．Z80 设置两组通用寄存器、两个累加器和两个标志寄存器。当一组处于工作状态时，另一组则处于备用状态，并可由交换指令进行快速交换信息。这样，不仅扩充了内部寄存器的容量，而且对单级中断可以实现快速响应。

2．增加了两个 16 位变址寄存器 IX 和 IY，使 Z80 具有比 8080 A 更加灵活的多样化的寻址方式。

3．Z80 芯片集成度同 Intel 8085A 一样，把 8080A CPU 模块中三片 LSI 电路的大部分功能都集成在单片 Z80μP 上，而且还扩大了其它的功能。

4．Z80A 采用单电源(+5)供电，又采用单相时钟，工作频率可提高到 4～8MHz。这些特点使 Z80 具有明显的速度优势。

5．Z80 和 8080A 在软件方面是兼容的。Z80 的基本指令系统不仅包括了 8080A 的全部 72 条指令，而且又增加了成组传送、成块搜索，位操作和双字长处理等指令及其它的寻址方式，指令数大大增加(基本指令 150 条)，因而处理能力大大增强，程序长度大为缩短。

6．Z80 为动态存贮器设有自动刷新(或再生)的逻辑电路，使动态 RAM 能与系统直接连接，而不需要外加刷新电路。而在 8080A/8085 A 系统中，如果要用动态 RAM，则需要另外加刷新控制电路来执行必要的刷新操作。

7．Z80 比 8080A 增加了一条中断输入线，即它除了有一般的中断请求($\overline{INT}$)输入外，还有一条不可屏蔽中断请求输入线($\overline{NMI}$)。因而，在中断处理能力方面比 8080A 系统强得多。关于 Z80 的中断系统将在第十二章详细介绍。

四、Z80 与 8080A 信号的对比

在使用 Z80 代替 8080A 时应注意，尽管 Z80 的指令系统对于 8080A 是向上兼容的（仅两者的助记符不同），然而，两者的信号线不能兼容。对于 Z80，可用图5-20电路产生与 8080A 相应的控制信号。

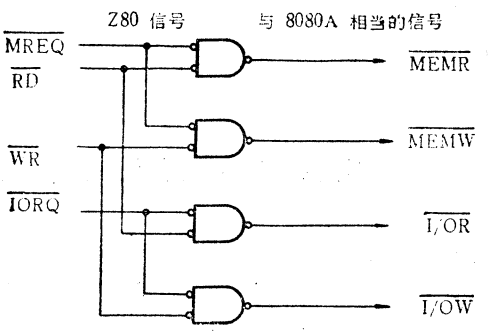


图 5-20　Z80 产生与8080A 相应的存贮器和 I/O 读写信号

Z80 没有专门的中断响应信号，这只要把 $\overline{IORQ}$ 和 $\overline{M_1}$ 组合起来就可以获得，见图 5-21。

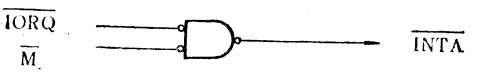


图 5-21　Z80 产生与 8080A 相应的中断响应信号

8080A 的 HOLD(请求保持)和 HLDA(保持响应)相当于 Z80 的 $\overline{BUSRQ}$ 和 $\overline{BUSAK}$信号。

Z80 中断请求 $\overline{INT}$ 类似于 8080A 的 INT 线，区别仅在于 Z80 产生中断请求是低电平有效。

Z80 的 $\overline{WAIT}$ 请求线类似于 8080A 中的 READY 输入线。

在 8080A CPU 中，8228 的 $\overline{BUSEN}$ 输入是给外部逻辑用来“浮空”系统总线的。而在

Z80 系统中，系统总线是由 CPU 自己浮空的。

Z80 实际上并没有真正的 HALT 状态。在执行 HALT 指令时，Z80 输出 $\overline{\text{HALT}}$ 变为低电平，然后就连续执行 NOP 指令，以保持动态 RAM 刷新逻辑继续工作。只要有不可屏蔽中断请求或屏蔽中断请求(而且允许中断触发器已置"1")，则 CPU 就在下一个时钟周期的前沿退出 HALT 状态，而进入一个中断响应周期。

§5-6 Motorola 6800 微处理器

MC6800 是 Motorola 公司于 1974 年研制成的一种第二代 8 位微处理器。它的基本指令 72 条，寻址方式 6 种，指令操作总共 197 种，中断功能 4 级。它采用单电源(+5V)。所有 MC 6800 系列的电路都可以直接与其 CPU 的地址和数据总线相接，这些总线能驱动一个 TTL 负载。它具有直接存贮器存取(DMA)的能力，并有暂停功能。时钟脉冲是两相(主频为 1 兆赫)非重迭的，它由外部加入。

一、MC6800 CPU 的结构

MC6800 的内部结构见图 5-22。

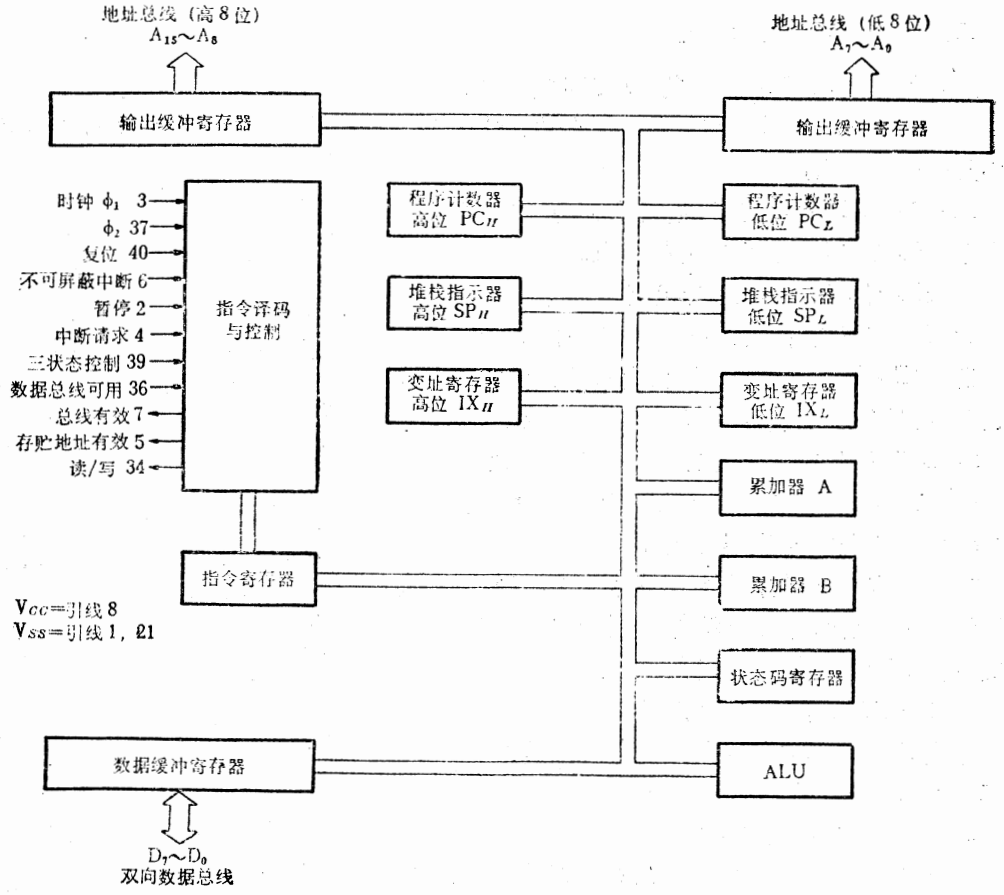


图 5-22 MC6800 内部结构框图

为了与外界交换数据，CPU 通过数据缓冲器与双向数据总线 D_7～D_0 相连。地址是通过输出缓冲器送到 16 位地址总线(A_{15}～A_0)上的。由于这两种总线与 CPU 相连的缓冲器都是三态的，故实现 DMA 操作比较方便。在 MC6800 中，存贮器地址和 I/O 设备地址是编在一起的，即采用存贮器映象 I/O 寻址结构。

MC6800 CPU 同样包括寄存器阵列、运算器和控制器三大部分，现分述如下：

1．寄存器阵列

(1) 程序计数器 PC(16 位)。

(2) 堆栈指示器 SP(16 位)。

(3) 两个累加器 A、B(8 位)。

(4) 条件码寄存器(即标志寄存器，8 位，仅用 6 位)。

它们是：符号位(N)、全零位(Z)、溢出位(V)、进位位(C)、半进位位(H)、中断屏蔽位(I)。

(5) 变址寄存器 IX(16 位)。

(6) 暂存寄存器(8 位)。

(7) 加 1 减 1 器(16 位)。

2．算术逻辑部件 ALU(8 位)

3．指令译码与控制逻辑，包括中断与再启动逻辑、定时发生器、总线控制与停机逻辑等。

二、MC6800 的引线功能

图 5-23 示出了 MC6800 的 40 条引线信号。

图中 A_{15}～A_0 地址总线是单向三态的，CPU 通过它们输出地址到存贮器或 I/O 接口，以选择要操作的单元。

D_7～D_0 是 8 位双向三态数据总线，用来实现CPU与内存贮器和 I/O 设备之间交换信息。

MC6800 的控制总线中的控制信号有：

1．TSC 三状态控制线(输入，高电平有效)

当 TSC = 1 时，地址总线和读写控制线处于高阻抗状态；当 TSC = 0 时，则两者都为工作状态。

2．DBE 数据总线可用(输入，高电平有效)

当 DBE = 1 时，数据总线处于工作状态；当 DBE = 0 时，数据总线处于高阻抗状态。

TSC 和 DBE 两条控制线提供了使 CPU 瞬时脱离总线的方法，它们可用于直接存贮器存取操作。一般情况下，DBE 可与时钟 ϕ_2 连在一起。

3．$\overline{\text{HALT}}$ 暂停(输入，低电平有效)

它是外部使 CPU 暂停执行的信号。若 $\overline{\text{HALT}}$ = 1 时，则执行程序；反之则处于停止或空转状态。

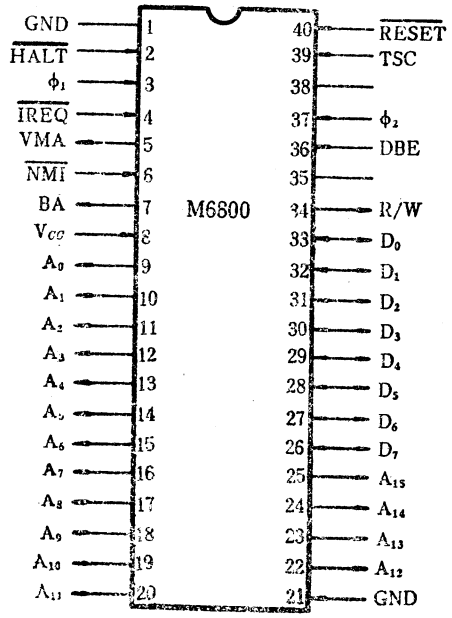


图 5-23 MC6800 引线图

4．BA 总线有效(输出)

这个信号提供了一个 CPU 现行状态的标志。当 BA = 0 时，表明 CPU 正在使用数据总线和地址总线；当 CPU 处于暂停状态或中断等待状态时，它输出高电平，即 BA = 1。此时系统

总线可作为DMA或其它应用。

5．R/W 读写控制线和 VMA(存贮地址有效)

R/W 用于控制数据总线上数据传输的方向，它连到存贮器和 I/O 接口。当 R/W = 1 时，表示读出；当 R/W = 0 时，则表示写入。TSC 线变高或 CPU 暂停时，R/W 线便处于高阻抗状态。

VMA 输出线为高电平时，表明 CPU 正在执行读或写操作，即表示输出地址有效。它常用来控制地址译码器，以决定译码器是否有输出。

6．$\overline{\text{IREQ}}$ 中断请求(输入，低电平有效)

它是一条外部中断请求线，当外部有中断请求时，CPU 要根据条件码寄存器中 I 的状态来决定是否响应。仅当 I = 0 时，CPU 才能响应外部的中断请求。

7．$\overline{\text{NMI}}$ 不可屏蔽中断(输入，低电平有效)

这个信号不同于 $\overline{\text{IREQ}}$，一旦 $\overline{\text{NMI}}$ 有效，不管条件码寄存器 I 位的状态如何，只要等正在执行的那条指令执行完毕，它就要强迫 CPU 中断正在执行的程序，从而转入不可屏蔽中断服务程序。因此它是一种不能禁止的中断请求，一般用于掉电保护等紧急事故。

8．$\overline{\text{RESET}}$ 复位(输入，低电平有效)

这是用外部电路使 CPU 复位或重新启动的输入控制线。当 CPU 接受复位信号后，就回到初始状态，从某一特定的地址开始执行程序。

§5-7 几种微处理器的总线接口信号之比较

微处理器是通过系统总线和存贮器、I/O 接口电路连接组成微型计算机的。因而，深入理解微处理器的总线接口信号是正确设计微型计算机应用系统的必要前提。为了进一步加深印象，本节将上述 4 种微处理器的总线接口信号进行分类比较。

图 5-24 至图 5-27 示出了 Intel 8080A、8085A、Z80 和 MC6800 4 种典型微处理器的总线接口信号图。表 5-3 对它们进行分类比较。

下面我们对表 5-3 作扼要的说明。

1．时钟

目前大多数微处理器用外部时钟进行驱动。如 Z80 需要一个单相的 TTL 电平的脉冲源；MC6800 需要双相非重迭的 TTL 电平的脉冲源；8080A 需要双相非重迭、幅度为 12 伏的脉冲源。

Intel 8085 具有内部时钟电路，只须外接一个晶体来决定时钟的频率。

2．地址和数据总线

上述四种 8 位微处理器的地址总线都为 16 位，数据总线都为 8 位。其中 8085 采用多重总线结构，即把低 8 位地址总线和数据总线合在一起分时使用，根据时间上的差别来区分。为此，8085A 专门设置一条允许地址锁存线 ALE，用以指明多重总线上 $AD_7 \sim AD_0$ 的信息是地址而不是数据。

8080A 的数据总线 ($D_7 \sim D_0$) 在每个机器周期的第一个时钟周期 (T_1) 发出一组 8 位的周期信息码，这组周期信息码是在 SYNC 信号作用期间送来的，它用来判明现行机器周期的类型，并被锁存至外部锁存器，以便产生对外部电路的控制信号。

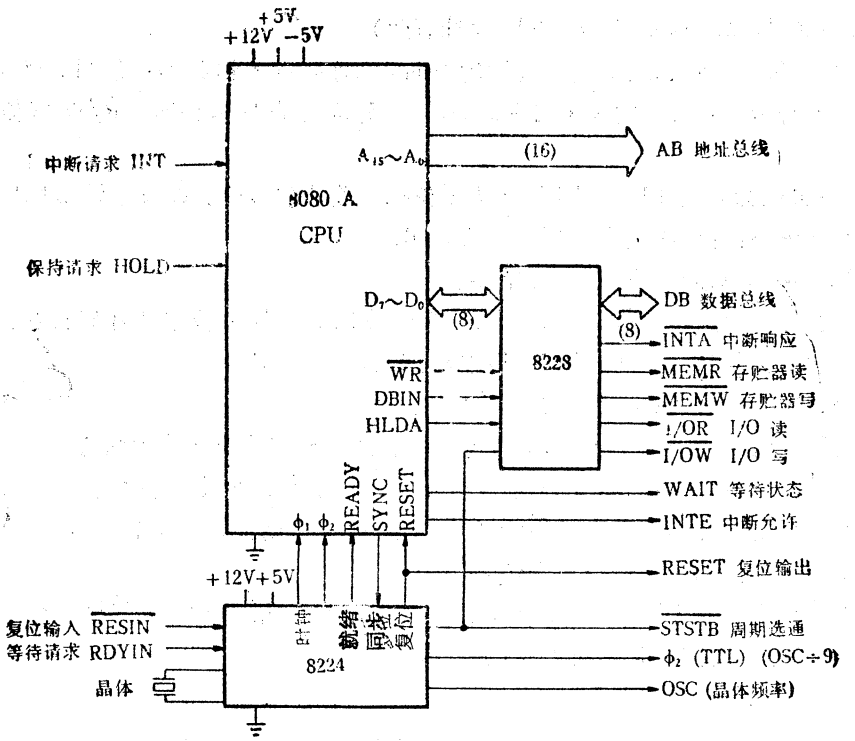


图 5-24　8080A 的总线接口信号

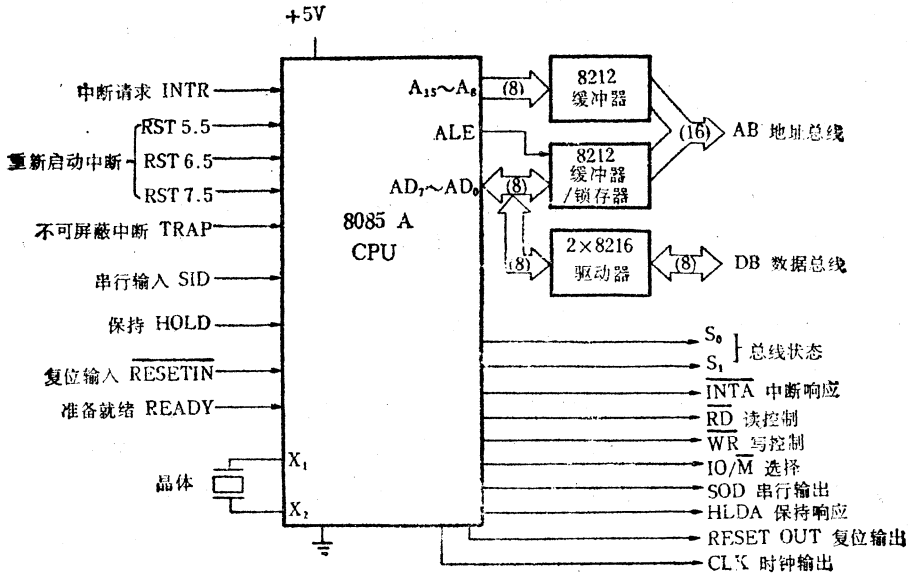


图 5-25　8085A 的总线接口信号

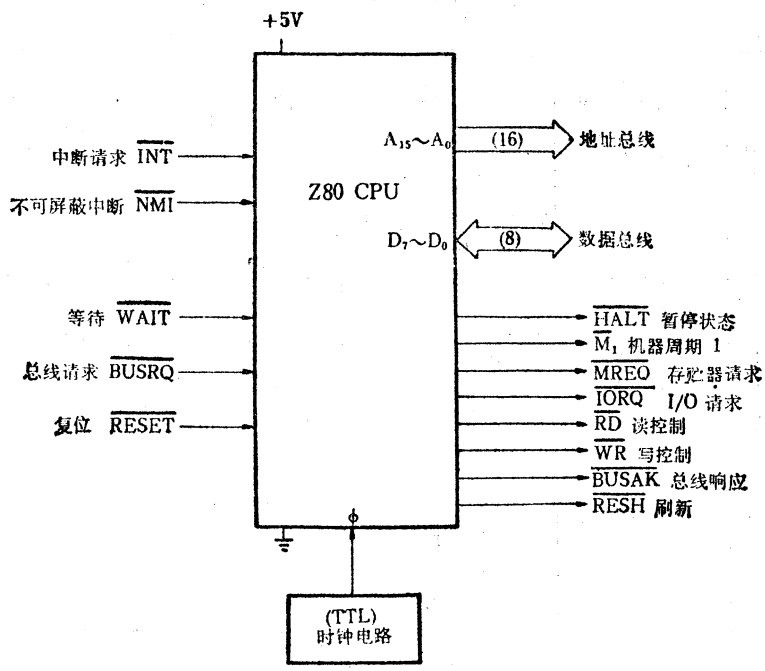


图 5-26　Z80 的总线接口信号

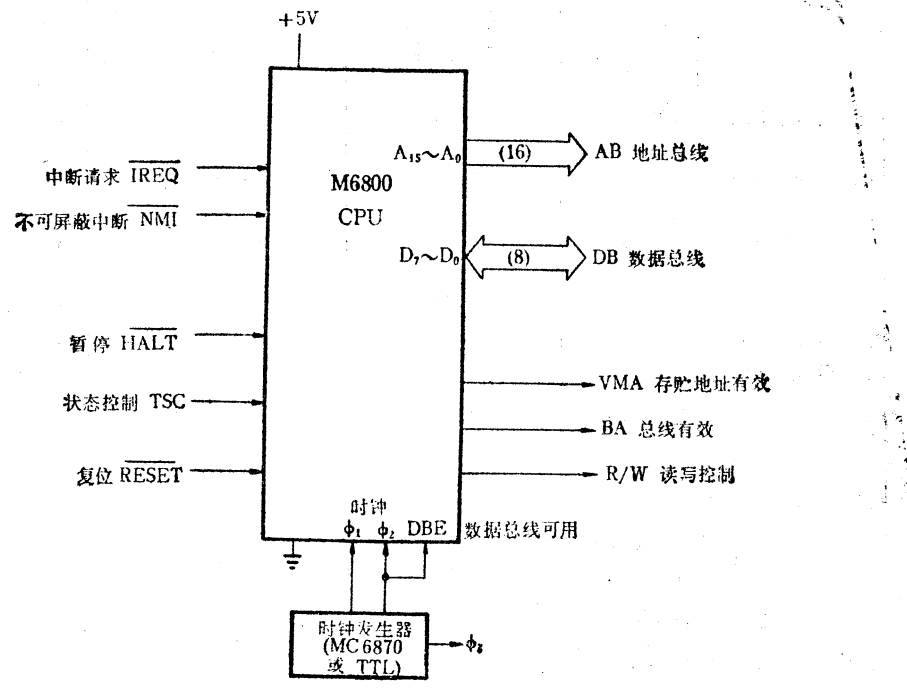


图 5-27　MC6800 的总线接口信号

3．复位

它是确定微处理器初始状态的信号。当复位信号(RESET)有效时，迫使程序计数器 PC 置“0”，并使 CPU 置于初始状态。它包括复位中断允许触发器，禁止接受中断请求。此时地址

表 5-3　微处理器总线接口信号比较

项目		微处理器				
		Intel 8080A*	Intel 8085A	Z80	MC6800	说明
总线	地址	A_{15}～A_0	A_{15}～A_8	A_{15}～A_0	A_{15}～A_0	地址总线(单向、三态)
	地址/数据		AD_7～AD_0			地址/数据多重总线
			(T_1-锁存低8位地址)			(分时使用)
			ALE			地址允许锁存
	数据	D_7～D_0		D_7～D_0	D_7～D_0	数据总线(双向、三态)
		(T_1-周期信息)				
控制输入	复位	$\overline{RESIN}$	$\overline{RESET\ IN}$	$\overline{RESET}$	RESET	异步复位
	准备就绪	RDYIN	READY	$\overline{WAIT}$		CPU 进入等待状态
	总线控制	HOLD	HOLD	$\overline{BUSRQ}$	TSC	使系统总线脱离 CPU 控制
					HALT	CPU 暂停
					DBE	数据总线有效
	中断请求	INT	INTR	INT	$\overline{IREQ}$	一般的中断请求
			RST 5.5			重新启动中断
			RST 6.5			
			RST 7.5			
			TRAP	NMI	NMI	不可屏蔽中断
	特殊信号		SID			串行输入数据
控制输出	数据传送控制	$\overline{MEMR}$ $\overline{MEMW}$ $\overline{I/O\ R}$ $\overline{I/O\ W}$	$\overline{RD}$ $\overline{WR}$ IO/$\overline{M}$	$\overline{RD}$ $\overline{WR}$ $\overline{MREQ}$ $\overline{IORQ}$	R/W VMA	这组输出信号经组合后，允许信息沿着数据总线在规定的单元和设备之间定时传送。
	中断响应	$\overline{INTA}$	$\overline{INTA}$	M_1 $\overline{IORQ}$		从数据总线读取中断指令用的定时信号
	CPU 状态	WAIT	SOD	$\overline{HALT}$		见控制信号引线图
		INTE				
	总线状态	HLDA	HLDA	$\overline{BUSAK}$	BA	总线响应、使系统总线脱离 CPU 控制
			S1 S0			总线周期编码
	同步输出信号	RESET	RESETOUT			供系统使用
	特殊信号			$\overline{RFSH}$		动态 RAM 的刷新定时脉冲
定时	时钟	ϕ_1 ϕ_2	单相	ϕ单相(TTL)	ϕ_1 ϕ_2	
		双相非重迭			双相非重迭	
		(由 8224 提供)	X_1, X_2			
	时钟输出	ϕ_2(TTL)	CLK			
		OSC(晶体频率)				
	主频	2MHz	3MHz	(2.5,4)MHz	1MHz	
电源		+5V、−5V、+12V	+5V	+5V	+5V	

注：8080A*=8080A+8224+8228（见图 5-24）

总线和数据总线都处于高阻抗状态。

4．准备就绪和等待状态

设置这两条控制信号线是为了使微处理器与慢速的存贮器或 I/O 设备同步工作。Intel 8080A/8085 的准备就绪信号 READY，Z80 的等待请求，$\overline{WAIT}$，它们的作用都是用来指明所寻址的存贮器或 I/O 设备是否准备就绪。

5．总线控制和总线状态

8080A/8085A 的保持请求信号 HOLD 和保持响应信号 HLDA 与 Z80 的总线请求$\overline{\text{BUSRQ}}$和总线响应 $\overline{\text{BUSAK}}$ 类似，它们的作用都是为了使微型计算机具有直接存贮器存取(DMA)操作的功能。区别仅在于有效电平不同而已。

MC6800 用三态控制线 TSC，数据总线有效 DBE、和暂停 $\overline{\text{HALT}}$ 进行总线控制，而用总线有效 BA 来表明总线状态。

6．中断控制

中断有三种类型：一般中断、重新启动中断和不可屏蔽中断。8080A 只有一般中断 INT；Z80 和 MC6800 具有一般中断 $\overline{\text{INT}}$(或 $\overline{\text{IREQ}}$)和不可屏蔽中断 $\overline{\text{NMI}}$；8085A 则兼有三种类型的中断，即一般中断 INTR、重新启动中断 RST 5.5、RST 6.5、RST 7.5 和不可屏蔽中断 TRAP。

8080A 的 INT、8085A 的 INTR 和 Z80 的 $\overline{\text{INT}}$ 都用作一般中断，区别仅在于有效电平不同而已。一般中断是用允许中断(EI)指令和禁止中断(DI)指令来控制是否允许中断的。8085A 的重新启动中断则是用 S M 指令来屏蔽的。它们都属于可屏蔽中断。8085A 可屏蔽中断的优先级由高到低的次序是：RST 7.5, RST 6.5, RST 5.5, INTR。

不可屏蔽中断是级别最高的中断，它不受软件控制，通常用来处理掉电等紧急事故。

Z80 还有 3 条设置中断方式的 IM 指令，它们用来指明能响应的中断是可屏蔽中断请求的方式 0 或方式 1 或方式 2 。

与上述 8080A/8085A 和 Z80 相比，MC6800 处理中断时，CPU 内部寄存器的保护(入栈)或恢复(出栈)，都是由片内硬件自动完成的，这是 MC6800 的一个优点。

中断处理的内容很丰富，将放在第十二章专题讨论。

7．数据传送控制

这组输出信号用来控制 CPU 和存贮单元(或 I/O 端口)之间的数据交换。8080A 的读控制信号($\overline{\text{MEMR}}$ 存贮器读，$\overline{\text{I/OR}}$ 输入读)是由特定的周期信息位和总线允许输入控制信号 DBIN 逻辑组合而得；写控制信号($\overline{\text{MEWR}}$ 存贮器写，I/OW 输出写)是由特定的周期信息和 $\overline{\text{WR}}$ 逻辑组合而得。8085A 的读写控制信号是通过选择信号 IO/$\overline{\text{M}}$ 和 $\overline{\text{RD}}$、$\overline{\text{WR}}$ 逻辑组合而得。Z80 的读写控制信号是通过存贮器请求 $\overline{\text{MREQ}}$、输入/输出请求 $\overline{\text{IORQ}}$ 和 $\overline{\text{RD}}$、$\overline{\text{WR}}$ 逻辑组合而得。只要用简单的逻辑门电路就可以将 8085A 和 Z 80 的一组读写控制信号转换为与 8080A 相兼容的读写控制信号。MC6800 的读写控制信号由读/写控制线(R/W)和存贮地址有效(VMA)逻辑组合而得。

8．特殊信号

8 85A CPU 增加了一个串行控制器，有两条用于串行输入(SID)和串行输出 (SOD)的数据线，使 8085A CPU 能直接与串行输入/输出设备进行串行数据传送，这是8080A、Z80 和 MC 6800 CPU 所不具备的。

Z80 CPU 为动态存贮器设置有自动刷新(再生)电路，使动态 RAM 能与系统直接连接，而不必外加刷新电路。

微处理器通过总线接口信号控制和管理整个微型计算机系统有条不紊地工作，因此，详细地了解微处理器总线接口信号的功能，以便和存贮器、I/O 接口电路以及其它逻辑部件组成一个完整的微型计算机系统，应当视为本章的重点之一。

第六章　微处理器的指令系统

微型计算机的功能是从外部接受数据，在CPU中处理数据，然后把处理结果送到外部去。设计一台计算机，首先需要提供具有执行一套特定操作的命令，称之为指令。CPU所能执行的各种指令的集合，称之为该微处理器的指令系统。

按人们的要求把一系列指令组成一个连贯的程序，程序员就能通过该程序去指挥微处理器执行所要求的任务。因此，现代设计一种微处理器，往往从设计它的指令系统开始。

微处理器只能直接识别和执行以二进制代码形式表示的指令，这种代码称为机器代码。为了便于编程，人们采用了各种形式的程序设计语言，它们可以通过计算机翻译成为机器代码。当前，微型计算机主要还是应用汇编语言来编写程序。这就要求使用者认真地去掌握该微处理器的指令系统。对8位微处理器来说，其基本的指令系统是大致相同的。

本章将讨论指令的基本格式，指令的寻址方式，指令系统的分类以及条件标志，并以Intel 8080A/8085A和Z80为实例具体分析微处理器的指令系统。

§6-1　指令的基本格式

微处理器接受形式相同的指令和数据，两者都以二进制数的形式存于存贮器中，并通过数据总线送往CPU。因此，微处理器把指令也当作与数据的字长相同的字加以处理。它们之间唯一的区别是微处理器把指令送往指令寄存器和指令译码器，而把数据送往数据寄存器和算术逻辑部件(ALU)。

对于一条指令来说，它应该说明以下几个问题：

1．操作码：表示这一条指令执行什么性质和类型的操作，例如传送、加、减等等。

2．操作数的来源：指令中有一部分数值用来说明操作数存放的地址，称之为地址码。例如加法指令，有被加数和加数两个操作数。指令必须指明这两个操作数的地址。然后，CPU才能取数进行操作。对于移位、取反等只有一个操作数的指令只要指明这一操作数的地址。

3．操作结果存放的地址：例如，两数相加后的结果存放的地址。

4．下一条指令存放的地址。

因此，一条完整的指令字应包括5个字段：操作码字段和4个地址字段(它们分别存放两个操作数的地址码、操作结果的地址码、下一条指令的地址码)，其格式如下所示：

操作码	第一操作数的地址	第二操作数的地址	结果的地　址	下一条指令的地址

假设地址码为16位二进制数，则上述格式中的4个地址共需64位。如果有256种操作，则操作码应有8位($2^8=256$)。这样的指令共需72位。而一般的微处理器的字长仅为8位或16位。为此，必须设法缩短指令的长度，其方法如下：

1．指令按顺序排列，由程序计数器(PC)指出下一条指令的地址，每取指令的一个字节，

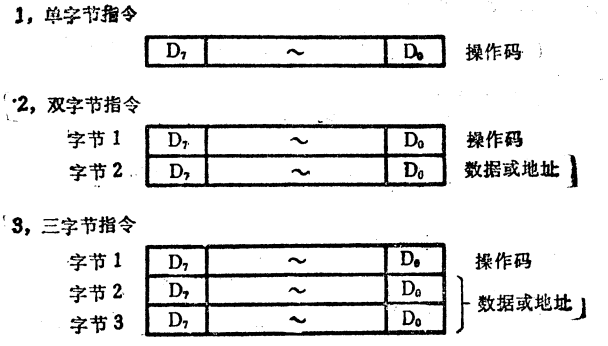
PC就自动加1，这样在指令中就不必指明下一条指令的地址。如果遇到程序需要转移的情况，则用转移指令来指出下一条指令的地址。

2．约定操作数事先存放在某寄存器中，只要知道寄存器的地址即可寻到操作数。或者把操作数的地址存在寄存器中，只要知道寄存器的地址，即可找到操作数的地址。

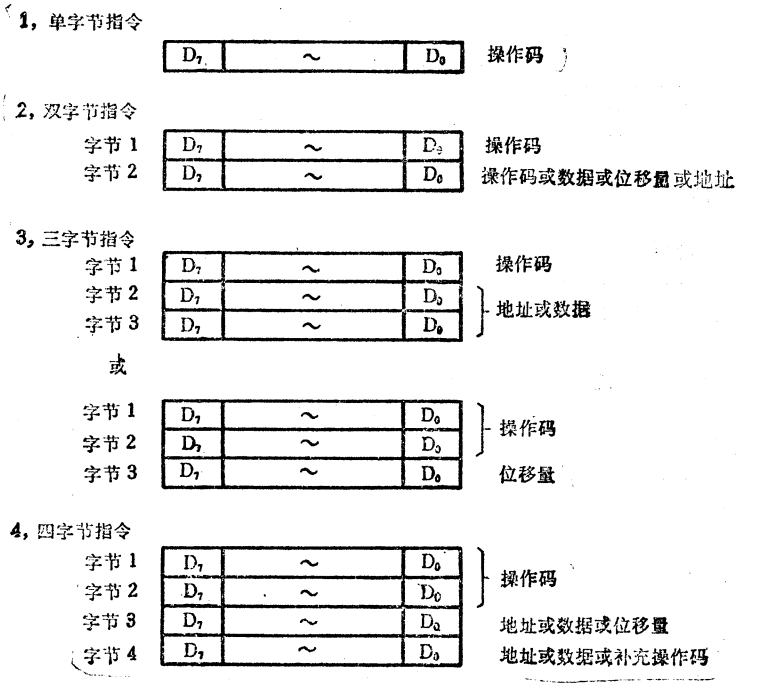
由于CPU仅有少数几个寄存器，因此所需要的寄存器的地址码位数很少。例如，只要用三位二进制数即可表示8个寄存器的地址。

3．约定结果数的地址和一个操作数的地址相同。例如，先把一个被加数存于累加器中，当它和另一个加数相加后，其结果仍放在累加器之中。这样就不必在指令中指明结果的地址。当然，这时累加器原来的内容将被相加的结果所取代。一般来说，这是程序设计所允许的。事实上，对算法的分析表明，大部分中间结果并不需要保存。

4．把指令分段。在微处理器中，把一条长指令分解成二段或三段来存放和处理，每一段



(a) 8080A/8085A指令的格式



(b) Z80指令的格式

图6-1 指令的基本格式

长度与微处理器的字长相匹配。如果微处理器的存贮器容量有 64 K，则相应的地址码为 16 位，若加上操作码，则单地址指令长度就可能达到 24 位。这样，8 位字长的微处理器的指令可分成 3 段。有些 I/O 指令需要指出 256 种 I/O 设备，其地址码为 8 位，全指令字长为 16 位。此时，其指令可分成二段。

综上所述，按照指令长度的分段，微处理器的指令格式又可分成单字节、双字节和三字节指令。例如，典型的 8 位微处理器如 Intel 8080 A/8085 A 指令的基本格式有单字节、双字节和三字节等三种。Z 80 指令中还有四字节指令格式。多字节指令必须顺序地存放在存贮单元里，首先存放低位字节，接着存放高位字节。它们的格式如图 6-1 所示。

不论哪种格式，第一个字节总是操作码，它决定了微处理器执行操作的类型。单字节指令的操作码本身就隐含着操作数在微处理器内部的某个寄存器中。双字节指令的第二字节以及三字节指令的第二、三字节或是直接给出操作数、或是给出操作数所在的地址。显然，指令越短，执行指令的速度越快。指令的具体格式取决于指令的操作内容。通常，指令用汇编符号(即助记符)来书写。例如，LD r_1，r_2 是单字节的传送指令，ADD A，n 是双字节的加法指令，而 LD HL，(nn)则是三字节的取数指令。

采用了上述缩短指令的方法之后，就可将四地址格式的指令简化为单地址或二地址格式的指令了。由于微处理器字长较短，因此这种单地址或二地址格式的指令得到了广泛的应用。

§6-2 寻 址 方 式

所谓寻址方式是指如何寻找指令的操作数或操作数的真实地址的方式。一般说来，微处

表 6-1 几种微处理器寻址方式比较表

寻址方式		微处理器			
		Intel8080A/8085A	Z80	MC 6800	说明
立即寻址	立即寻址(8 位立即数)	有	有	有	直观、方便，执行速度较快，但不够灵活。
	立即扩展寻址(16 位立即数)	有	有	有	
寄存器寻址	隐含寻址	有	有	有	指令短，执行速度最快，效率高。
	寄存器寻址	有	有	有	
存贮器寻址	寄存器间接寻址	有	有		执行速度较快，灵活性大。
	直接(扩展)寻址	有	有	有	直观、方便，但不够灵活。
	变址寻址		有	有	灵活性大，但执行速度慢。
	相对寻址		有	有	指令较短，节省内存空间
	零页寻址	有	有		仅用于 RST 指令
位寻址	位寻址		有		执行速度快，使用方便，用于位操作。

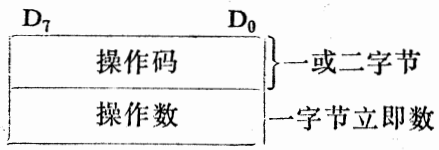
理器的寻址方式越多样，指令系统的功能就越强，灵活性也越大。

表 6-1 列出几种微处理器寻址方式的比较表，从表中可知：Z80 所使用的寻址方式是比较全面的。下面我们以 Z80 为例，说明微处理器的寻址方式。

一、立即寻址(Immediate Addressing)

在这种寻址方式中，指令的操作数包含在指令中。也就是说，指令操作码(可以是一或二个字节)后面，跟着一个字节的实际操作数(称为立即数)，参加指令所规定的操作。

它的指令格式如下：



【例】 LD A, 45 H 这条指令是将常数 45H 取入累加器 A。这是一条两字节指令，它的操作码为 3EH，在操作码后面紧跟着操作数 45H。执行该指令的示意图如图 6-2 所示。

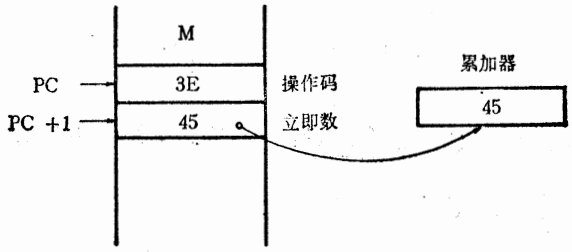
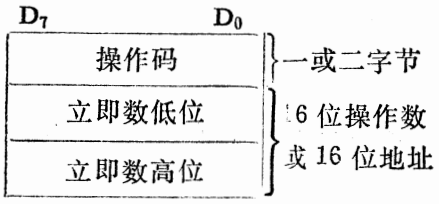


图 6-2 立即寻址示意图

二、立即扩展寻址(Immediate Extended Addressing)

这种寻址方式就是上述立即寻址方式的扩充，即操作码后面的两个字节为实际操作数。其中，紧跟操作码之后的为操作数的低 8 位字节，再下面是操作数的高 8 位字节。它的指令格式如下：



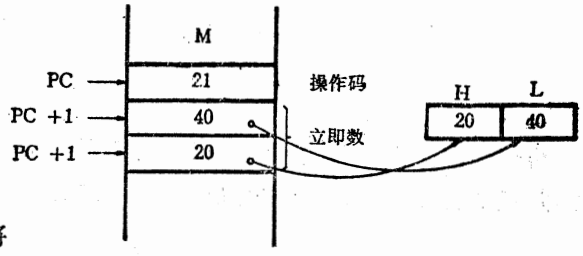


图 6-3 立即扩展寻址示意图

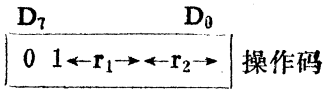
【例】 LD HL, 2040 H 这条指令是将 16 位立即数 2040 H 送入 HL 寄存器对。这是一条三字节指令，它的操作码为 21 H。执行这条指令后，立即数 20 H 送 H，40 H 送 L。如图 6-3 所示。

三、寄存器寻址 (Register Addressing)

这是单字节指令的寻址方式，指令操作码中有若干信息位用以指定 CPU 内的某一寄存器

的内容为操作数。

【例】 LD r_1, r_2 指令格式如下：

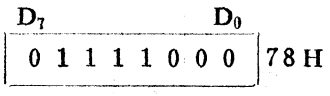


表示把 r_2 寄存器的内容送到 r_1 寄存器里。r_1、r_2 是寄存器编码，可以选定 CPU 内 7 个寄存器中的一个。其编码分配如表 6-2 所列。

表 6-2 寄存器号码表

寄存器 r_1, r_2	B	C	D	E	H	L	A
编　　码	000	001	010	011	100	101	111

本例中，若设 r_1 = A, r_2 = B, 即 LD A, B, 其指令码为：



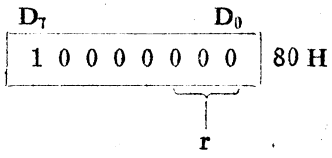
即其操作码为 78H, 执行这条指令后，寄存器 B 的内容送入寄存器 A 中。

四、隐含寻址(Implied Addressing)

在这种寻址方式中，操作码自动暗示出存放操作数的 CPU 寄存器(通常是指累加器 A)，尽管形式上操作码并没有明显地指出来。

如 Z80 中的算术和逻辑指令，其操作码本身隐含着 A 中的内容为另一操作数。

【例】 ADD A, B,其指令码为：



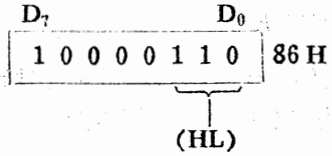
指令中 D_2～D_0 三位指定 CPU 的内部寄存器 r 的内容作为一个操作数（本例为 B 寄存器)，并且隐含着另一操作数为累加器 A 的内容。

五、寄存器间接寻址(Register Indirect Addressing)

在这种寻址方式中，指令指定某一寄存器对的内容作为操作数的地址，即寄存器对的内容并不是操作数，而是操作数的地址，从这个地址指定的单元中取出来的内容才是操作数。由于可变地址有 16 位，因此，寻址范围可达 65536。这类指令功能极强，实现简单，用途广泛。

Z80 中的数据块转移和搜索指令是这一寻址方式的扩展。

【例】 ADD A,(HL),其指令码为：



这条指令是以 HL 中的内容为地址，从这个地址单元中取出的内容作为操作数，与累加器 A 中的内容相加，结果放在 A 中，如图 6-4 所示。

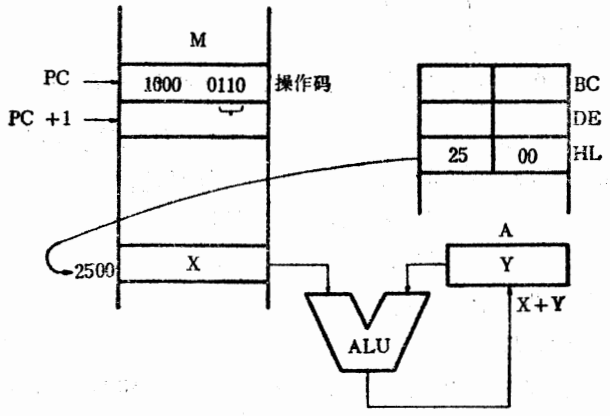
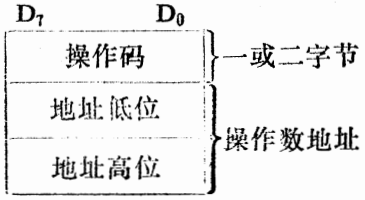


图 6-4　寄存器间接寻址方式示意图

六、直接寻址(Direct Addressing)或扩展寻址(Extended Addressing)

在这种寻址方式中，操作码后面直接给出了两个字节组成的 16 位地址码。这个地址码可以是存放数据的存贮单元的地址，也可以是程序的转移地址。它的指令格式如下：



【例】 LD A, (2040 H)

这条指令表示以操作码后面两个字节 2040 H 为地址，从内存中取出这个地址单元的内容送入累加器 A，其指令执行的示意图如图 6-5 所示。

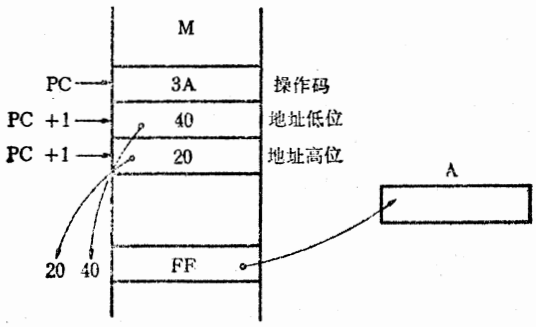
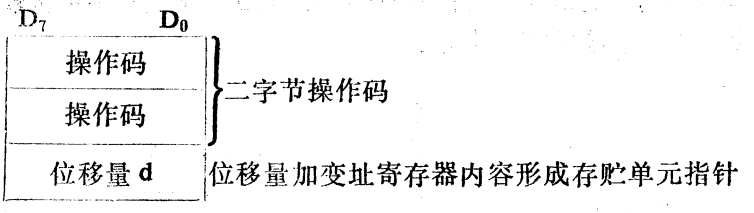


图 6-5　直接寻址示意图

七、变址寻址(Indexed Addressing)

变址寻址指令中，操作码后面的一个字节是相对于变址寄存器中基地址的位移量。该位移量和变址寄存器 IX 或 IY 中的内容(基址)相加即得到一个指明操作数的 16 位地址指针。它的指令格式如下：



当然，在使用这种寻址方式前，必须对变址寄存器预先设定基地址。Z80 中位移量 d 是带符号的对 2 补码，可变范围是 +127～−128，因此变址寻址范围是：

$$IX+127 \sim IX-128$$

$$IY+127 \sim IY-128$$

变址寻址大大简化了应用数据表的程序。

【例】 ADD A,(IY+30H)

执行这条指令时，先把 IY 中的内容与指令给定的位移量 30H 相加作为操作数的地址，由这个地址单元取出的内容 X 和累加器 A 中的内容 Y 相加，结果送至 A 中，如图 6-6 所示。

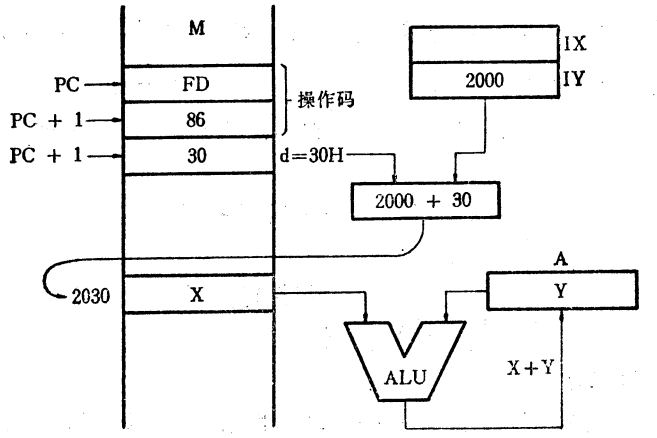
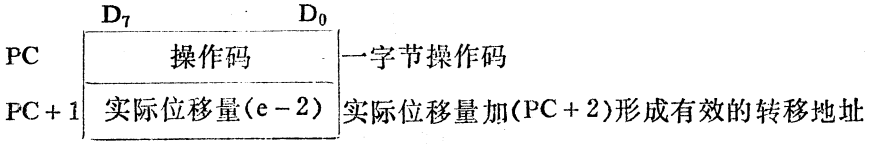


图 6-6 变址寻址示意图

八、相对寻址(Relative Addressing)

相对寻址是两字节的指令，第一字节为操作码，第二字节为相对于现行程序计数器(PC)地址的位移量，它是一个带符号数的对 2 补码，将它加到现行 PC 地址的低 8 位上，就形成下一条所要转移的地址。它的指令格式如下：



【例】 JR e(或JR $+e, $表示现行的PC值)

这条相对转移指令表示将程序转移到PC+e的地址上去。其中e为相对于现行指令第一字节地址的位移量。设现行指令的第一字节地址为A。由于这类指令都是两字节的，因此取完两字节后，程序计数器的地址为：PC=A+2。所以，实际的相对位移量应取e-2。这样，有效的转移地址为：PC=(A+2)+(e-2)=A+e

e-2的范围是 +127～-128，由此可得出相对寻址的范围是：

A+129～A-126。

相对寻址的示意图如图6-7所示。

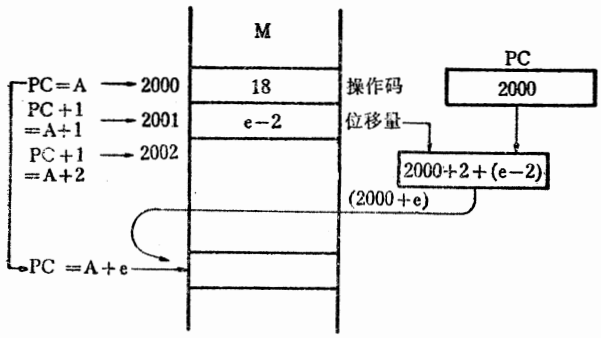


图6-7 相对寻址示意图

应当注意，在程序中用汇编符号表示相对转移指令时，它的位移量为e，当用机器代码表示送入存贮器的相对指令时，它的第二字节虽也表示位移量，但其值应为e-2。计算机在执行时能够自动地进行PC+2处理。因而，相对于现行转移指令码地址的位移量仍然为e。

相对寻址的实际意义在于只需两个字节就可以转移到该指令附近的单元。这样，可以大大节省存贮空间，并且便于修改程序。

九、零页寻址(Modified Page Zero Addressing)

Intel 8080A/8085A 和 Z80 有一条单字节的重新启动指令 RST (Restart)，它指定零页(内存单元 0000 H～00FF H 为零页)中的某些规定的单元作为子程序的入口地址。它的指令格式及执行情况如图 6-8 所示。

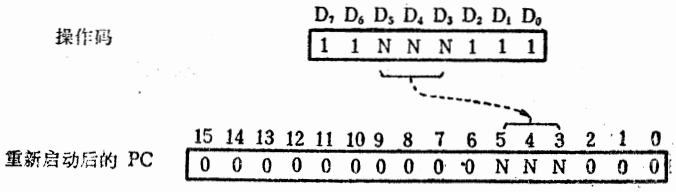


图6-8 零页寻址示意图

图中：NNN 范围为 000～111，根据 NNN 的值，就可以控制程序转移到 8×NNN的地址上去。NNN 值与规定的入口地址如表 6-3 所示。

表 6-3 RST P 指令入口地址

p	NNN	入口地址
00H	000	0000H
08H	001	0008H
10H	010	0010H
18H	011	0018H
20H	100	0020H
28H	101	0028H
30H	110	0030H
38H	111	0038H

零页寻址方式主要用于形成中断矢量。这时，RST p 指令是由申请中断的外设提供的，而不是由程序员在程序中编写的(详见第十二章)。

十、位寻址(Bit Addressing)

Z80 CPU 有大量的位操作(置位、复位和测试)指令。这类指令能对任一内部寄存器或存贮单元的任一位进行操作(置 0、置 1 和测试等)。它们可通过上述的寄存器寻址、寄存器间接寻址或变址寻址方式之一进行寻址。另外，用操作码的三位(b)来指定对 8 位操作数中的哪一位进行操作。指令中 b 所指定的操作数中的某位，其规定如表 6-4 所示。

位寻址(位测试)示意图如图 6-9 所示。

表 6-4

b	000	001	010	011	100	101	110	111
信息位	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7

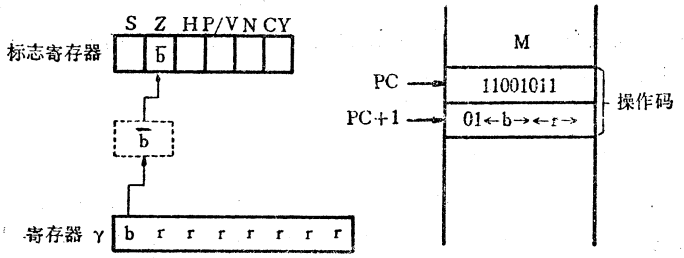
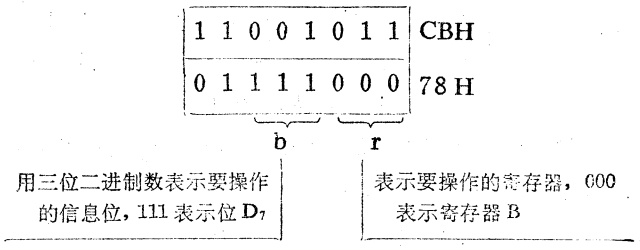


图 6-9 位寻址(位测试)示意图

【例】 BIT 7,B

这条指令表示对寄存器 B 中的第 7 位进行测试，并将测试的结果置于标志位 Z 中。其指令格式如下：



如果 B 寄存器中的 D_7 位为 1，则测试后使零标志 Z 置"0"，见图 6-9。

§6-3 指令系统的分类

指令系统表示一台微处理器功能的强弱。通常，指令系统的条数为几十条到一百多条。Intel 8080A 有 72 条基本指令，244 种操作码；Intel 8085A 有 74 条基本指令，246 种操作码；MC 6800 有 72 条基本指令，197 种操作码；Z80 有 150 条基本指令，696 种操作码。显然，在 8 位微处理器的指令功能方面，Z80 处于遥遥领先的地位。

微处理器指令分类的方法有很多种。可以按其指令长短、寻址方式以及指令功能等进行分类。通常，按指令功能可以分为以下五类：

1．数据传送指令(只进行传送而不改变数据值)

包括：存贮器传送指令
堆栈指令
内部传送指令
输入/输出指令

2．数据处理指令(一般常用 ALU 处理数据)

包括：算术运算指令
逻辑指令
移位指令
比较指令
专用指令

3．程序控制指令(用于控制程序执行的次序，改变 PC 的内容)

包括：转移指令 { 无条件转移指令 / 条件转移指令
子程序调用指令
返回指令

4．CPU 控制指令

包括：管理中断指令
暂停指令
空操作指令

5．其它指令

下面分别介绍各类指令的操作功能。

一、数据传送类

这类指令用于在 CPU 内部寄存器之间，或寄存器与存贮器之间，或寄存器与 I/O 端口之间传送数据，即把源单元的内容复制到目标单元中去，但源单元的内容不变。除个别指令外，这类指令不影响条件标志。

根据数据传送方向的不同，这类指令又可分为四类：

1．存贮器传送指令

(1) 取数指令

这类指令将指令所直接给出的操作数，或者把有效地址所指定的存贮单元的内容读入微处理器内部某个寄存器中。例如，LD A,(nn) 表示把由指令第三字节和第二字节的地址码所指定的存贮单元的内容送至累加器中。

(2) 存数指令

这类指令是将微处理器内某一寄存器的内容写入由指令的地址码所指定的存贮单元。例如，LD (nn), A 表示将累加器的内容送至由指令第三字节和第二字节的地址码所指定的存贮单元中。

2．堆栈指令

堆栈操作指令实质上也是取数和存数指令。例如推入指令(PUSH　AF)，表示将累加器和标志寄存器的内容推入到堆栈存贮器的顶部。又如弹出指令(POP　AF)，表示将堆栈顶部的内容弹出到累加器和标志寄存器中。

3．寄存器传送指令

取数和存数都是在微处理器内部寄存器和存贮器之间进行数据传送。寄存器传送指令则是在微处理器内部寄存器之间进行数据传送。显然，这类指令的执行速度最快。例如 LD r_1,r_2 表示寄存器 r_2 的内容送至寄存器 r_1 中。

4．输入/输出(I/O)指令

I/O 指令可实现微处理器内部寄存器与 I/O 设备之间的数据交换。输入指令(IN)是将某输入设备的数据送入累加器A中，而输出指令(OUT)则将累加器的数据送至某输出设备中。

二、数据处理类

这类指令是以微处理器内部寄存器或存贮器的内容为对象，以某种形式进行运算处理，在指令执行后将使内部寄存器或存贮单元的内容发生变化，而且运算处理后的特征将保留在标志寄存器中。根据运算处理的功能不同，又可以分为以下几种：

1．算术运算指令

这类指令能完成不带进(借)位的加法和减法，带进(借)位的加法和减法，以及加1、减1等二进制算术运算。加法和减法都是以累加器作为第一操作数，某寄存器或某存贮单元的内容作为第二操作数，运算结果存于累加器中。例如，ADD　A,(HL)表示由寄存器H和L中的地址码所指定的存贮单元中的内容加上累加器的内容，结果存于累加器中。加1、减1也是以某寄存器和存贮单元的内容作为操作对象。例如，INC r 表示寄存器 r 的内容加1。带进(借)位的加(减)指令，是为实现多字节的数据运算而设置的指令，在执行这类指令时，除了两数相加(减)外，还要加上(减去)前一字节运算所得的进(借)位。例如，ADC　A,r 表示寄存器 r 的内容及进位位的内容加上累加器的内容，结果放在累加器中。

算术运算指令还包括有十进制调整指令(DAA)，它是二-十进制数运算的辅助性指令。因为当一个字节表示两个二-十进制数时，计算机对它们只能进行二进制的运算，这样运算的结果有可能得到不正确的答案。例如，有两个二-十进制数 00101001 BCD = $(29)_{10}$ 和 01000101 BCD = $(45)_{10}$ 相加，结果为 01101110B = $(6E)_{16}$，此数对二-十进制数来说是没有意义的。故在加法之后应执行一条十进制数调整指令(DAA)，使在结果的低位再加上0110，其和为 01110100BCD = $(74)_{10}$，这样便可得到二-十进制运算的正确结果。

目前大部分8位微处理器都没有乘/除运算指令，这是因为要实现这种运算所需要的逻辑线路将占去相当大的芯片面积。当然也有已具有乘除指令的16位微处理器，如 Intel 8086、MC68000、TM9900 等。

2．逻辑运算指令

这类指令也是以内部寄存器或存贮单元为操作对象，一般有“与”、“或”、“非”和“异或”4种逻辑运算，运算结果保存在累加器中。例如，OR　r 表示寄存器 r 的内容和累加器A内容相“或”，结果存于累加器中。

逻辑“或”用于强迫给定位置“1”；逻辑“与”用来析取一个字节中某几位的代码，逻辑“异

或"则用来测试两个数是否完全相同等。

3．移位指令

这类指令能使某寄存器或某一存贮单元的内容，以一定方式左移或右移一位。移位指令主要有：逻辑移位、算术移位及循环移位，它们又分为左移及右移。

逻辑移位是使移位的空位置"0"，并使从数据字端移出的一位放入进位位，如图 6-10 所示。例如，逻辑右移即将寄存器中最低位移入进位位，而将最高位补"0"。逻辑左移与右移的情况类似。由图 6-10 可知，逻辑移位可用来分离数据字中的一位或一部分。

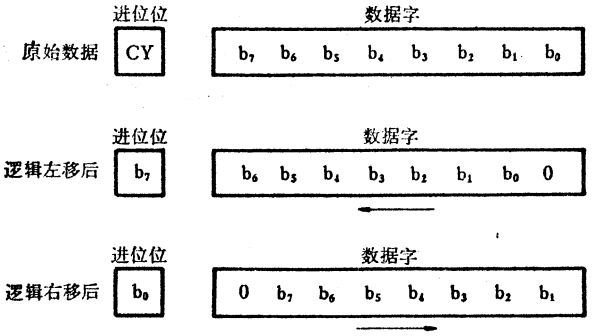


图 6-10　逻辑移位示意图

算术移位是指不改变符号位的移位，某些微处理器在处理补码时要用到它。它把 b_7 看成符号位，当左移时，右端一位置"0"，但符号位 b_7 保留不动，左端 b_6 进入进位位；当右移时，最高位 b_7 仍保持原符号，而又把 b_7 复制在其右面下一位，这个过程叫符号扩展。如图 6-11 所示。

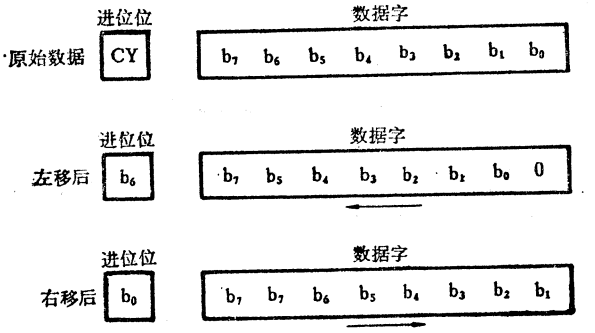


图 6-11　算术移位示意图

循环移位是使最高位和最低位首尾相接，8 位数码都保留，只是使其位置循环移动，但进位位内容可能变化。带进位循环移位与循环移位相似，只是把进位位也包括在这个循环之中，即 b_7 与 CY 相接，CY 又与最低位 b_0 相接。所以，在左移和右移时，8 个数位与进位位的数字都保留，只不过首尾循环移动位置。

循环移位和带进位位循环移位示意图如图 6-12 所示。

4．比较指令

比较指令可用来比较或测试内部寄存器或存贮单元的数据，但并不改变其内容，而只是影响标志状态。

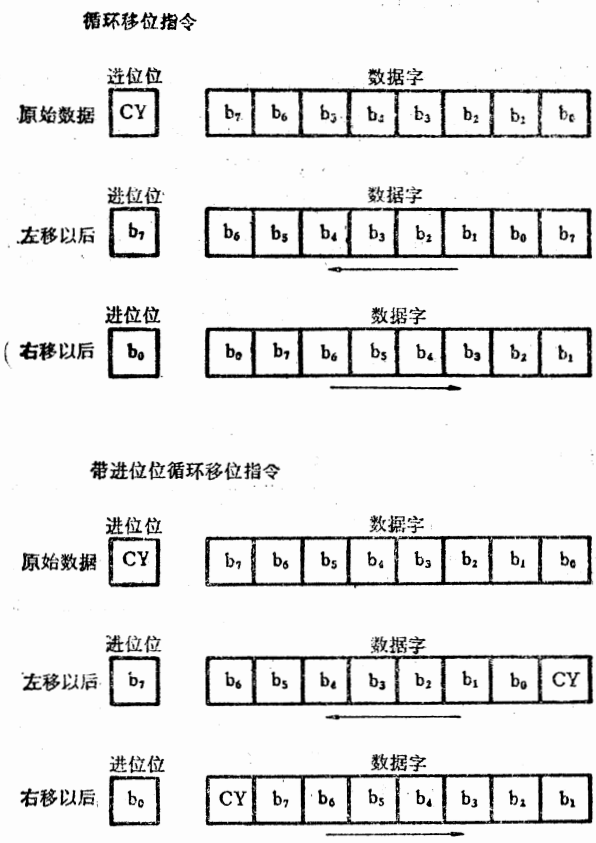


图 6-12　循环移位和带进位循环移位示意图

常见的有累加器与某一寄存器或某一存贮单元之间数据的比较。例如 CP r 表示累加器 A 的内容与寄存器 r 的内容相比较，即相减，但其结果不送回累加器。若 A = r，则标志位 Z = 1；若 A < r，则 CY = 1；若 A ≥ r，则 CY = 0。这些指令常用来作为条件转移指令的先行指令，执行比较指令后，条件转移指令就可以依据比较的结果来判定转移的条件是否满足。

5．专用指令

这类专用指令包括：对累加器取反、取补，对进位位取反或置“1”等指令。

三、程序控制类（即转移控制类）

这类控制程序流程的指令，可进一步分为转移指令、调用指令、返回指令、中断控制指令等。

1．转移指令

这类指令能够改变程序的正常次序。它有两种类型：无条件转移指令和条件转移指令。无条件转移指令强制使程序转移到所规定的指令地址上再开始执行。例如，JP nn 表示程序转移到现行指令第三字节和第二字节所确定的地址的那条指令上再开始执行。

条件转移指令通常根据标志寄存器中某一标志位的状态来决定是否转移。标志寄存器中某一位的状态，如果是“1”（或者是“0”）就满足条件，则要产生转移；若这位状态是“0”（或者是

"1")就不满足条件，则不产生转移，程序就按正常次序执行下去。因此不同的条件有不同的条件转移指令。例如，JPC 表示：如果 CY = 1，程序转移到现行指令第三字节和第二字节所确定地址的那条指令；否则就顺序执行。又如，JP NC 表示：如果 CY = 0，程序转移到现行指令第三字节和第二字节所确定地址的那条指令；否则就顺序执行。

条件转移指令赋于计算机以逻辑判断的功能，因而使计算机成为一种智能控制器。

2．调用和返回指令

这类指令能使正在执行的主程序改变原有顺序，而转去执行子程序，或者使正在执行的程序改变原有顺序，而返回去执行主程序。前者称转子(或调用)指令，后者称子程序返回指令。在执行调用指令(CALL)时，CPU 先将程序计数器中下一条指令的地址作为返回地址推入堆栈，然后将子程序的首地址送入程序计数器，这样就可以使程序转入执行子程序。当执行到子程序返回指令(RET)时，原来保存在堆栈中的返回地址就被弹出，并送入程序计数器中。这样，微处理器又继续执行原来被打断的主程序。

调用和返回指令也可分为无条件调用和返回指令及有条件调用和返回指令。

3．中断控制指令如 RST 指令，用于中断场合。

四、CPU 控制指令

1．空操作指令和暂停指令

空操作(NOP)指令除了使程序计数器加 1 外，不进行任何操作，并且微处理器内部寄存器也不会发生任何变化。暂停(HALT)指令在使程序计数器加 1 之后，微处理器就暂停工作，直到有中断或者复位(RESET)出现时，才脱离暂停状态。

2．状态管理指令

最常用的状态管理指令有：允许中断 EI (Enable Interrupt) 指令和禁止中断 DI (Disable Interrupt)指令等。这两条指令的功能将放在第十二章中断系统中详细分析。

§6-4 条件标志

微处理器中的标志寄存器(又称状态寄存器)是用来保存运算结果的一些特征的。这些特征可以由专门的指令来测试，根据测试结果决定条件转移指令是否要转移到程序的其它位置上。标志寄存器的位数由微处理器需要检测运算结果的状态位的数目而定，一般由 8 位触发器组成。在 8 位标志寄存器中，Intel 8080A/8085A 用到 5 位条件标志，Z80 用到 6 位条件标志，它们的格式如图 6-13(a)、(b)和(c)所示。

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
S	Z	0	AC	0	P	1	CY

(a) 8080A 标志寄存器 F 的格式

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
S	Z	X	AC	X	P	X	CY

(b) 8085A 标志寄存器 F 的格式

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
S	Z	X	H	X	P/V	N	C

(c) Z80标志寄存器 F 的格式

图 6-13 微处理器的状态标志

注：图中X为任意值(1或0)。

下面将微处理器中常用的标志说明如下：

Intel 8080A/8085A CPU 中有一个标志寄存器 F，Z80 CPU 中有两个标志寄存器 F 和 F′。标志寄存器中的状态标志位能够被有关的指令置"1"或置"0"，其中 4 位可以作测试用，即可以用作转移、调用或返回指令的条件。这 4 个测试是：

进位标志 CY
零标志 Z
符号标志 S
奇偶/溢出标志 P/V

一、进位标志 CY(Carry)

在进行算术运算时，如果运算结果的最高有效位(第七位)产生进位或借位，则 CY = 1；否则 CY = 0。多字节运算时需要用到 CY，在测试中也要用到 CY。例如，通过一系列的移位，可使累加器中的各位依次地移入进位位 CY 中。然后，再对 CY 进行测试。这样，便能了解累加器的各位状态。

二、全零标志 Z(Zero)

如果运算结果的各位全为零时，则 Z = 1；否则 Z = 0。这里，应当注意：当结果值为"0"时，Z 为"1"，除此之外 Z 都为"0"。

在比较操作或算术运算中，常常要测试全零标志位 Z 的状态，以确定两数是否相同，并根据比较结果确定程序是否转移。

三、符号标志 S(Sign)

符号标志用来反映运算结果是正还是负。如果运算结果的最高有效位(即符号位)$D_7 = 1$，则 S = 1 表示符号为负；否则，S = 0 表示符号为正。可用条件转移、条件转子(调用)和条件返回指令来测试符号标志。

四、奇偶标志 P(Parity)

奇偶标志通常用来检测传输过程是否正确。奇偶校验的方法有两种：一种是奇校验法，另一种是偶校验法。通常，若运算结果中 1 的总个数为偶数，则 P = 1；否则 P = 0。

应当指出，在 Z80 CPU 中，处于第 2 位的奇偶/溢出标志是一个双用途的标志位。当 μP 进行逻辑操作时，P/V 指出操作结果的奇偶性；当 CPU 进行算术运算时，P/V 表示溢出。

五、溢出标志 V(Overflow)

溢出标志 V 用作检测在二进制补码算术运算过程中是否产生溢出以致使结果出错的状态标志。如果算术运算的结果有溢出，则 V = 1；反之 V = 0。

由 §2-6 知，当两个带符号的 8 位二进制数进行算术运算时，只有在两个同号数相加或两个异号数相减的情况下，而且运算结果 $> +127$ 或者 < -128 时，才会产生溢出。因此，可能发生下述两种情况：

(1) 当两个带符号的 8 位二进制数作算术运算时，会出现没有进位，但却发生溢出的情况。

(2) 当两个带符号的 8 位二进制数作算术运算时，也会出现虽有进位，但却未发生溢出的情况。

上述两种情况的举例，已在 §2-6 节介绍过，这里不再重复。

由此可知，溢出标志与进位标志是性质不同的两个标志。

六、半进位位 H(Half-Carry)或 AC(Auxiliary Carry)

半进位位(又称辅助进位位)用于二-十进制数(BCD 码)的运算。在二-十进制数的表示法中，每一个十进制数需要用 4 位二进制数表示。当两个二-十进制数相加时，可能会产生从第 3 位向第 4 位的进位，而这个半进位可能导致结果出错。因而，必须对结果进行加 06 的修正，即进行十进制调整。为此，需要把半进位位保存在标志位 H(或 AC)中。如果两个 BCD 码相加时，运算结果产生从第 3 位向第 4 位进位时，H(或 AC) = 1；反之，H (或 AC) = 0。

应当指出，在 Intel 8080 A/8085 A 中，十进制调整指令 DAA 仅能对 BCD 码的加法运算进行调整，而 Z80 的 DAA 指令既能对 BCD 码的加法运算，又能对 BCD 码的减法运算进行十进制调整。

因此，在 Z80 中，半进位位 H 还应能反映减法运算时第 3 位从第 4 位借位的情况。若有借位，则 H = 1；否则 H = 0。并且根据 H 的状态，对 BCD 码的减法运算结果进行修正。

七、加/减标志 N(Add/Subtract 或 Wegative)

在 Z80 中，当对两个 BCD 码进行算术运算时，十进制调整指令 DAA 利用此标志来区别加法和减法。如果执行的操作为减法，则 N = 1；反之，N = 0。因为在加法、减法操作后，对 BCD 码进行修正的算法不同，所以利用 N 标志与 H 标志结合起来确定 DAA 指令如何进行修正。

§6-5 Intel 8080A/8085A 指令系统

8080 A/8085 A 的指令格式有单字节、双字节和三字节等三种，见图 6-1(a)。指令的寻址方式共有七种：立即寻址；立即扩展寻址；寄存器寻址；寄存器间接寻址；直接(扩展)寻址；隐含寻址；零页寻址。

8080A 指令系统有基本指令 72 条，8085A 比 8080A 增加了两条指令，共有基本指令 74 条。

一、Intel 8080A/8085A 指令系统的分类

1. 数据传送指令(13 条)
2. 算术运算指令(20 条)
3. 逻辑操作指令(19 条)

它包括逻辑运算、比较、循环移位等。

4. 转移、调用和返回指令(8 条)
5. 堆栈、输入/输出指令(8 条)
6. CPU 控制指令(4 条)

此外，Intel 8085A 还增加了两条指令：

读中断屏蔽位指令 RIM 和设置中断屏蔽位指令 SIM。

二、8080A/8085A 指令系统的功能说明

下面按照上述指令的分类，扼要介绍 8080A/8085A 指令系统的功能，并选择一些典型的指令举例说明。由于 Intel 8080A 的指令系统和 Z80 的指令系统是向上兼容的。因而，读者可以从 §6-6 Z80 指令系统的功能说明中，进一步了解 8080A 的指令功能。应当指出，Intel 8085A 新增加的两条指令 RIM 和 SIM 与 Z80 指令系统是不兼容的。

关于 Intel 8080A/8085A 指令系统的详细功能，读者可参阅本章末的表 6-8，该表列出了 Intel 8080A/8085A 和 Z80 CPU 通用的指令表。附录 3 列出了 Intel 8080A/8085A 十六进制指令码，并指出各类指令执行时对标志位的影响。附录 4 列出了 Intel 8080A 各指令在各机器周期、时钟周期中的功能的详细说明。

为了方便地表示指令的功能，我们给出一些表示指令的符号。r、r_1 及 r_2 表示寄存器 A、B、C、D、E、H、L 等 7 个中的任一个。M 表示由寄存器对 HL 提供地址的存贮单元。SSS 表示源寄存器的二进制编码，DDD 表示目的寄存器的二进制编码。它们可以表示 A、B、C、D、E、H、L 7 个寄存器及由 HL 提供地址的存贮单元 M，如表 6-5 所示。

表 6-5 寄存器的二进制编码表

SSS 或 DDD	000	001	010	011	100	101	110	111
寄存器名称	B	C	D	E	H	L	M	A

rp 表示寄存器对 BC、DE、HL 及 SP，其中 B、D、H、SP_H 为高位寄存器，C、E、L、SP_L 为低位寄存器。RP 表示 BC、DE、HL、SP 4 个寄存器对中之一的编码，如表 6-6 所示。有时也用 B、D、H 表示寄存器对 BC、DE、HL。

表 6-6 寄存器对的二进制编码

RP	00	01	10	11*
寄存器对	BC	DE	HL	SP

注：* 编码 11 用于堆栈指令。

addr 表示 16 位地址，data、data 16 分别表示 8 位、16 位数据，byte 2、byte 3 分别表示指令第二、三字节的内容，port 表示 I/O 端口的 8 位地址。用括号( )表示以( )中的内容作为地址，从此地址所指定的内存单元、寄存器对或 I/O 端口中取出操作数。

下面按指令功能分类作简要的说明。

(一) 数据传送指令 (13 条)

1. 存贮器传送 (11 条)

(1) 累加器与存贮器之间传送

```
LD   A    addr  ;   A←(byte 3 byte 2)    直接寻址
ST   A    addr  ;   (byte3 byte2)←A       直接寻址
LDAX      rp    ;   A←(rp)                寄存器间接寻址
STAX      rp    ;   (rp)←A                寄存器间接寻址
```

(2) 寄存器与存贮器之间传送

MOV r, M; r←(HL)　　寄存器间接寻址

MOV M, r; (HL)←r　　寄存器间接寻址

(3) 寄存器对与存贮器之间传送

LHLD addr; L←(byte3 byte2), H←(byte3 byte2+1)　　直接寻址

SHLD addr; (byte3 byte2)←L, (byte3 byte2+1)←H　　直接寻址

(4) 立即数送寄存器、存贮器、寄存器对

MVI r, data; r←byte2　　立即寻址

MVI M, data; (HL)←n　　立即/寄存器间接寻址

LXI rp, data16; rh←byte3, rl←byte2　　立即扩展寻址

2. 内部寄存器传送 (2 条)

(1) 寄存器之间传送

MOV r_1, r_2; r_1←r_2　　寄存器寻址

(2) 寄存器对之间交换数据

XCHG; H⟷D, L⟷E　　寄存器寻址

【例 1】 存贮器内容送寄存器指令 MOV r, M (Move from memory),

这是用 HL 寄存器对间接寻址的 8 位数传送指令。指令中的 M 表示以寄存器对 HL 的内容为地址的存贮单元。这一指令表示把地址在寄存器对 HL 中的存贮单元的内容传送到 CPU 内的任一寄存器 r 中。它的指令码为：

D_7							D_0
0	1	D	D	D	1	1	0

其中：$D_2D_1D_0 = 110$ 为 M 的代码。

例如，若 HL 寄存器对的内容为 2050 H，而地址为 2050 H 的存贮单元的内容为 58 H，则执行 MOV C, M 指令后，C = 58 H。

【例 2】 立即数送寄存器对指令 LXI rp, data 16(Load register pair immediate)

这是立即寻址的 16 位数传送指令。它表示指令的第三字节的内容送到寄存器对 rp 的高位寄存器 (rh)，指令的第二字节的内容送到寄存器对 rp 的低位寄存器(rl)。它的指令码为：

0	0	R	P	0	0	0	1
低位数据							
高位数据							

例如，将立即数 2200 H 送入寄存器对 HL，可执行 LXI H, 2200 H，结果：HL = 2200 H。

(二) 算术运算指令 (20 条)

算术运算指令一般是把累加器中内容与寄存器或存贮器中的内容进行算术运算，其结果放在累加器中。所有这类指令的执行都对有关的标志位产生影响。

1. 加法指令(7 条)

ADD r ; A←A+r

ADD M; A←A+(HL)

ADC r ; A←A+r+CY

ADC M ; A←A+(HL)+CY

ADI data ; A←A+byte2

ACI data ; A←A+byte2+CY

DAD rp ; HL←HL+rh rL

2. 减法指令(6条)

SUB r ; A←A−r

SUB M ; A←A−(HL)

SBB r ; A←A−r−CY

SBB M ; A←A−(HL)−CY

SUI data ; A←A−byte2

SBI data ; A←A−byte2−CY

3. 加1(增量)指令(3条)

INR r ; r←r+1

INR M ; (HL)←(HL)+1

INX rp ; rp←rp+1

4. 减1(减量)指令(3条)

DCR r ; r←r−1

DCR M ; (HL)←(HL)−1

DCX rp ; rp←rp−1

5. 十进制调整指令(1条)

DAA

【例1】 累加器内容和寄存器内容连同借位相减指令SBB r(Subtract register with borrow)它表示累加器内容减去寄存器r的内容和CY标志的内容，其结果放在累加器。它用于多字节减法。其指令码为：

1	0	0	1	1	S	S	S

例如，设B=02H，A=04H，CY=1，则执行SBB B指令的操作过程如下：

先对B与CY之和变补，然后与A的内容相加。即

$$02H+1=03H=00000011B$$

$$[-03H]_{补}=FDH=11111101B$$

$$A=04H=00000100B$$

$$+)\quad [-03H]_{补}=FDH=11111101B$$

$$\text{进位输出}=\boxed{1}\quad 00000001B=01H$$

各标志位为：CY=0，S=0，Z=0，P=0，AC=1

应当指出：这里的进位位意义已经转化，因为此时正在处理变补了的数据，所以进位输出为"1"时，表示无借位，即CY=0；反之，进位输出为"0"时，表示有借位，CY=1。

【例2】 累加器的十进制调整指令DAA(Decimal Adjust Accumulator)

累加器的 8 位数码按以下步骤调整为两个 4 位 BCD 码(即二进制编码的十进制码)形式:

(1) 若累加器低 4 位值大于 9 或 AC 标志被置位,则累加器的低 4 位加 6。

(2) 若累加器高 4 位值大于 9, 或 CY 标志被置位,则累加器的高 4 位加 6。

执行 DAA 指令对所有标志位都有影响。

它的指令码为:

0	0	1	0	0	1	1	1

例如,若将 15(BCD 码)和 27(BCD 码)相加,则用简单的十进制算术运算即可得出结果:

$$\begin{array}{r} 15 \\ +\ 27 \\ \hline 42 \end{array}$$

但是在累加器中将两个二进制数相加时,按标准二进制运算规则,可得出:

$$\begin{array}{l} \left.\begin{array}{l} 00010101 = 15 \\ +)\ 00100111 = 27 \end{array}\right\}\text{BCD 码} \\ \hline 00111100 = 3\,\text{CH} \end{array}$$

显然,该运算结果不符合 BCD 码要求。此时 DAA 指令将对结果进行调整, 以得到正确的 BCD 码。

$$\begin{array}{l} 00111100 \\ +)\ 00000110 \\ \hline 01000010 = 42\,(\text{BCD}) \end{array}$$

CY = 0, AC = 1。

应当指出, Intel 8080A/8085A 的 DAA 指令只能对累加器且只限于加法操作时才起作用,它必须跟在 BCD 码数的每条加法指令之后。

(三) 逻辑操作指令(19 条)

1. 逻辑运算指令(12 条)

```
ANA  r      ; A←A∧r,          且 CY = 0
ANA  M      ; A←A∧(HL),       且 CY = 0
ANI  data   ; A←A∧ byte2,     且 CY = 0
ORA  r      ; A←A∨r,          且 CY = 0
ORA  M      ; A←A∨(HL),       且 CY = 0
ORI  data   ; A←A∨byte2,      且 CY = 0
XRA  r      ; A←A⊻r,          且 CY = 0
XRA  M      ; A←A⊻(HL),       且 CY = 0
XRI  data   ; A←A⊻byte2,      且 CY = 0
CMA         ; A←Ā
CMC         ; CY←C̄Y
STC         ; CY←1
```

2. 比较操作指令(3 条)

CMP r ; A－r

CMP M ; A－(HL)

CPI data ; A－byte2

上述三条比较指令执行后，A、r、M、byte2 的内容均不改变，只是对标志位产生影响。

3．移位操作指令(4 条)

(1) 循环移位

RLC ; $A_{n+1}\leftarrow A_n$, $A_0\leftarrow A_7$, $CY\leftarrow A_7$; n＝0～6

RRC ; $A_n\leftarrow A_{n+1}$, $A_7\leftarrow A_0$, $CY\leftarrow A_0$

(2) 带进位循环移位

RAL ; $A_{n+1}\leftarrow A_n$, $CY\leftarrow A_7$, $A_0\leftarrow CY$

RAR ; $A_n\leftarrow A_{n+1}$, $CY\leftarrow A_0$, $A_7\leftarrow CY$

【例 1】 寄存器内容和累加器内容逻辑与指令 ANA r (AND Register)

它表示寄存器 r 的内容和累加器的内容进行逻辑“与”，其结果放在累加器。CY 标志被清除，AC 标志被置位。由于任何一位数码与“0”相“与”，其结果为“0”，而与“1”相“与”，其结果不变。因而，该指令可用来屏蔽无用的位，保留有用的位。它的指令码为：

1 0 1 0 0 S S S

例如，将累加器 A＝0FCH 与 C＝0FH 相“与”，可使累加器高 4 位为“0”，而低 4 位不变。

$$\begin{array}{r} A = 11111100 = FCH \\ \underline{C = 00001111 = 0FH} \\ \underbrace{0000}_{\text{屏蔽}}\underbrace{1100}_{\text{保留}} = 0CH \end{array}$$

【例 2】 寄存器内容和累加器内容逻辑或指令 ORA r(OR Register)

它表示寄存器内容和累加器内容进行“或”操作，其结果放入累加器。CY 和 AC 标志被清除。它可用来在给定位置上强迫置“1”。它的指令码为

1 0 1 1 0 S S S

例如，若要使累加器 A 中的第 5 位置“1”，则可在 r 寄存器中存一个数 00100000 后，利用指令 ORA r 来实现。

此外，还常用 ORA A 指令清进位位(即 CY＝0)，而保留 A 的内容。

【例 3】 寄存器内容和累加器内容异或操作指令 XRA r(Exclusive OR Register)

它表示寄存器 r 的内容和累加器内容“异或”，其结果放在累加器。CY 和 AC 标志被清除。它可用来测试两个数是否相等。这是因为异或的功能是相同为“0”，不同为“1”。它的指令码为

1 0 1 0 1 S S S

通常用 XRA A 去清除累加器 A 和进位位 CY。

例如，下列指令可用于清除 A、B、C 寄存器：

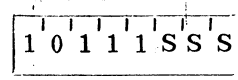
XRA A

MOV B, A

MOV C, A

【例 4】 寄存器内容和累加器内容相比较指令 CMP r(Compare Register)

它表示累加器内容减去寄存器 r 的内容，但累加器的内容保持不变。根据减的结果决定标志位是否置位。若 A = r，则 Z 标志位置"1"，若 A < r，则 CY 标志位置"1"，若 A ≥ r，则 CY 标志位置"0"。它的指令码为



例如，设累加器内容为 0AH，而 E 寄存器内容为 05 H，则执行 CMP E 指令时，进行减法操作如下：

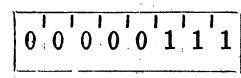
$$\begin{array}{r} (A) = 00001010B \\ +)\ [-E]_{补} = 11111011B \\ \hline 进位输出 = \boxed{1}\ 00000101B \end{array}$$

各条件标志：CY = 0，Z = 0，说明 A > E

这里，对进位位的处理见前述的 SBB r 例子。

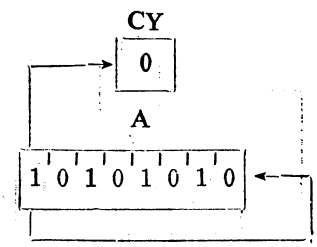
【例 5】 循环左移指令 RLC(Rotate A Left)

它表示累加器的内容循环左移一位，高位移出的值置入低位和 CY 标志，仅有 CY 标志受影响。它的指令码为

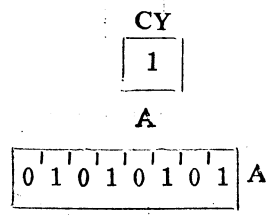


例如，设累加器的内容为 0AAH，而 CY = 0，可用下图说明 RLC 指令的作用。

执行前



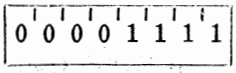
执行后



【例 6】 循环右移指令 RRC(Rotate A right)

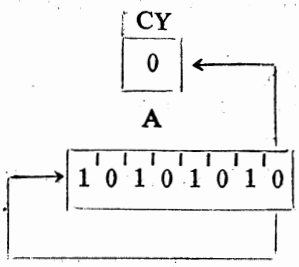
它表示累加器内容循环右移一位，低位移出的值同时置入高位和 CY 标志，仅有 CY 标志

受影响。它的指令码为

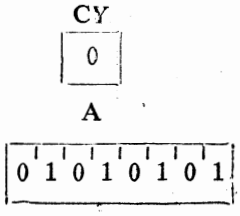


例如，设累加器的内容为0AAH，而CY＝0可用下图说明RRC指令的作用。

执行前

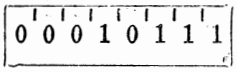


执行后



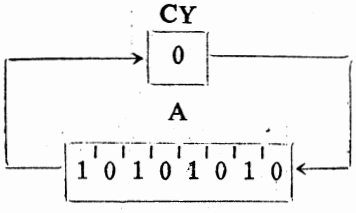
【例7】 通过进位循环左移RAL(Rotate Left through carry)

它表示累加器的内容通过CY标志循环左移一位。CY标志置入低位，高位值移出置入CY标志，仅有CY标志受影响。它的指令码为

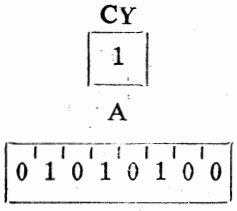


例如，设累加器A的内容为0AAH，CY＝0，执行RAL指令如下：

执行前

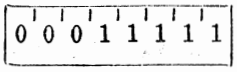


执行后



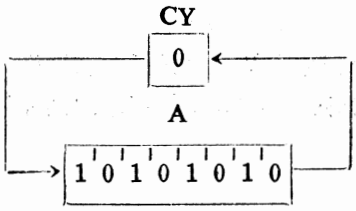
【例8】 通过进位循环右移RAR (Rotate right through carry)

它表示累加器内容通过CY标志循环右移一位。CY标志置入高位，低位值移出置入CY标志，仅有CY标志受影响。它的指令码为

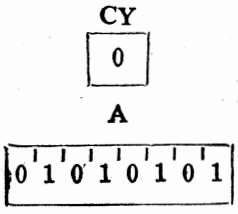


例如，设累加器的内容为 0AAH，CY = 0，执行 RAR 指令如下：

执行前



执行后



（四）程序控制指令（转移指令 8 条）

转移类指令包括条件转移和无条件转移指令，子程序调用指令和返回指令。

1．转移指令

转移指令分无条件转移及条件转移指令，均为三字节指令。

(1) 无条件转移指令

JMP addr；PC←byte3 byte2

(2) 条件转移指令

Jcondition addr；If(CCC)，PC←byte3 byte2

条件转移指令根据条件 (CCC) 不同而有 8 条。当满足给定条件时，则 PC←byte3 byte2，否则，顺序执行下一条指令。

CCC

JNZ addr ；IF Z = 0， 000，PC←byte3 byte2

JN addr ；IF Z = 1， 001，PC←byte3 byte2

JNC addr ；IF CY = 0，010，PC←byte3 byte2

JC addr ；IF CY = 1，011，PC←byte3 byte2

JPO addr ；IF P = 0， 100，PC←byte3 byte2

JPE addr ；IF P = 1， 101，PC←byte3 byte2

JP addr ；IF S = 0， 110，PC←byte3 byte2

JM addr ；IF S = 1， 111，PC←byte3 byte2

2．子程序调用指令（又称转子指令）

转子指令分无条件转子及条件转子指令，均为三字节指令。

(1) 无条件转子指令

CALL addr；$(SP-1)\leftarrow PC_H$, $(SP-2)\leftarrow PC_L$, $SP\leftarrow SP-2$, $PC\leftarrow$byte3 byte2

(2) 条件转子指令

C condition addr；IF (CCC),

$(SP-1)\leftarrow PC_H$, $(SP-2)\leftarrow PC_L$, $SP\leftarrow SP-2$, $PC\leftarrow$ byte3 byte2

同上述条件转移一样，有 8 条条件转子指令。

转子指令可以使当前的程序计数器(PC)内容保存在堆栈中，然后把子程序的起始地址置入 PC，当子程序执行完毕后，用返回指令 RET 使程序返回原来的断点处，即把堆栈中保存的内容弹回 PC。

3．返回指令

(1) 无条件返回指令

RET；$PC_L\leftarrow(SP)$, $PC_H\leftarrow(SP+1)$, $SP\leftarrow SP+2$

(2) 条件返回指令

R condition；IF (CCC), $PC_L\leftarrow(SP)$, $PC_H\leftarrow(SP+1)$, $SP\leftarrow SP+2$

同上述条件转移一样，有 8 条条件返回指令。返回指令都是单字节的指令。

4．重新启动指令

RST n；$(SP-1)\leftarrow PC_H$, $(SP-2)\leftarrow PC_L$, $SP\leftarrow SP-2$, $PC\leftarrow 8*NNN$

其中，NNN 范围为 000～111。

5．间接转移指令

PCHL；$PC_H\leftarrow H$, $PC_L\leftarrow L$

此指令与无条件转移指令的功能相同。

(五) 堆栈、输入/输出指令 (8 条)

1．堆栈指令 (6 条)

(1) 推入指令 (2 条)

PUSH rp ；$(SP-1)\leftarrow rh$, $(SP-2)\leftarrow rl$, $SP\leftarrow SP-2$

PUSH PSW ；$(SP-1)\leftarrow A$, $(SP-2)\leftarrow F$, $SP\leftarrow SP-2$

(2) 弹出指令 (2 条)

POP rp ；$rL\leftarrow(SP)$, $rh\leftarrow(SP+1)$, $SP\leftarrow SP+2$

POP PSW ；$F\leftarrow(SP)$, $A\leftarrow(SP+1)$, $SP\leftarrow SP+2$

(3) 栈顶单元与 HL 交换指令 (1 条)

XTHL ；$L\longleftrightarrow(SP)$, $H\longleftrightarrow(SP+1)$

(4) HL 的内容装入 PC 指令(1 条)

SPHL ；$SP\leftarrow HL$

2．输入/输出指令 (2 条)

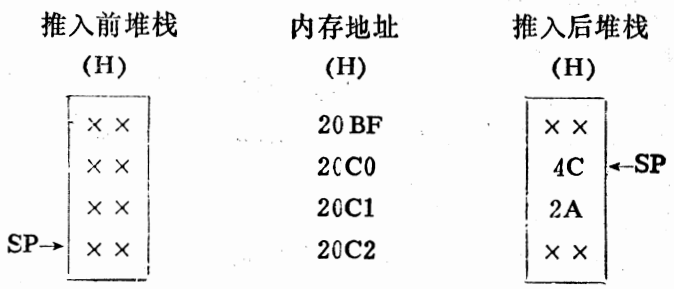
(1) 输入指令 (1 条)

IN port ；$A\leftarrow$(port)

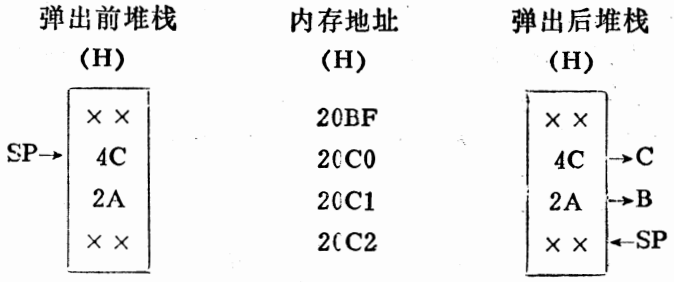
(2) 输出指令 (1 条)

OUT port ；(port)$\leftarrow A$

【例 1】 设 B = 2AH, C = 4CH, SP = 20C2H, 则执行 PUSH B 指令的过程如下：



【例 2】 内存单元 (20COH) = 4CH, (20C1H) = 2AH, SP = 20C0H, 则执行 POP B 指令的过程如下：



【例 3】 输入指令 IN port (Input)

其中 Port 表示输入端口的地址。这一指令表示将指定地址的端口的数据放到 8 位双向数据总线上，再送到寄存器 A。端口的编址范围为 00H～0FFH。当执行 I/O 指令时，CPU 将 Byte2 的端口地址复制到高 8 位和低 8 位地址总线上去选择 I/O 端口。它的指令码为

例如，若累加器 A 的内容为 55H，外设待传送的字节为 80H，该外设相应的输入端口地址为 01H，则在执行 IN 01H 指令后，累加器 A = 80H。

【例 4】 输出指令 OUT port(Output)

这一指令表示把累加器 A 的内容放到 8 位双向数据总线上，然后传送到指定的输出端口。端口的编址范围为 00H～0FFH。它的指令码为

例如，若累加器内容为 55H，在执行 OUT 02H 指令之后，字节 55H 将被写入至相应于输出端口 02H 的外设中。

(六) CPU 控制指令 (4 条)

1. 允许中断指令 EI(Enable interrupts)

2. 禁止中断指令 DI(Disable interrupts)

EI 和DI 指令将在第十二章中详细介绍。

3. 暂停指令 HLT(Halt)

这一指令执行后，使 CPU 暂停执行程序，直至有中断请求或复位命令时，才重新开始执

行程序，调试程序时，常用到 HLT 指令。

4．空操作 NOP(NO Operation)

空操作指令除了能使程序计数器 PC 加 1 以外，不执行其它操作，也不影响寄存器和标志位的状态。设置 NOP 指令的目的有两个：一是当编程发生错误而删去一条指令后，可用 NOP 指令填充；二是可以提供短时间的精确延时。

（七）8085A 新增加的两条指令 RIM 和 SIM

1．读中断屏蔽位指令 RIM(Read Interrupt Masks)

执行 R M 指令以后，可将重新启动屏蔽位、中断允许标志、中断判定条件和串行输入数据线(SID)的内容读入累加器。

通常在执行 TRAP 中断以后，为了能准确地恢复 TRAP 中断前原来的中断允许状态，必须在 CPU 响应 TRAP 中断之后，紧接着执行一条 RIM 指令。这样，当执行 RIM 指令时，累加器的 IE 位就反映出在 TRAP 出现以前原来的中断允许状态。

R M 指令的操作码为 20H，其执行过程如图 6-14 所示。

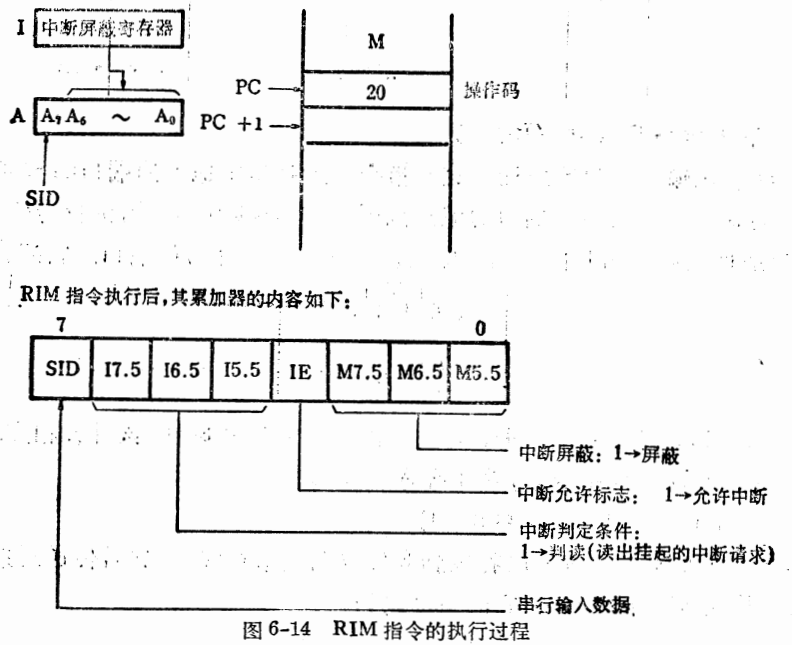


图 6-14 RIM 指令的执行过程

该指令占用 1 个机器周期，4 个时态，对标志位无影响。

2．设置中断屏蔽位 SIM(Set Interrupt Masks)

在执行 SIM 指令期间，累加器的内容用来设置重新启动中断屏蔽位(见图 6-15)。若第 3 位是“1”(置位)，对于中断屏蔽寄存器的 RST5.5、6.5 和 7.5，由第 0～第 2 位来置位/复位屏蔽位。第 3 位是用来控制“允许设置屏蔽”。

屏蔽位一经设置(即屏蔽位 = 1)，CPU 就禁止相应的重新启动中断。

重新启动指令		置位（屏蔽）	复位（允许）
RST 5.5	MASK	若第 0 位=1	若第 0 位=0
RST 6.5	MASK	第 1 位=1	第 1 位=0
RST 7.5	MASK	第 2 位=1	第 2 位=0

若累加器的第4位是1，则不管RST 7.5是否屏蔽，RST 7.5请求触发器将被复位。

8085 A的$\overline{\text{RESET IN}}$信号将使所有的RST MASK置位。

若累加器A的第6位置位，则执行SIM指令后，A的第7位状态将被送至SOD锁存器。若A的第6位是置零，则锁存器不受影响。$\overline{\text{RESETIN}}$输入信号将使SOD锁存器复位。

应当指出，DI指令可以超越SIM指令，即当DI指令有效时，不论屏蔽与否，RST 7.5, RST6.5, RST5.5都被禁止中断。

SIM指令的操作码为30H，其执行过程如图6-15所示。

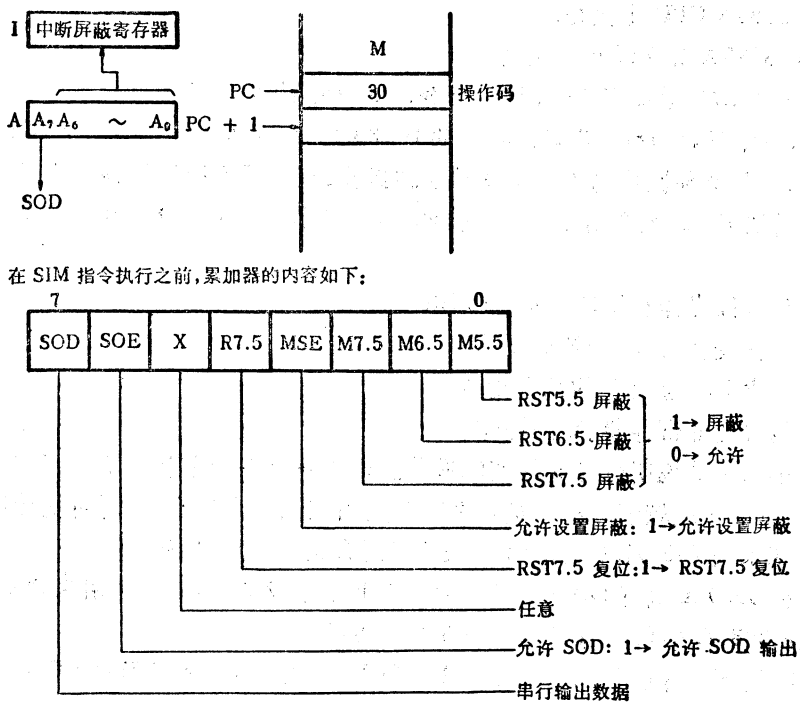


图6-15 SIM指令的执行过程

该指令占用1个机器周期，4个时态，对标志无影响。

【例1】 设累加器内容的位模式为11001111B，则执行SIM指令之后，RST5.5, RST6.5, RST7.5都被设置屏蔽，并把A_7位的内容输送到SOD输出。

【例2】 要求解除RST5.5屏蔽(SDK-85单板机中键盘的中断输入端为RST5.5)。

可用下列两条指令：

```
MVI   A,  0EH  ; 00001110B
                  └→允许 RST5.5 中断
SIM
```

§6-6 Z80的指令系统

本节将在分析Z80与Intel 8080A指令系统异同点的基础上，将Z80指令系统加以归纳分类。然后，着重从指令的应用方面，举例说明各类指令的功能。

一、8080A和Z80指令系统的异同点

Z80 CPU有150类基本指令，能识别和执行696种操作码，其中包括8080A CPU的全部

72 种基本指令，244 种操作码。

1．8080A 和 Z80 的兼容性

(1) 8080A 和 Z80 在机器语言级上是兼容的。但是它们在汇编语言级上并不兼容，这是因为两者采用了不同的操作码助记符。本章末表 6-8 列出了 Intel 8080A/8085A 和 Z80 微处理器通用的指令表。从表中可知：这三种微处理器通用的操作码有 111 种，它们大都是常用的指令，熟悉这些常用指令的功能，就足以编写出所需要的程序。

(2) 利用 8080 A 的机器语言指令编写的程序都能够在 Z80 CPU 上执行，但是 Z80 的程序并不都能在 8080A CPU 上执行。

2．8080A/8085A 和 Z80 的差异点

(1) Z80 比 8080A 增加了 78 种基本指令，452 种操作码。Z80 利用 8080A 中未用到的 12 种操作码：08H、10H、18H、20H、28H、30H、38H、D9H、CBH、DDH、EDH 和 FDH 等进行扩充，其中利用前 8 种操作码扩充了 8 种新的操作，利用后 4 种操作码组合成为双字节的操作码，从而又增加了 444 种操作。这些指令的第一字节是 CB、DD、ED、FD，它们的意义是：

CB——包含位操作和循环及移位操作；

DD——包含变址寄存器 IX 的操作；

ED——包含 Z80 新增加的部分指令；

FD——包含变址寄存器 IY 的操作。

操作码的第二字节说明待完成的实际操作。由于这类多字节指令增加了访问存贮器的次数，故其指令执行的速度较慢。程序员在编程时应当注意这个特点，并尽可能选用执行速度较快的指令。本章末表 6-9 列出上述 12 种操作码的含义表，以供比较。

(2) Z80 和 8080 A 执行 DAA 指令时有所不同。在 Z80 中该指令用来调整十进制的加/减法运算，而在 8080A 中仅能用于调整十进制加法运算。

(3) Z80 的循环指令能清除 H 状态标志。8080A 的循环指令不影响 AC 标志。

(4) Z80 的 P/V 标志是双重用途的标志。当逻辑操作时用作奇偶标志；当算术运算时用作溢出标志。而在 8080A 中仅用作奇偶标志。

(5) Z80 中的 N 标志，位于 F 中的 D_1 位，而在 8080A 中没有该标志，且该 D_1 总是置“1”。

(6) Z80 与 8085A 新增加的两条指令是不兼容的。8085A 中用于 RIM 和 SIM 的机器代码，在 Z80 上用于相对转移(NZ 和 NC)指令。

(7) 8080A、8085A 和 Z80 CPU 的指令时序是不同的。因而，凡是需要依靠指令精确定时的程序或延时程序，只有在为之编写该程序的微处理器上才能正确执行。

现将 Intel 8080A/8085A、Z80 和 Z80A 的时钟频率、时钟周期以及执行一条指令所需要的时间比较如下：

微处理器	时钟频率	时钟周期	执行一条指令的时间
Intel 8080A	2MHz	500ns	2～9μs
Intel 8085A	3MHz	330ns	1.3～5.94μs
Z80	2.5MHz	400ns	1.6～9.2μs
Z80A	4MHz	250ns	1～5.75μs

二、Z80 指令系统的分类

Z80 的指令系统可以分成下列 8 大类，共 150 条基本指令[注]：

1．数的传送和交换指令（47 条）
2．数据块传送和搜索指令（8 条）
3．算术和逻辑运算指令（33 条）
4．循环和移位指令（16 条）
5．位操作（置位、复位和测试）指令（9 条）
6．转移、调用和返回指令（18 条）
7．输入/输出指令（12 条）
8．CPU 控制指令（7 条）

下面分别加以说明，其中将输入/输出指令、中断指令放在后面相应的章节进行分析。

三、Z80 指令系统的功能说明

（一）数的传送和交换指令（Load and Exchange）

这类指令用于 CPU 内各寄存器之间或寄存器与存贮器之间传送数据，或将指令中包含的立即数送至寄存器或存贮器中去。这类指令中，除个别指令外，一般不影响各标志位的状态。

1．8 位数的传送

8 位数传送指令共有 21 条基本指令。如附录 1 表 A1-1 和附录 2 表 A2-1 所列。它们的汇编语言助记符是 LD，其后是目的和源。目的地写在源的左面。从表 A2-1 中可知，除 LD A，I 和 LD A，R 两条指令影响标志位 Z 和 S 外，该类中的其它指令执行时都不会影响标志位的状态。附录 1 表 A 1-1 表示所有 8 位传送指令的操作码及用于每条指令上的寻址方式。表格上面的横栏里指出数据源。左边纵向栏则指出传送的目的地。表中粗线框内表示与 8080A/8085A 相兼容的操作码。

（1）立即数送寄存器

指令格式：LD r，n；r←n

这是两字节指令，它的功能是将 8 位的立即数 n 送至 CPU 内任意一个寄存器 r 中。该指令常用于给某一寄存器赋初值，或在计算中处理一些常数。r 的编码与 8080A 的规定相同。指令功能示意如图 6-16 所示。

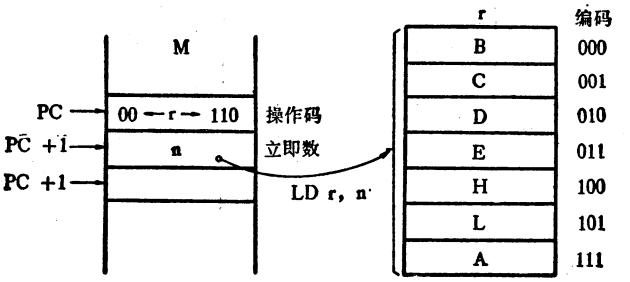


图 6-16　立即数送寄存器指令示意图

【注】一般书刊、资料都写为 158 条，但 Z80 指令表中只有 150 条。

(2) 寄存器之间传送

指令格式：LD r_1, r_2; $r_1 \leftarrow r_2$

这是单字节指令，它表示将寄存器 r_2 的内容送至寄存器 r_1 中。r_1、r_2 的编码与上述的 r 相同，其功能示意如图 6-17 所示。

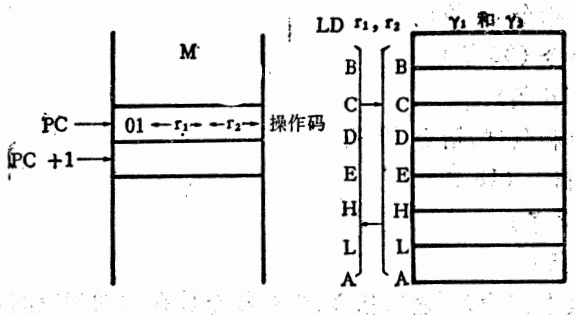


图 6-17 寄存器间传送指令示意图

在寄存器传送指令中，还有 4 条指令，它们执行时，对有关标志将产生影响(见表 A2-1)。

指令格式：① LD A , I ; A←I

② LD I , A ; I←A

③ LD A , R ; A←R

④ LD R , A ; R←A

(3) 用 HL 寄存器对间接寻址的 8 位数传送

指令格式：① LD r, (HL) ; r←(HL)

② LD (HL), r ; (HL)←r

③ LD (HL), n ; (HL)←n

①、②是一组单字节的 8 位数传送指令；

③是一组两字节指令。

寄存器间接寻址方式就是以 CPU 内的 H 和 L 寄存器对中的内容 (16 位)来指明存放操作数的存贮单元地址，用(HL)表示之。这组指令能将 CPU 内部任一寄存器的内容存入指定的内存单元，或把指定的内存单元的内容取入 CPU 内的任一寄存器，也可以把一个 8 位的立即数存入指定的内存单元，如图 6-18 所示。

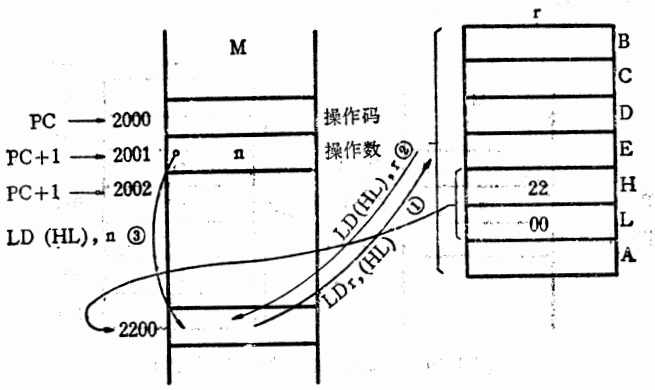


图 6-18 用 HL 间接寻址的传送指令示意图

【例 1】 把寄存器 A、B、C 中的内容分别存入到 2050H、2060H 及 2070H 内存单元中。

根据题意，可用上述指令实现，程序如下：

```
LD  HL,  2050H    ; HL← 内存地址号 2050H
LD  (HL), A       ; 存A的内容
LD  HL,  2060H    ; HL← 内存地址号 2060H
LD  (HL), B       ; 存B的内容
LD  HL,  2070H    ; HL← 内存地址号 2070H
LD  (HL), C       ; 存C的内容
```

每次新的寄存器内容被存入内存单元时，HL 必须重新赋值，以指出要存入的内存单元的地址。这里，LD HL, 2050H, LD HL, 2060H, LD HL, 2070H, 是一种立即扩展寻址方式的指令，它是把 16 位立即数赋给寄存器对 HL。关于这类指令下面还要介绍。

(4) 用 BC、DE 寄存器对间接寻址的 8 位数传送：

```
指令格式：① LD  A   ,(BC)  ; A←(BC)
          ② LD  A   ,(DE)  ; A←(DE)
          ③ LD  (BC) ,A    ; (BC)←A
          ④ LD  (DE) ,A    ; (DE)←A
```

这也是一组单字节的间接寻址指令。它们与以 HL 寄存器对间接寻址指令的重要区别在于寄存器对 BC 及 DE 中的内容所指出的内存单元仅能与累加器 A 交换数据，其功能示意如图 6-19 所示。

利用这类间接寻址指令，可以很方便地在通用寄存器中修改地址指针，以实现数据块传送。

(5) 用直接寻址(扩展寻址)的 8 位数传送

```
指令格式：① LD  A,(nn)  ; A←(nn)
          ② LD  (nn),A   ; (nn)←A
```

这类指令是三字节的直接寻址指令。指令的第三、二字节指明实际操作数的有效地址(第三字节为地址高位)，并在此地址所指示的内存单元和累加器之间进行数据传送，其功能如图 6-20 所示。

应当指出，只有累加器 A 才能与直接寻址的内存单元之间进行数据传送。CPU 内其它的寄存器也只有通过累加器 A 作为桥梁，才能与所指定内存单元进行数据传送，这点务必引起注意。

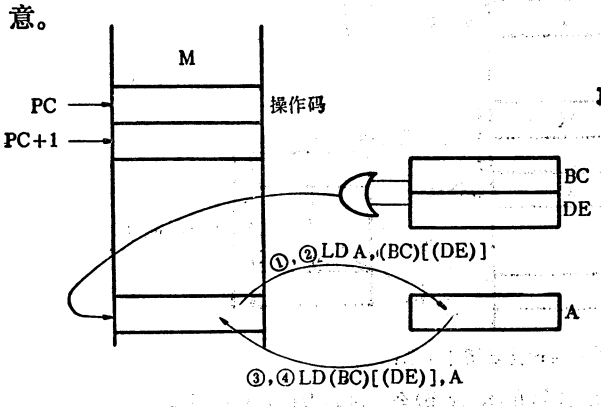


图 6-19 用 BC、DE 间接寻址的传送指令示意图

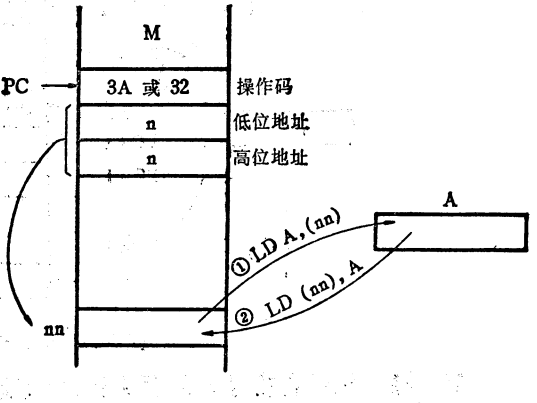


图 6-20 直接寻址(扩展寻址)的传送指令示意图

【例 2】 将寄存器 A、B、C 的内容分别存入内存单元 2400H、2401H、2402H 中。然后，再将 2600H、2601H、2602H 的内容分别送入 A、B、C 中。

根据题意编出的程序如下：

```
LD  (2400H) , A        ;  存 A
LD  A       , B        ;
LD  (2401H) , A        ;  } 存 B
LD  A       , C        ;
LD  (2402H) , A        ;  } 存 C
LD  A       , (2602H)  ;
LD  C       , A        ;  } 从内存单元 2602H 取数至 C
LD  A       , (2601H)  ;
LD  B       , A        ;  } 从内存单元 2601H 取数至 B
LD  A       , (2600H)  ;  从内存单元 2600H 取数至 A
```

(6) 用变址寄存器寻址的 8 位数传送

指令格式：

```
① LD  r       , (IX+d)   ; r←(IX+d)
② LD  r       , (IY+d)   ; r←(IY+d)
③ LD  (IX+d)  , r        ; (IX+d)←r
④ LD  (IY+d)  , r        ; (IY+d)←r
⑤ LD  (IX+d)  , n        ; (IX+d)←n
⑥ LD  (IY+d)  , n        ; (IY+d)←n
```

前 4 条是两组三字节的传送指令，后两条是四字节的立即数送内存单元的指令，内存单元用变址寄存器寻址，即用变址寄存器 IX(或 IY)的内容与位移量 d 之和确定的内存地址。d 是一个由指令给定的位移量，它是一个带符号的对 2 的补码，其变化范围为 +127～−128。指令功能示意如图 6-21 所示。

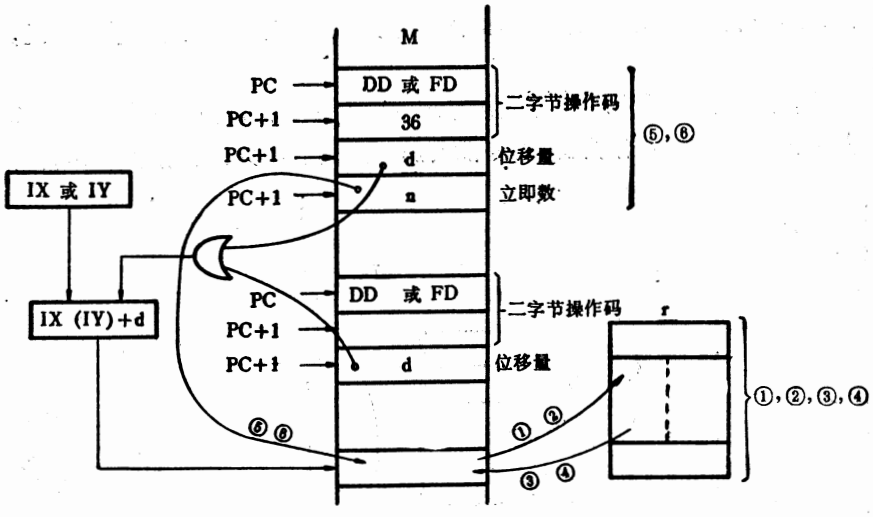


图 6-21 用变址寻址的传送指令示意图

【例 3】 仍以直接寻址例题为例，用变址寻址的传送指令，则可用以下程序：

```
LD IX,  2400H  ; 把立即数(地址)送 IX
```

```
LD  (IX+0),  A  ; 存A
LD  (IX+1),  B  ; 存B
LD  (IX+2),  C  ; 存C
```

显而易见，利用变址寻址传送指令，比直接寻址灵活得多，并可减少程序的指令数。但是，变址寻址指令执行时间较长，这是因为它必须执行一次加法后才能得到有效地址。

2．16位数的传送

16位数传送共有20条基本指令，如附录1表A1-2和附录2表A2-2所列。从表A2-2中可知，这类指令对标志状态无影响。

(1) 立即扩展寻址的16位数传送

dd	00	01	10	11
寄存器对	BC	DE	HL	SP

指令格式：① LD dd, nn ; dd←nn

② LD IX, nn ; IX←nn

③ LD IY, nn ; IY←nn

这类立即扩展寻址的16位数传送指令，是将指令操作码后的第二、三字节的16位立即数nn送到寄存器对BC、DE、HL、堆栈指示器SP和变址寄存器IX、IY中去。

通常，这类指令用于预置初值，尤其是设置地址初值。其功能示意图如图6-22所示。

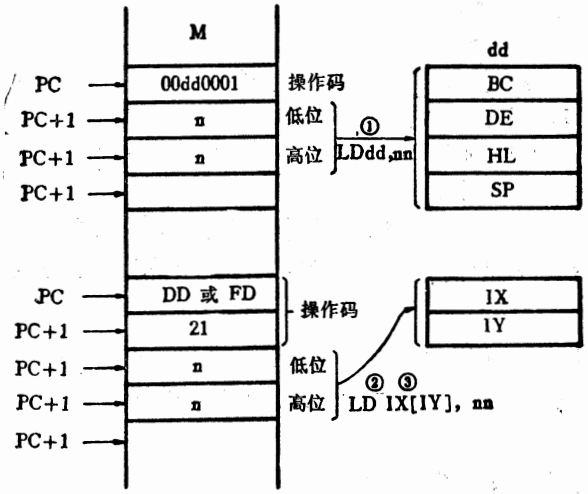


图6-22 立即数送寄存器对指令示意图

(2) 直接寻址方式的16位数的传送

指令格式：① LD HL, (nn) ; L←(nn), H←(nn+1)

② LD dd, (nn) ; dd_L←(nn), dd_H←(nn+1)

③ LD IX, (nn) ; IX_L←(nn), IX_H←(nn+1)

④ LD IY, (nn) ; IY_L←(nn), IY_H←(nn+1)

⑤ LD (nn), HL ; (nn)←L, (nn+1)←H

⑥ LD (nn), dd ; (nn)←dd_L, (nn+1)←dd_H

⑦ LD (nn), IX ; (nn)←IX_L, (nn+1)←X_H

⑧ LD (nn), IY ; (nn)←IY_L, (nn+1)←IY_H

上述指令组中，除①为3字节指令外，其余都为4字节指令。这类指令利用直接寻址，把地址nn单元的内容送入寄存器对BC、DE、HL、SP、IX、IY的低位，而把下一地址nn+1单元的内容送入相应寄存器对的高位，或进行反向传送。其功能示意图如图6-23所示。

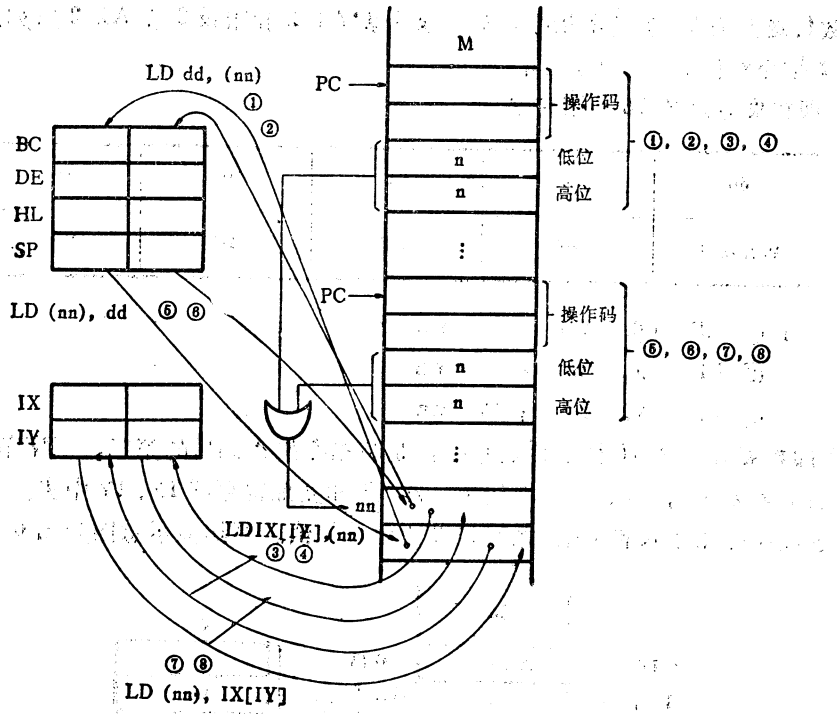


图6-23 扩展寻址(直接寻址)的16位数传送指令示意图

这里，应当注意下面两种区别：

① Z80的16位扩展寻址指令可以用作任何寄存器对和指定的内存单元之间进行16位数的传送。而8080A/8085A中只有HL寄存器对才能和指定的内存单元之间进行16位数的传送。

② Z80中LD dd, nn和LD dd, (nn)是两条性质不同的指令。前者是将16位的立即数nn取入寄存器对，而后者则是将指定的内存单元nn及nn+1中的16位操作数取入寄存器对。

即LD dd, nn 指的是： dd←nn

LD dd, (nn)指的是：dd_L←(nn)，dd_H←(nn+1)

(3) 16位数传送到堆栈指示器SP

指令格式：

① LD SP, HL ; SP←HL

② LD SP, IX ; SP←IX

③ LD SP, IY ; SP←IY

上述指令组中，① 为单字节指令；②、③ 为两字节指令。这类指令允许把HL、IX和IY

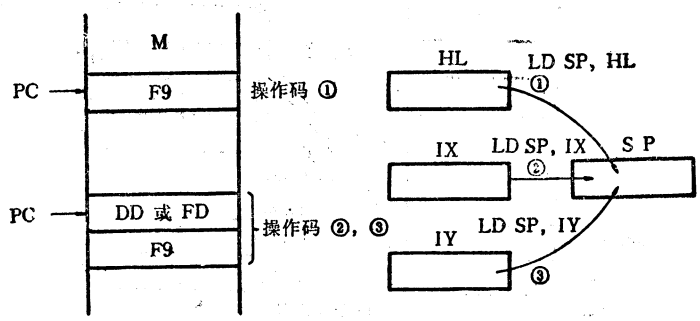


图 6-24　16 位数传送至堆栈指针的指令示意图

寄存器对的数据传送到堆栈指示器 SP，但是不允许反向传送。它们主要用于设置堆栈指针，其功能示意图如图 6-24 所示。

(4) 16 位堆栈操作

寄存器对	BC	DE	HL	AF
qq	00	01	10	11

指令格式：① PUSH qq ；$(SP-1)\leftarrow qq_H$, $(SP-2)\leftarrow qq_L$, $SP\leftarrow SP-2$

② PUSH IX ；$(SP-1)\leftarrow IX_H$, $(SP-2)\leftarrow IX_L$, $SP\leftarrow SP-2$

③ PUSH IY ；$(SP-1)\leftarrow IY_H$, $(SP-2)\leftarrow IY_L$, $SP\leftarrow SP-2$

④ POP qq ；$qq_L\leftarrow(SP)$, $qq_H\leftarrow(SP+1)$, $SP\leftarrow SP+2$

⑤ POP IX ；$IX_L\leftarrow(SP)$, $IX_H\leftarrow(SP+1)$, $SP\leftarrow SP+2$

⑥ POP IY ；$IY_L\leftarrow(SP)$, $IY_H\leftarrow(SP+1)$, $SP\leftarrow SP+2$

关于堆栈的概念已在 §4-4 中介绍过，这里不再重复。

堆栈操作的指令有两种：一是把 CPU 内部寄存器对的内容推入堆栈的指令——PUSH；二是由堆栈弹出到 CPC 内的寄存器对的指令——POP。它们都是采用寄存器间接寻址方式的。

从上述堆栈操作符号表示式中可知：当执行推入堆栈指令 PUSH 时，先要使 $SP\leftarrow SP-1$，即使 SP 指向一个新单元，接着把寄存器对的高位的内容推入 SP 所指的内存单元，然后再使 $SP-1\rightarrow SP$，把寄存器对的低位的内容推入 SP 所指的内存单元(也就是说，入栈时“先高后低”)。推入后，SP 指向最后推入的单元，也即 SP 指向堆栈的顶部。当执行弹出堆栈指令 POP 时，则要先把 SP 所指的单元的内容弹出送至寄存器对的低位，接着 $SP\leftarrow SP+1$，再把 SP 所指的单元的内容弹出，送至寄存器对的高位(也就是说，出栈时“先低后高”)。然后 $SP\leftarrow SP+1$ 使 SP 始终指向堆栈的顶部。

应当指出，上述堆栈指令，都是对 16 位数据而言，而且高位字节总是先推入后弹出。不能单独把 8 位数据推入和弹出堆栈，当只有一个寄存器的内容要暂时保存在堆栈时，可以通过一些辅助操作来实现。堆栈指令的功能示意图如图 6-25 所示。

由于堆栈操作指令的特点是按后进先出的原则工作的，因此它们非常适合于调用子程序时保护返回地址和中断处理时保护现场，尤其适用于子程序嵌套和多重中断的场合。

通常，子程序或中断服务程序都有可能用到一至几个寄存器，而这些寄存器可能正好存放

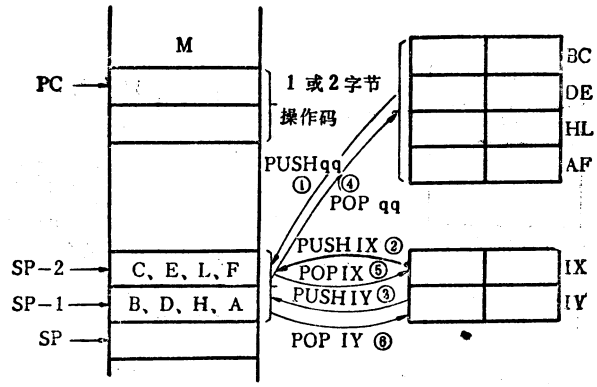


图 6-25　堆栈操作示意图

着主程序所需要的数据，因此规定子程序或中断服务程序在用到这些寄存器前必须先把该寄存器的内容保存起来(即保护现场)，而在返回主程序前又必须使这些寄存器的内容恢复原样(即恢复现场)。为此，子程序或中断服务程序可以先使这些寄存器的内容进栈，然后，在执行返主之前先将它们退栈。应当注意，退栈时必须按与进栈相反的次序进行，否则将发生错误。

下列的指令序列可以实现保护和恢复 CPU 内部所有工作寄存器的状态。

子程序：

```
PUSH  AF
PUSH  BC        } 保护现场
PUSH  DE
PUSH  HL
  ⋮             } 子程序处理程序
POP   HL
POP   DE
POP   BC        } 恢复现场
POP   AF
RET
```

在上面指令序列中，最后一条 RET 是返主指令，其功能将在本节第六部分中介绍。

堆栈指令除了上述功用外，还可以用来进行寄存器对之间的数据传送。

【例 4】 试用堆栈操作指令将寄存器对 BC 和 DE 的内容进行交换，设 SP = 2FC0H

根据题意可编写出如下程序段：

```
LD     SP , 2FC0H
PUSH   BC          ; 寄存器对 BC 的内容入栈
PUSH   DE          ; 寄存器对 DE 的内容入栈
POP    BC          ; 寄存器对 BC 的内容出栈
POP    DE          ; 寄存器对 DE 的内容出栈
```

3．寄存器对之间的数据互相交换

这类指令共有 6 条，见附录 1 表 A1-3 和附录 2 表 A2-3。

(1) DE 和 HL 之间的交换

指令格式：EX　DE, HL　；D⟷H, E⟷L

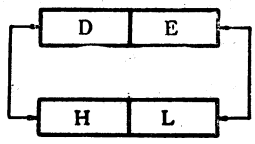


图 6-26　DE 与 HL 交换指令示意图

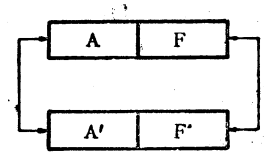


图 6-27　AF 与 A′F′ 交换指令示意图

这条单字节指令表示寄存器对 HL 和 DE 的内容互相交换，如图 6-26 所示。

(2) AF 和辅助寄存器中的 A′F′ 的内容互相交换

指令格式：EX　AF，　AF′　；A⟷A′，F⟷F′

其功能示意如图 6-27 所示。

(3) CPU 通用寄存器组与辅助寄存器中的相应寄存器的内容同时进行交换。

指令格式：EXX　；$\begin{pmatrix} BC \\ DE \\ HL \end{pmatrix} \longrightarrow \begin{pmatrix} B'C' \\ D'E' \\ H'L' \end{pmatrix}$

其功能示意图如图 6-28 所示。

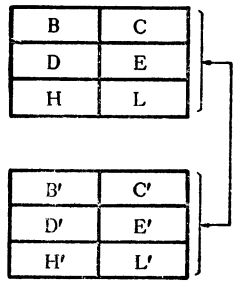


图 6-28　EXX 指令操作示意图

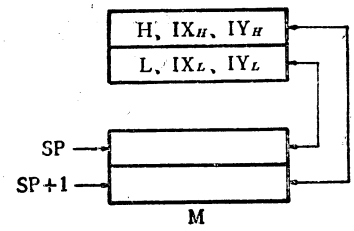


图 6-29　寄存器对与栈顶交换示意图

(4) 寄存器对 HL、IX、IY 与栈顶的内容互相交换

指令格式：

① EX　(SP)，　HL　；H⟷(SP+1)，L⟷(SP)

② EX　(SP)，　IX　；IX_H⟷(SP+1)，IX_L⟷(SP)

③ EX　(SP)，　IY　；IY_H⟷(SP+1)，IY_L⟷(SP)

其功能示意图如图 6-29 所示。

应当指出，堆栈和堆栈指示器有时可能混淆。

例如，当堆栈指示器作为目的操作数，LD 指令将影响 SP 的内容，但并不影响堆栈的内容。而和堆栈有关的交换指令(EX(SP)，HL；EX(SP)，IX；EX(SP)，IY)只改变堆栈的内容，并不改变堆栈指示器的内容。

(二) 数据块传送和搜索(Block Transfer and Block Search)指令

1. 数据块传送指令

数据块传送指令用来实现存贮器中任何两区域之间进行成组的数据传送。这类指令共有 4 条，见附录 1 表 A1-3(b) 和附录 2 表 A2-3，它们分为单个传送和成组传送两类。

(1) 单个传送

指令格式：① LDI ；(DE)←(HL)，DE←DE+1，HL←HL+1，BC←BC−1

② LDD ；(DE)←(HL)，DE←DE−1，HL←HL−1，BC←BC−1

所有数据块传送指令都用 HL 寄存器对作源地址，DE 寄存器对作目的地，BC 寄存器对作字节计数器。

上述两条指令的功能，是将以 HL 为地址指针的内存单元的内容，传送到以 DE 为地址指针的单元中去。且每执行一条，在数据块传送后自动修改地址指针，同时使字节数减 1(以 BC 作为字节计数器)，表明已按要求传送了一个数据。如果 BC≠0，则标志寄存器中的 P/V 标志位置"1"；反之，P/V 置"0"。其功能示意图如图 6-30 所示。

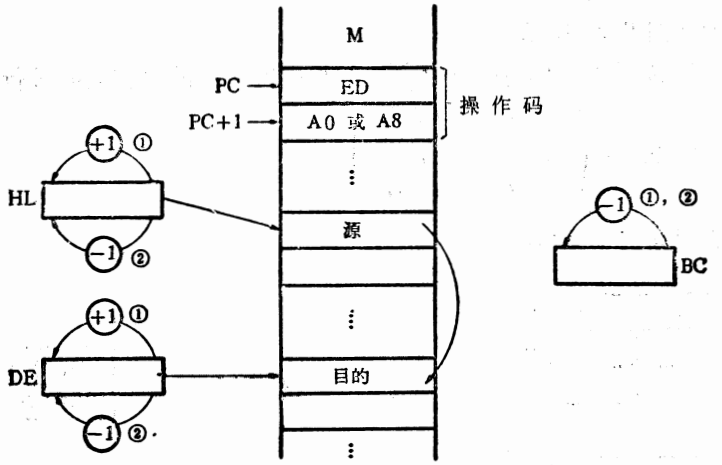


图 6-30 数据块单个传送指令示意图

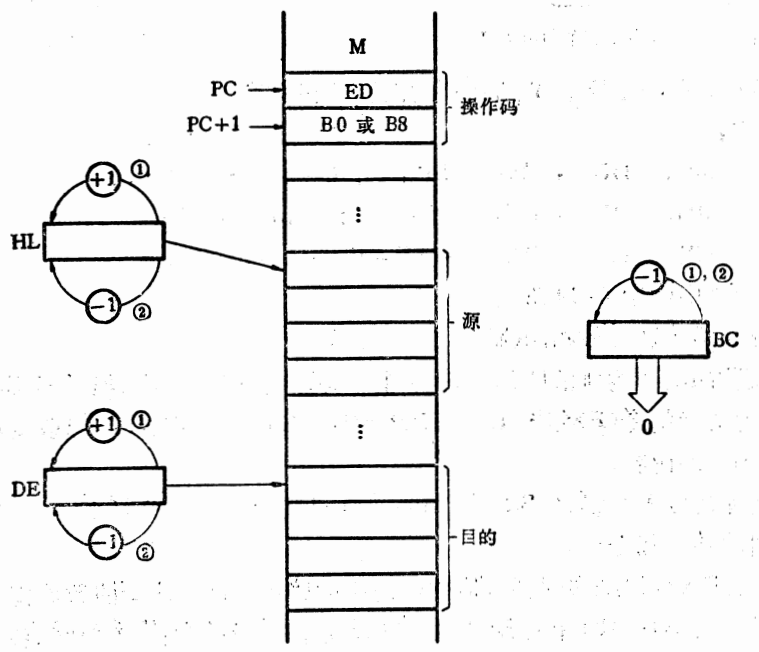


图 6-31 数据块成组传送指令示意图

(2) 重复传送(成组传送)

指令格式：① LD R ；(DE)←(HL)，DE←DE+1，HL←HL+1，BC←BC−1，重复直至 BC=0

② LDDR ；(DE)←(HL)，DE←DE−1，HL←HL−1，BC←BC−1，重复直至 BC=0

重复传送指令与单个传送指令类似，只不过此时用一条指令即可完成数据块传送，直到 BC (计数器)等于 0 为止，如图 6-31 所示。

【例 1】 把放在存贮区域 2400H～24FFH 中的 256 个字节数据传送到 2800H～28FFH 中去。其中，2400H 和 2800H 分别为数据源地址和目的地址的起始地址。

根据题意，可编出如下程序：

```
LD HL, 2400H   ; 源地址指针
LD DE, 2800H   ; 目的地址指针
LD BC, 0100H   ; 字节数
LDIR           ; 传送数据块，即把 HL 指示的内存单元的内容传送到 DE 指示的内存单元中去，过
               ; 程重复直至 BC=0 结束。
HALT
```

2. 数据块搜索指令

数据块搜索指令用来在一个数据块中搜索规定的关键字，一旦找到这一关键字，该数据块搜索完毕。

这类指令共有 4 条，见附录 1 表 A1-3(c) 和附录 2 表 A2-3。它们分为单条搜索和成组搜索两类。

(1) 单条搜索指令

指令格式：① CPI ；A−(HL)，HL←HL+1，BC←BC−1

② CPD ；A−(HL)，HL←HL−1，BC←BC−1

在执行搜索指令前，将要搜索的关键字先送入累加器 A；用 HL 寄存器对作为当前要搜索的数据块的起始地址，用 BC 寄存器对作为字节计数器，将累加器的内容与由 HL 寄存器对指示地址的内存单元的内容进行比较。比较结果反映在标志寄存器的 Z 和 P/V 中。

如果比较的结果不是全零，则 Z=0，这时若 BC≠1 即 P/V=1，则继续搜索；如果比较的

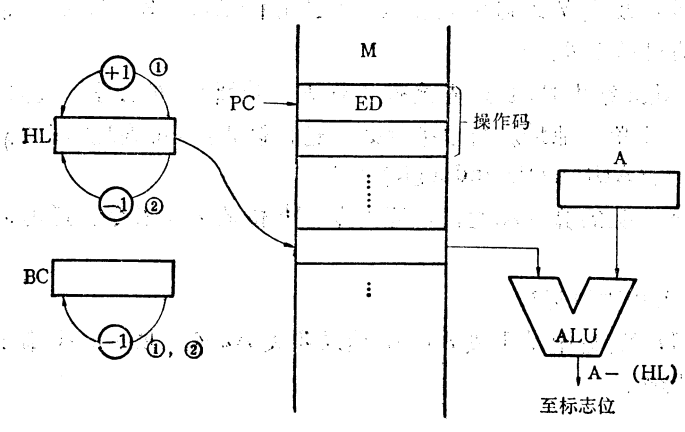


图 6-32 搜索指令示意图

结果是全零，即 A = (HL)，Z = 1 表示已搜索到关键字，就停止搜索。如果查遍整个数据块未搜索到关键字，这时 BC = 0，即 P/V = 0，也就停止搜索。其功能示意图如图 6-32 所示。

(2) 成组搜索指令

指令格式：① CPIR ；A − (HL)，HL←HL + 1，BC←BC − 1，重复直至 A = (HL) 或 BC = 0

② CPDR ；A − (HL)，HL←HL − 1，BC←BC − 1，重复直至 A = (HL) 或 BC = 0

这两种成组搜索指令与单个搜索指令的执行过程类似，仅仅是增加了重复检查的操作功能，一直进行到出现下述两种情况之一时才停止执行：

① 要搜索的关键字已搜索到，即 A = (HL)，Z = 1；

② 整个数据块已搜索结束，无关键字，即 A≠(HL)，Z = 0；BC = 0，P/V = 0。

【例 2】 假设内存中有一个由 100 个字符组成的字符串(以 ASCII 码表示)，起始地址为 2400H，要搜索其中有否 ASCII 码的符号 $ (它的 ASCII 码为 24H)，若有则记下其地址，并保存在 2800H 开始的单元中。

根据题意可编出如下程序：

```
START:  LD      A       , 24H          ; 取符号 $ 的 ASCII 代码
        LD      HL      , 2400H        ; 字符串起始地址送入 HL 寄存器对中
        LD      BC      , 0064H        ; 字节数 0064H = 100
        CPIR                           ; 在字符串中搜索
        JR      Z       , FOUND - $    ; 若 Z = 1，即 A = (HL) 已搜索到，则转 FOUND
DONE:   HALT                           ; 若 Z = 0，即未搜索到，则暂停
FOUND:  DEC     HL      ,              ; 指向"$"的数据字节地址
        LD      (2800H), HL            ; 存"$"的地址
        JP      DONE                   ; 转暂停
```

程序中 START、DONE 和 FOUND 是指令的符号地址，其含义见第八章微型计算机的汇编语言。

在执行上述程序中，若关键字搜索到，则 Z = 1，停止搜索并执行下一条指令。若"$"符号位于字符串最后，则 Z 及 P/V 均可用来表示指令终止的情况。Z = 1，表示"$"被搜索到，P/V = 0 表示字节计数值已减至 0。

因而，在字符串处理结束时，必须先检查 Z 标志。若"$"已被搜索到，这时 HL 寄存器对中的内容为此码的下一个单元地址。所以指针必须进行修正，以正确地指向"$"码的单元。

(三) 算术和逻辑(Arithmetic and logic)指令

这类指令包括 8 位数的算术和逻辑运算指令、16 位数运算指令、通用运算指令，总共 33 条。现分述如下：

1．8 位数的算术和逻辑运算

这类指令共有 17 条，见附录 1 表 A1-4，附录 2 表 A2-4。所有这类指令都对标志寄存器的有关标志位产生影响。

(1) 加法运算

加法指令有 6 条(包括带进位加法指令)

指令格式：ADD A, r ；A←A+r

ADD A, n ；A←A+n

ADD A, (HL) ；A←A+(HL)

ADD A, (IX+d) ；A←A+(IX+d)

ADD A, (IY+d) ；A←A+(IY+d)

ADC A, S ；A←A+S+CY

说明：S≡r, n, (HL), (IX+d), (IY+d)

8位数的算术运算都是在累加器与源地址中的数据之间进行的，其运算的结果都放在累加器A中，并且对有关的标志位产生影响。源地址中的数据可以是任一内部寄存器或任一内存单元的内容或是立即数n。它的地址可用HL寄存器对间接寻址，也可用变址寻址方式得到。指令功能示意图如图6-33所示。

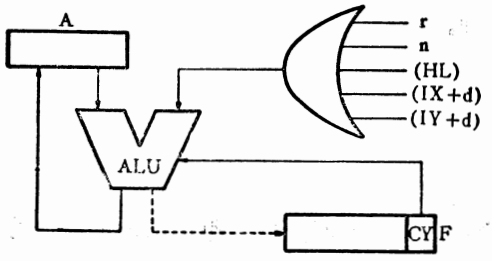


图6-33 加法指令示意图

带进位加法运算与上述加法运算类似，只是还要加上低位字节相加时产生的进位，它主要用于多字节的加法运算。下面举几个例子加以说明：

【例1】 试求两个8位数之和。设这两个操作数分别存放在地址为2400H和2401H的内存单元中，相加结果存放在2042H内存单元中。

解：根据题意可编出如下简单加法程序：

```
START: LD   A     , (2400H) ; 将2400H单元中的数取入累加器A
       LD   B     , A       ; 将累加器A的内容送寄存器B暂存
       LD   A     , (2401H) ; 将2401H单元中的数送累加器A
       ADD  A     , B       ; 将累加器A中的数和寄存器B中的数相加，结果在累加器A中
       LD   (2402H), A      ; 将累加器A中的结果送内存单元2402H保存
       HALT                 ; 暂停
2400H 操作数1
2401H 操作数2
2402H 操作结果
```

【例2】 试编出双精度(16位)相加的程序。

示范题：672AH+14F8H=7C22H

(2040H)=2AH

(2041H)=67H

(2042H)=F8H

(2043H)=14H

结果：(2044H)=22H

(2045H)=7CH

解：用8位数加法指令实现16位数的加法必须分两步进行，先做低8位，后做高8位其源程序如下：

```
LD    HL      , (2040H)    ; 被加数低位字节地址 2040H 送 HL 寄存器对
LD    A       , (2042H)    ; 加数低位字节地址 2042H 的内容取入 A
ADD   A       , (HL)       ; 形成和的低位部分
LD    (2044H) , A          ; 和的低位字节存入 2044H 单元
LD    A       , (2043H)    ; 加数高位字节地址 2043H 的内容送 A
INC   HL                   ; 地址指针加 1，指出被加数高位字节地址 2041H
ADC   A       , (HL)       ; 形成和的高位部分并与低位部分的进位相加
LD    (2045H) , A          ; 和的高位部分存入 2045H 单元
HALT                       ; 暂停
```

(2) 减法运算

减法指令有两条。包括不带借位和带借位减法两种。

指令格式：

SUB S ; A←A－S

SBC A, S ; A←A－S－CY

S≡r, n, (HL), (IX＋d), (IY＋d)

与带进位加法运算指令类似，执行带借位减法指令时，在考虑两个操作数相减的同时，还必须考虑从低位来的借位，即必须减去低位字节相减时产生的借位。

【例 1】 假设累加器 A＝3AH, CY＝1，执行 SBC A, 7CH 指令的执行过程如下：

实际上，这时机器先将减数 7CH 与进位 1 之和转换为对 2 的补码，然后再和累加器中的被减数相加，即把减法变为补码相加。

首先对 7CH＋进位＝7DH＝01111101B 变补得：

$$[-7DH]_{补} = 10000011$$

然后执行 A＋$[-7DH]_{补}$

$$\begin{array}{r} A = 00111010B \\ +)\ [-7DH]_{补} = 10000011B \\ \hline BDH = 10111101B \end{array}$$

符号位是 1, S＝1 ←

无进位，有借位 CY＝1 ←

→ 结果非 0, Z＝0

→ 无进位，有借位，H＝1

0 ⊕ 0＝0, P/V＝0

减指令，N＝1

应当指出，上式补码相加时，产生的进位输出将由机器硬件逻辑求反后置入标志位 CY 中。本例中补码相加结果进位输出为 0，经机器内部求反得 CY＝1，，表示有借位。

同时，还应注意，进位位的建立不考虑运算结果的代数符号。无论结果是正或是负，都可能出现进位。而对溢出的检测是监视符号位进位的进出情况。当进出符号位的进位不同时，就称该状态为“溢出”，也就是说，此时运算结果超过带符号数所能容纳的范围。这种判断溢出的方法就是第二章所介绍的双进位位判断溢出的方法。

最后，还要指出，补码相加的结果必是补码，如果需要求出该数的原码，还得将结果再次求补得：11000011B＝－43H。

【例 2】 试编出双精度(16 位)相减程序。

示范题：E303H－A004H＝42FFH

(2040H)＝03H

(2041H)＝E3H

(2042H)＝04H

(2043H)＝A0H

结果：(2044H)＝FFH

(2045H)＝42H

解：8位数减法指令实现16位数相减，也得分两步进行，先做低8位，后做高8位。其步骤如下：

首先做低位字节减法：

$$\begin{array}{r} 03H = 00000011B \\ +)\ [-04]_{补} = 11111100B \\ \hline 11111111B \end{array}$$

无进位、有借位、CY＝1

然后再做高8位字节减法，又分两小步。先将减数A0H与借位1之和转换为对2的补码，然后再和被减数相加。

即先对A0H＋1＝A1H变补得：01011111B＝5FH

$$\begin{array}{r} E3H = 11100011B \\ +)\ [-A1H]_{补} = 01011111B \\ \hline 01000010B \end{array}$$

符号位是0，S＝0　　结果非0，Z＝0

有进位，无借位 CY＝0　　有进位，无借位，H＝0

减指令 N＝1

1 ⊕ 1＝0，P/V＝0

编写源程序如下：

```
LD    HL      , 2042H     ; 减数低位字节地址2042H送HL寄存器对
LD    A       , (2040H)   ; 被减数低位单元2040H的内容取入A
SUB   (HL)                ; 形成差的低位部分
LD    (2044H) , A         ; 差的减位字节存入2044H
LD    A       , (2041H)   ; 被减数高位单元2041H的内容送A
INC   HL                  ; 地址指针加1，指出减数高位单元地址2043H
SBC   A       , (HL)      ; 形成差的高位部分减去低位部分的借位
LD    (2045H) , A         ; 差的高位部分存入2045H单元
HALT                      ; 暂停
```

(3) 比较指令

指令格式：CP S　；A－S　S≡r, n, (HL), (IX＋d), (IY＋d)

它与减法指令相似，它们的区别仅在于相减之后的结果不送回累加器A，即累加器中的内容保持不变。相减的结果反映在有关标志位中。若A＝S，则Z＝1，否则Z＝0。所以可以由Z标志来判断A是否与S相等。

【例 1】 要求从一串 ASCII 码字符 (7 位,其最高位为 0) 中寻找第一个非空格符。

字符串从内存单元 2042H 开始存放。第一个非空格符的地址放入内存单元 2040H 和 2041H (高 8 位放 2041H 单元)。空格字符在 ASCII 码中是 20H, 用 SP 表示。

示范题: 内存单元 (2042H) = 20H = SP
(2043H) = 20H = SP
(2044H) = 20H = SP
(2045H) = 46H = F
(2046H) = 20H = SP

结果: (2040H) = 45H
(2041H) = 20H

完成上述任务的程序如下:

```
        LD    HL        , 2041H     ; 指向字符串前一字节的地址
        LD    A         , 20H       ; 取 ASCII 空格符,以供比较
CHBLK:  INC   HL                    ; 地址指针 HL 加 1
        CP    (HL)                  ; 字符 = ASCII 空格字符吗?
        JR    Z         , CHBLK     ; 是,继续检查字符
        LD    (2040H)   , HL        ; 否,存第一个非空格字符的地址
        HALT
```

通常,比较指令更多的是用来判断两数的大小。可分两种情况讨论:

① 算术比较,即两个不带符号的正数进行比较

第二章已述及,两个不带符号的数相减,结果的正负是以借位 CY 状态来表示的。

若 A 为累加器的内容, X 为 CP 指令的第二操作数,则执行比较指令后对标志位产生的影响如下:

若 A = X, 则 Z = 1; A ≠ X, 则 Z = 0。

若 A < X, 则 CY = 1; A ≥ X, 则 CY = 0。

(A 和 X 为不带符号的二进制数)

【例 2】 要求将内存单元 2040H 和 2041H 中较大的数放入内存单元 2042H, 假设内存单元 2040H 和 2041H 的内容都是不带符号的二进制数。

设: (2040H) = 3FH
(2041H) = 2BH

结果: (2042H) = 3FH

编写的源程序如下:

```
        LD    HL    , 2040H
        LD    A     , (HL)      ; 取第一操作数
        INC   HL
        CP    (HL)              ; 第二个操作数较大吗?
        JR    NC    , DONE      ; 否,转 DONE
        LD    A     , (HL)      ; 是,取第二操作数
DONE:   INC   HL
        LD    (HL)  , A         ; 存较大操作数(第一操作数)
        HALT
```

② 代数比较，即两个带符号的数的比较

这时，又可分以下几种情况：

第一，若两个都是带符号的正数，则可由符号位标志 S 来判断大小。若结果为正，则 A＞X；结果为负，则 A＜X。

【例 3】 若有两个正数分别存放在内存单元 2040H 和 2041H 中，要求比较它们的大小，并把较大的数存入内存单元 2042H 中去。其源程序如下：

```
        LD      A,      (2040H)
        LD      HL,     2041H
        CP      (HL)
        JP      P,      GREAT   ；若 S=0，即 (2040H)＞(2041H)，则转出
        LD      A,      (HL)
CTREAT: LD      (2042H), A
        HALT
```

第二，若参与比较的数有正有负，应先判断结果是否溢出，然后再利用符号标志位 S 判断大小。

下面我们分四种情况分别加以说明：

若 A＞0，X＞0，执行 CP S 指令后，若符号标志 S＝0，则 A≥X；反之 A＜X。

若 A＜0，X＜0，运算时不会发生溢出，因此，也可用符号标志来判断两个数的大小。

若 A＞0，X＜0，按理比较的结果应是 A＞X，且比较的结果应是正数。但若相减后的结果超出了带符号数的允许范围 ＋127～－128，就产生溢出错误。因此，此时必须先确定运算的结果是否有溢出，若结果无溢出，即 P/V＝0，则仍为 S＝0 时 A≥X；S＝1 时 A＜X。若结果有溢出，即 P/V＝1，则 S＝0 时，A＜X；S＝1 时，A＞X。

若 A＜0，B＞0，按理比较的结果应是 A＜X，且比较的结果应是负数。但若相减后的结果超出了带符号数的允许范围 ＋127～－128，也会发生溢出错误。

综上所述，判断两个带符号数的大小的方法可概括为：

① 当进行比较的两个数符号相同时，不会发生溢出，即 P/V＝0，故可用符号标志 S 判断两个数的大小，即若 S＝0 时，A＞X；S＝1 时，A＜X。

② 当进行比较的两个数符号相异时，可能发生溢出，即 P/V＝1，此时，S＝1，A＞X；S＝0，A＜X。

Z80 CPU 有专用于检测溢出的标志位，但是 Intel 8080A/8085A CPU 没有设置检测溢出的硬件，如何保证运算结果不发生溢出错误，这是程序员的责任。图 6-34 示出一种用同号异号方法判断两个带符号数 A 和 B 的大小的流程图。

(4) 增量、减量操作

指令格式：

```
① INC  r                ；r←r＋1
② DEC  r                ；r←r－1
③ INC  (HL)             ；(HL)←(HL)＋1
④ DEC  (HL)             ；(HL)←(HL)－1
⑤ INC  (IX＋d)/(IY＋d)  ；(IX＋d)/(IY＋d)←(IX＋d)/(IY＋d)＋1
⑥ DEC  (IX＋d)/(IY＋d)  ；(IX＋d)/(IY＋d)←(IX＋d)/(IY＋d)－1
```

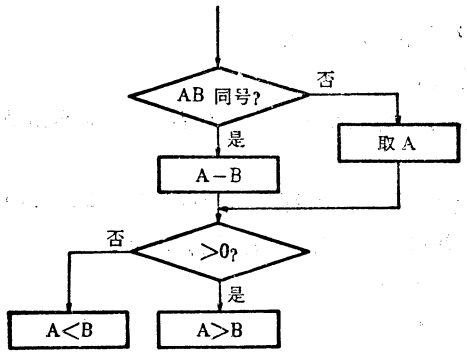


图 6-34　用同号异号法判断两个带符号数大小的流程图

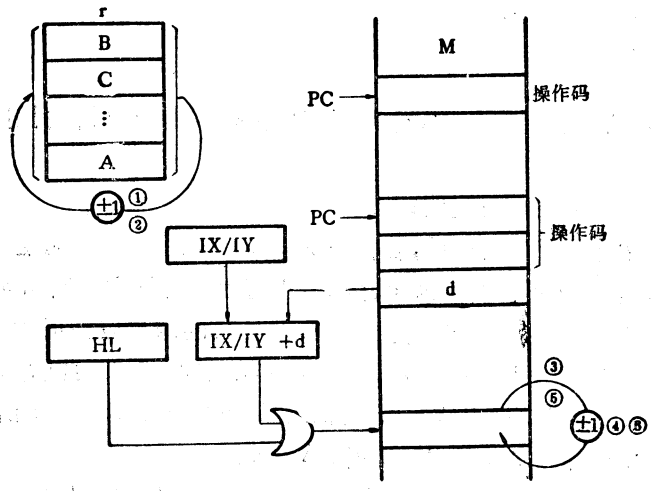


图 6-35　增量、减量指令功能示意图

这类指令中，①、②、③、④为单字节指令，⑤、⑥为三字节指令。

这类指令可以使 CPU 内部寄存器或存贮单元加 1 或减 1。执行时，除了进位标志 CY 外，对其它标志位都产生影响，其功能示意如图 6-35 所示。

通常利用这类指令作为计数器来控制循环次数。

(5) 逻辑操作

指令格式：

```
AND   S   ; A←A∧S
OR    S   ; A←A∨S
XOR   S   ; A←A⊕S
      S≡r, n, (HL), (IX+d), (IY+d)
```

这类指令有 3 种，都是单字节指令。当它们执行时，对有关标志位将产生影响（见附录 2 表 A2-4）。逻辑操作的特点是按位进行的。各位的操作对其它位毫无影响。因而，对逻辑操作指令而言，不存在进位或半进位问题。其所以设置这两个标志位，主要是为了产生附带的功能。

AND 指令用来屏蔽数据字节中无用的部分，保留有用的部分。

【例 1】 将内存单元 2040H 的低 4 位存入内存单元 2041H 的低 4 位，清除内存单元的高

4位，可用如下源程序：

```
LD      A,        (2040H)   ；取数据
AND     00001111B           ；屏蔽高4位
LD      (2041H),  A         ；存结果
HALT                        ；暂停
```

源程序中用二进制代码表示屏蔽字，可以使人一目了然。

OR 指令可以用来组合信息，如把两个数组合为一个数，或者可以在一个数据的某些确定位上置位。

【例 2】 若累加器A中的 BCD 码为 0000jjjj，另一个 BCD 码 kkkk0000 在寄存器 B 中，将它们合并为一个数的源程序如下：

```
LD  A,  0000jjjj    ；第一个 BCD 码送 A
LD  B,  kkkk0000    ；第二个 BCD 码送 B
OR  B               ；合并两个 BCD 码
```

应当指出，AND A；OR A；XOR A 都可以用于清进位位 CY。XOR A 还可用于清累加器 A(用自身相异或，结果必是每一位均为 0)。AND A，OR A 对累加器内容无影响。

2．16位数算术运算

这类指令共有 11 类。包括 16 位加法指令 3 条，16 位带进位加法和带借位减法指令各 1 条，16 位增量指令和 16 位减量指令各 3 条。如附录 1 表 A1-5 和附录 2 表 A 2-5 所列。这类指令中，16 位加法仅影响标志位 CY 和 N。16 位带进位的加法和带借位的减法指令执行后，其结果对所有标志位都产生影响，但 16 位增量和减量指令执行后其结果对标志位无影响。

(1) 加法运算

指令格式：

```
ADD  HL,  SS  ；HL←HL+SS, SS≡BC、DE、HL、SP
ADD  IX,  PP  ；IX←IX+PP, PP≡BC、DE、IX、SP
ADD  IY,  qq  ；IY←IY+qq, qq≡BC, DE, IY、SP
ADC  HL,  SS  ；HL←HL+SS+CY, SS≡BC, DE, HL, SP
```

【例 1】 设寄存器对 HL = 034AH，BC = 214CH 执行 ADD HL, BC 指令的过程如下：

```
      034AH = 0000001101001010B
+)    214CH = 0010000101001100B
------------------------------
      2496H = 0010010010010110B
              ↑                ↑
无进位，CY = 0←┘                └→无进位 H = 0   加指令 N = 0
```

此外，执行 ADD HL, HL 指令可以实现 16 位逻辑左移。

【例 2】 前述 8 位数加法运算的例 2，若用 16 位加法运算指令编程，则其源程序可从 9 条指令缩短至 4 条指令。

```
LD    HL,       (2040H)   ；取第一个 16 位数
LD    DE,       (2042H)   ；取第二个 16 位数
ADD   HL,       DE        ；两个 16 位数相加
LD    (2044H),  HL        ；存 16 位结果
```

HALT　　；暂停

【例 3】 设寄存器对 HL = A536H, BC = 1044H, 设二数均为无符号数, CY = 1

执行 ADC HL, BC 指令的过程如下：

A536H = 1010 0101 0011 0110B

1044H = 0001 0000 0100 0100B

+）进位 CY = 1

B57BH = 1011 0101 0111 1011B

符号位是 1, S = 1 ←

无进位, CY = 0 ←

→ 结果非 0, Z = 0

→ 无进位　加指令 N = 0

0 ⊕ 0 = 0, P/V = 0

ADC 指令常用于多字节数的加法程序。

(2) 减法指令

SBC HL, SS ；HL←HL − SS − CY ，SS≡BC、DE、HL、SP

【例 1】 设寄存器对 HL = F4A2H, BC = A034H (设二数均为无符号数), CY = 0, 执行 SBC HL, BC 指令过程如下：

F4A2H = 1111 0100 1010 0010B

+）[−(A034H + 0)]补 = 0101 1111 1100 1100B

546EH = 0101 0100 0110 1110B

符号位是 0, S = 0 ←

有进位, 无借位 CY = 0 ←

→ 结果非 0, Z = 0

→ 有进位, 无借位　减指令, N = 1

1 ⊕ 1 = 0　P/V = 0

【例 2】 试编制两个 4 字节数的减法程序，设两个操作数分别存放在以 2100H 和 2200 H 为起始地址的内存单元中，低位字节放在前面。结果存放在以 2100H 为起始地址的内存单元中。

其源程序如下：

```
LD    HL,    (2100H)   ；取操作数 1 的两个低位字节
LD    DE,    (2200H)   ；取操作数 2 的两个低位字节
OR    A                ；清 CY 标志
SBC   HL,    DE        ；形成差的低位字节
LD    (2100H), HL      ；存结果的两个低位字节
LD    HL,    (2102H)   ；取操作数 1 的两个高位字节
LD    DE,    (2202H)   ；取操作数 2 的两个高位字节
SBC   HL, DE           ；形成差的两个高位字节
LD    (2102H), HL      ；存结果的两个高位字节
HALT                   ；暂停
```

由于 16 位数的减法指令没有不带借位的，因此在作两个低位字节减法之前应用 ORA

指令清除 CY 标志。

(3) 增量和减量指令

指令格式：

① INC SS ；SS←SS+1，SS=BC、DE、HL、SP

② DEC SS ；SS←SS−1

③ INC IX ；IX←IX+1

④ DEC IX ；IX←IX−1

⑤ INC IY ；IY←IY+1

⑥ DEC IY ；IY←IY−1

这类指令通常用于修改地址指针或用作计数器，如前面例子所示。应当指出，这类指令对标志位毫无影响，应用时必须特别注意。

例如，当循环次数超过 256 时，如果用某一寄存器对作为计数器，此时，不能用下列指令组来判断循环次数是否执行完毕。

```
      ⋮
    DEC  BC
    JR   NZ, LOOP
```

因为 DEC BC 并不影响标志位，故条件转移指令中判断 Z 标志位并不取决于 DEC BC，这就无法得到正确的结果。在这种情况下，若要正确判断标志位，须用下列指令：

```
      ⋮
DEC  BC
LD   A,   B
OR   C          ；只有当 B 与 C 全为 0 时，或的结果才为 0
JR   NZ,  LOOP
      ⋮
```

3. 十进制算术调整和通用算术指令

这类指令有 5 条，见附录 1 表 A1-6(a) 和附录 2 表 A2-6。

(1) 十进制算术调整 DAA

DAA 指令是为完成二-十进制的加减运算而对累加器的内容进行调整的指令。应当注意，DAA 指令必须紧跟在每条加或减法指令的后面，在一系列加或减指令之后，只用一条 DAA 指令是不行的。

关于两个 BCD 码进行加法或减法运算时，进行修正的规定和方法已在第二章分析过。

简单地说，对于两个 BCD 码加法而言，若相加后的低 4 位(或高 4 位)二进制数大于 9 或大于 15 (即低 4 位有进位 H=1 或高 4 位有进位 CY=1)，则应对低 4 位(或高 4 位)加 6 修正；对两个 BCD 码减法而言，若个位向十位有借位，即 H=1 或十位向更高位有借位，即 CY=1，则应对低 4 位或高 4 位减 6 修正。DAA 指令的这些调整操作由 CPU 内的十进制调整电路自动进行，使用者只要在上述加、减运算指令后面紧跟一条 DAA 指令即可。

下面举例说明 DAA 指令的应用：

【例】 试编制两个 4 字节 BCD 制数多精度相加的程序，设两个数分别为 36701985 和 12663459，并分别存放于下列内存单元：

(2040H) = 04　(数的长度)
(2041H) = 85　; (2061H) = 59
(2042H) = 19　; (2062H) = 34
(2043H) = 70　; (2063H) = 66
(2044H) = 36　; (2064H) = 12

相加结果 49365444 存放在从 2041H 开始的内存单元中。

编制的源程序如下：

```
START:  LD    HL,    2040H
        LD    B,     (HL)      ; 计数 = 数的长度(以字节计)
        INC   HL               ; 指针 1 指向数 1 的第一个字节
        LD    DE,    2061H     ; 指针 2 指向数 2 的第一字节
        XOR   A                ; 清 CY 标志
LOOP:   LD    A,     (DE)      ; 带进位加
        ADC   A,     (HL)
        DAA                    ; 进行十进制调整
        LD    (HL),  A         ; 存结果
        INC   DE
        INC   HL               ; 修改指针
        DEC   B                ; 计数
        JR    NZ,    LOOP-$    ; 若 B≠0, 转下一字节相加
        HALT
```

在 Z80 CPU 中实现多精度减法，只要把上述程序中的 ADC 换成 SBC，即将带进位加换成带借位减就可以把上述程序变成多字节减法程序。同样地，DAA 指令可以对减法的结果进行相应的修正。

(2) 通用算术类指令

指令格式：

① CPL　; $A \leftarrow \overline{A}$
② NEG　; $A \leftarrow \overline{A} + 1$
③ CCF　; $CY \leftarrow \overline{CY}$
④ SCF　; $CY \leftarrow 1$

前两条指令是对累加器 A 的内容求对 1 的补码和对 2 的补码；后两条指令是对进位标志 CY 操作的。这些指令的用法简单，这里不一一举例了。

(四) 循环(Rotate)和移位(Shift)指令

Z80 指令系统中的循环与移位指令有 16 条，可以对累加器 A，任一通用寄存器 B、C、D、E、H 或 L 的内容，或任一内存单元中的内容进行循环或移位操作。如附录 1 表 A1-7 和附录 2 表 A2-7 所示。执行这类指令将对标志产生不同情况的影响。按照这类指令的功能可分以下几类：

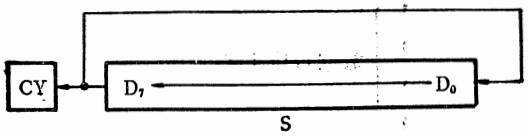
1. 循环移位

这类指令又分为不带进位和带进位的循环移位。所有的循环都保留全部的 8 位或 9 位数据，并把 CPU 寄存器或内存单元的内容向左或向右移动 1 位。

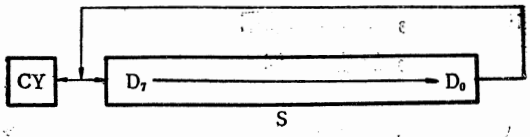
(1) 不带进位的循环移位

指令格式：

RLC S (Rotate Left Circular); S=r, (HL), (IX+d), (IY+d)



RRC S (Rotate Right Circular)

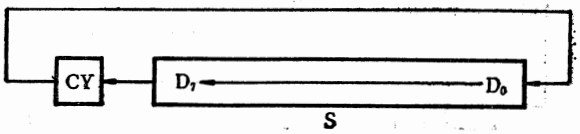


这类指令当移位 8 次后，S 中的内容恢复初始状态。所以此类指令主要用于要求按位检测 S 中的内容(移至 CY 中检测)，而又不允许破坏 S 中的内容的场合。

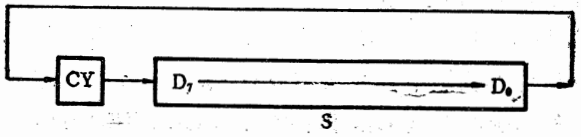
(2) 带进位循环移位

指令格式：

RL S (Rotate Left)



RR S (Rotate Right)



这类指令将进位位 CY 包含在循环之内。实际参加移位的是 9 位。当移位 9 次后，S 中及 CY 的内容恢复初始状态。

Z80 中还有另外 4 条与 8080A/8085A 相同的对累加器 A 的循环移位指令，其执行时间只有 4 个 T 时态，可以缩短指令的执行时间和减少指令的字节数。而对内部寄存器循环移位需要 8 个 T 时态，用 HL 间接寻址需要 15 个 T 时态，用变址寻址则需 23 个 T 时态，所以应尽可能采用对累加器 A 循环移位指令。

【例 1】 将内存单元 2040H 的内容拆成两段，每段 4 位，并将它们分别存入内存单元 2041H 和 2042H，即内存单元 2040H 的高 4 位放入 2041H 的低 4 位，2040H 的低 4 位放入 2042H 的低 4 位。并清除内存单元 2041H 和 2042H 的高 4 位。

示范题：(2040H) = 3FH

结果：　(2041H) = 03H

　　(2042H) = 0FH

编制的源程序如下：

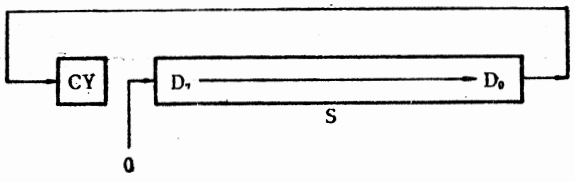
```
LD      HL,     2040H
LD      A,      (HL)        ; 取数据
LD      B,      A
RRA                         ;
RRA                         ;  } 数据右移4次
RRA                         ;
RRA                         ;
AND     00001111B           ; 屏蔽高4位
INC     HL
LD      (HL),   A           ; 存高4位
LD      A,      B           ; 恢复原始数据
AND     00001111B           ; 屏蔽高4位
INC     HL
LD      (HL),   A           ; 存低4位
HALT
```

2．逻辑和算术移位

(1) 逻辑右移指令

指令格式：

SRL S (Shift Right Logical)



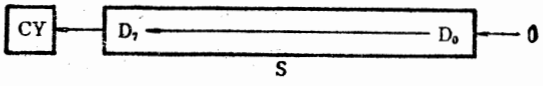
此指令使指定单元的内容右移一位，最高有效位置"0"。若S中的内容为无符号数，则右移一位相当于S内容除2；若S为带符号的正数，则与算术右移指令SRA S的功能相同。

例如上述循环移位指令的例子中，也可用SRL指令直接进行逻辑移位。且最后不必用AND指令。但是SRL A指令所需的时间和存贮单元都比RRA的多一倍。

(2) 算术左移指令(逻辑左移指令)

指令格式：

SLA S (Shife Left Arithmetic)



此指令将指定单元的内容左移一位，最低有效位置"0"。结果相当于原内容乘2。显然，这是一种典型的逻辑左移操作，尽管Z80中命名其为算术左移指令。

【例1】 要求对内存单元2400H的内容X乘10。而X*10 = X*(8+2)。

可用下列源程序：

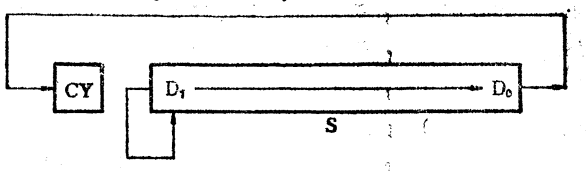
```
START:  LD      HL,     2400H
        LD      A,      (HL)        ; 取操作数
        SLA     A                   ; *2
        LD      B,      A           ; 暂存于B
```

```
SLA     A          ; *4
SLA     A          ; *8
ADD     A,      B  ; X*(8+2)=X*10
LD      (HL),   A
HALT
```

(3) 算术右移指令

指令格式：

SRA S (Shift Right Arithmetic)



此指令将指定单元的内容右移一位，最高有效位内容不变，即带符号数右移后符号位不变。它可用于带符号数的运算，右移一位符号不变，数值部分除 2。

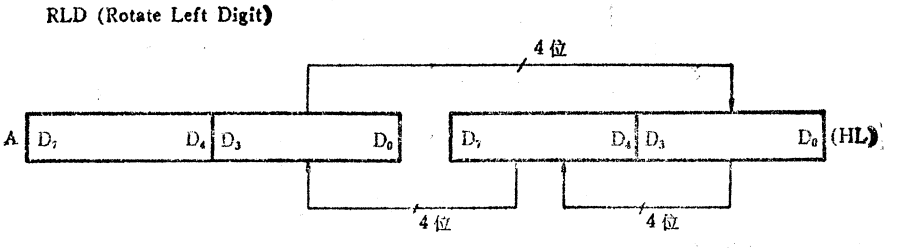
【例 2】 要求对内存单元 2500H 的内容除以 8，把小数部分保留下来，设小数部分保留在寄存器 B 中，小数点设在 A、B 两寄存器之间，则 B 中各位的权分别为 1/2、1/4、1/8… 等。

编制源程序如下：

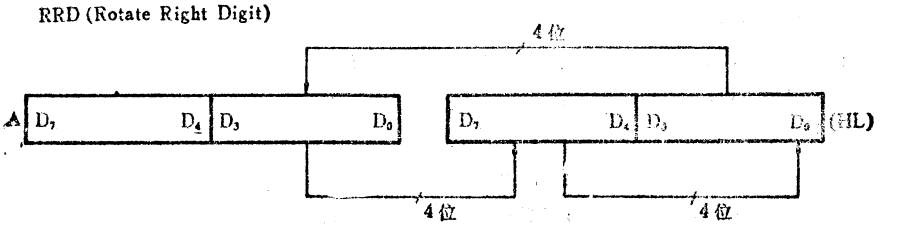
```
DIVER:  LD      A,   (2500H)   ; 取操作数 Nx
        LD      B,   00        ; 清小数部分寄存器 B
        SRA     A              ; A 的最低位进入 CY, Nx/2
        RR      B              ; 由 B←CY 的最高位保存 1/8
        SRA     A              ; Nx/4
        RR      B              ; 保存 1/4
        SRA     A              ; Nx/8
        RR      B              ; 保存 1/2
        HALT
```

(4) BCD 码移位

RLD (Rotate Left Digit)



RRD (Rotate Right Digit)



这类BCD码移位指令与上述的移位指令的区别在于，每执行一次，它们可对指定单元的内容左移或右移4位，而不是一位。所以，利用这类指令很容易实现ASCII码与BCD码之间的转换。

【例3】 若把从键盘输入的用ASCII码表示的0～9，存放在以3000H为起始地址的10个单元中，用以下程序可把它们转换成BCD码仍存放在以3000H为起始地址的单元中，这时仅需5个单元。

编制源程序如下：

```
START: LD    IX,  3000H   ; 设置存放ASCII码的内存起始地址指针
       LD    HL,  3000H   ; 设置存放BCD码的内存起始地址指针
       LD    B,   05      ; 设置计数值
LOOP:  LD    A,   (IX+0)  ; 取ASCII码字符
       RRD                ; BCD码移入以HL为地址指针的内存单元
       LD    A,   (IX+1)  ; 取下一个ASCII码字符
       RRD                ; BCD码移入以HL为地址指针的内存单元
       INC   IX           ; }
       INC   IX           ; } 得到下一个ASCII码地址
       INC   HL           ; 得到下一个BCD码的内存单元地址
       DEC   B            ; 计数值减1
       JR    NZ,  LOOP    ; 未转换结束，转继续
       HALT
```

程序执行前后，内存单元内容变化的示意图如图6-36所示。

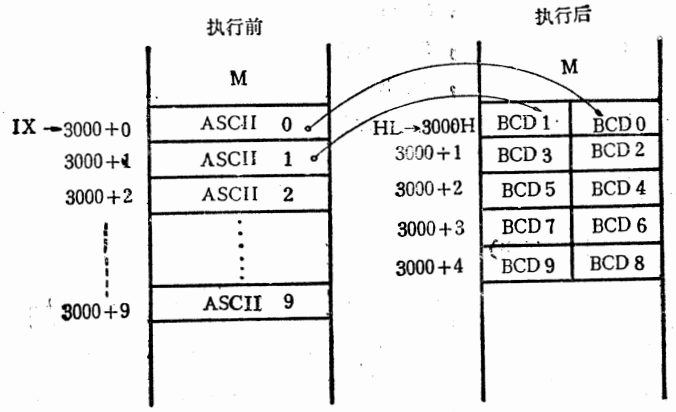


图6-36 由ASCII码→BCD码存贮器示意图

【例4】 若有100个两位BCD码，放在以3500H为起始地址的内存单元中，要求将它们转换成用ASCII码表示并放在以3000H为起始地址的内存单元内。

编制源程序如下：

```
START: LD    IX,  3000H   ; 设置存放ASCII码的内存起始地址指针
       LD    HL,  3500H   ; 设置存放BCD码的内存起始地址指针
       LD    B,   64H     ; 设置计数值
       XOR   A            ; 清A
       OR    30H          ; 为转换成ASCII码，将A先置成30H
```

```
LOOP:  RRD                      ; BCD 码移入 A，组合成 ASCII 码
       LD      (IX+0),  A       ; 将组合的 ASCII 码放入以 3000H 为起始地址的内存单元
       RRD
       LD      (IX+1),  A
       INC     IX               ;
       INC     IX               ; } 得到下一个 ASCII 码地址
       INC     HL               ; 得到下一个 BCD 码地址
       DEC     B                ; 计数值减 1
       JR      NZ,      LOOP    ; 未转换结束则继续
       HALT
```

如上所述，Z80 具有丰富的移位指令。Intel 8080A/8085A 只有 4 条且只能对累加器 A 的内容进行循环移位的指令，而 Z80 具有 16 类 72 条循环和移位指令，它们能对任何寄存器或内存单元的内容进行逻辑、算术和循环移位以及在累加器和内存单元之间进行 BCD 码移位。

（五）位操作 (Bit Manipulation) 指令

Z80 的位操作有 9 类基本指令，240 种操作码，它们可归结为位测试、位置位和位复位三类操作。这类指令可以对任一内部寄存器中的任一位或对以 HL 间接寻址和以 IX、IY 变址寻址的内存的任一单元的任一位进行测试、置位和复位。如附录 1 表 A1-8 和附录 2 表 A2-8 所示。这类指令中，除位测试指令外，其余两类对标志位都不产生影响。

指令格式：

① BIT b, S ; $Z \leftarrow \overline{S_b}$

② SET b, S ; $S_b \leftarrow 1$

③ RES b, S ; $S_b \leftarrow 0$

S≡r, (HL), (IX+d), (IY+d)

其中 b 用三位二进制数表示 0—7，它指明 S 中的某一位。

【例 1】 要求对一组带符号数求补后，再放回原内存单元，数组长度放在 2040H 单元，数组起始地址为 2041H。

示范题：假设：(2040H) = 06H
(2041H) = 2EH
(2042H) = A6H
(2043H) = F5H
(2044H) = 41H
(2045H) = 81H
(2046H) = 35H

结果：(2041H) = 2EH
(2042H) = DAH
(2043H) = 8BH
(2044H) = 41H
(2045H) = FFH
(2046H) = 35H

编制的源程序如下：

```
START: LD   HL,  2040H  ; }
       LD   B,   (HL)   ; } 计数值送寄存器 B
   LO: INC  HL          ; HL 指向数组起始地址
       LD   A,   (HL)   ; 取数组中的一个带符号组
       BIT  7,   A      ; 测试符号位
       JR   Z,   LI-$   ; 是正数则转 LI
       NEG              ; 是负数则对其变补(连符号位求反加 1)
       SET  7,   A      ; 恢复符号位
       LD   (HL), A     ; 存入原内存单元
   LI: DEC  B           ; 计数值减 1
       JR   NZ,  LO-$   ; 未结束则转出
       HALT             ; 暂停
```

【例 2】 要求将累加器A中数据的最高位代码取反。

编制源程序如下：

```
START: BIT   7,      A
       JR    NZ,     LOOP-$
       SET   7,      A
       JR    LOOP2-$
LOOP 1: RES  7,      A
LOOP 2: HALT
```

(六) 转移(Jump)指令

Z80 的转移指令有 11 条基本指令，18 种操作码。如附录 1 表 A1-9 和附录 2 表 A2-9 所示。由表中得知，所有转移指令都不影响标志位状态。

Z80 除了具有 8080A/8085A 标准的转移指令(JP)以外，还新增加了以下几种：

1．两字节的相对寻址转移指令(JR)，这类指令的第二字节是一个 8 位的带符号的对 2 补码的位移量。这与 8080A 的三字节转移指令相比，可以节省内存容量。

2．两条变址值转移指令用来把变址寄存器 IX (或 IY)的内容置入程序计数器 PC 中去。

3．循环控制转移指令(DJNZ)，它对于各种迭代循环程序非常有用。

现分述如下：

1．无条件转移(Unconditional Jump)

(1) 标准的无条件转移指令

指令格式：

① JP nn ; PC←nn(三字节)

② JP (HL) ; PC←HL(单字节)

③ JP (IX) ; PC←IX }
④ JP (IY) ; PC←IY } (双字节)

这类指令是无条件地使程序转移到指定的地址。

应当指出，②、③、④指令中，(HL)、(IX)、(IY)，虽加了一个括号，但实际上是将其本身的内容置入 PC 而不是指以它们为地址的单元的内容。

这类指令的功能示意图如图 6-37 所示。

(2) 无条件相对转移指令

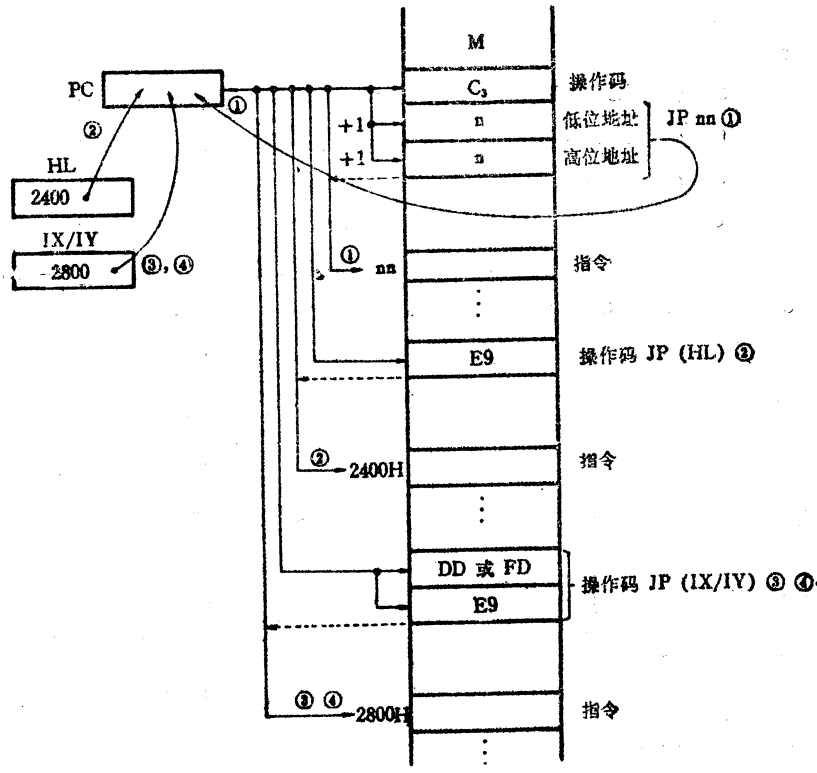


图 6-37　标准的转移指令功能示意图

指令格式：JR　e　；PC←PC+e

其中，e 是相对于现行程序计数器(PC)地址的位移量，它是一个带符号数的对 2 补码。关于相对转移寻址方式，我们已在 §6-2 寻址方式中详述过，这里不再重复。

2．条件转移(Conditional Jump)

(1) 标准的条件转移指令

指令格式：

JP　CC,　nn

若条件 CC 满足，则 PC←nn，否则顺序执行下一条指令。其中转移条件 CC 如表 6-7 所示。指令功能示意图如图 6-38 所示。

表 6-7　转移指令条件表

CC	条　　件
000	NZ(Non Zero)
001	Z(Zero)
010	NC(Non Carry)
011	C(Carry)
100	PO 奇(P/V=0)(Parity Odd)
101	PE 偶(P/V=1)(Parity Even)
110	P(Plus)
111	M(Minus)

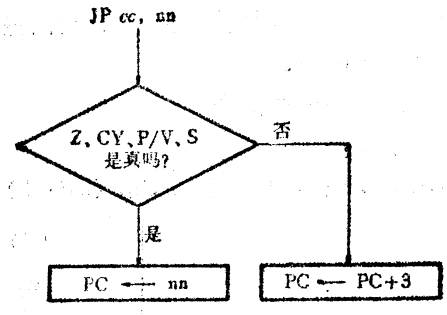


图 6-38　条件转移指令功能示意图

(2) 条件相对转移指令

指令格式：

① JR C, e ；若CY=1，则PC←PC+e，否则执行下一条指令

② JR NC, e ；若CY=0，则PC←PC+e，否则执行下一条指令

③ JR Z, e ；若Z=1，则PC←PC+e，否则执行下一条指令

④ JR NZ, e ；若Z=0，则PC←PC+e，否则执行下一条指令

条件相对转移指令都是两字节的指令，它们的转移条件仅取决于标志CY和Z的状态，故只有4条指令。其功能示意图如图6-39所示。

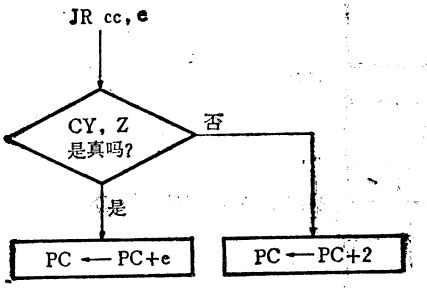


图6-39 条件相对转移指令功能示意图

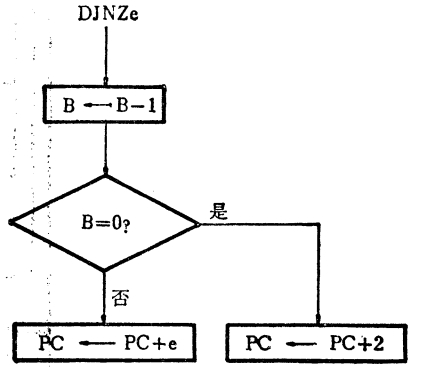


图6-40 DJNZ指令功能示意图

(3) 循环控制转移指令

指令格式：

DJNZ e ；B←B−1 ；若B≠0，则PC←PC+e；若B=0，则执行下一条指令

实际上，这也是一条两字节的条件相对转移指令，它规定以B作为计数器，每执行一次，使B←B−1，根据判断B是否为零，来决定程序是否转移。其功能示意图如图6-40所示。

其功能实际上是以下两条指令的组合：

```
DEC  B
JR   NZ,  e
```

转移指令的应用例子，我们在前面例子中已经多次运用过，这里不再重复。

(七) 子程序调用(CALL)和返回(RET)指令

Z80的调用与返回指令组有7类基本指令，12种操作码，如附录1表A1-10和附录2表A2-10所示。这类指令对标志状态没有影响。

1. 调用指令

调用指令CALL是转移指令的一种特殊形式，这是一条三字节指令。可分为无条件调用和条件调用指令两种：

(1) 无条件调用指令

指令格式：

CALL nn；$(SP-1)\leftarrow PC_H$，$(SP-2)\leftarrow PC_L$，$SP\leftarrow SP-2$，$PC\leftarrow nn$。

CD	操作码
n_L	低位地址
n_H	高位地址

执行 CALL 指令时，首先把程序计数器 (PC) 的现行地址(紧跟在 CALL 指令后面的字节地址)推入堆栈，同时把调用的子程序入口地址 nn 置入 PC 中，转去执行子程序。

(2) 条件调用

指令格式：

CALL cc, nn

当满足条件时，功能与无条件调用一样；当条件不满足时，就顺序执行下一条指令。其功能示意图如图 6-41 所示。

条件 cc 与条件转移指令中的 cc 一样。

2. 返回指令

RET 指令是 CALL 指令的逆过程。它的作用是在子程序结束时，使程序返回到原主程序段继续执行。返回是由弹出原来执行 CALL 时，保存在栈顶两个单元中的原程序计数器的内容来实现的。因此，当使用调用指令 CALL 时，在其子程序的末尾必然设置一条返回指令 RET。它也分无条件返回和条件返回两种：

(1) 无条件返回

指令格式：

RET; $PC_L \leftarrow (SP)$, $PC_H \leftarrow (SP+1)$, $SP \leftarrow SP+2$

(2) 条件返回

指令格式：

RET cc

当满足条件时，功能与无条件返回一样；当条件不满足时，顺序执行下一条指令。其功能示意图如图 6-42 所示。

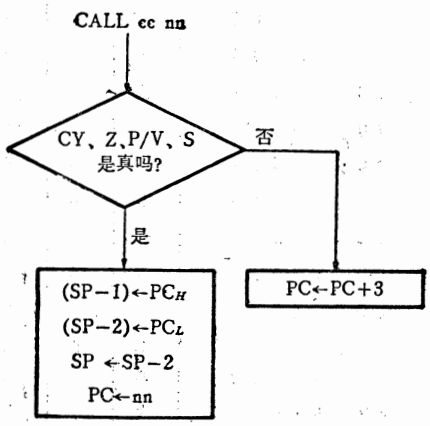


图 6-41 条件调用指令功能示意图

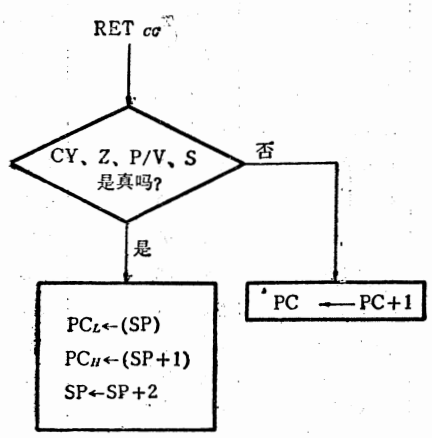


图 6-42 条件返回指令功能示意图

条件 cc 与条件转移指令中的 cc 一样。

Z80 返回指令与 8080A/8085A 返回指令相比，还增加了两条专用的返回指令，即从中断返回指令(RETI)和从不可屏蔽中断返回(RETN)指令，专用于中断处理时返回主程序。这部分内容将放在第十二章中断系统专门讨论。

表 6-8 Intel 8080A/8085A 和 Z80 微处理器通用的指令表

	助记符 Z80	助记符 8080A/8085A	功能说明	指令码(操作码) $D_7D_6D_5D_4D_3D_2D_1D_0$	时钟周期 T 8080A	8085A	Z80
数据传送指令 21 条	LD r_1, r_2	MOV r_1, r_2	$r_1 \leftarrow r_2$	0 1 D D D S S S	5	4	4
	LD (HL), r	MOV M, r	(HL)←r	0 1 1 1 0 S S S	7	7	7
	LD r, (HL)	MOV r, M	r←(HL)	0 1 D D D 1 1 0	7	7	7
	LD r, n	MVI r, n	r←n	0 0 D D D 1 1 0	7	7	7
	LD (HL), n	MVI M, n	(HL)←n	0 0 1 1 0 1 1 0	10	10	10
	LD BC, nn	LXI B, nn	BC←nn	0 0 0 0 0 0 0 1	10	10	10
	LD DE, nn	LXI D, nn	DE←nn	0 0 0 1 0 0 0 1	10	10	10
	LD HL, nn	LXI H, nn	HL←nn	0 0 1 0 0 0 0 1	10	10	10
	LD SP, nn	LXI SP, nn	SP←nn	0 0 1 1 0 0 0 1	10	10	10
	LD (BC), A	STAX B	(BC)←A	0 0 0 0 0 0 1 0	7	7	7
	LD (DE), A	STAX D	(DE)←A	0 0 0 1 0 0 1 0	7	7	7
	LD A, (BC)	LDAX B	A←(BC)	0 0 0 0 1 0 1 0	7	7	7
	LD A, (DE)	LDAX D	A←(DE)	0 0 0 1 1 0 1 0	7	7	7
	LD (nn), HL	SHLD nn	(nn)←L, (nn+1)←H	0 0 1 0 0 0 1 0	16	16	16
	LD HL, (nn)	LHLD nn	L←(nn), H←(nn+1)	0 0 1 0 1 0 1 0	16	16	16
	LD (nn), A	STA nn	(nn)←A	0 0 1 1 0 0 1 0	13	13	13
	LD A, (nn)	LDA nn	A←(nn)	0 0 1 1 1 0 1 0	13	13	13
	EX DE, HL	XCHG	D←→H, E←→L	1 1 1 0 1 0 1 1	4	4	4
	EX (SP), HL	XTHL	L←→(SP), H←→(SP+1)	1 1 1 0 0 0 1 1	18	16	19
	LD SP, HL	SPHL	SP←HL	1 1 1 1 1 0 0 1	5	6	6
	JP (HL)	PCHL	PC←HL	1 1 1 0 1 0 0 1	5	6	4
增量与减量指令 12 条	INC r	INR r	r←r+1	0 0 D D D 1 0 0	5	4	4
	DEC r	DCR r	r←r−1	0 0 D D D 1 0 1	5	4	4
	INC (HL)	INR M	(HL)←(HL)+1	0 0 1 1 0 1 0 0	10	10	11
	DEC (HL)	DCR M	(HL)←(HL)−1	0 0 1 1 0 1 0 1	10	10	11
	INC BC	INX B	BC←BC+1	0 0 0 0 0 0 1 1	5	6	6
	INC DE	INX D	DE←DE+1	0 0 0 1 0 0 1 1	5	6	6
	INC HL	INX H	HL←HL+1	0 0 1 0 0 0 1 1	5	6	6
	INC SP	INX SP	SP←SP+1	0 0 1 1 0 0 1 1	5	6	6
	DEC BC	DCX B	BC←BC−1	0 0 0 0 1 0 1 1	5	6	6
	DEC DE	DCX D	DE←DE−1	0 0 0 1 1 0 1 1	5	6	6
	DEC HL	DCX H	HL←HL−1	0 0 1 0 1 0 1 1	5	6	6
	DEC SP	DCX SP	SP←SP−1	0 0 1 1 1 0 1 1	5	6	6
算术与逻辑运算指令	ADD A, r	ADD r	A←A+r	1 0 0 0 0 S S S	4	4	4
	ADC A, r	ADC r	A←A+r+CY	1 0 0 0 1 S S S	4	4	4
	SUB r	SUB r	A←A−r	1 0 0 1 0 S S S	4	4	4
	SBC A, r	SBB r	A←A−r−CY	1 0 0 1 1 S S S	4	4	4
	AND r	ANA r	A←A∧r	1 0 1 0 0 S S S	4	4	4
	XOR r	XRA r	A←A∨r	1 0 1 0 1 S S S	4	4	4
	OR r	ORA r	A←A∨r	1 0 1 1 0 S S S	4	4	4
	CP r	CMP r	A−r	1 0 1 1 1 S S S	4	4	4

续表 6-8

	助记符		功能说明	指令码(操作码)	时钟周期 T		
	Z80	8080A/8085A		$D_7D_6D_5D_4D_3D_2D_1D_0$	8080A	8085A	Z80
算术与逻辑运算指令32条	ADD A, (HL)	ADD M	A←A+(HL)	10000110	7	7	7
	ADC A, (HL)	ADC M	A←A+(HL)+CY	10001110	7	7	7
	SUB (HL)	SUB M	A←A−(HL)	10010110	7	7	7
	SBC A, (HL)	SBB M	A←A−(HL)−CY	10011110	7	7	7
	AND (HL)	ANA M	A←A∧(HL)	10100110	7	7	7
	XOR (HL)	XRA M	A←A∀(HL)	10101110	7	7	7
	OR (HL)	ORA M	A←A∨(HL)	10110110	7	7	7
	CP (HL)	CMP M	A−(HL)	10111110	7	7	7
	ADD A,n	ADI n	A←A+n	11000110	7	7	7
	ADC A,n	ACI n	A←A+n+CY	11001110	7	7	7
	SUB n	SUI n	A←A−n	11010110	7	7	7
	SBC A,n	SBI n	A←A−n−CY	11011110	7	7	7
	AND n	ANI n	A←A∧n	11100110	7	7	7
	XOR n	XRI n	A←A∀n	11101110	7	7	7
	OR n	ORI n	A←A∨n	11110110	7	7	7
	CP n	CPI n	A−n	11111110	7	7	7
	ADD HL,BC	DAD B	HL←HL+BC	00001001	10	10	10
	ADD HL,DE	DAD D	HL←HL+DE	00011001	10	10	10
	ADD HL,HL	DAD H	HL←HL+HL	00101001	10	10	10
	ADD HL,SP	DAD SP	HL←HL+SP	00111001	10	10	10
	DAA	DAA	A的十进制调整	00100111	4	4	4
	SCF	STC	CY←1	00110111	4	4	4
	CCF	CMC	CY←$\overline{CY}$	00111111	4	4	4
	CPL	CMA	A←$\overline{A}$	00101111	4	4	4
循环指令4条	RLCA	RLC	$A_{n+1}\leftarrow A_n$, $A_0\leftarrow A_7$, $CY\leftarrow A_7$	00000111	4	4	4
	RRCA	RRC	$A_n\leftarrow A_{n+1}$, $A_7\leftarrow A_0$ $CY\leftarrow A_0$	00001111	4	4	4
	RLA	RAL	$A_{n+1}\leftarrow A_n$, $A_0\leftarrow CY\leftarrow A_7$	00010111	4	4	4
	RRA	RAR	$A_n\leftarrow A_{n+1}$, $A_7\leftarrow CY\leftarrow A_0$	00011111	4	4	4
转移指令9条	JP nn	JMP nn	PC←nn	11000011	10	10	10
	JP NZ nn	JNZ nn	IF Z=0,PC←nn	11000010	10	7/10	10
	JP Z nn	JZ nn	IF Z=1,PC←nn	11001010	10	7/10	10
	JP NC nn	JNC nn	IF CY=0,PC←nn	11010010	10	7/10	10
	JP C nn	JC nn	IF CY=1,PC←nn	11011010	10	7/10	10
	JP PO nn	JPO nn	IF P=0,PC←nn	11100010	10	7/10	10
	JP PE nn	JPE nn	IF P=1,PC←nn	11101010	10	7/10	10
	JP P nn	JP nn	IF S=0,PC←nn	11110010	10	7/10	10
	JP M nn	JM nn	IF S=1,PC←nn	11111010	10	7/10	10

续表 6-8

	助记符 Z80	助记符 8080A/8085A	功能说明	指令码(操作码) $D_7D_6D_5D_4D_3D_2D_1D_0$	时钟周期 T 8080A	8085A	Z80
子程序调用指令10条	CALL nn	CALL nn	$(SP-1)\leftarrow PC_H$, $(SP-2)\leftarrow PC_L$ $SP\leftarrow SP-2$, $PC\leftarrow nn$	1 1 0 0 1 1 0 1	17	18	17
	CALL NZ, nn	CNZ nn	IF Z=0 $(SP-1)\leftarrow PC_H$, $(SP-2)\leftarrow PC_L$ $SP\leftarrow SP-2$, $PC\leftarrow nn$	1 1 0 0 0 1 0 0	11/17	9/18	10/17
	CALL Z, nn	CZ nn	IF Z=1, 同上	1 1 0 0 1 1 0 0	11/17	9/18	10/17
	CALL NC, nn	CNC nn	IF CY=0, 同上	1 1 0 1 0 1 0 0	11/17	9/18	10/17
	CALL C, nn	CC nn	IF CY=1, 同上	1 1 0 1 1 1 0 0	11/17	9/18	10/17
	CALL PO, nn	CPO nn	IF P=0, 同上	1 1 1 0 0 1 0 0	11/17	9/18	10/17
	CALL PE, nn	CPE nn	IF P=1, 同上	1 1 1 0 1 1 0 0	11/17	9/18	10/17
	CALL P, nn	CP nn	IF S=0, 同上	1 1 1 1 0 1 0 0	11/17	9/18	10/17
	CALL M, nn	CM nn	IF S=1, 同上	1 1 1 1 1 1 0 0	11/17	9/18	10/17
	RST n	RST n	$(SP-1)\leftarrow PC_H$, $(SP-2)\leftarrow PC_L$, $SP\leftarrow SP-2$, $PC\leftarrow 8*NNN$ NNN=000～111	1 1 N N N 1 1 1	12	11	11
返回指令9条	RET	RET	$PC_L\leftarrow(SP)$, $PC_H\leftarrow(SP+1)$, $SP\leftarrow SP+2$	1 1 0 0 1 0 0 1	10	10	10
	RET NZ	RNZ	IF Z=0 $PC_L\leftarrow(SP)$, $PC_H\leftarrow(SP+1)$, $SP\leftarrow SP+2$	1 1 0 0 0 0 0 0	5/11	6/12	5/11
	RET Z	RZ	IF Z=1, 同上	1 1 0 0 1 0 0 0	5/11	6/12	5/11
	RET NC	RNC	IF CY=0, 同上	1 1 0 1 0 0 0 0	5/11	6/12	5/11
	RET C	RC	IF CY=1, 同上	1 1 0 1 1 0 0 0	5/11	6/12	5/11
	RET PO	RPO	IF P=0, 同上	1 1 1 0 0 0 0 0	5/11	6/12	5/11
	RET PE	RPE	IF P=1, 同上	1 1 1 0 1 0 0 0	5/11	6/12	5/11
	RET P	RP	IF S=0, 同上	1 1 1 1 0 0 0 0	5/11	6/12	5/11
	RET M	RM	IF S=1, 同上	1 1 1 1 1 0 0 0	5/11	6/12	5/11
堆栈操作指令8条	PUSH BC	PUSH B	$(SP-1)\leftarrow B$, $(SP-2)\leftarrow C$, $SP\leftarrow SP-2$	1 1 0 0 0 1 0 1	11	12	11
	PUSH DE	PUSH D	$(SP-1)\leftarrow D$, $(SP-2)\leftarrow E$, $SP\leftarrow SP-2$	1 1 0 1 0 1 0 1	11	12	11
	PUSH HL	PUSH H	$(SP-1)\leftarrow H$, $(SP-2)\leftarrow L$, $SP\leftarrow SP-2$	1 1 1 0 0 1 0 1	11	12	11
	PUSH AF	PUSH PSW	$(SP-1)\leftarrow A$, $(SP-2)\leftarrow F$, $SP\leftarrow SP-2$	1 1 1 1 0 1 0 1	11	12	11
	POP BC	POP B	$B\leftarrow(SP+1)$, $C\leftarrow(SP)$, $SP\leftarrow SP+2$	1 1 0 0 0 0 0 1	10	10	10
	POP DE	POP D	$D\leftarrow(SP+1)$, $E\leftarrow(SP)$, $SP\leftarrow SP+2$	1 1 0 1 0 0 0 1	10	10	10
	POP HL	POP H	$H\leftarrow(SP+1)$, $L\leftarrow(SP)$, $SP\leftarrow SP+2$	1 1 1 0 0 0 0 1	10	10	10
	POP AF	POP PSW	$A\leftarrow(SP+1)$, $F\leftarrow(SP)$, $SP\leftarrow SP+2$	1 1 1 1 0 0 0 1	10	10	10

续表 6-8

	助记符		功能说明	指令码(操作码)	时钟周期 T		
	Z80	8080A/8085A		$D_7D_6D_5D_4D_3D_2D_1D_0$	8080A	8085A	Z80
I/O指令2条	IN A, (n)	IN n	A←(n)	1 1 0 1 1 0 1 1	10	10	11
	OUT (n), A	OUT n	(n)←A	1 1 0 1 0 0 1 1	10	10	11
状态管理指令4条	EI	EI	允许中断	1 1 1 1 1 0 1 1	4	4	4
	DI	DI	禁止中断	1 1 1 1 0 0 1 1	4	4	4
	NOP	NOP	空操作	0 0 0 0 0 0 0 0	4	4	4
	HALT	HLT	暂停	0 1 1 1 0 1 1 0	4	4	4

注 (1) DDD或SSS: 000(B); 001(C); 010(D); 011(E); 100(H); 101(L); 110存贮单元; 111(A)

(2) 某些指令有两种时钟周期数(5/11)需根据条件标志所指示的情况来决定。斜线下面的数表示当条件满足时所需要的时态数;斜线上面的表示当条件不满足时所需要的时态数。

(3) M和(HL)表示HL寄存器对所指出的地址的存贮单元。

(4) PSW是程序状态字,它有16位,由A、F寄存器组成。

(5) n表示8位数据, nn表示16位数据或地址。

(6) (nn)表示以( )中的内容nn作为地址,从它所指示的内存单元中取出操作数。

【例】 试编写软件延时子程序,延时秒数K可事先任意设定。其源程序如下:

```
        LD    B, K
AGAIN:  CALL  DELAYI
        DEC   B
        JR    NZ, AGAIN
        ⋮
```

1秒延时子程序

```
DELAYI:  LD    C, 05
WAIT 1:  LD    D, 63H
WAIT 2:  LD    E, 00
WAIT 3:  DEC   E
         JR    NZ, WAIT 3
         DEC   D
         JR    NZ, WAIT 2
         DEC   C
         JR    NZ, WAIT 1
         RET
```

注: 设μC的时钟频率为2 MHz。

以上我们分门别类地从应用指令的角度,系统地阐述了Z80指令系统的功能和特点。并通过指令的应用实例来帮助读者理解和掌握各类指令的特点和用途,以便为正确而合理地编制汇编语言源程序打下扎实的基础。

关于输入输出和中断的指令,我们放在相应的章节中进行分析。

表6-8列出了Intel 8080A/8085A和Z80微处理器通用的指令表;表6-9列出了Z80微

处理器新增加的指令表；附录 1 和附录 2 详细地列出了 Z80 指令系统的寻址方式、操作码以及功能表。读者在学习本章内容和编制程序的实践中，应充分利用这些指令系统表。

表 6-9　Z80 新增加的指令表（与 8080A 相比）

操作码 (H)	助记符
08	EX AF, AF′
10	DJNZ disp*
18	JR disp
20(8085A 中的 RIM)	JR NZ, disp
28	JR Z disp
30(8085A 中的 SIM)	JR NC, disp
38	JR C, disp
CB	BIT, RES, SET, RL, RLC, RR, RRC, SLA, SRA, SRL
D9	EXX
DD	包含寄存器 IX 的所有指令
ED	ADC HL,SS ; LD A,I ; NEG CPD ; LD A,R ; OTDR CPDR ; LD(nn),dd ; OTIR CPI ; LD I,A OUT(C),r CPIR ; LD R,A OUTD IM m ; LD dd,(nn) ; OUTI IN r,(CC) ; LDD ; RETI IND ; LDDR ; RETN INDR ; LDI ; RLD INI ; LDIR ; RRD INIR , SBC HL,SS
FD	包含寄存器 IY 的所有指令

注：disp 表示 8 位带符号的二进制地址位移量

第七章　微处理器的操作时序

本章将从微观的角度深入分析指令的执行过程及典型的操作时序。

我们学习微处理器操作时序的意义在于：

1．有利于深入理解微处理器内部的操作原理。

2．有利于合理地选用指令，编制出具有较佳时空指标的源程序。

3．有利于更好地解决 CPU 与存贮器和各种外设之间接口的时序配合问题。

4．有利于更准确地估计 CPU 完成操作的时间，这对于微型计算机实时控制系统是十分重要的。

下面我们以 Intel 8080 A/8085 A 和 Z 80 为例，深入分析微处理器典型的操作时序。

§7-1　Intel 8080 A 微处理器的操作时序

一、8080 A 的机器周期和周期信息码

如前所述，8080A 的一个指令周期由 1～5 个机器周期(M_1～M_5)组成。8080 A 共有十种类型的机器周期：

1．FETCH　(M_1)——取指周期；

2．MEMORY　READ——存贮器读周期；

3．MEMORY　WRITE——存贮器写周期；

4．STACK　READ——堆栈读周期；

5．STACK　WRITE——堆栈写周期；

6．INPUT——输入周期；

7．OUTPUT——输出周期；

8．INTERRUPT——中断周期；

9．HALT——暂停周期；

10．HALT · INTERRUPT——暂停 · 中断周期。

在一个指令周期内，究竟执行哪些机器周期，这取决于该指令的类型。但是，8080 A 规定，任何指令周期中的第一个机器周期必定是取指机器周期。

为了判明机器周期的类型，8080 A 在每个机器周期的第一个时态 T_1，将发出一组 8 位的周期信息码，这个周期信息码在同步信号 SYNC 作用期间送到 8080A 的数据总线 D_7～D_0 上，并被锁存在外部锁存器中，以便用它来产生对外部电路的控制信号。

表 7-1 表明这 8 种周期信息位的含义。

这 8 位周期信息码的内容将在 T_1 的 SYNC 信号作用期间指明本机器周期的类型。表 7-2 列出了数据总线上的周期信息码和执行的机器周期的关系，以及在该机器周期中，系统控制器 8228 发出的控制信号的情况。

表 7-1　周期信息码的定义

数据总线位	符　号	定　义
D_0	INTA	中断响应状态。当 CPU 接受中断请求时，它为高电平。该信号在 DBIN 为高电平作用时，用来门控一个重新启动指令 RST 放到数据总线上去。
D_1	$\overline{WO}$	写入型判断状态。它指出当前指令周期的操作将是一个存贮器写或者输出操作，此时，$\overline{WO}$ 为低电平。否则，将执行存贮器读或输入操作。
D_2	STACK	堆栈状态。当它为高电平时，指明地址总线上有从堆栈指示器(SP)来的堆栈地址。
D_3	HLTA	暂停响应状态。当它为高电平时，指明 CPU 处于对暂停指令(HALT)的响应状态。
D_4	OUT	输出指令响应状态。当它为高电平时，指明地址总线上的地址是输出设备的地址，而且在 $\overline{WR}$ 作用时，将使输出数据放到数据总线上。
D_5	M_1	M_1 状态。当 M_1 为高电平时，指明 CPU 处于指令取第一字节的基本操作周期。
D_6	INP	输入指令响应状态。当它为高电平时，指明地址总线上的地址是输入设备的地址，而且在 DBIN 作用时，将使输入数据放到数据总线上。
D_7	MEMR	存贮器读出状态。当它为高电平时，指明数据总线将用于读出存贮器的数据。

表 7-2　机器周期和周期信息码的关系

数据总线上数位	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	8228 发出控制信号				
周期信息 / 机器周期类型	INTA	$\overline{WO}$	STACK	HLTA	OUT	M_1	INP	MEMR	$\overline{MEMR}$	MEMW	$\overline{I/OR}$	$\overline{I/OW}$	$\overline{INTA}$
取指周期	0	1	0	0	0	1	0	1	0	1	1	1	1
存贮器读周期	0	1	0	0	0	0	0	1	0	1	1	1	1
存贮器写周期	0	0	0	0	0	0	0	0	1	0	1	1	1
堆栈读周期	0	1	1	0	0	0	0	1	0	1	1	1	1
堆栈写周期	0	0	1	0	0	0	0	0	1	0	1	1	1
输入周期	0	1	0	0	0	0	1	0	1	1	0	1	1
输出周期	0	0	0	0	1	0	0	0	1	1	1	0	1
中断响应周期	1	1	0	0	0	1	0	0	1	1	1	1	0
暂停响应周期	0	1	0	1	0	0	0	1	1	1	1	1	1
暂停中断周期	1	1	0	1	0	1	0	0	1	1	1	1	0

二、8080A 基本指令周期和时态转换流程

1．8080A 基本指令周期的时序

8080A除暂停、中断、保持三种基本操作外，一个机器周期可包含 3～5 个时钟周期(称时态)，即 T_1～T_5。在每个机器周期中，前三个时态 T_1、T_2 和 T_3 是必须的。而等待时态 T_W 则要视 READY (准备就绪)信号的情况来选用。

每个时态的操作内容如表 7-3 所示。

表 7-3 各时态的操作内容

时　　态	有　　关　　操　　作
T_1	由时钟信号 ϕ_2 的上升沿产生一个同步信号 SYNC，它表示每个机器周期的开始，并送至外电路作选通信号用。此时，在数据总线 $D_{7\sim0}$ 上出现本机器周期的周期信息码，用以产生外电路的控制信号。同时，在 ϕ_2 的上升沿，CPU 的地址总线接受 PC 送来的指令地址信息，这一信息一直保持到 T_3 时态，以使 CPU 有足够时间，从存贮器读出数据。
T_2	CPU 查询 READY (准备就绪) 和 HOLD (保持)信号，并检测是否为暂停指令 (HLT)，如果 READY 为高电平，CPU 就从 T_2 进入 T_3 时态，否则将插入 T_W 时态。
T_W (选用)	如果 READY 为低电平或已检测到是执行暂停指令，则 CPU 进入等待时态。此时，CPU "空转"，并通过 WAIT 线输出高电平，一直到 READY 变为高电平。T_W 的延续时间为时钟周期的整数倍。
T_3	当 READY 是高电平且没有 HOLD 信号或者不是暂停指令时，CPU 即进入 T_3 时态。此时态内的操作内容取决于机器周期的类型。在取指周期，CPU 就将数据总线内容送入指令寄存器并译码；在存贮器读、堆栈读或输入周期时，数据总线内容被 CPU 当作数据字节而接收；而在存贮器写、堆栈写或输出周期，CPU 将数据字节送至数据总线。
T_4、T_5 (选用)	如果执行一条特定指令，则需要有 T_4、T_5 时态　否则，就可跳过其中的一个或二个。T_4、T_5 多用于内部操作，例如指令译码、寄存器之间的数据传送。

应当指出，并不是所有机器周期都有以上各种时态，这取决于该指令的类型和在这个指令周期内具体的机器周期。任何一个机器周期，只要其处理工作完成了，即可直接进入下一个机器周期，而不需要每次都进入到 T_4 及 T_5 时态。

图 7-1 展示了 8080A 的基本指令周期中的时序关系波形图

8080A 基本指令周期内各个时态所要完成的操作时序如表 7-3 所列。在图 7-1 中，ϕ_1、ϕ_2 是时钟脉冲，ϕ_1 的两个正跳沿间隔为一个时态 T。每个时态中的动作都发生在 ϕ_1 和 ϕ_2 时钟脉冲的跳变时。在任何一个机器周期中，都以同步信号 SYNC 有效时作为开始的标志。SYNC 是由 ϕ_2 脉冲前沿产生的，但 SYNC 前沿比 ϕ_2 前沿略有延迟。同样，SYNC 的后沿比第二个时态 T_2 的 ϕ_2 脉冲前沿也有所延迟。周期信息码是在 SYNC 作用期间送到数据总线 D_7～D_0 上去的。

在 T_1 的 ϕ_2 上升沿时，8080A CPU 输出地址到地址总线 A_{15}～A_0 上，但地址比 ϕ_2 的上升沿略为延迟一段时间后才变为稳定，并保持稳定直至 T_3 过后的第一个 ϕ_2 脉冲为止。这样，就使 CPU 有足够的时间从存贮器中读出数据。

每当 CPU 向存贮器发出一个地址，它就给存贮器一个申请 WAIT (等待)的机会。如果存贮器存取时间小于 CPU 系统存取时间，则它可以很快地向 CPU 送出一个数据；如果存贮器存

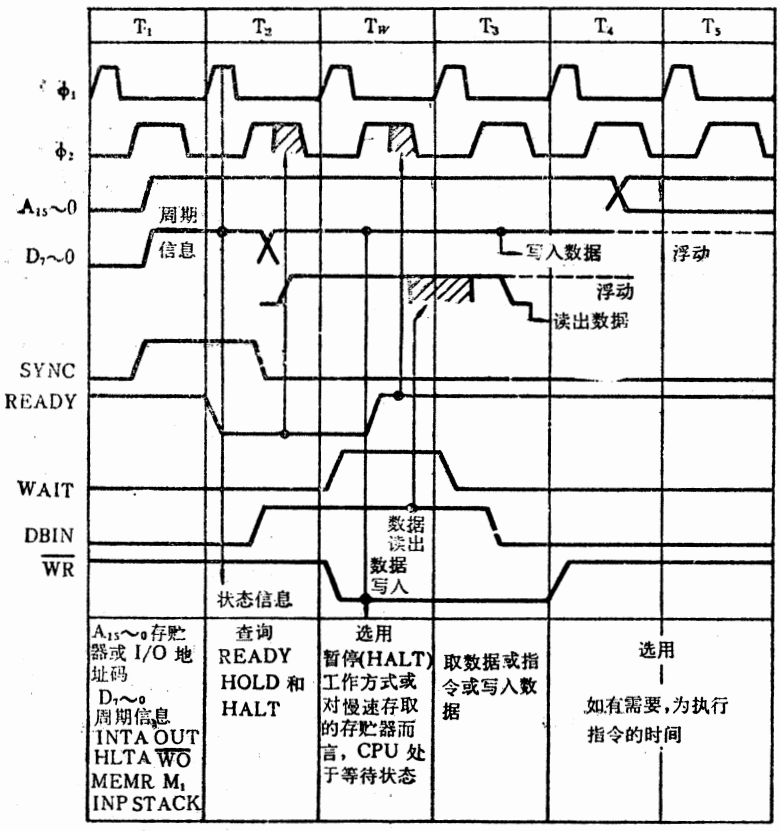


图 7-1　8080 A 基本指令周期

取时间大于 CPU 的系统存取时间，则要花费足够的时间准备数据。此时，它使 CPU 的 READY（准备就绪）线变为低电平，通知 CPU 转入“等待”时态，以便给存贮器足够的时间准备好数据。

CPU 为了响应存贮器的等待请求，在 T_2 结束时插入一个等待时态(T_W)，而不直接进入 T_3 时态。CPU 进入 T_W 时态时，将向外部发出 WAIT 信号，以表明 CPU 正在响应存贮器的等待请求。8080A 输出线 WAIT 电平从低变高，是由 ϕ_1 时钟的上升沿触发的。

CPU 等待的时间取决于存贮器的存取时间。CPU 在 READY 线重新升高以前一直处于等待状态。当 READY 线升为高电平时，表明存贮器已经准备就绪。如果 READY 线电平的上升沿在时钟 ϕ_2 的下降沿以前一段时间内完成，那末就可以在 ϕ_2 结束以后，退出 T_W 时态而进入 T_3 时态。由于等待时间是从 ϕ_1 开始的，因此总的等待时间必然既是 T_W 时态，又是时钟周期 T 的整数倍。

进入 T_3 时态以后，CPU 究竟执行什么操作，这将取决于现行机器周期的类型。若是 FETCH（取指）机器周期，则 CPU 就将数据总线的内容当作指令送入指令寄存器并送往指令译码器译码；若是 MEMORY READ（存贮器读）机器周期或 STACK READ（堆栈读）机器周期，则 CPU 就将数据总线的内容视为一个数据字节；若是 MEMORY WRITE 机器周期，则 CPU就将一个数据字节送至数据总线上。在读出数据时，DB.N 信号线应为高电平。在写入数据时，$\overline{WR}$ 信号线应为低电平。

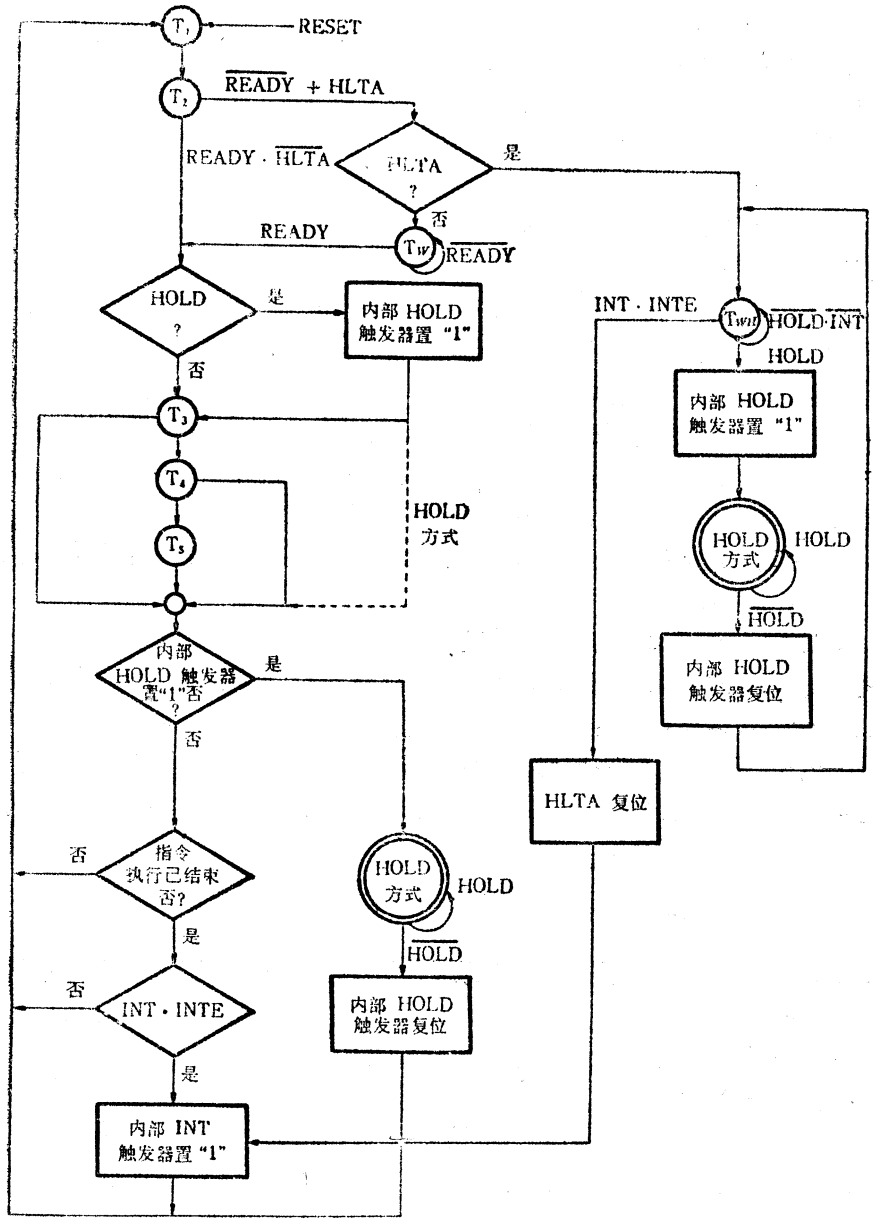


图 7-2 Intel 8080A 时态转换流程图

2．8080A 的时态转换流程

在了解了基本指令周期以后，就可以进一步讨论 8080A 时态转换流程图(图 7-2)

这张图展示了一个机器周期的过程。说明 8080A 是怎样从一个时态转换到另一个时态的。此图还展示了机器周期内 READY(准备)、HOLD(保持)及 INTERRUPT(中断)线的信号是怎样被采样的，以及在这些线处于什么样的条件下可能修改基本的转换时序。读者在分析这张流程图时，请复习一下 §5-2 中的 8080A 控制信号线的功能。下面分析一下 8080A在一个机器周期中各时态的转换过程。

首先，在复位信号 RESET 作用下，机器周期从 T_1 开始进入 T_2 时态，此后有两种情况：

(1) READY = "0" 或为 HLT 指令

这时，若是 HLT 指令，则 CPU 在 T_2 后进入 T_{WH} (暂停等待)时态。若不是 HLT 指令，而是 READY = "0"，则表示外面存贮器或 I/O 设备尚未准备就绪。CPU 在 T_2 后将进入 T_W 等待时态，直到 READY = "1" 为止。

同时查询 HOLD 信号线。

(2) READY = "1"，又不是 HALT 指令，则将直接查询 HOLD 信号线。

这时，又有两种情况：

① 若 T_2 期间检查出有 HOLD 请求，且 READY = "1"，则将把 HOLD 内部触发器置"1"。

② 若不是 HOLD，则进入 T_3、T_4、T_5 时态。

在一个机器周期结束以前，要检查是否内部 HOLD 触发器置"1"。如果是，则进入 HOLD 方式；如果不是，则要看指令是否结束。若指令已经结束，则要看是否外面有中断请求信号(INT)，以及允许中断信号(INTE)是否为"1"。如果这两者都满足，则 CPU 把内部中断触发器置"1"，并进入中断机器周期；如果这两者不能同时满足，则将开始另一个新的机器周期。

上面讲到，在 T_2 时态，CPU 可能进入 T_{WH} 暂停(HALT)时态。这时，8080A 只有通过下述三种途径才能退出暂停时态：

① RESET 线为"1"，使 8080A 恢复到 T_1 时态。

② 一个请求 HOLD 输入，能使 8080A 暂时进入保持(HOLD)时态。这时，内部 HOLD 触发器置"1"。

③ 若有一个中断请求(INT = "1")，且 CPU 的允许中断线 INTE = "1"，即若 INT·INTE = 1，则将使 8080A 脱离暂停(HALT)时态，并使内部 INT 触发器置"1"，从而进入中断机器周期。

三、8080A 典型指令的操作时序

Intel 8080A 指令各时态的功能表见附录 4。此表详尽地列出了 8080A μP 所有指令在各个时态的操作内容，仔细地分析这张时序表，将有助于进一步理解微处理器内指令执行的具体过程。在这里，我们仅举若干典型的指令略加分析说明。

1。寄存器传送：MOV C，B；C←B

这是一条单机器周期(M_1)的指令，只需访问一次存贮器。任何指令的执行过程，都包括取指令和执行指令两个阶段。在 M_1 中，T_1、T_2、T_3 是完成取指操作所必需的时态。T_4、T_5 则是执行指令的时态，视不同的指令类型来决定是需要 T_4 还是需要 T_4、T_5 时态。

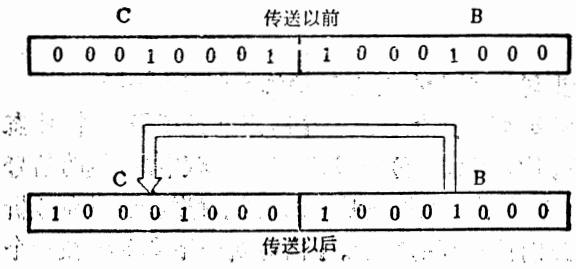


图 7-3 寄存器 B 的内容传送到 C

这条指令执行过程如下：

在 M_1 机器周期内，经 T_1、T_2、T_3 时态，指令即从存贮器中取出。在 T_3 末，指令码已送至 IR 中，随后即进行译码，并执行指令操作。

M_1 T_4：TMP←SSS (本例 SSS = B)

寄存器B的内容送至暂存寄存器TMP中。

T_5：DDD←TMP (本例 DDD = C)

暂存寄存器 TMP 的内容送至寄存器 C 中。由于在 8080A μP 寄存器(单通道)阵列内，一

个时态只能访问一个字，而不能同时对两个寄存器进行读写，故需用到暂存寄存器 TMP。这种局限性随着双通道 RAM 的出现将自然消失。

至此，指令执行结束。寄存器 B 的内容已传送到目标寄存器 C。

指令执行的时间可以计算出来，标准 8080A 的每一个时态（T）的持续时间为 500 ns，MOV r_1, r_2 指令的执行时间为 5 个时态(T)，所以这条指令执行的时间为 $5 \times 500 = 2500\,\text{ns} = 2.5\,\mu\text{s}$。

指令执行示意图如图 7-3，图 7-4 所示。

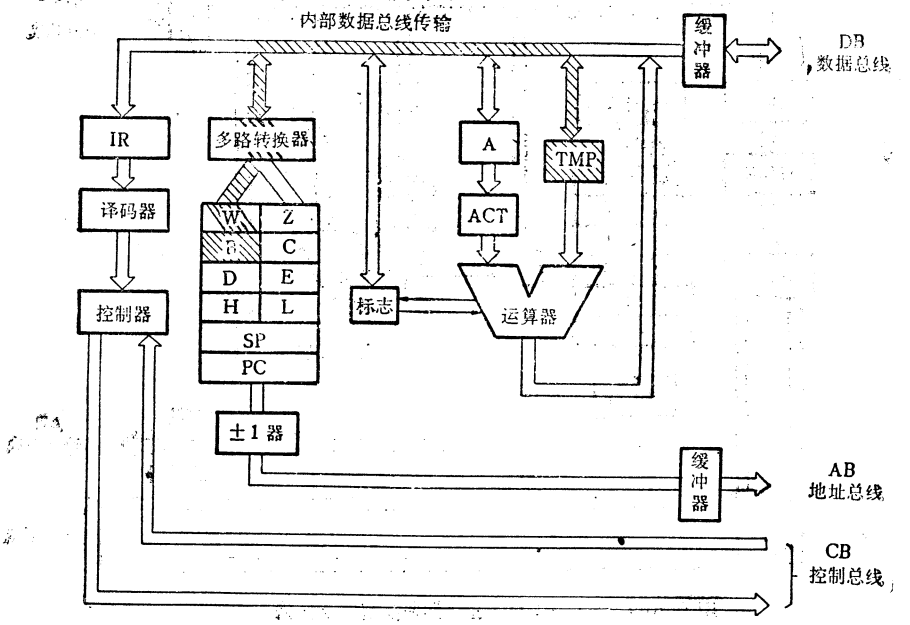


图 7-4(a)　寄存器 B 的内容传送到暂时寄存器 TMP

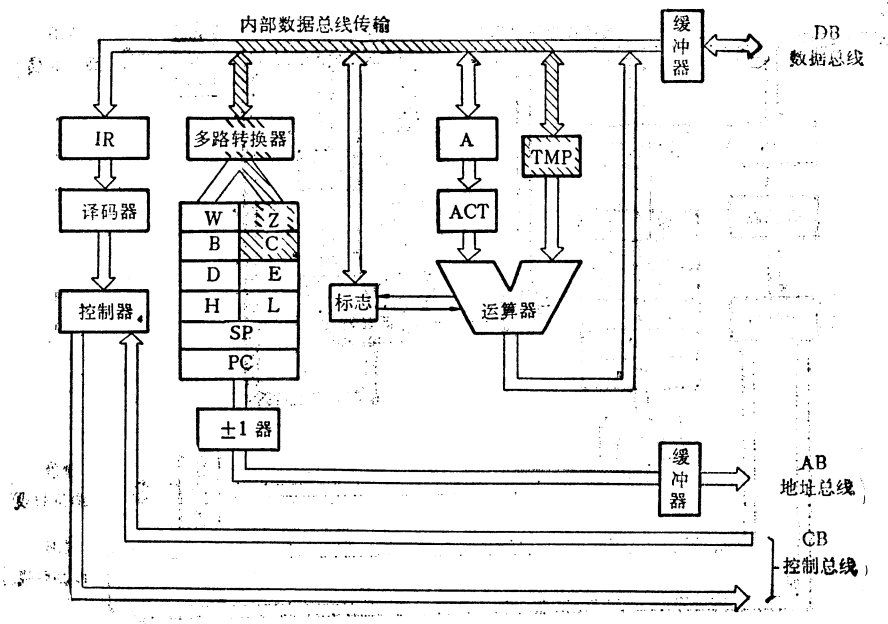


图 7-4(b)　暂时寄存器 TMP 的内容传送到寄存器 C

2. ADD r (加寄存器)；A←A+r

这是一条隐指令，也是单机器周期的指令。它只指出寄存器r，而把第二个寄存器隐含在累加器A中。其优点是指令码只需一个字节，为指明寄存器r只需3位二进制代码即可。所以这是完成加法操作的快速途径。

ADD r指令执行示意图如图7-5所示。

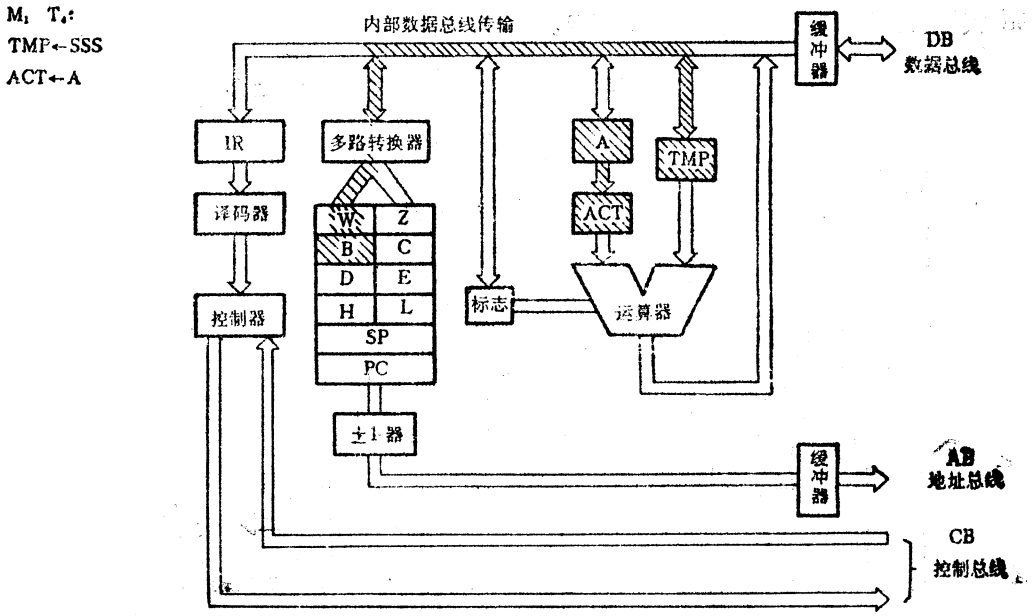


图7-5(a) 执行ADD r指令同时实现两次传输

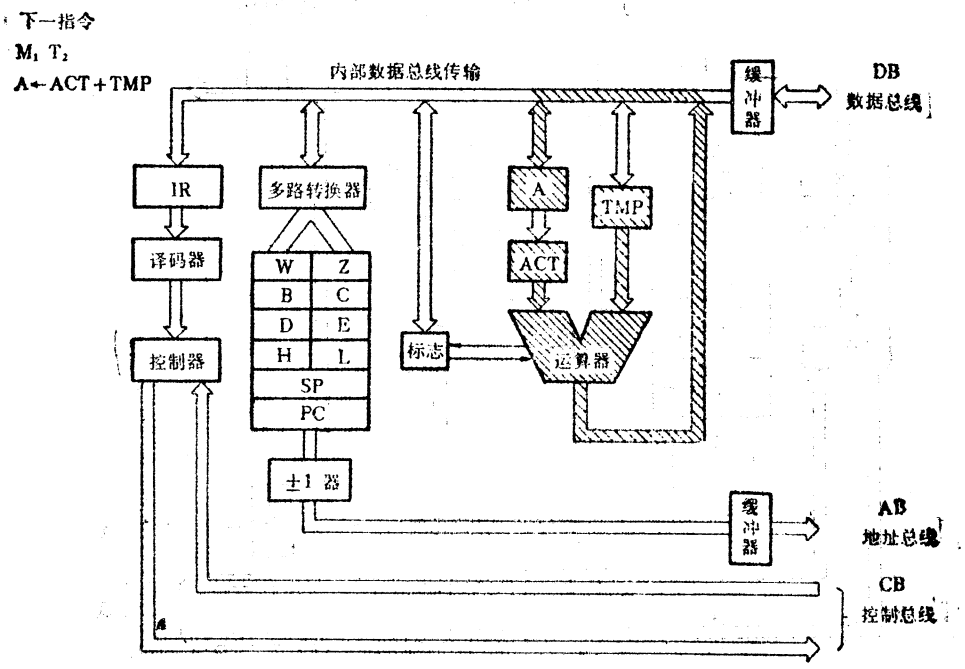


图7-5(b) 执行ADD r指令实现加法操作示意图

ADD r 指令执行过程如下：

M_1：T_1、T_2、T_3：指令从存贮器读出并送至指令寄存器 IR；

T_4：指令译码并执行：

即 TMP←SSS(本例 SSS = B)

ACT←A

首先指定源寄存器的内容(B)送至 TMP，同时累加器 A 的内容送至 ACT，为了缩短时间，使两次传输操作同时进行。至此，似乎可以进行加法操作了。但是，指令在实际执行过程中，却跳过了 M_1 内的 T_5 和下一指令的 M_1 的 T_1 时态，加法尚未完成。

在下一指令的 M_1 机器周期的 T_2 时态，才实现 ACT 和 TMP 相加并存入累加器 A，加法操作至此才告完成。即

M_1：T_2：　　A←ACT + TMP

ADD r 指令所采用的这种取数/执行重迭操作法，它的基本构思是：加法的实际执行，只要求使用运算器和数据总线。CPU 知道任一指令执行完毕后，将要继续执行的下面两个时态是下一指令 M_1 机器周期 T_1、T_2。而这两个时态的执行操作，只要求访问 PC 和应用一下地址总线即可，故执行加法操作与下一指令 M_1 的 T_1、T_2 的执行操作不会发生冲突，故有可能实现取数/执行重迭操作。

应当指出，实际上由于在下一指令的 M_1 的 T_1 时态，数据总线上送出的是周期信息，故此时还不能用于完成加法操作，必须等到 T_2 时态才能实现加法操作。

采用取数/执行重迭操作法的优点是加快执行指令的速度。如果按流水线法执行 ADD r 指令，则加法将在 M_1 的 T_5 时态完成，此时，加法的执行时间为 $5 \times 500\,ns = 2.5\,\mu s$ 时；而采用重迭操作法，可以在指令执行到 T_4 时态时，立即读取下一指令。这时，从概念上讲，M_1 将是这一加法指令的第二个机器周期，而实际上 M_1 是重迭操作的。因此，加法指令的实际的执行时间只有 4 个时态，即 $4 \times 500\,ns = 2\,\mu s$，显然，速度加快了 20%。

3．ADD M (加存贮器)；A←A + (HL)

这是一条两个机器周期的指令，它需要访问两次存贮器，故需两个机器周期(M_1, M_2)。M_1 用于取指令，M_2 用于执行指令。

当执行这条指令时，为了提供加到累加器去的数据的第二字节，必须再次访问存贮器。这一数据字节的地址存贮于 HL 寄存器对中。ADD M 指令执行示意图如图 7-6 所示。

其执行过程如下：

M_1：T_1、T_2、T_3：读取指令 IR←Inst；

T_4：ACT←A；

M_2：T_1：HL out；

Status out；

T_2、T_3：TMP←DATA 即从指定的内存单元中读出数据并送至暂存寄存器 TMP 中。

至此，运算器的两个输入端都准备好了数据，以后的情况与 ADD r 时的情况相似，即指令跳过了 M_2 的 T_4、T_5 和下一指令的 M_1 的 T_1 时态。

在下一指令的 M_1 的 T_2 时态，完成了加法操作。即

M_1：T_2　　A←ACT + TMP

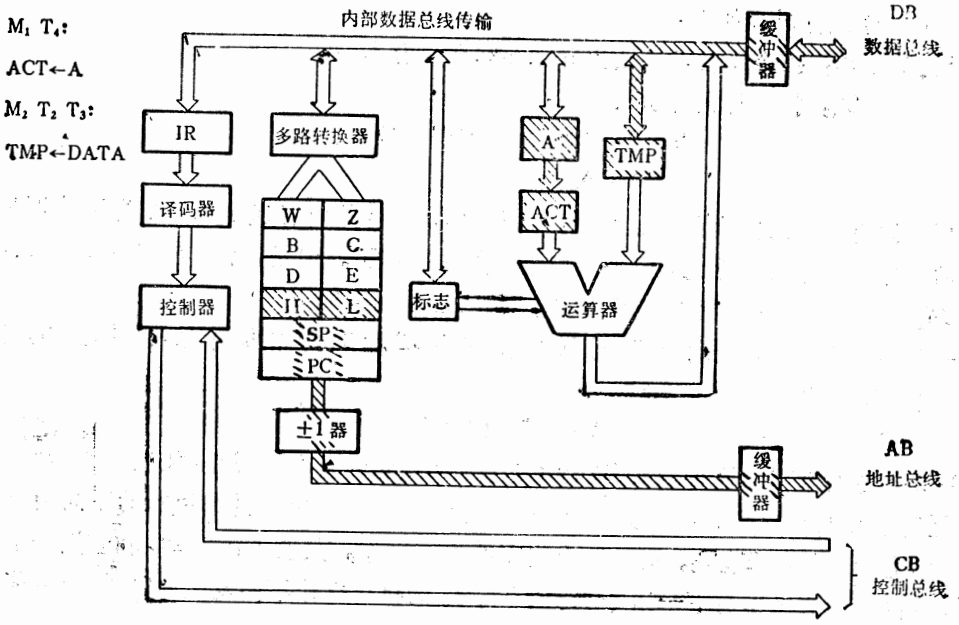


图 7-6(a)　执行 ADD M 指令取操作数操作示意图

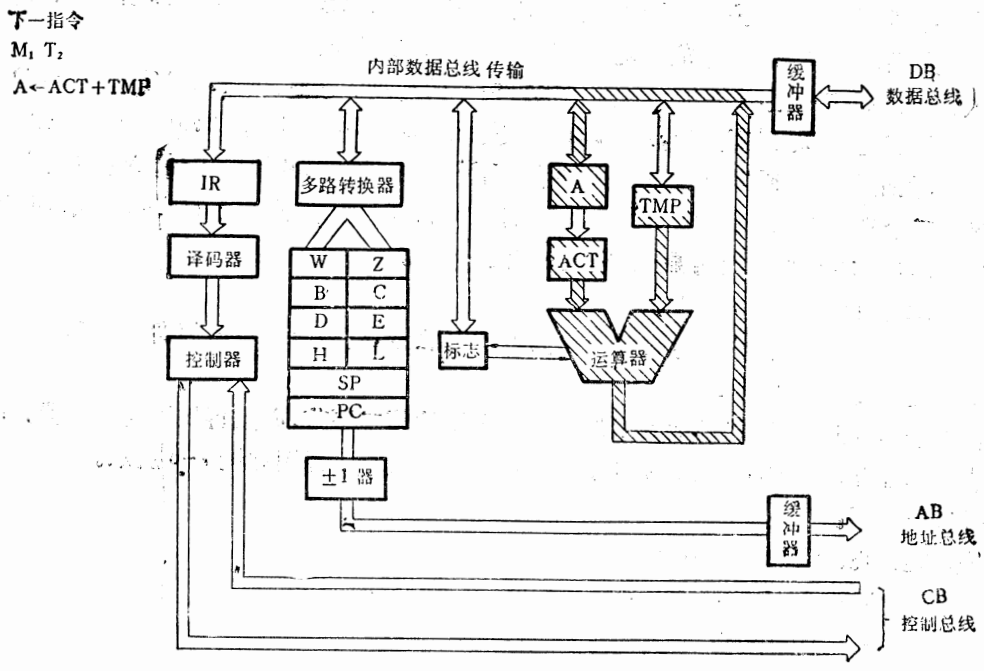


图 7-6(b)　执行 ADD M 指令实现加法操作示意图

至此加法过程已告成，完成这条指令实际上只用了两个机器周期 7 个时态。

4. LDA addr (直接取数送累加器)；A←(B_3B_2)【注】

这条指令指明：地址由指令的第三和第二字节规定的内存单元的内容送到累加器。它是

【注】 B_3B_2 为 Byte3 Byte2 的简写。

一条三字节的指令，需要访问存贮器 4 次，因此需要 4 个机器周期。前 3 个机器周期 M_1、M_2、M_3 用于取指令，M_4 用于执行指令。

指令执行示意图如图 7-7 和图 7-8 所示。

指令执行过程如下：

M_1：T_1、T_2、T_3：读取指令；

T_4：指令译码；

M_2：T_1：PC OUT 和 Status OUT；

T_2：PC←PC＋1；

T_3：Z←B_2；

M_3：T_1：PC OUT 和 Status OUT；

T_2：PC←PC＋1；

T_3：W←B_3；

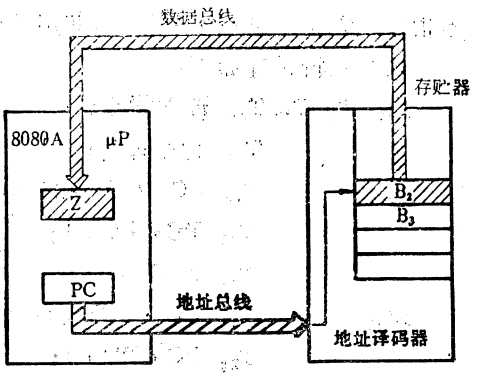


图 7-7　指令第二字节进入Z

至此，WZ 寄存器对已含有 16 位地址，将这个地址送到存贮器去读取数据。

M_4：T_1：WZ OUT 和 Status OUT；

T_2、T_5：A←DATA

至此，取数指令执行完毕，完成这条指令共用了 4 个机器周期 13 个时态。

图 7-8 为指令 LDA 执行前后各单元数据变化情况。设该指令的 3 个字节分别置于存贮器的 2200H、2201H、2202H 地址单元中，指令第二字节 B_2 的内容为 20H，第三字节 B_3 的内容为 23，存贮器 2320H 中的内容为 56H。

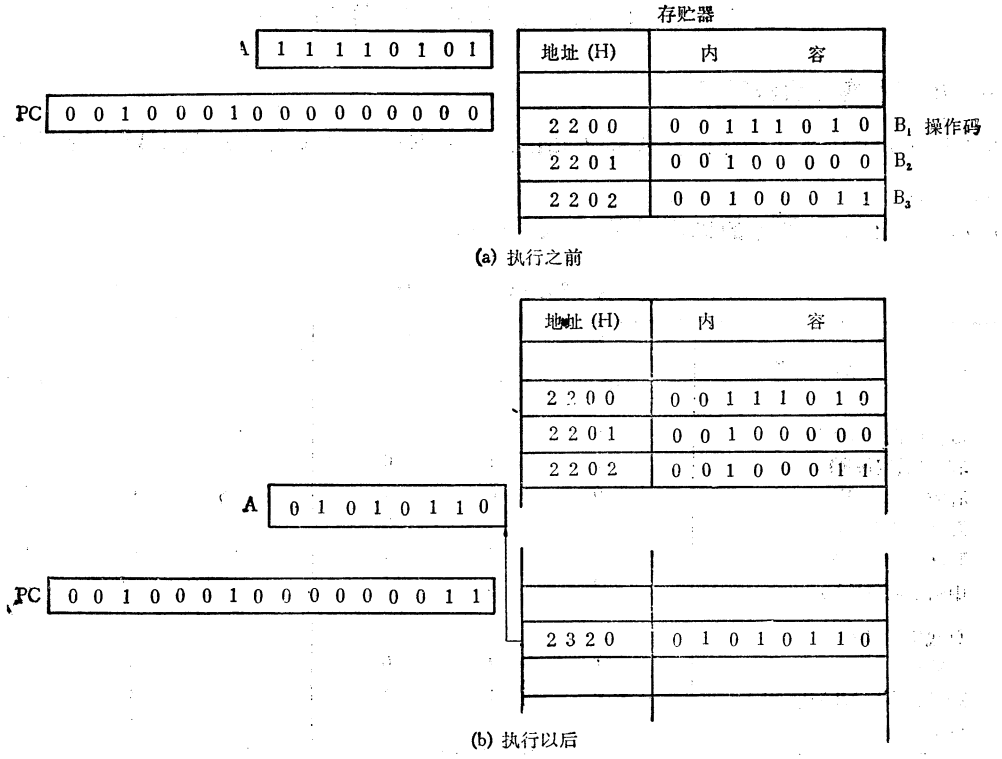


图 7-8　执行 LDA 指令前后各单元的内容

5. JMP addr (无条件转移指令)；PC←B_3B_2

这条指令的功能是使程序转移到现行指令的第三字节 B_3 和第二字节 B_2 所指定地址的那条指令。它是一条三字节指令，共用 3 个机器周期 10 个时态完成这一指令的操作。

指令执行过程如下：

M_1；T_1、T_2、T_3：读取指令；

T_4：指令译码；

M_2：T_1：PC OUT 和 Status OUT；

T_2：PC←PC + 1；

T_3：Z←B_2；

M_3：T_1：PC OUT 和 Status OUT；

T_2：PC←PC + 1；

T_3：W←B_3；

经过 M_1、M_2、M_3 机器周期已经将程序的转移地址 B_3、B_2 送入 W、Z 寄存器对内，随后在下一条指令的 M_1 周期内，WZ 的内容代替 PC 的内容输出到地址总线，从而完成了程序转移。即在下一条指令的 M_1 机器周期内：

T_1：WZ OUT 和 Status OUT；

T_2：PC←WZ + 1；

至此，完成了程序的转移。

8080A 其它指令的执行周期详见附录 4，这里不再一一列举了。

§7-2 8085A 微处理器的操作时序

一、8085A 指令执行周期

8085A CPU 是根据指令的操作性质和访问存贮器或 I/O 设备的次数来划分执行指令的机器周期的。

8085A 共有 7 种类型的机器周期，如表 7-4 所示。

表 7-4 8085A 机器周期分类表

机器周期	状态			控制		
	IO/M	S_1	S_0	RD	WR	INTA
取操作码(OF)	0	1	1	0	1	1
存贮器读(MR)	0	1	0	0	1	1
存贮器写(MW)	0	0	1	1	0	1
I/O 读 (IOR)	1	1	0	0	1	1
I/O 写 (IOW)	1	0	1	1	0	1
中断响应(INA)	1	1	1	1	1	0
总线空闲(BI)：						
DAD	0	1	0	1	1	1
INA(RST/TRAP)	1	1	1	1	1	1
HALT	T_S	0	0	T_S	T_S	1

说明：0=逻辑"0"，1=逻辑"1"，T_S=高阻抗

二、8085A 指令时序分析

现以 STA 指令为例，说明 CPU 是怎样根据指令的操作性质取指令和执行指令的。

STA 指令的功能是把累加器内容存入指令第三字节和第二字节所指出的直接地址的存贮单元中，即$(B_3B_2)\leftarrow A$。

STA 指令的 CPU 时序分析表如表 7-5 所示。

表7—5　STA 指令的CPU 时序分析表

机器周期	指令周期			
	M_1	M_2	M_3	M_4
时钟周期	T_1 T_2 T_3 T_4	T_1 T_2 T_3	T_1 T_2 T_3	T_1 T_2 T_3
机器周期类型	取操作码 (OF)	存贮器读 (MR)	存贮器读 (MR)	存贮器写 (MW)
地址总线	PC OUT 指出指令的第一个字节 B_1	PC+1 OUT 指出指令的第二个字节 B_2	PC+2 OUT 指出指令的第三个字节 B_3	(WZ) OUT: M_2 M_3 取出的直接地址 B_3 B_2
数据总线	指令操作码 $IR\leftarrow B_1$	直接地址 低位字节 B_2 $Z\leftarrow B_2$	直接地址 高位字节 B_3 $W\leftarrow B_3$	累加器 A 的内容 $DB\leftarrow A$

8085A 的指令周期包含 1～5 个机器周期，每个机器周期包含 3～6 个时钟周期。所有指令周期的第一个机器周期必是取指周期。

8085A CPU 的指令系统中，有 74 条基本指令，246 种操作码。其中，最短的指令需要 4 个 T 时钟周期，最长的指令需要 18 个 T 时钟周期。

根据 STA 指令的时序波形图(如图 7-9 所示)，具体分析如下：

1．取操作码机器周期(OF)

(1) 8085A 在每一机器周期开始时，首先在控制线上送出 3 个状态信号($IO/\overline{M}$, S_1, S_0)以识别是哪一类型的机器周期。在 M_1 的 T_1 时态，由于 $IO/\overline{M}=0$, $S_1=1$, $S_0=1$，表明是取操作码机器周期(OF)；

(2) 与此同时 8085A CPU 送出 16 位地址以指明所寻址的存贮单元或 I/O 端口的地址。对 OF 周期来说，PC 的内容被送至地址总线上，其中高 8 位(PC_H)被送至A_{15}～A_8线上，低 8 位 (PC_L) 被送至 AD_7～AD_0 线上。但是，AD_7～AD_0 上的地址信息只保留一个时钟周期 (T_1)，此后总线驱动器就进入虚线所示的高阻抗状态。这是因为 AD_7～AD_0 是分时的多重地址/数据总线，即在每一个机器周期的 T_1 时态，AD_7～AD_0 输出地址的低 8 位，而 T_2、T_3 时态 AD_7～AD_0 中出现所需要的信息(操作码或数据)。

(3) 由于现在是取指周期，因此读入 CPU 的信息是 STA 指令的操作码。又因为 AD_7～AD_0 线上的地址信息只保留一个时钟周期 (T_1)，所以必须把此地址信息锁存在外部的地址锁

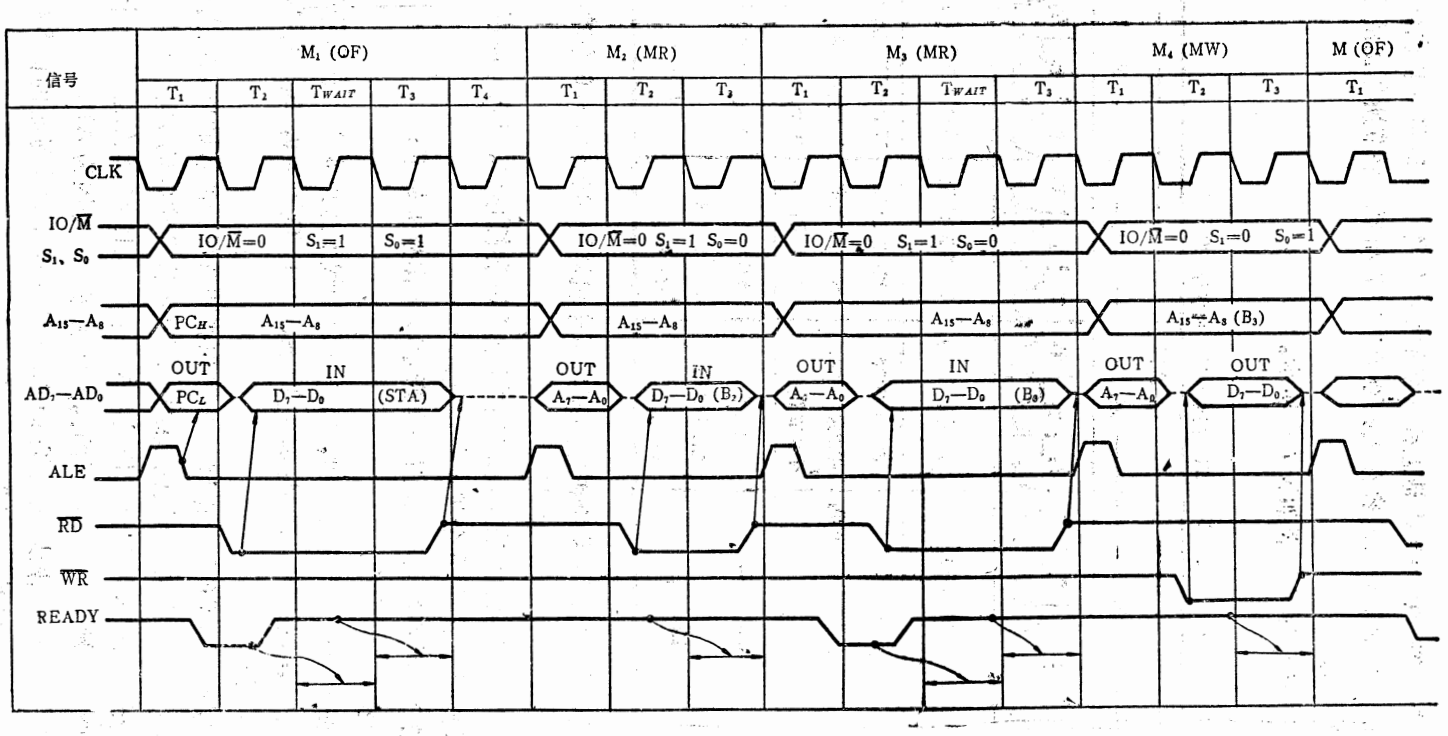


图 7-9　STA 指令的执行时序图

存器里。8085A CPU 必须在每一个机器周期的 T_1 时态发出一个允许地址锁存信号 ALE，并用ALE 的脉冲下降沿，将低 8 位地址信息 A_7～A_0 锁入锁存器。这样，形成 16 位地址经地址总线送至 RAM，以寻找所指定的内存单元。

(4) 接着，在 T_2 时态，CPU 发出 $\overline{RD}$ 读控制信号(低电平有效)，启动所寻址的存贮器。

(5) 随后，这个存贮器开始驱动 AD_7～AD_0 线(即图中虚线转为实线)，即开始输出 STA 操作码。

(6) 经过一个存取时间之后，在 T_3 时态，AD_7～AD_0 线上的数据已经稳定下来，此时 CPU 把 AD_7～AD_0 线上的操作码读入指令寄存器。然后，使 $\overline{RD}$ 信号变为高电平(无效)，禁止对所寻址的存贮器进行读数。至此，8085A 完成了取指令操作码的过程。

由于是指令的第一个机器周期(OF)，因此这时 CPU 会自动进入 T_4 时态，并对 IR 中的操作码进行译码，以判定是进入下一个机器周期的 T_1 时态，还是进入取指周期的 T_5 时态(有些指令在取操作码周期需要 T_5、T_6 时态，如 DCX 指令)。

在每一个机器周期的 T_2 时态，CPU不断查询 READY 线，如果存贮器(或 I/O 设备)的存取时间大于 CPU 的系统存取时间，则存贮器将使 CPU 的 READY 线变为低电平，通知 CPU 进入等待状态(T_{WAIT})直到 READY 线变成高电平，CPU 方才退出 T_{WAIT} 状态，而进入T_3时态。T_{WAIT} 是时钟周期 T 的整数倍，它的长短取决于存贮器的存取时间。

2．存贮器读机器周期(MR)

(1) 当 STA 指令操作码被取出后，PC 自动加 1(即 PC＋1→PC)，并在 M_2 机器周期的 T_1 时态发出总线状态信号 $IO/\overline{M}=0$，$S_1=1$，$S_0=0$，以识别 M_2 是存贮器读周期。

(2) 与此同时，CPU 将 PC_H 中的内容送至 A_{15}～A_8 线上，而把 PC_L 中的内容在 ALE 脉冲下降沿经 AD_7～AD_0 线锁入地址锁存器中，所形成的 16 位地址，经地址总线送至 RAM，选中指令第二字节的地址单元。

(3) 在 T_2 时态，CPU 发出 $\overline{RD}$ 信号，启动所选中的内存单元的数据，经 AD_7～AD_0 数据总线读入 CPU。

(4) 随后进入 T_3 时态，在 T_3 时态结束之前，STA 指令的第二字节(B_2)已被取入 CPU 内部的 Z 寄存器中。

除了取操作码机器周期 M_1 外，所有的机器周期都由 3 个时钟周期组成。因此，在 M_2 的 T_3 时态之后，接着就进入 M_3 机器周期。

M_3 机器周期仍为存贮器读机器周期，这时 PC 再次自动加 1 后，把 16 位地址经地址总线送至 RAM，选中 STA 指令的第三字节(B_3)，并在 $\overline{RD}$ 信号控制下，把 B_2 取入 CPU 内部的W 寄存器中。当 M_3 的 T_3 时态结束时，CPU 已经全部取出 STA 指令的 3 个字节，随即进入 M_4 机器周期，执行指令所规定的操作。

3．存贮器写机器周期(MW)

M_4 是存贮器写机器周期。(1) 在 M_4 机器周期的 T_1 时态，CPU 发出总线状态信号 $IO/\overline{M}=0$，$S_1=0$，$S_0=1$，以识别 M_3 是存贮器写机器周期。(2) 与此同时，CPU 将 M_2、M_3 机器周期取出的两个字节数据(B_2)、(B_3)送至地址总线上，而把累加器(A)的内容放在数据总线上。(3) 在 $\overline{WR}$ 写控制信号的作用下，在 T_2、T_3 时态，CPU 将累加器 (A) 中的内容存入地址由 STA 指令的第三、二字节(B_3、B_2)所指定的 RAM 单元中，从而完成 STA 指令的全部操作。随即转入下一个指令的机器周期 M_1。

在 MR、MW 机器周期中，T_2 时态仍然不断地查询 READY 线的状态，以决定是否要插入等待状态 T_{WAIT}。

4．I/O 读和 I/O 写周期(IOR、IOW)

上面以 STA 指令为例分析了访问存贮器的读、写过程。显然，对于 I/O 端口的读、写过程，除了在 MR (MW) 时 $IO/\overline{M}=0$，而在 IOR(IOW) 时 $IO/\overline{M}=1$ 有所区别外，两者的时序大致相同。在这里，$IO/\overline{M}$ 状态信号用于识别现行机器周期的类型，究竟是存贮器读写周期，还是 I/O 读写周期，这一点是关键所在。

在 IOR 周期，来自 INPUT 指令的第二字节 (B_2) 的地址信息重迭地进入 $AD_7\sim AD_0$ 和 $A_{15}\sim A_8$。而在 IOW 周期，来自 OUTPUT 指令的第二字节(B_2)的地址信息重迭地进入 $AD_7\sim AD_0$ 和 $A_{15}\sim A_8$ 线。它们的执行周期只能发生在第三个机器周期。

5．中断响应周期

在现行指令的执行过程中，一旦外设发出中断请求信号 INTR，如果此时 CPU 允许中断，即事先已执行过 EI 指令，使中断允许触发器 INTE 置位(INTE = 1)，则 CPU 接受中断请求后会自动判别出是 TRAP 还是 RST，或者是 INTR 的中断请求，并先响应具有最高优先级的中断请求。于是，CPU 就进入中断响应周期(INA)。这个周期与取指周期(OF)相比，除了这时送出的是中断响应信号 $\overline{INTA}$ 而不是 $\overline{RD}$ 信号外，其它情况大致相同。

其过程大致如下：(1) 首先在 CPU 完成现行的指令周期后，令 PC 停止计数，从而进入中断响应周期(INA)。(2) 使中断允许触发器复位(INTE = 0)。(3) CPU 发出 $\overline{INTA}$ 信号，此时，外部中断逻辑应提供相应的指令操作码。显然，RST 和 CALL 指令是最合理的选择对象，因为这两条指令会使 CPU 在转移到新的地址单元之前，把 PC 的内容推入堆栈并保存起来。(4)如果这时是片外中断请求 INTR，那么，①在 INA 周期的 M_1 机器周期，可编程序的中断控制器(如 8259)会自动地把 CALL 操作码送入 CPU。CPU 接受操作码后，立即进行译码，同时确定 CALL 指令需要的另外两个字节。② 当CPU 执行 INA 周期的 M_2 机器周期时，从 8259 器件中取出指令的第二字节(B_2)。③ 在 INA 周期的 M_3 机器周期，再从 8259 器件中取出 CALL 指令的第三字节(B_3)。至此，CPU 已经取出用于中断响应的完整指令。应当指出，在 INA 周期的 3 个机器周期内，CPU 禁止程序计算器(PC)递增。这样，在 M_4 和 M_5 期间可以把正确的 PC 值置入堆栈。④ 在 M_4 和 M_5 期间，CPU 执行存贮器写周期，先把 PC 的高位字节推入堆栈，接着把 PC 的低位字节推入堆栈，然后再把在 M_2 和 M_3 时取出的两个字节置入 PC。这样，就使程序转移到 CALL 指令所指出的地址单元上。

6．总线空闲周期(BI)

从上述可知，8085A 中的大多数机器周期与读或写有关，但有两个例外：

(1) 在 DAD 指令(寄存器对相加)的 M_2 和 M_3 机器周期。因为 8085A CPU 需要用 6 个时钟周期去执行 DAD 指令所要求的双倍字长的加法操作，但是又不希望 M_1 具有 10 个时钟周期(4 个正常的，6 个额外的)。所以，由 CPU 产生两个不进行存取的存贮器机器周期，用于 CPU 内部的操作。这种机器周期称为总线空闲机器周期。此时，除 $\overline{RD}$ 信号维持高电平及不发出 ALE 信号之外，总线空闲周期与存贮器读周期的情况相同。

(2) 在 RST 5.5、RST 6.5、RST 7.5 或 TRAP 中断内部操作码产生期间，它与正常的 OF 周期的区别在于此时不发出 $\overline{RD}$ 信号，而是由内部产生 RESTART 操作码，使数据总线空闲起来。经过 $T_1\sim T_6$ 时态之后，进入 M_2 机器周期，把 PC 内容送入堆栈保护起来。

应当指出，在 BI 周期内不查询 READY 线状态，而由 T_2 时态直接进入 T_3 时态。

§7-3 Z80 微处理器的操作时序

一、Z80 CPU 的定时

Z80 CPU 的指令周期是由 1～6 个机器周期组成的。所有指令周期的第一个机器周期必是取指周期。

每个机器周期 (M) 由 3～6 个时钟周期(T 时态)组成。T 时态是 Z80 CPU 中处理动作的最小时间单位。

在 Z80 CPU 的指令系统中，有 150 条基本指令，696 种操作码。其中最短的指令需要 4 个 T 时态，而最长的指令需要 23 个 T 时态。尽管如此，Z80 CPU 同 Intel 8080A/8085A CPU 一样，仍然是通过一些基本的操作来完成各类指令操作的。这些基本操作包括：

1．存贮器读或写；

2．I/O 设备读或写；

3．中断响应。

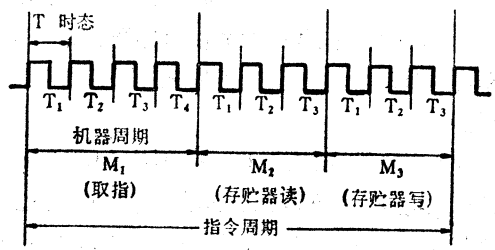


图 7-10 Z80 CPU 基本时序图

Z80 的所有指令都是这些基本操作的组合。图 7-10 示出了 Z80 CPU 基本指令周期的时序波形图，图中指令周期包含 3 个机器周期，即取指周期、存贮器读和存贮器写周期。其中，取指周期由 4 个 T 时态组成；存贮器读或写机器周期都由 3 个 T 时态组成。

二、Z80 CPU 的典型时序分析

我们可以把 Z80 CPU 完成各类指令的时序分解成为 7 种最基本的典型时序进行分析。即 Z80指令包括以下 7 种基本机器周期：

1．取指令操作码(M_1)周期；

2．存贮器读或写周期 (MEM R/W)；

3．输入/输出读或写周期 (I/O R/W)；

4．总线请求/响应周期(BUS RQ)

5．可屏蔽中断请求/响应周期 (INT RQ)；

6．不可屏蔽中断请求/响应周期 (NMI RQ)；

7．暂停响应周期及其解除。

现分述如下：

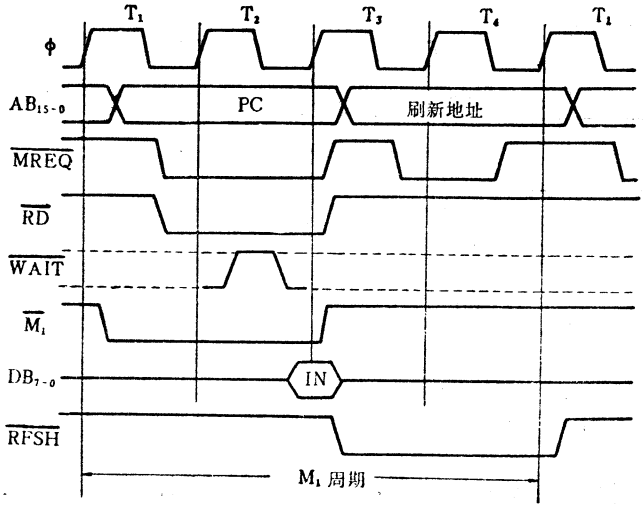


图 7-11(a) 取指周期时序波形图

(一)取指令操作周期(M_1 周期)

通常，执行任何一条指令总是分为三个步骤：取指——译码——执行。因而，任何一个指

令的第一个机器周期必然是取指周期。为了与别的机器周期相区别，用 M_1 信号来表示，故取指周期又称为 M_1 周期，它包括 4 个时钟周期 T_1～T_4（在没有插入 T_W 的情况下）。其时序如图 7-11(a) 所示。

1．当 CPU 开始进入 M_1 周期时（即 $\overline{M_1}$ 变为有效——低电平），在 T_1 上升沿后略为延迟一段时间，PC 的内容被送到地址总线上，它指出了存贮指令操作码的单元地址。

2．在 T_1 的半个时钟周期之后（即 T_1 下降沿后），$\overline{MREQ}$ 信号变为有效（低电平），此时存贮器的地址信号已经稳定下来，故 $\overline{MREQ}$ 的下降沿可直接用作存贮器的片选信号。

3．与此同时 $\overline{RD}$ 信号也变为有效，它命令存贮器执行读操作，并把读出的数据放到数据总线上。

4．每个机器周期的 T_2 下降沿和 T_W（等待时态）下降沿，都查询 $\overline{WAIT}$ 信号，若有等待请求，则下一时态进入 T_W；若无等待请求，则进入 T_3。

5．CPU 在 T_3 上升沿采样数据总线，而取入指令操作码。同时，用此上升沿脉冲使 $\overline{RD}$、$\overline{MREQ}$ 和 $\overline{M_1}$ 信号失效（恢复高电平）。因而在 $\overline{RD}$ 信号失效前，数据已被 CPU 采样。

6．取指周期的 T_3、T_4 时态，CPU 对取入的指令进行译码并在 CPU 内部执行，这时不会访问存贮器，故可用来刷新动态存贮器。(1) 在 T_3～T_4，由 R 寄存器将刷新地址送到地址总线低 7 位上（由R寄存器提供）。(2) 与此同时，刷新信号 $\overline{RFSH}$ 变为有效。(3)随后，用 T_3 时态 ϕ 的下降沿使存贮器请求信号 $\overline{MREQ}$ 再次有效。只有当 $\overline{RFSH}$ 和 $\overline{MREQ}$ 两个信号同时有效时，才能保证动态存贮器按刷新地址所指定的行，进行该行的刷新操作。然后 R 寄存器内容自动增 1。由于刷新期间，$\overline{RD}$ 无效。因此，信号不会读到数据总线上。

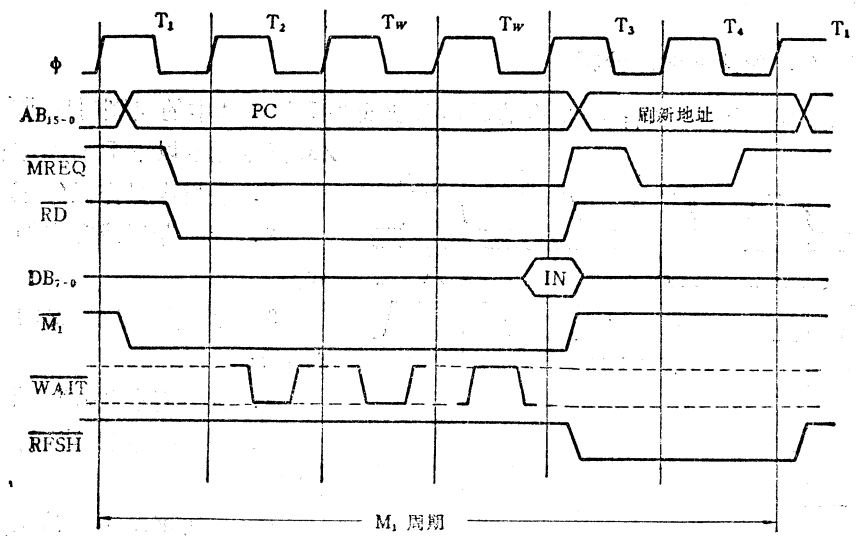


图 7-11(b)　具有 T_W 时态的取指周期时序波形图

7．如果存贮器的存取速度较慢，则利用 $\overline{WAIT}$ 线，可在 T_2 时态后插入任意个 T_W 时态。CPU 在 T_2 以后每一个 T_W 时态 ϕ 的下降沿采样 $\overline{WAIT}$ 线。若其为低（低电平有效）则插入 T_W 时态，具有 T_W 时态的取指周期的时序如图 7-11(b) 所示。

Z80 CPU 指令分一、二、三、四字节指令。每个单字节指令只有一个取指周期 M_1，取出一个 8 位操作码。但是，在二、三字节的指令中，具有两个字节操作码的指令和四字节指令，一定具有两个 M_1 周期。

(二)存贮器读或写周期(M_1)

如上所述，在 M_1 周期内，CPU 将对所取的指令操作码进行译码，然后根据译码的结果决定下一步执行什么操作。

存贮器读或写是最基本的操作，它用于 CPU 与存贮器之间交换数据。这些操作是在存贮器读或写周期内完成的。在没有 T_W 插入时的存贮器读或写周期是由 T_1～T_3 时态组成的。其时序分别如图 7-12(a) 和(b)所示。

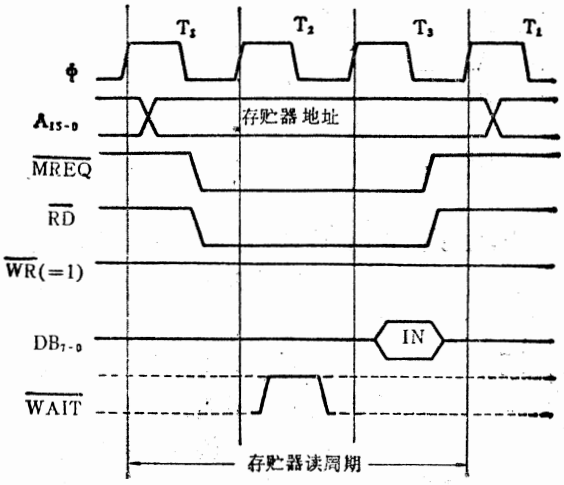


图 7-12(a) 存贮器读周期时序波形图

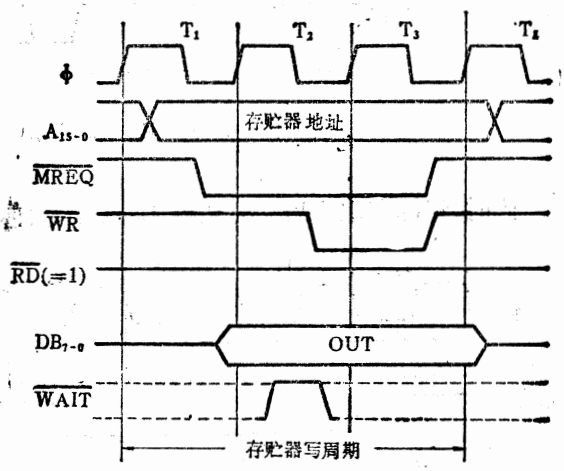


图 7-12(b) 存贮器写周期时序波形图

1．存贮器读周期时序

(1) T_1 上升沿，CPU 将地址码送到地址总线上；(2) 在 T_1 下降沿至 T_3 下降沿，$\overline{MREQ}$ 与 $\overline{RD}$ 变为有效，准备向存贮器读取信息；(3) 在 T_3 下降沿，CPU 从数据总线上采样数据，送入 CPU 内的有关寄存器；(4) 在 T_2 和 T_W 下降沿，查询 $\overline{WAIT}$ 信号，若有等待请求，则进入 T_W。

应当指出，存贮器读与取指周期在时序上是有区别的。其区别在于：存贮器读是在 T_3 下降沿，CPU 采样数据总线，同时使 $\overline{MREQ}$ 与 $\overline{RD}$ 信号变为无效；而在 M_1 周期内，则是在 T_3 上升沿采样数据总线上的指令操作码。此外，存贮器读写周期都不对动态 RAM 进行刷新。

2．存贮器写周期时序

(1) T_1 上升沿，CPU 将地址码送到地址总线上；(2) 在 T_1 下降沿，CPU 将欲写入存贮器的数据送至数据总线上；(3) 在 T_1 下降沿至 T_3 下降沿 $\overline{MREQ}$ 变为有效；(4) 在 T_2 下降沿 $\overline{WR}$ 变为有效。这时在 $\overline{MREQ}$ 与 $\overline{WR}$ 同时有效作用下，将数据写入存贮器。(5) 在 T_2 和 T_W 下降沿，查询 $\overline{WAIT}$ 信号，若有等待请求，则进入 T_W。

具有 T_W 时态的存贮器读或写周期的时序如图 7-12(c)所示。

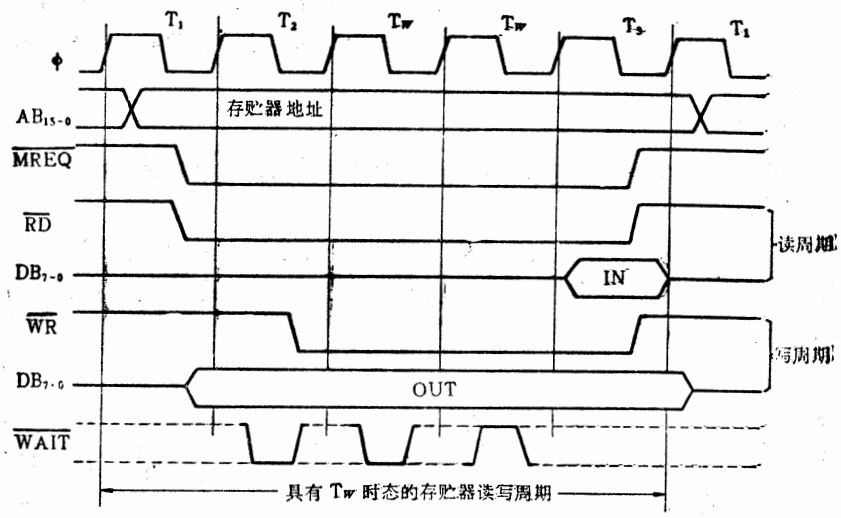


图 7-12(c) 具有 T_W 时态的存贮器读写周期时序波形图

此时，情况与取指周期时插有 T_W 时态一样。这里，为了简化，将读、写周期画在一张图上。

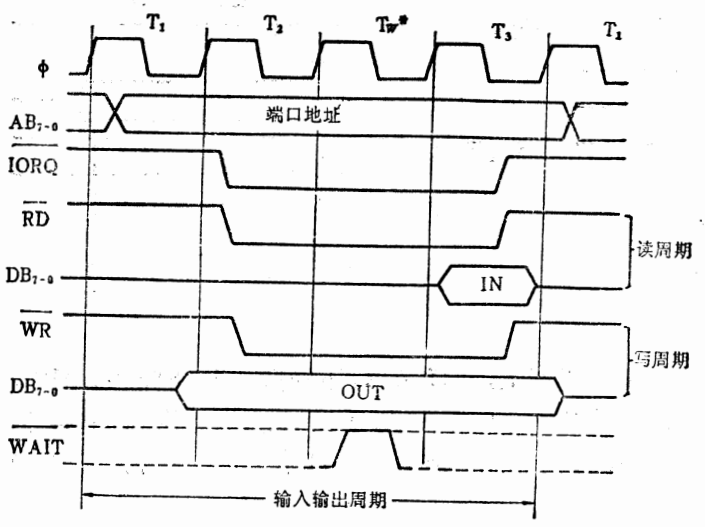


图 7-13(a) 输入输出时序波形图

（三）输入或输出（I/O）周期

输入输出周期是用来实现 CPU 与外部设备之间进行数据交换的。每个输入或输出周期包括 4 个 T 时态 T_1、T_2、T_W^*、T_3。其时序如图 7-13(a) 所示。

（1）T_1 上升沿，CPU 将 I/O 端口地址送到地址总线的低 8 位（$A_7 \sim A_0$）上；（2）在 T_2 上升沿后至 T_3 下降沿，$\overline{IORQ}$ 变为有效；（3）与此同时，I/O 读信号 $\overline{RD}$ 也变为有效；（4）T_2 时态结束后，由 CPU 内部自动插入一个 T_W^* 等待时态。这是因为在 I/O 操作期间，从 $\overline{IORQ}$ 变为有效到 CPU 对 $\overline{WAIT}$ 线（T_2 下降沿）进行采样的时间间隔很短（小于半个 T 时态），如果不插入 T_W^*，则即使 I/O 端口要求等待，也来不及对其地址译码并发出 $\overline{WAIT}$ 请求。同时还考虑到目前还难以设计出速度完全赶得上 CPU 的 MOS 型的 I/O 接口器件。（5）对于 I/O 读周期，在 T_3 下降沿，在 $\overline{IORQ}$ 和 $\overline{RD}$ 信号共同作用下，把所寻址的端口中的数据放到数据总线上，这与存贮器读时的情形相似。（6）对于 I/O 写周期，在 T_3 下降沿，在 $\overline{IORQ}$ 和 $\overline{WR}$ 信号共同作用下，把数据总线的数据写入所选中的 I/O 端口中。

在 I/O 读写周期，CPU 自动插入一个 T_W^* 时态后，在 T_W^* 下降沿采样 $\overline{WAIT}$ 线，如果外设仍然跟不上 CPU 的速度，还可继续插入所需要的 T_W 时态。具有 T_W 时态的 I/O 时序如图

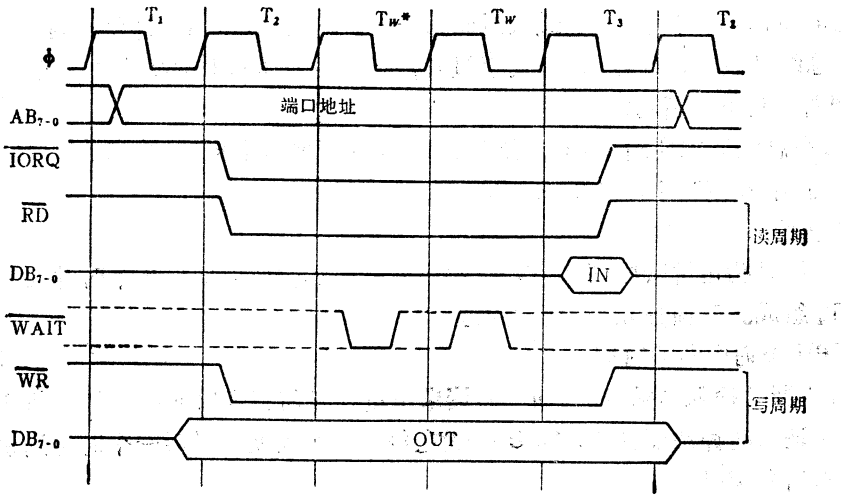


图 7-13(b)　具有 T_W 时态的 I/O 时序波形图

7-13(b)所示，其原理同前所述。

(四)总线请求和响应周期

总线请求和响应周期用于直接存贮器存取方式(DMA 方式)。此时，要求 CPU 的地址总线和数据总线处于高阻抗状态，即让 CPU 暂时交出对系统总线的控制权，由请求的外设控制使用。当外设需要控制总线进行 DMA 操作时，就向 CPU 发出总线请求 $\overline{BUSRQ}$ 信号，CPU 在每一个机器周期的最后一个 T 时态的上升沿，采样 $\overline{BUSRQ}$ 线。若此信号有效(低电平)，则 CPU 就在下一时钟脉冲的上升沿，向请求使用总线的外设发出总线响应 $\overline{BUSAK}$ 信号。它表明 CPU 已将其地址、数据和三态控制信号线置成高阻抗状态。此时，外设就可以控制系统总线，以便在存贮器与 I/O 设备之间直接传送数据。总线请求和响应时序如图 7-14 所示。

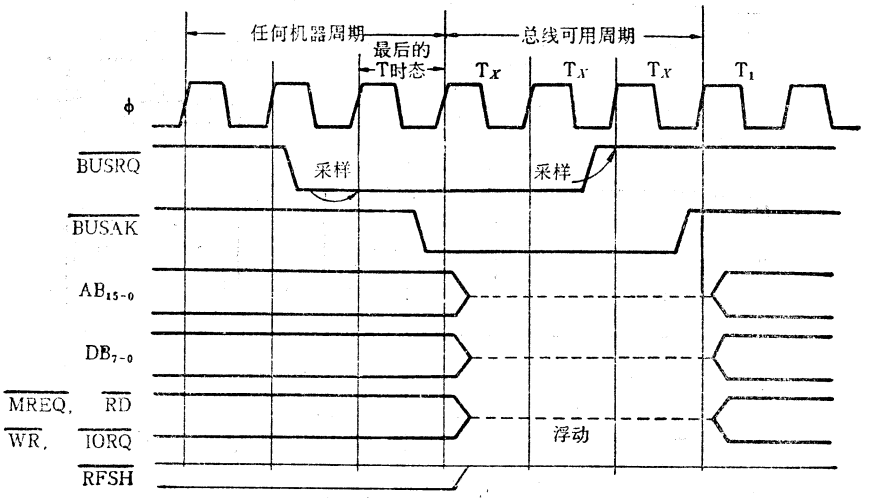


图 7-14　总线请求和响应时序波形图

CPU 进入总线响应周期后，在每个 T_X 时态的 ϕ 上升沿，查询 $\overline{BUSRQ}$ 信号，一旦发现 $\overline{BUSRQ}$ 变为无效时，则在该 T_X 下降沿，使 $\overline{BUSAK}$ 失效，通知外设总线响应周期已告结束。在下一个机器周期的 T_1 时态，CPU 恢复对系统总线的控制权。

由上可知，CPU 响应总线的最长时间为一个机器周期，原则上外部控制器可以根据需要的时间占用总线。但是，如果 DMA 周期过长，可能致使动态 RAM 丢失信息。在这种情况下，外部控制器应具备刷新功能。

此外，在总线请求期间，不可屏蔽中断 NMI 和可屏蔽中断 $\overline{INT}$ 都不能中断 CPU。换言之，$\overline{BUSRQ}$ 请求级别高于 $\overline{INT}$ 和 NMI。

（五）中断请求和响应周期

这里中断是指可屏蔽中断 $\overline{INT}$，这种中断请求和响应周期由 6 个时钟周期 T_1、T_2、T_W^*、T_W^*、T_3、T_4 组成。下面分几个问题讨论。

1．CPU 查询中断的时间

通常，中断的输入是随机的，即中断请求可以在任何指令周期的任何时态发生。CPU 内部的逻辑能保证用时钟给予同步。CPU 一般在每一指令周期的最后一个时钟周期的上升沿去查询中断请求信号 $\overline{INT}$。

2．CPU 响应中断的条件

在出现下列任何一种情况时，CPU 不能响应中断：(1) 由软件控制的中断允许触发器 IFF_1（IFF_2）复位时；（2）有 $\overline{BUSRQ}$ 请求时；(3，有 NMI 请求时；(4）有复位信号 $\overline{RESET}$ 时。

3．CPU 响应中断的时序

当 CPU 符合响应中断条件后，就响应外设的中断请求，产生一个特殊的 M_1 周期作为中断响应周期，其时序如图 7-15(a) 所示。

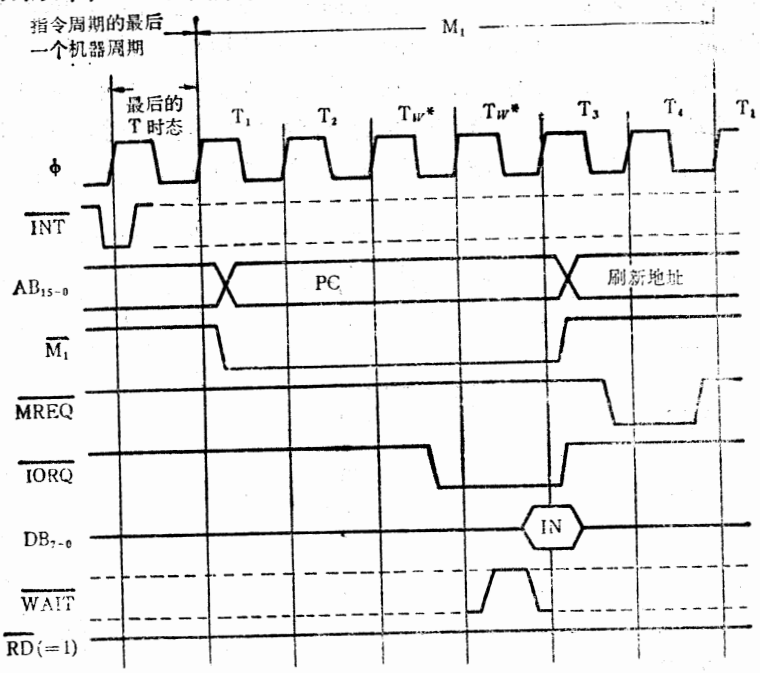


图 7-15(a)　中断请求和响应时序波形图

这里的 M_1 周期与取指周期 M_1 不同，其差别如下：(1) 不产生 $\overline{MREQ}$ 信号（仅在刷新期间产生）；(2) 产生 $\overline{IORQ}$ 信号，以表明是中断响应周期；(3) CPU 利用 $\overline{M_1}$ 和 $\overline{IORQ}$ 同时有效（低电平），作为 $\overline{INT}$ 中断响应信号，通知请求中断设备，按中断选择方式，向数据总线送一指令或中断矢量；(4) 在中断响应周期 T_2 时态后，CPU 自动插入两个等待时态 T_W^*，其目的

是当有多个外设同时请求中断时，有足够时间实现链式优先权中断方式。(5) CPU 在 T_3 的 ϕ 上升沿，读取数据总线上的信息；(6) T_3、T_4 时态也用于动态 RAM 的刷新，故在 T_3 的 ϕ 下降沿使 $\overline{MREQ}$ 有效。

应当指出，Z80 CPU 虽然自动插入两个 T_W^* 时态，但若外设仍然来不及检测中断响应信号并形成中断矢量时，则可在两个 T_W^* 时态后，再插入外设请求等待的时态。自动插入的第二个 T_W^* 下降沿用来对 $\overline{WAIT}$ 信号采样。当 CPU 识别出 $\overline{WAIT}$ 信号有效时，就在第二个 T_W^* 结束之后，插入一个 T_W 时态而不进入 T_3 时态。并利用 T_W 下降沿继续对 $\overline{WAIT}$ 信号进行查询。如果它仍然处于低电平有效状态，就再插入一个 T_W 时态，直到对它查询发现 $\overline{WAIT}$ 信号变为无效为止。此时，表明外设的数据已准备好。在 T_3 到来之前，将 8 位中断矢量放到数据总线上，随即进入 T_3 时态。图 7-15(b) 表示具有等待状态的中断请求/响应周期时序。

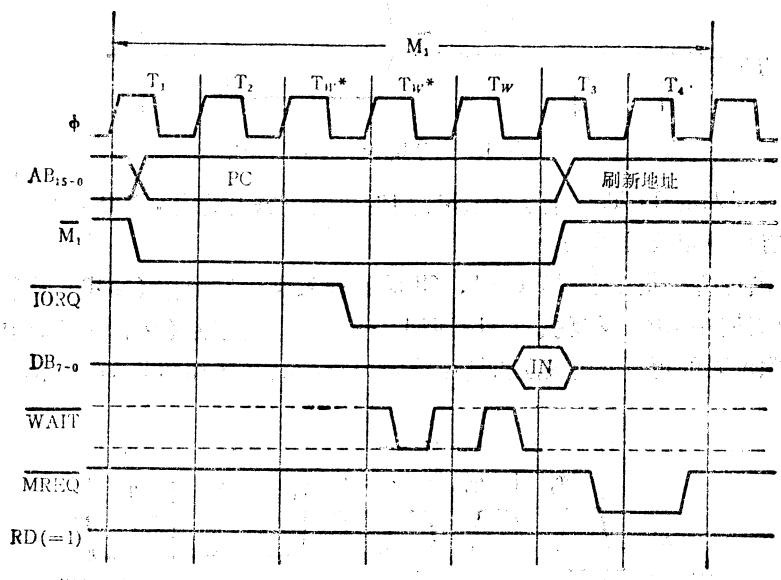


图 7-15(b) 具有 T_W 时态的中断响应时序波形图

值得指出，在中断响应周期内，$\overline{RD}$ 信号始终无效。

(六)不可屏蔽中断请求和响应周期

Z 80 CPU 设有不可屏蔽中断请求线 $\overline{NMI}$，以用于在一些紧急事故时要求 CPU 立即处理，例如电源掉电等。它的优先权高于任何可屏蔽中断，而且不受内部中断允许触发器状态的影响。也就是说，不能用软件方法来禁止不可屏蔽中断请求。图 7-16 示出了不可屏蔽中断请求/响应周期的时序。

$\overline{NMI}$ 信号和其它中断请求信号同时被采样，但此中断线比屏蔽中断线具有更高的优先权，因而 CPU 首先响应 $\overline{NMI}$ 中断请求，并立即在下一个机器周期进入不可屏蔽中断请求/响应周期。它与取指周期 M_1 很相似，唯一的差别是对数据总线上的内容置之不理，即当 $\overline{M_1}$、$\overline{MREQ}$ 和 $\overline{RD}$ 同时有效时，CPU 并不读取数据总线上的内容，而是在内部自动产生一个 RST 指令(详见第十二章)，把 PC 存入堆栈，然后转移到 0066H 地址单元执行不可屏蔽中断服务程序。不可屏蔽中断请求/响应周期的 T_3、T_4 时态，仍然用来作为动态存贮器的刷新周期。其

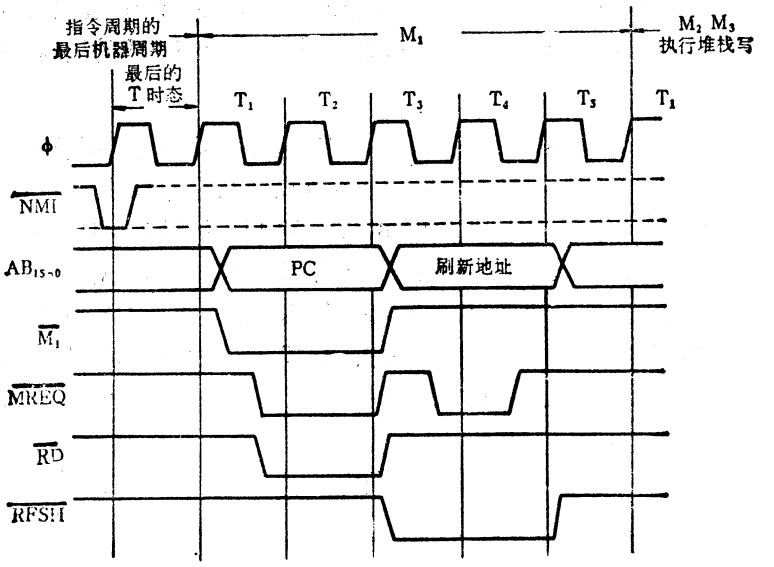


图 7-16　不可屏蔽中断时序波形图

操作过程与取指周期的 T_3、T_4 一样。M_2、M_3 周期执行堆栈写操作，把 PC 推入堆栈保护起来。

（七）暂停响应周期及其解除

每当 CPU 执行暂停指令（HALT）时，CPU 就进入执行空操作（NOP）指令的暂停响应周期，在暂停响应周期中 CPU 反复执行 M_1 周期，一直到接受中断（或不可屏蔽中断，或中断允许触发器 IFF 置位时的可屏蔽中断）和复位信号为止。其时序如图 7-17 所示。

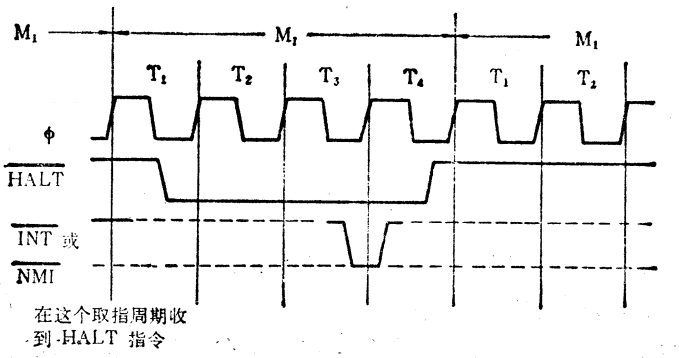


图 7-17　暂停响应周期及其解除时序波形图

CPU 在每一个 M_1 周期的 T_4 上升沿对这两条中断线采样，检测有无 $\overline{INT}$、$\overline{NMI}$ 和 $\overline{RESET}$ 信号。如果收到不可屏蔽中断或可屏蔽中断（中断允许触发器为置位时），则在下一时钟脉冲的上升沿将解除暂停状态，转入相应的中断响应周期。如果同时接收到不可屏蔽中断和可屏蔽中断，则将首先响应不可屏蔽中断。如果收到 $\overline{RESET}$ 信号，就转入复位状态。在暂停状态执行空操作的目的是为了维持存贮器刷新信号有效，以对存贮器进行刷新。暂停状态中的每个机器周期和一般的取指周期 M_1 相似，差别在于对从存贮器来的数据不予理睬并强制 CPU 执行空操作指令。在暂停状态时，CPU 使暂停信号 $\overline{HALT}$ 变为有效，以表明微处理器处于暂停状态。

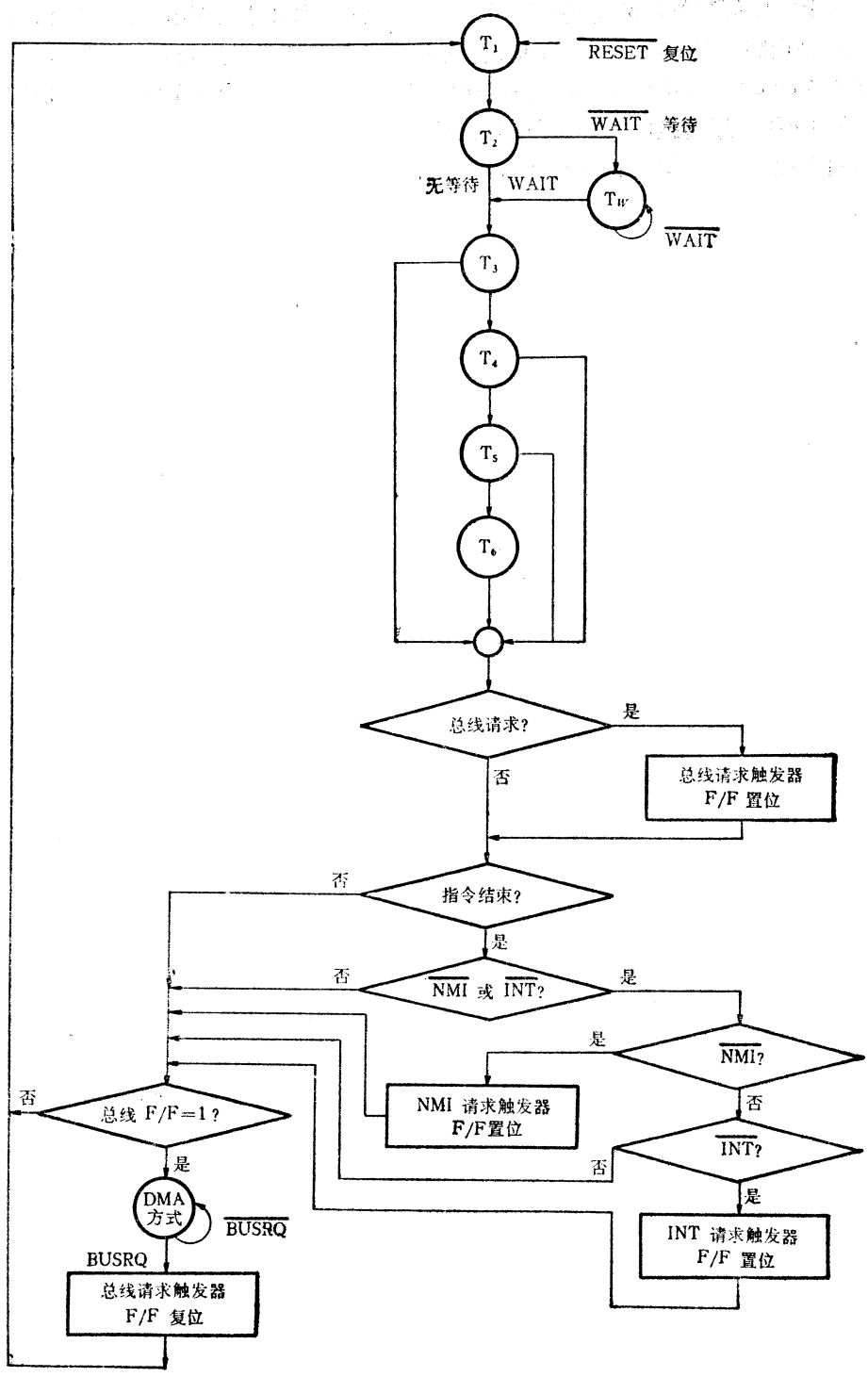


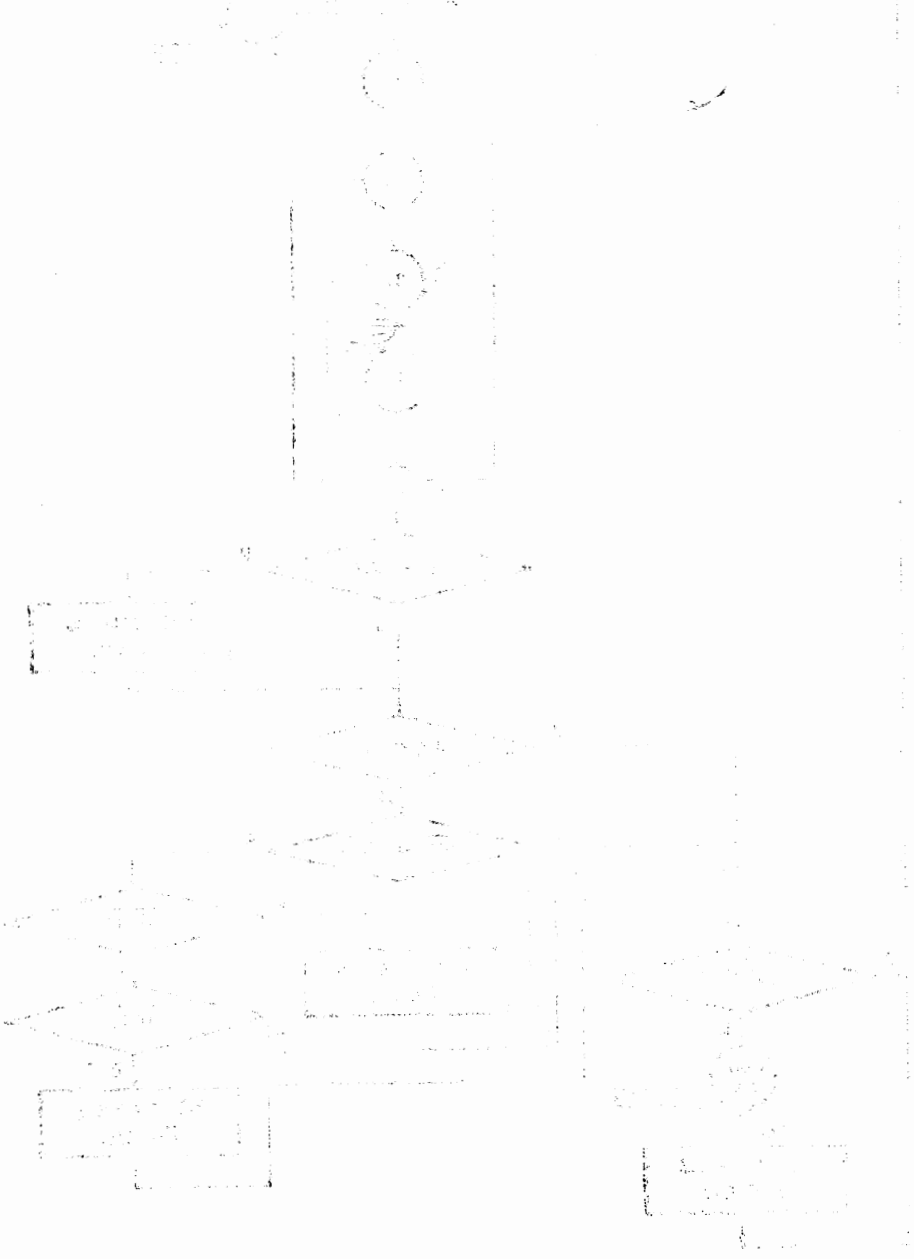
图 7-19 Z80 CPU 时序转换图

三、Z 80 CPU 时态转换流程

如上所述，任何一条 Z 80 指令都是由上述的 7 种基本机器周期组成的，可由附录 3 上查到每条指令是由哪些基本机器周期组成的，它共需多少个机器周期和多少个 T 时态。

为了对 Z80 CPU 的操作时序建立起完整的概念，我们可将上述 7 种基本机器周期有机地连接成 Z80 CPU 时序转换图。如图 7-18 所示。

读者可参照 Intel 8080A 时态转换图的分析方法自行分析。



第八章　微型计算机的汇编语言

本章以 Intel 8080A/8085A 和 Z80 汇编语言为例，着重讨论汇编语言的结构、规范以及汇编程序的功能。

§8-1　程序设计语言

程序设计语言(Programming language)通常分为机器语言、汇编语言和高级语言等三类。

一、机器语言

如前所述，微型计算机指令系统本身就是一种程序设计语言，这是一种用二进制代码"0"或"1"来表示指令和数据的最原始的程序设计语言，但却是机器唯一能够识别的程序设计语言，习惯上称之为机器语言。

但是，用这种机器语言编程，非常繁琐、费时，因而一般都不用机器语言编程。

二、汇编语言(初级语言)

汇编语言是一种用助记符来表示的面向机器的程序设计语言。这种语言比较直观、易懂，而且容易记忆。对指令中的操作码和操作数，也容易区分清楚。例如若要求计算机执行两个数 32 和 16 相加，则用助记符形式表示的程序为：

```
LD      A, 20H
ADD     A, 10H
HALT
```

微型计算机并不能直接识别汇编语言程序。因此，用汇编语言编写的源程序，在交付微型计算机执行之前，必须借助于汇编程序翻译成用机器语言表示的目标程序。当然，对于容量不大的汇编语言程序也可以用原始的手工汇编方法，将其表示为机器语言代码。在这里，机器语言程序是目标程序，而汇编语言程序是源程序。汇编程序是执行把源程序翻译成目标程序的程序，它与汇编语言程序是两回事。

由于汇编语言与特定的计算机的结构和指令系统密切相关，其符号操作码和机器语言操作码成一一对应关系。因此，对于不同类型的微型计算机，针对同一问题所编的汇编语言程序往往是互不通用的。程序员必须先熟悉该机器的硬件结构、指令系统以及寻址方式，然后方能进行程序设计。由此可知，汇编语言虽比机器语言有所进步，但仍然比较繁琐、费时。由于，用汇编语言编程时，可以直接操作到机器内部的寄存器，能把计算过程刻划得非常具体，因而经过推敲能编制非常紧凑的程序，既可节省内存贮器的容量，又能提高程序执行的速度，在时间及空间上都能充分发挥计算机的潜力，很适合于实时控制的场合。

三、高级语言

高级语言是一种面向过程的、独立于计算机的通用语言。利用高级语言编程，人们可以不

必去了解计算机的内部逻辑，而主要集中精力研究解题、算法和过程的描述。由于这种语言对解题过程的描述，比较接近于人们的习题，因而易学、易懂。程序员用高级语言编程，比用汇编语言编程约快10倍。同样，高级语言程序必须由其解释程序或编译程序翻译成目标程序，计算机方能加以执行。

四、机器语言、汇编语言和高级语言的比较

上述三种语言都有各自的特点，究竟哪种语言更适宜于微型计算机的应用，这取决于目前微型计算机应用的特点。当前微型机应用的特点是：大量用作专用机，特别是用于工业控制，其本身的存贮容量较小。由于用机器语言编程序太繁琐、费时，且容易出错，故不宜采用。而高级语言限于下述原因，目前还不宜广泛采用：① 高级语言程序是由计算机的编译程序自动地"硬译"成机器语言的，故其目标程序较长，与用汇编语言写的源程序相比，其目标程序大约长1～2倍。当然其所占的存贮容量也较大；② 高级语言编制的程序，执行时间也较长(约长1～3倍)；③ 一般编译程序要占16K～32K内存，要求机器的存贮容量大。而汇编语言则由于与微型机本身的结构特点密切相关，因此，可编写出紧凑而有效的汇编语言程序，以充分发挥微型机所特有的灵活性。汇编程序一般仅占2～4K内存，所以一般的微型计算机系统，都可配上汇编程序。

因此，从当前来看，在科学计算、企业管理方面采用高级语言较合适，而在工业控制智能仪器仪表中采用汇编语言更为适宜。

鉴于上述比较，本章将着重介绍汇编语言及其源程序的结构和规范。

§8-2 汇编语言源程序的规范

汇编语言是一种面向机器的程序设计语言，对于不同的计算机，其汇编语言是各不相同的。但是，它们之间所采用的语言规则有很多相似之处。本节以Intel 8080A/8085A和Z80汇编语言为例来说明汇编语言的规范。读者一旦掌握了这里所介绍的语言规则，对其它微型机的汇编语言，就可以举一反三，触类旁通。

一、汇编语言源程序的格式

汇编语言源程序是由汇编语句(即指令语句)组成的，汇编语句由四个部分(或段)组成，其格式如下：

标　　号 Label	操　作　码 Opcode	操作数(地址) Operand	注　　释 Comment

各段之间通常用冒号"：", 分号"；", 逗号"，"和空格等作为分界符分隔开来。

标号和注释部分是任选项，操作码部分是必须项，有的语句也可不带操作数。

下面分别说明组成汇编语句的四个段的含义：

1．标号(名字)段

标号是指令的符号地址，它用来给一条指令的地址赋予一个符号名字，其值为被汇编的那条指令所在单元的地址。这样我们便可以在程序的其它地方引用这个符号来表示一个地址单

元。标号是由英文字母、数字和其它特殊字符组成的，其长度可为1～6个符号，但第一个符号必须是一个英文字母或特殊符号"?"或@。标号必须以冒号":"结束。应当指出，不能用该机器指令系统的助记符、伪指令码或寄存器名称作为标号。标号是任选的，并非每条指令都有一个标号。但一旦使用标号指定某一地址，在程序的其它地方就不能再修改这个定义，即保持一义性(除了在几个SET或DEFL伪指令之前标号可以相同外)。

对Intel 8080A/8085A的伪指令EQU、SET和MACRO来说，它们要求一个名字，除了不是以冒号结束外，其余的规则与标号相同。这里应当注意，名字不一定要求用冒号结束，而标号必须以冒号结束。

按照上述规定，以下一些字符串都不能用作标号：如DAA:、4EC:、EQU:、LABEP等。

凡是多于6个字符的标号，汇编程序会自动截取前面6个字符作为标号，而舍去后面部分。

2．操作码段

这一个部分可能有三种码：一是指令助记符；二是伪指令码；三是宏指令的助记符。

3．操作数段

这一部分是操作码所要操作的数据。至于这个数据的形式如何，这与具体微处理器的指令形式有关。在8080A/8085A和Z80中，这个操作数部分可以有以下四种类型的信息：寄存器、寄存器对、立即数(8位或16位)，以及16位的存贮器地址。而表示这些信息，可能有九种方法。这些方法归纳在表8-1中。

表8-1 操作数段的表示形式

所需信息	表示方法	
(1) 寄存器	(1) 十六进制数据(H)	(6) ASCII码
(2) 寄存器对	(2) 十进制数据(D或无字尾)	(7) 符号赋值
(3) 立即数(8位或16位)	(3) 八进制数据(Q)	(8) 指令的标号
(4) 16位存贮器地址	(4) 二进制数据(B)	(9) 表达式
	(5) 当前指令地址($)	

(1) 十六进制数据

十六进制数据是目前最常用的微型计算机的工业标准数制。

例如，在8080A(或Z80)中，

```
标号      操作码      操作数      注释
HERE:   MVI(LD)    A,2DH      ；把十六进制数2D送入寄存器A
```

十六进制数后面要写字母"H"。一般在十六进制中，不宜以字母开头来表示数，若开头是字母的话，必须冠以数字0，以避免与以字符表示的名字相混淆。

(2) 十进制数据

例如，在8080A(或Z80)中，

```
标号    操作码        操作数   注释
ABC:   MVI(LD)     C,45     ；把十进制数45送入寄存器C
```

十进制数据后面可用字母D注明，亦可不注。

(3) 八进制数据

表示方法与上述两种相同，但八进制数后面要写字母"Q"。(八进制为 Octal 本应用首字母 O 表示，但为了避免与数字 0 相混淆，故用字母 Q 注明)。

例如，8080A (或 Z80)中，

```
标号        操作码        操作数      注释
LABEL:  MVI(LD)      A,72Q      ; 把八进制数 72 送入寄存器 A
```

(4) 二进制数据

例如，在 8080A (或 Z80)中，

```
标号        操作码        操作数          注释
BINARY:  MVI(LD)     D,01100110B     ; 把 66H 送入寄存器 D
```

二进制数后面要用字母"B"注明。

(5) 现行指令地址($)

符号 $ 表示现行程序计数器内容。即指现行指令的第一字节地址。

例如，在 8080A 中，

```
标号  操作码  操作数  注释
GO:  JMP    $+5    ; 从本指令第一字节的地址开始向后跳 5 个单元
```

由于 8080A/8085A 中没有相对寻址方式。因此，用此方法并不能节省内存单元，故很少采用。

又如，在 Z80 中，

```
标号      操作码  操作数     注释
LOOP:  JR      C, $+5    ; 同上。
```

由于 Z 80 中有相对寻址指令，因此，常采用符号 $。

(6) ASCII 码

在汇编程序中规定，ASCII 字符必须用单引号括起来。

例如，在 8080A (或 Z80)中，

```
标号   操作码       操作数     注释
KIM:  MVI(LD)   E, '*'     ; 将用 ASCII 码表示的'*'(2AH)送入 E 寄存器
```

ASCII 码的表示方法见表 2-6。

(7) 已赋值的符号

下面我们将要提到，伪指令 EQU 能把数值赋予一个符号。假定我们把某一符号 VALUE 作为名字(或标号)，并用伪指令 EQU 为它赋值 2DH，则下面例子中的第一个指令语句和第二个指令语句是等效的：

例如，在 8080A (或 Z80)中，

```
标号        操作码          操作数       注释
LABEL1:  MVI(LD)  A,    VALUE
LABEL2:  MVI(LD)  A,    2DH
```

(8) 指令的标号

当某一指令具有一个标号时，这个标号即代表该指令第一字节的地址。以后，在程序中凡要提到这条指令的地址时，可以用其标号表示。

例如，在 8080A (或 Z80)中，

```
标号      操作码        操作数      注释
LOOP:   MVI(LD)       C,35H
          ⋮             ⋮
        JMP(JP)       LOOP      ；转移到标号为 LOOP 的指令
```

JMP LOOP(或 JP LOOP)指令中操作数段 LOOP，就是指 MVI C, 35H(LD C, 35H)这一指令的第一字节地址。

(9) 表达式

在操作数部分，可以是算术表达式或逻辑表达式，在汇编过程中，这个表达式的值将被计算出来。汇编程序对表达式的操作顺序是先乘除后加减；先括号内后括号外；先算术运算后逻辑运算。

例如，在 8080A(或 Z80)中，

```
标号   操作码       操作数              注释
       MVI(LD)     A, (18+48H)/2       ；把(18+72)/2=45 放累加器 A 中
       JMP(JP)     COUNT+2             ；转移到 COUNT 地址以后两单元的地址
```

4．注释部分

注释字段中要遵守的法则是：它必须以分号“;”开始，若一行不够写，另起一行时也必须以“;”开始。注释段可以使程序文件更加清楚易读。注释部分不会被译成任何机器代码。

在写注释时，应说明为什么要执行这条指令，或者在什么条件下执行这条指令。此外，在调用子程序时，要解释进入子程序和退出子程序的条件，以及这段子程序执行什么内容。一般用英语或 ASCII 符号书写注释。

二、汇编命令(伪指令——Pseudo-Instruction)

汇编命令在形式上与指令相似，但它并不直接被翻译成机器代码。对汇编程序来说，它仅是一种命令，以便在汇编时提供必要的控制信息，从而执行一些特殊操作。例如：为程序指定一个存贮区域；把一些表格和数据放入存贮器；定义一些符号等等。

伪指令主要有下述 8 种，如表 8-2 所示。

表 8-2 Intel 8080A/8085A 和 Z80 的伪指令

伪指令名称	Intel 8080A/8085A	Z80	伪指令名称	Intel 8080A/8085A	Z80
程序起始	ORG	ORG	定义数据字	DW	DEFW
程序结束	END	END	定义存贮区	DS	DEFS
符号赋值	EQU	EQU	定义字符串		DEFM
重新赋值	SET	DEFL	条件汇编	(1) IF ENDIF	IF〈表达式〉语句组
定义数据字节	DB	DEFB		(2) IF ELSE END IF	END IF

现分述如下：

1．ORG(Origin)程序起始

伪指令 ORG 用来规定主程序、子程序或数据在存贮器中的起始地址。在汇编程序中有一个汇编地址计数器，它相当于一个程序计数器，该计数器指出下一条指令或下一项要处理的数据的存贮单元位置。ORG 能把位置计数器置成由操作数表达式 exp 所规定的值，exp 值必须是一个 16 位地址值。它的格式为

```
标号      操作码      操作数
          ORG         表达式(exp)
```

例如，在 8080A（或 Z80）中，

```
存贮地址(H)  标号      操作码          操作数
                      ORG (ORG)   2000H
2000                  MOV(LD)     A,C
2001                  ADI (ADD)   12H(A,12H)
2003                  JMP (JP)    NEXT
             HERE:    ORG         2050H
2050         NEXT:    XRA (XOR)   A
```

第一条 ORG 伪指令通知汇编程序，以下这段程序的目标程序的起始地址将从存贮地址 2000H 开始；第二条 ORG 伪指令把 2050H 置入汇编地址计数器，通知汇编程序从该地址开始继续汇编机器指令码或数据。ORG 伪指令的操作数段中也可使用表达式，与前相同，式中的符号必须在引用前定义过。如果程序员在程序开始时不设置 ORG 伪指令，则汇编程序将从存贮地址“0000H”号开始存放目标程序。

2．END 程序结束

END 伪指令用于通知汇编程序，该源程序已经结束。在一个程序中，只能允许出现一个 END 语句，而且必须安排在源程序的末尾。伪指令 END 格式为

```
标号      操作码      操作数
          END
```

3．EQU(Equate)符号赋值

伪指令 EQU 格式为

名字　EQU 表达式（8080A/8085A）

标号　EQU 表达式（Z80）

EQU 伪指令把操作数段中的地址或数据赋值给这个名字或标号，并且只能赋值一次。

例如，在 8080A 中，

```
标号    操作码    操作数    注释
TTY     EQU       5
LAST    EQU       3000
```

在以后程序中，若遇到符号 TTY，就可用 5 代替；若遇到符号 LAST，就可用 3000 代替。对于大多数汇编程序，都可以把一个名字赋值给另一个名字，或者把一个表达式赋值给一个名字，但在操作数段中的名字或表达式中的名字，必须在引用前被定义过。

例如：

```
LAST    EQU    FINAL
ST1     EQU    START+2*5
```

其中 FINAL 和 START 必须在以前已经定义过。

又如，在 Z80 中用 EQU 语句定义表格长度。

```
存贮地址                    源程序
(H)
2100        TABLE:      ⋮
⋮
2106        LENGTH:  EQU    $-TABLE
⋮                       ⋮
203F                 LD     IX,LENGTH
```

这里 $ 字符表示汇编程序当前所指的内存地址，本例中应为 2106H，而 TABLE 标号相应的单元地址为 2100H，则表达式 $-TABLE = 6。故语句 LENGTH: EQU $-TABLE 表示将数值 6 赋值给 LENGTH(即 2106H)符号。当源程序执行语句 LD IX, LENGTH 时，就表示将数值 6 送入 IX 变址寄存器。

显然，采用这种方法可大大简化数据表程序。

4．SET 和 DEFL(Define Label)符号重新赋值

伪指令 SET 和 DEFL 格式为

名字 SET 表达式 (8080A/8085A)

标号：DEFL 表达式 (Z80)

伪指令 SET、DEFL 与 EQU 一样能对符号赋值，唯一的区别是：在同一程序中，伪指令 SET、DEFL 可以对一个名字(或标号)重新予以赋值(即多次赋值)。

例如，在 8080A 中，

```
标号      操作码   操作数     汇编后的目标代码(H)
IMMED     SET     5
          ADI     IMMED      C605
IMMED     SET     10H-6
          ADI     IMMED      C60A
```

此例说明在第一个 SET 之后，语句 ADI 表示累加器和立即数 5 相加，而在第二个 SET 之后，语句则表示累加器和立即数 10 相加。

又如，在 Z80 中有如下程序：

```
          ⋮
COUNT:    DEFL   4
          LD     A,COUNT
          ⋮
COUNT:    DEFL   COUNT-1
          LD     B,COUNT
          ADD    A,B          ; A←4+3
          ⋮
          END
```

在这段程序中 COUNT 开始等于 4，后来等于 3。若在重新汇编时要改变两个 COUNT 的值，只需要改变起始定义的 COUNT 即可。

5．DB 和 DEFB(Define Byte)定义数据字节

DB 为定义数据字节指令。这个指令用来确定在程序中存放到存贮器中的数据。该数据可以是 8 位的单字节，也可以是一个字符串。

DB 伪指令的格式为

标号：DB 表达式或字符串

表达式可以是算术表达式，也可以是逻辑操作表达式，它将计算出一个 8 位数据字节。如果是字符串，则应用单引号括起来。操作数段中，如果有多个项目，则各项之间应用逗号分开。8080A/8085A 规定：若项数超过 8 项，就需要再增加 DB 命令。当某一语句的标号确定以后，这个标号所指的是 DB 指令操作数段中第一个字节的存贮地址。例如，下例中的标号 STR 应指字符串 'TIME' 的第一个字母T的地址。

例如，

```
标号      操作码  操作数        注释
STR:      DB      'TIME'    ;  把54494D45存于STR开始的相继单元中
HERE:     DB      0A3H      ;  把A3H存于HERE单元中
WORD:     DB      -03H,5*2  ;  把FD0A存于WORD和WORD+1单元中
```

6．DW 和 DEFB(Define Word)定义数据字指令。

DW 伪指令格式为

标号：DW 表达式或字符串

DW 的作用与 DB 基本相同，但其表达式计算出的是16位字长的值(一个数据字包含两个字节)。它的低8位存贮在标号所表示的单元地址中，高8位存贮在标号加1的单元地址中。如果操作数段中有多个项目(8080A/808A 规定不超过八项)，则各个项目之间用逗号分开，而每个项目都是16位字长，其低8位和高8位的存贮地址依次如上所述排列。

例如：

```
标号    操作码  操作数      注释
ADDR:   DW      4645H   ;  将45H存于ADDR单元，46H存于ADDR+1单元
```

7．DS 和 DEFS(Define Storage)定义存贮区

DS 伪指令用来定义一个存贮区。

DS 伪指令格式为

标号：DS 表达式

操作数段中应是算术或逻辑表达式。这个表达式的值指出了为存贮数据而需要保留的字节的数目。DS 伪指令与 DB、DW 伪指令不同，在汇编过程中，需要保留的这些字节的值并不汇编出来，它仅要求知道需要保留的这些字节的数目。如果这个语句有标号，则把汇编地址计数器的当前值赋给这个标号，这样就指出了保留存贮区域的第一字节地址。然后，DS 伪指令使汇编地址计数器增量，增量的值应等于操作数段中表达式的值。

例如：

```
存贮地址  标号     操作码  操作数   注释
                   ORG     2040H
2040H     TEMP:    DS      10     ;  保留其后10个字节，下一条指令将从TEMP+0AH处开始汇编
204AH     PRES:    DS      10H    ;  保留其后16个字节，下一条指令将从PRES+10H处开始汇编
```

在 DS 伪指令的操作数段中出现的任何符号，应在使用该指令之前预先定义过。

8．DEFM 定义字符串(仅 Z80 有)

伪指令 DEFM 的功能是将字符串翻译成7位 ASCII 代码。

DEFM 语句的格式为

标号：DEFM $←中间为字符串→$

符号$为定界符，在字符串前、后各用一个定界符，字符串经转换为相应的7位 ASCII 代码后顺序存入到以标号为起始地址的内存单元中，字符串中的空格(SPACE)也应当用相应的代码表示。

例如，ERROR: DEFM 'ERROR'

此伪操作将7位的 ASCII 字符 E、R、R、O 和 R 存入以符号 ERROR 为起始地址的5个单

元中。

下面举例说明用汇编语言编写的源程序。

例如：双倍精度数的加法程序

一个双倍精度数 N_1，存放在存贮单元 AUGEND 和 AUGEND+1，另一个双倍精度数 N_2，存放在单元 ADDEND 和 ADDEND+1。$N_1(N_2)$ 的低位字节放在 AUGEND (ADDEND)，而高位字节放在 AUGEND+1(ADDEND+1)。N_1 和 N_2 相加后，其结果放在 AUGEND 和 AUGEND+1，结果的低位字节在 AUGEND。

用 Z80 汇编语言编写的源程序如下：

```
标号        操作码  操作数                    注释
            ORG     2040H
            LD      HL,       (AUGEND)    ; 把被加数放入 HL
            LD      DE,       (ADDEND)    ; 把加数放入 DE
            ADD     HL,DE                 ; 被加数与加数相加
            LD      (AUGEND), HL          ; 保存 16 位结果
            HALT
AUGEND:     DW      AF8AH
ADDEND:     DW      0A90H
            END
```

图 8-1 说明上例中，在执行两个双精度数相加的程序前后，存贮器中内容的变化情况。

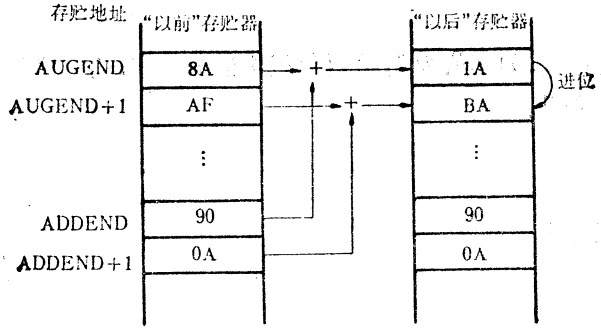


图 8-1 双倍精度加法示意图

三、条件汇编(IF 语句)

条件汇编伪指令用来在汇编过程中，有条件地汇编或不予汇编某部分源程序。即仅当某个特定条件满足时，才对相应的源程序进行汇编，否则不予以汇编。这样当一个通用的源程序，用在某个具体情况时，就可去掉无关的程序段，而得到最简单的目标代码。

1. 8080A/8085A 条件汇编

它有两种格式：

格式 1：IF　ENDIF

```
标号：  IF   表达式
        语句组
标号：  ENDIF
```

汇编程序将计算条件表达式。如果其值为真，则在 IF 与 ENDIF 之间的语句组将被汇编成目标程序；否则，这段语句组将不予以汇编。

例如：

```
标号    操作码   操作数          汇编后的目标代码(H)
COND    SET      1
        IF       COND
        MOV      A,     C        79
        ENDIF
COND    SET      0
        IF       COND
        MOV      A,     C        不汇编
        ENDIF
```

格式 2：IF ELSE ENDIF

```
标号    操作码   操作数
        IF       表达式
        语句组 1
        ELSE
        语句组 2
        ENDIF
```

汇编程序将计算表达式，如其值为 1，将语句组 1 汇编出来，而忽略语句组 2；否则，只汇编语句组 2，而不汇编语句组 1。

2．Z80 条件汇编

它的格式为

```
        IF       表达式
        语句组
        ENDIF
```

同样地，在计算了表达式之后，只有当表达式的值为真时，汇编程序才汇编 IF 之后的语句组，否则，这段语句组将不予以汇编。

四、宏(Macros)指令

1．宏定义，宏调用，宏扩展

一个较长的程序段往往包含了许多重复的指令序列。为了简化源程序，我们可以把这些重复出现的指令序列用一条宏指令来代替，而由汇编程序在汇编时产生所需要的代码。一般用宏指令定义的指令序列是很短的，不应超过 10～15 条指令。对于更长的指令序列，为节省存贮空间，应将它们汇编成子程序。

为了用宏指令来代替某一指令序列，首先应给这一指令序列定义一个名字，叫做宏定义。8080A/8085A 和 Z80 的宏定义都用 “MACRO” 语句开始，而以 “ENDM” 语句结束。在 “MACRO” 与 “ENDM” 之间的指令序列，是构成宏指令的主体，称为宏体。8080A/8085A 和

Z80 宏定义的一般格式为

标号	操作码	操作数		
宏名字[注](:)	MACRO	宏参数表		
	重复使用的指令序列		(宏体)	宏定义
	ENDM			

宏指令一旦定义后，就可以象汇编语言中其它指令那样任意调用，称为宏调用。由于在调用前必须先进行定义，因此，宏定义通常放在源程序的开始部分。源程序中出现宏名字时，汇编程序将它替换为宏体，这个过程称为宏扩展。源程序中的宏指令如何在目标程序中多次扩展为一串指令序列 YYY，如图 8-2 所示。

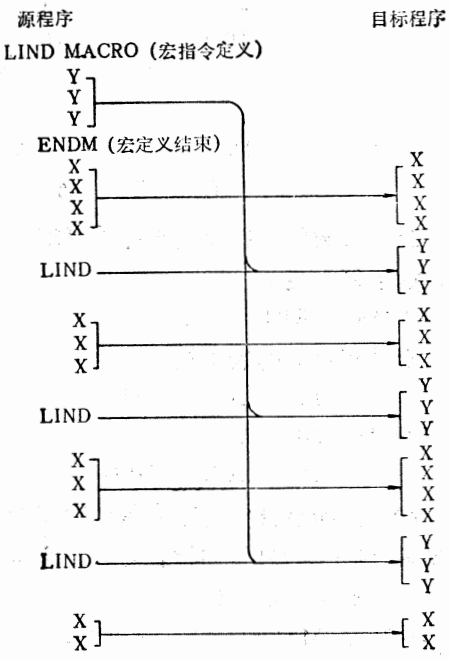


图 8-2　宏指令多次扩展为宏体

2．带参数的宏指令

宏指令不仅能使源程序更加简洁明了，而且它还具有接受参数的能力，因而功能更加灵活。

带参数的宏指令是指在宏指令中带有参数（操作数）的宏指令。带参数的宏定义格式为

```
宏名字：    MACRO  〈形式参数 1，  形式参数 2〉
            宏指令体
            ENDM
```

例如，在 Z80 中有一个带有延时参数的宏指令定义如下：

```
带参数的宏定义
DVMS:  MACRO  #TIME
       LD     A,    #TIME
LOPD:  DEC    A
       JR     NZ,   LOPD      } 宏体
       ENDM
```

上面表示宏名字为“DVMS”带有一个形式参数 TIME。前面加一个“#”号，是为了与其他标号区分开。当程序中出现此宏名字（即宏调用），并且在其操作数部分带有一个实在的参数时，则此宏指令就是“带参数的宏指令”。

宏名字必须与宏定义名一致。若宏定义中没有形式参数，则在宏调用中也没有参数。

宏调用的格式为

```
〈标号〉宏名字：  〈实在参数 1，实在参数 2〉
    （如 DVMS）  （如操作数为 0）
```

其中标号并不是必须项，可根据程序中的要求来选定。

这样，当宏指令被定义后，只要在程序中一旦出现带参数的宏指令，汇编程序就会用宏体取代宏指令，而且宏体中的形式参数将被宏指令中所规定的实在参数所代替。

例如：

【注】 8080A/8085A 宏指令的宏名字不要求以冒号结束，Z80 宏指令的宏名字要求以冒号结束。

原来的源程序（带参数的宏定义）

```
DVMS:   MACRO   TIME
        LD      A,   #TIME
LOPD:   DEC     A
        JR      NZ,  LOPD
        ENDM
PWPOS:  IN      A,   (02H)
        BIT     5,   A
        JR      Z,   PWPOS
        DVMS    0
PHFIR:  IN      A,   (02H)
        BIT     7,   A
```

带参数的宏指令宏扩展后的等效源程序

```
PWPOS:  IN      A,   (02H)
        BIT     5,   A
        JR      Z,   PWPOS
        LD      A,0
LOPD:   DEC     A
        JR      NZ,  LOPD
PHFIR:  IN      A,   (02H)
        BIT     7,A
```

当带参数的宏指令的参数改变时，仍可完成相应的操作，而原先带参数的宏定义可以不变。如上例中变为

```
DVMS    80H
```

则其相对应的宏扩展为

```
        LD    A,    80H
LOPD:   DEC   A
        JR    NZ,   LOPD
```

Z80 汇编语言中的带参数的宏指令，其中实在参数可以是操作数，也可以是字符。若有两个参数，则中间要用逗号“,”分开。例如：

原来的源程序

```
DVMS:   MACRO #X,   #TIME
        LD    #X,   #TIME
LOPD:   DEC   #X
        JR    NZ,   LOPD
        ENDM
        ⋮
        DVMS  C,    3DH
```

展开后的源程序

```
        LD    C,    3DH
LOPD:   DEC   C
        JR    NZ,   LOPD
```

从上述可以看出，带参数的宏指令中的参数，提供了产生有限改变的能力，这就使宏指令具有更大的灵活性，从而使宏指令得到广泛的应用。

3．宏指令和子程序的异同点

使用宏指令和子程序有以下共同的优点：

(1) 用户只需写一次指令序列，就可以多次使用这组指令序列。

(2) 程序更为清楚和易读。

(3) 如果要修改一组指令序列，则只要修改一次，而不必在每次遇到指令序列时都加以修改。

(4) 用户更易于使用现成的程序库、标准指令序列或某组指令的集合等。

宏指令与子程序的不同点可归纳如下几点：

(1) 在处理方式上，宏指令是在汇编时进行处理并对宏调用进行宏扩展的；而子程序是在运行时处理，并由主程序调用子程序，然后再从子程序返回主程序的。

(2) 在目标程序中，宏指令是把宏调用替换为相应的宏体，即宏指令并没有简化目标程序。原程序中宏调用的次数越多，得到的目标代码就越长。而子程序不存在扩展的问题。在源程序中，不论有多少转子指令，在目标程序中，同一个子程序也只出现一次。而且，每次转子程序，只有一条调用指令。因而，总的目标代码较短。

(3) 在执行时间上，宏指令是把宏体直接插入到目标代码中，故执行时间较短。而对于子程序，除了执行子程序的基本功能外，还需增加额外的处理，如调用和返回，保护现场和恢复现场，所以时间较长。

(4) 在灵活性上，由于宏指令可以引入参数，并可与条件汇编结合起来使用，因此，可以根据特定的情况产生特定的目标代码，使用起来比子程序灵活。

(5) 从使用的角度上，宏指令和子程序各有利弊：用宏指令得到的目标代码长，占内存容量大，但执行时，不需要额外的处理，故执行时间短。而子程序恰好相反，它占用内存容量小，但执行时间较长，因此，在进行程序设计时，需依据具体情况，权衡利弊，决定取舍。通常，为了减少整个程序的长度，使用子程序更为有利。但是，在含有较多参数的情况下，使用宏指令较为方便。

(6) 在实现方式上，宏指令是由软件即汇编程序处理的，而子程序则是由硬件处理的，即通过 CPU 执行调用和返回指令实现的。

综上所述可知，一般对于较短的常用指令序列才使用宏指令，而对于较长的指令序列，则采用子程序为宜。

§8-3 汇 编 程 序

一、汇编程序的功能

汇编程序的主要功能是把用汇编语言写的源程序，翻译成机器能够识别并执行的目标程序，如图 8-3 所示。

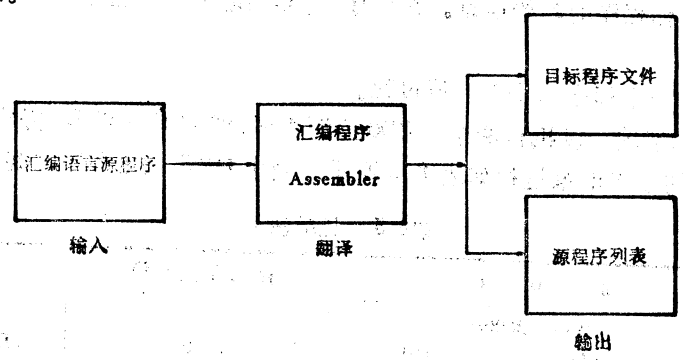


图 8-3 汇编程序功能示意图

汇编程序把源程序文件作为输入，产生两种输出文件——目标程序文件和源程序列表。目标程序文件就是经过汇编、翻译出来的机器语言程序，它将由机器执行。源程序列表将列出源程序、目标程序的机器语言代码以及符号表。符号表是汇编程序所提供的一种诊断手段，它包

括程序中所用的所有符号和名字，以及这些标号和名字指定的地址值。如果程序出错，则可比较容易地在这个符号表中检查出错误。

实际上，汇编程序能识别两种类型的源程序语句：一种是指令语句；另一种是汇编命令语句(或称伪指令语句)。前一种就是用助记符写出的微处理器的指令；后一种是为汇编程序提供控制信息的伪指令。在编写源程序时，这两种语句都要严格遵守汇编语言的规范，否则将出现错误。

二、汇编程序的汇编过程

为了深入地理解汇编程序的结构及汇编的过程。我们先从手工汇编过程开始分析。

手工汇编是把用汇编语言助记符编写的源程序直接表示为机器语言的目标程序。这是一项基本的编程技术。

在微型计算机中进行编程时，常用十六进制码表示存贮单元地址、操作码及操作数；用十进制码表示数值常数；用二进制码表示屏蔽字。有时还用 ASCII 码 (写在单引号中)来表示字符。八进制码现在比较少用。

对一个程序进行手工汇编时，首先要明确每一条指令的字节数。从第六章可知，8080A/8085A 微处理器指令可分为单字节、双字节和三字节三种，Z80 指令还有四字节的。单字节指令可以直接从指令表中查出。例如：传送指令 LD A,B 的十六进制码为 78H；增量指令 INC B 为 04H；算术运算指令 ADC A,(HL)为 8EH。双字节指令的第一字节为操作码，也可以从指令表中查出，第二字节的数据可化为十六进制数，把它和第一字节连接在一起，即为该指令的机器代码。例如将十进制数 45 送入累加器 A 的指令为 LD A,45，通过查指令表可知：LD A,n 对应的 16 进制目标代码为 3EH；而第二字节数据 45 的 16 进制码为 2DH。这两个字节代码 3E2DH，就是指令 LD A,45 的机器代码。同样，对于三字节指令，第一字节为操作码；第二字节为 16 位数据的低位字节；第三字节为 16 位数据的高位字节。例如传送指令 LD HL,2052H，这个指令码的第一、第二和第三字节分别为 21H、52H 和 20H。

严格地说，十六进制码并不是机器语言代码，称它为监控语言代码更合适些，因大多数微型计算机的监控程序接受十六进制数码后，将会自动地把它们转换为二进制的机器语言代码。

其次，要明确目标程序的起始地址。在程序中，起始地址用伪指令 ORG 表示，它指出该段程序的起始地址。

下面，举例说明一个程序的手工汇编过程。

【例 1】 若要求汇编一段执行两个数相减的源程序，即 7FH－52H，并将其差值放在寄存器 C 中。这个程序的手工汇编过程如表 8-3 所示。表中 ORG 指出程序的起始地址为 2000H。

表8-3 汇编例1

语句号	汇编码	存贮单元(H)	机器码(H)
	ORG 2000H		
1	LD A, 7FH	2000	3E 7F
2	LD C, 52H	2002	0E 52
3	SUB C	2004	91
4	LD C, A	2005	4F
5	HERE: JP HERE	2006	C3 06 20

从表 8-3 中得知：手工汇编过程可以归纳如下：

(1) 确定程序的起始地址。这是程序中第一条指令的第一字节的地址，即程序计数器 PC 的初始地址值。

(2) 下一条指令的地址取决于当前指令的字节长度。

(3) 重复第(2)步，找出程序中所有指令的地址值。

(4) 从第一条指令开始，通过查指令表，找出各条指令的机器代码。

同样地，汇编程序要把用汇编代码写的源程序汇编成目标程序，也要解决上述问题。为此，汇编程序必须具有以下数据库：

(1) 用来确定指令地址的汇编地址计数器(Location Counter)。

(2) 机器操作码表 MOT (Machine Operation Table)。

(3) 输入的源程序。

(4) 输出的目标程序。

同时，还需要以下简单的算法：

(1) 追踪汇编地址计数器。

(2) 逐条地读输入源程序的指令。

(3) 查机器操作码表 MOT，以得到相应的操作码。

(4) 计算地址(每条指令的起始地址加上指令字节数)。

(5) 逐条地写出目标程序中各条指令的机器代码。

【例 2】 若有表 8-4 左边的源程序，要求汇编为目标程序。

表 8-4 汇编例 2

(a)				(b)	
语句号	源	程	序	存贮单元	目标程序
		ORG	2000 H	(H)	(H)
1.	START:	LD	HL,DATA	2000	21 |0E| |20|
2.		XOR	A	2003	AF
3.		LD	B,COUNT	2004	06 |05|
4.	LOOP:	ADD	A,(HL)	2006	86
5.		INC	HL	2007	23
6.		DJNZ	LOOP-$	2008	10 |FC|
7.		LD	(RESULT),A	200A	32 |13| |20|
8.		HALT		200D	76
9.	DATA:	DEFB	1,2,3,4,5	200E	01 02 03 04 05
10.	COUNT:	EQU	$-DATA		0005
11.	RESULT:	DEFS	1	2013	
		END			

首先，设程序第一条指令的起始地址为 2000H 单元，即 START 的值为 2000H；接着，查 MOT 表确定每一条指令的字节数，由此可以计算出每一条指令的起始地址如表 8-4(b) 所示。这就解决了地址追踪问题。

第二步，查 MOT 表，找到每条指令的机器码，填入到相应的空格内。

本例中，经查 MOT 表，知第 1 条指令为立即数送 HL，是三字节指令，其操作码为 21H，第二、三字节应是由两个立即数组成的地址，但在指令中用符号地址 DATA 代替，故先空出两

格(用符号 ␣ 表示)。由第 1 条指令的字节数，可找到第 2 条指令的地址，填入相应的格内。照此方法将这段程序中的所有指令查找完毕。但是在程序中，第 3 条指令中的符号立即数 COUNT，第 7 条指令中的符号地址 RESULT 都是未知数，故均应留出空格。

第 6 条相对转移指令需要计算位移量。

按指令的规定 PC + e = LOOP (目标地址)

对第 6 条相对转移指令来说，其源地址为 2008H，由于它是两字节指令，因此把这条指令取出后，PC 已指向 2008 + 2 = 200AH，即

PC = 源地址(转移指令所在的地址) + 2

因此，实际位移量　　e − 2 = LOOP − 源地址 − 2 = 2006H − 2008H − 2 = −4H

而 −4 的补码为 0FCH，故第 6 条指令的实际位移量为 0FCH。

第三步，确定标号值。当汇编到第 9 条伪指令 DEFB 时，其标号 DATA 由上一条指令可以找到为 200EH。伪操作 DEFB 的作用是从 DATA 单元开始定义 5 个数：即 200EH 中放 01H，200FH 中放 02H，……2012H 中放 05H。

第 10 条伪指令 EQU 定义标号 COUNT，它本身不占用内存单元，但 EQU 右边的表达式为 $-DATA，$ 即当前程序计数器的值，此时它应当是 2013H，而 DATA 已知为 200EH，故 COUNT = 2013H − 200EH = 0005 H。

第 11 条伪指令 DEFS 是定义存贮区，即在其标号处留出一个存贮单元，标号 RESULT 按以上计算为 2013H。

至此，所有标号值都已确定，将它们分别填入相应的空格处。最后，END 语句结束整个汇编过程。

由上例可知，当在源程序中采用了符号地址以后，在第一次汇编(扫描)时，往往指令中的符号地址值还不能确定下来，只有在第一次扫描结束后，才能把程序中的所有标号的值确定下来。然后，第二次扫描时再把已确定的符号地址的值，代入相应的格内。即需要采用两次扫描的方法来完成整个源程序的汇编，故称两次扫描汇编程序。

第九章　汇编语言程序设计

在第六章中，我们曾援引了大量典型的程序段，作为微处理器指令系统应用的实例。本章将在此基础上系统地介绍程序设计的一般步骤、方法和技巧。同时，也介绍一些算术运算的常用程序和若干综合性的程序设计实例。

§9-1　程序设计（Programming）的基本方法和技巧

为了保证微型计算机能完成某一预定的任务（例如处理一批数据或控制某一生产过程），必须事先详细地安排好完整的解决问题的计划与步骤，并用机器指令或机器所能识别的语言描述出来，这就是“程序”。用程序设计语言编制程序的过程称为程序设计。程序设计的步骤通常有以下五点：

一、对任务加以正确的数学描述——建立数学模型

当我们接到任务时，首先要对任务进行详尽的调查研究和现场实验，搜集必要的设计数据。然后，在此基础上将计算任务或控制对象的物理过程归纳、抽象为数学表达式，这就是建立数学模型。在把微型机用于过程控制时，一个系统的数学模型及其控制算法的确定，常常是计算机过程控制系统的核心内容。

二、选择适当的计算方法

一般计算机只能进行算术和逻辑运算，而实际应用中的大量数学问题，往往不是单纯的算术运算所能描述的，这就要求借助于数值计算的方法，用近似的计算来代替计算机所不能直接进行的计算。从数学的角度来看，可能有几种不同的算法。在编制程序以前，先要对不同的算法进行分析比较，找出最适宜的算法。

三、程序结构的设计

程序结构的设计是把研究课题转化为程序的准备阶段。如果程序较短并且简单，那么此阶段可能仅仅是绘制一张流程图。如果程序较长或较复杂，则设计者就需考虑采用较为完善的方法。这些方法包括绘制流程图、模块化程序设计、结构式程序设计以及自顶向下设计等方法。无论采用哪一种方法，下列的几项原则都是适用的：

1．把大的作业分割成小的逻辑上相互独立的若干任务；

2．控制流程尽可能简单明了，以便于检查和修改；

3．尽可能用图形描述，这样比文字描述更为直观；

4．要以严谨而系统的方式进行设计，防止可能出现的混乱状况；

5．在程序结构设计阶段结束后，再着手编制程序。

下面简要介绍若干程序结构的设计方法：

1．编流程图

流程图是用预先约定的各种几何图形、指向线及必要的文字符号构成的，用以描述计算过程。流程图直观、清晰地体现了程序设计的思想和计算方法，是程序设计中常用的一种工具。标准流程图符号如图 9-1 所示。有了流程图，对于较大的程序，就便于分成若干模块，分头进行设计，最后再综合起来联调。有了流程图就可以使人们迅速抓住程序的基本线索，对全局有个完整的了解。

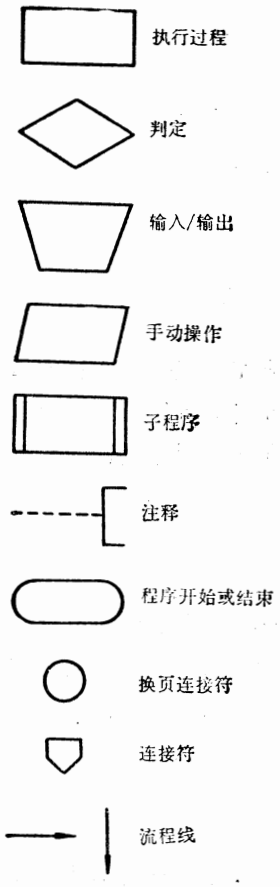


图 9-1　流程图符号

2．模块化程序设计

把整个程序分解成若干功能独立的模块，称为“模块化程序设计”。一般来说，模块化设计应遵循下列几项原则：

(1) 一个模块通常以包含 20 到 50 个语句为宜，因为过短的模块往往造成时间上的浪费，过长的模块则又缺乏通用性，并且可能难以装配；

(2) 力求使模块具有一定的通用性；

(3) 要精心设计延时程序、显示器处理程序、键盘处理程序等常用的模块程序；

(4) 力求使各模块尽可能地截然分开，并且在逻辑上相互独立。

3．结构式程序设计

采用“结构式程序设计”方法，可以使设计的程序每一部分都由若干环节组成，其中每个环节包含一个有限结构集，而每种结构只有一个入口和一个出口。这样，程序的操作顺序清晰，便于检查和修改。单入口、单出口模块程序的基本结构形式有：

(1) 顺序结构

顺序结构中的语句或结构都是顺序执行的。

(2) 分支结构

分支结构以给定的条件是否满足来改变程序的执行顺序，即可以根据需要，使用条件转移和无条件转移指令形成不同的分支程序。分支程序体现了计算机的判断功能，即“智能”作用。

(3) 循环结构

循环结构以给定的循环结束条件是否满足来决定程序的流程。若满足，则退出循环结构，否则返回去再次执行工作部分，直至循环结束条件满足为止。

结构式程序设计具有以下特点：

(1) 只允许三种基本结构，即顺序、分支、循环；

(2) 结构可以多重嵌套，以致任何程序又可包含任何结构；

(3) 每个结构仅有一个入口和一个出口。

采用结构式程序设计方法，可以使设计的程序更为系统化和条理化。而且，有助于查错、测试和编制文件。

4．自顶向下设计

自顶向下设计是从总体要求出发，采取逐步分解，逐步求精的方法，直至整个系统都得到实际的程序为止。

以上所述的是程序结构设计的基本方法，在实际的程序设计中往往把几种方法综合起来，互相补充，以求取得最佳的设计效果。

四、编制程序

经过程序结构设计以后，思路已经清楚，接下来的任务就是编制程序了。

对于较小的程序，可根据流程图选用合适的程序设计语言编写程序。如选用汇编语言编写程序，则可根据流程图选择合适的汇编指令，按流程图实现每一框图内的要求，从而编制出一个有序的指令流，这就是所谓"源程序"。当采用汇编语言或高级语言时，在源程序编制完成后，还必须经过翻译过程，以产生机器所能识别和执行的目标程序。

五、上机调试

程序编制好以后，难免会有缺点或问题，这就必须上机调试和检查。这一步可以在所选用的微型机上进行，也可在其它微型机上或微型机开发系统上模拟进行。经过调试、修正直至达到预定的要求，程序设计才告完成。

以上是程序设计的一般步骤，其过程如图 9-2 所示。

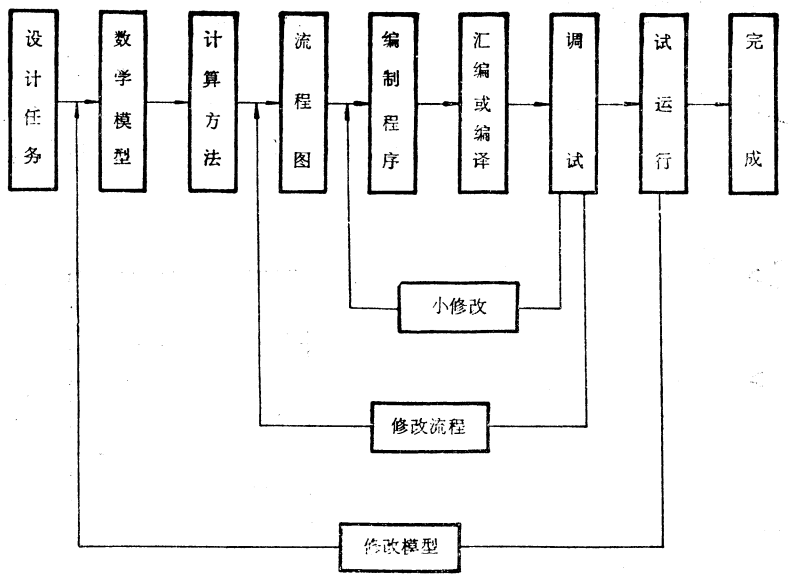


图 9-2　程序设计过程示意图

程序质量的优劣，一般是从执行程序的速度和程序的长度两个方面来衡量的，即称为时空指标(时间与空间)。对同一问题，若所编程序占据存贮空间越小，执行程序所花的时间越短，则说明这个程序质量较高。要编出质量好的程序，除了要熟悉上述每一步骤外，更重要的是要反复进行程序设计实践。

§9-2　汇编语言程序设计举例

如上所述，汇编语言程序设计的基本方法和技巧是采用模块化结构和结构式程序的设计

方法。其中最基本的程序结构有三种：顺序结构(简单程序)、分支结构和循环结构。对任何一种复杂问题的程序设计，基本上都可以由上述三种结构程序综合组成。在8位微处理器中，由于Z 80 μP有许多独特功能的指令(如数据块传送指令，搜索指令和位操作指令等)，可为程序员提供一系列灵活多样的操作。因此，对于同一任务，使用Z 80汇编语言就可以编制出更加简短而紧凑的程序，有利于提高运算速度和节省内存单元。

在实际的程序设计中，程序的走向始终是顺序执行的简单程序是很少见的。如果出现，也仅仅是在一些简单情况下。这些简单程序的实例，我们已在第六章中作过较详细的介绍，这里不再赘述了。下面着重介绍分支程序、循环程序、子程序以及算术运算常用程序的设计方法和实例。在例题中我们一般地写出Z 80的汇编语言源程序，有时也写出Intel 8080 A汇编语言源程序。

一、分支结构程序

在程序设计中，常常会遇到各种条件判断和比较操作，如“相等”或“不相等”；“大于”或“小于”；“满足条件”或“不满足条件”等等。依据这些判断和比较的结果，形成了相应的分支结构程序。编制分支程序的关键是要掌握好条件标志位的变化，使程序能够正确转移。在这里，条件转移、调用和返回指令是构成分支结构程序的基础。

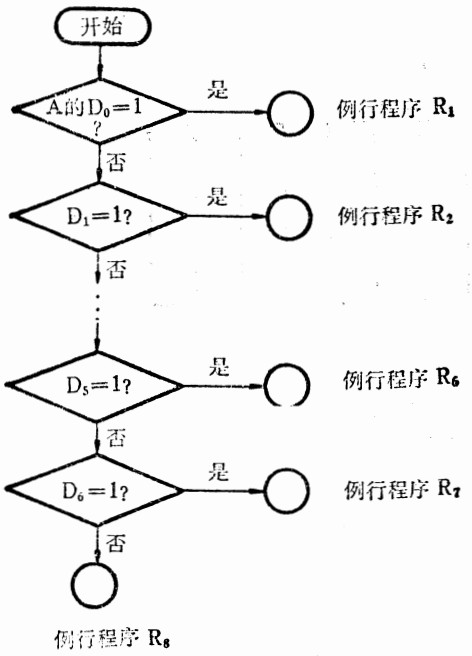


图 9-3 分支程序的流程图

【例1】 编制转移表程序（八中取一程序）。假设有8个例行程序，它们的首地址分别是R_1、R_2、……R_8。在累加器中事先存放例行程序的标志位，根据哪一位是1决定转去执行相应的例行程序。

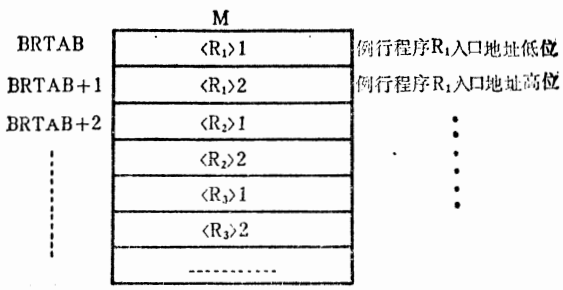


图 9-4 例行程序入口地址转移表

若累加器A内容为00000001，则程序转移到R_1

若累加器A内容为00000010，则程序转移到R_2

若累加器A内容为00000100，则程序转移到R_3

若累加器A内容为00001000，则程序转移到R_4

若累加器A内容为00010000，则程序转移到R_5

若累加器A内容为00100000，则程序转移到R_6

若累加器A内容为01000000，则程序转移到R_7

若累加器A内容为10000000，则程序转移到R_8

实现上述要求的程序流程图如图 9-3 所示。

转移表可存放在存贮器中，一个地址需要两个字节。其形式如图 9-4 所示。在图中 $\langle R_i \rangle_1$ 表示例行程序 R_i 的入口地址的低 8 位。$\langle R_i \rangle_2$ 表示高 8 位。

Z 80 的转移表源程序如下：

```
标号      指令(助记符)              注释
          ORG    2000H
START:    LD     HL,   BRTAB   ; 转移表首地址送入 HL 寄存器对
GTBIT:    RRA                  ; A的最低位移入进位位
          JP     C,    GETAD   ; CY=1, 转 GETAD
          INC    HL            ; CY=0, 转移表地址加 2
          INC    HL
          JP     GTBIT         ; 返回, 检查下一位
GETAD:    LD     E,    (HL)    ; 转移地址低 8 位送入寄存器 E
          INC    HL
          LD     D,    (HL)    ; 转移地址高 8 位送入寄存器 D
          EX     DE,   HL      ; 转移地址换入 HL
          JP     (HL)          ; 按转移地址转移到该例行程序
BRTAB:    DW     R1, R2, R3, R4, R5, R6, R7, R8
          END    START
```

对于本例，用 Intel 8080A/8085 A 汇编语言写出的源程序如下：

```
          ORG    2000H
START:    LXI    H,    BRTAB
GTBIT:    RAR
          JC     GETAD
          INX    H
          INX    H
          JMP    GTBIT
GETAD:    MOV    E,    M
          INX    H
          MOV    D,    M
          XCHG
          PCHL
BRTAB:    DW     R1, R2, R3, R4, R5, R6, R7, R8
          END    START
```

二、循环结构程序

在计算机的应用中，往往需要多次反复地执行某种相同的操作。在这种情况下，就需要用到循环结构程序。它能节省内存，简化程序。一个典型的循环结构程序由以下四个部分组成：

1．初始化部分：用来设置循环初态。循环初态分为循环工作部分初态和结束条件的初态。置循环工作部分的初态即置变量、数据指针(存放待处理数据的地址)的初值。通常作为累加和的单元(或寄存器)的初值总是置“0”。循环结束条件的初态往往是预置计数器的循环次数。

2．循环处理部分：这是指要求重复执行的数据处理程序段部分，是循环结构的基本部分。不同的程序需要解决的问题不同，因而处理部分的具体内容会有很大的差别。

3．循环控制部分：它的功能是实现对循环的判断和控制。它包括修改计数器的计数值，为下一次重复作好准备，以及每循环一次检查循环结束条件决定循环是否应该结束。例如，计数循环，当循环了一定次数后就结束循环。通常，用一个内部寄存器（或寄存器对）作为计数器。计数开始前，先赋以循环次数初值，每循环一次，计数器就减 1，当计数器减为 0 时就结束循环。当然也可以赋以初值为“0”，每循环一次加 1，再与循环次数比较，若两者相等，便结束循环。实际上，由于程序的循环结构一般是建立在条件转移指令的基础上的，即要用条件转移指令来控制循环，因此判断条件转移的标志（零标志 Z、进位标志 CY、符号标志 S、奇偶/溢出标志 P/V）都可作为判断循环结束的依据。循环处理部分和循环控制部分组成循环体，是循环程序的主体部分。

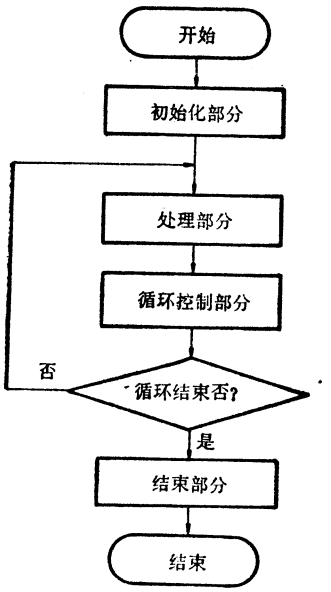


图 9-5　一般循环程序框图

4．循环结束部分：存贮或输出计算结果。

一般循环程序的框图如图 9-5 所示

【例 1】 多项单字节数据求和。

在许多情况下，我们常常需要求出一组数据的和。例如，从微型计算机输入端口输入传感器的一组数据；在控制应用中，某一段时间内的输入信息等等。对于这些数据，我们都可用数组来描述它们。假设我们把数组长度的值放入存贮单元 2041H，而将数组本身从存贮单元 2042 H 开始存放，假设其和的值小于 256，并存放在存贮单元 2040 H 中（8 位字长），即设：

(2041 H) = 03

(2042 H) = 11 H

(2043 H) = 22 H

(2044 H) = 33 H

结果：(2040 H) = 66 H

解决这道题的算法是：

1．置初始值

COUNT←(2041 H)（COUNT 为计数器）

SUM←0（开始时，累加和单元 SUM 清 0）

POINTER←2042 H（数据的地址指针 POINTER，其初始值为 2042 H）

2．求和

SUM←SUM + (POINTER)

3．每计算一次，计数器值 COUNT 减 1，地址指针 POINTER 加 1。即

COUNT←COUNT − 1

POINTER←POINTER + 1

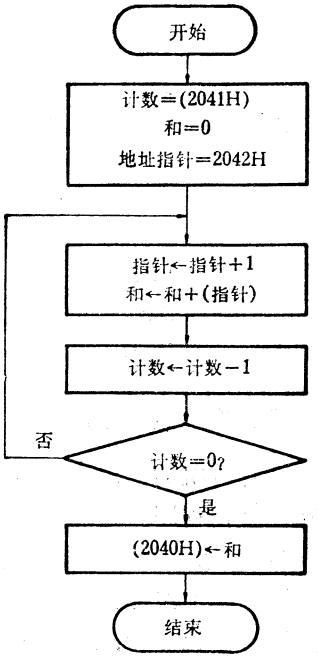


图 9-6　数组求和程序的框图

4．如果数组中的全部数据已加完，此时，COUNT = 0 就顺序执行 5,，否则转向 2，继续求和。

5．送结果至地址为 2040 H 的单元中，(2040 H)←SUM。

按照上述算法，画出程序框图如图 9-6 所示。

实现上述功能的 Z 80 源程序如下：

```
标号      指令(助记符)              注释
          ORG   2000H
          SUB   A                   ；清除存放累加和的累加器 A
          LD    HL,   2041H         ；设置计数值 = 数组长度
          LD    B,    (HL)
SUMD:     INC   HL                  ；HL 作为地址指针 POINTER
          ADD   A,    (HL)          ；和←和 + 数据
          DJNZ  SUMD-$              ；若 B←B－1 后，B≠0 则转 SUMD
          LD    (2040H),A           ；若 B = 0，则把结果存入 2040 H 中
          HALT                      ；暂停
```

与上述程序对应的 8080 A 源程序如下：

```
标号      指令(助记符)
          ORG   2000H
          SUB   A
          LXI   H,    2041H
          MOV   B,    M
SUMD:     INX   H
          ADD   M
          DCR   B
          JNZ   SUMD
          STA   2040H
          HLT
```

从上述两个程序看出：DJNZ 指令相当于 Intel 8080 A 的 DCR B 和 JNZ 两条指令，这是 Z 80 所特有的。这样，Z 80 程序就节省了两个字节的内存单元。

【例 2】 求数组中数据的最大值。

在计算机应用中，经常会遇到要找出一个数组中的最大值问题。例如，对数据分类；分析一组测试结果；按照优先次序输送数据等。现假定：数组长度放在存贮单元 2041 H 中；数组本身从存贮单元 2042 H 开始存放；把最大值放在存贮单元 2040 H 中；所有输入数据是 8 位长的不带符号的数；为简单起见，假设数组只有三个数据，各存贮单元的数据如下：

(2041 H) = 03

(2042 H) = 37 H

(2043 H) = F 2 H

(2044 H) = C 6 H

显然，上列无符号数的最大值是 F 2 H，故(2040 H) = F 2 H。

对本例，程序中先把第一个数据作为最大值，然后把新输入的数与它比较。如果新的数较大，则用它替换旧的最大值；否则，继续用下一个新的数进行比较。同时，用一个计数器计算数组中已比较过的数据的个数。这样逐个输入新的数，并与旧的最大值比较，一直把所有数据都比较完毕。

该程序的框图如图 9-7 所示。

根据这个流程图，就可以写出找最大值的源程序。

找最大值的 8080 A 程序如下：

```
标号      指令(助记符)              注释
          ORG   2000H
          LXI   H,    2041H   ; 数据数组长度
          MOV   B,    M
          DCR   B             ; B 的内容减 1
          INX   H             ; HL 的内容加 1,HL = 2042 H
          MOV   A,    M       ; 设第一个数为最大值
MAXM:     INX   H             ; HL 指示地址指针 2043 H
          CMP   M             ; 最大值是否比新进的数大?
          JNC   NOCHG         ; 是,仍为最大值
          MOV   A,    M       ; 否,将新进的数替换最大值
NOCHG:    DCR   B             ; B←B - 1
          JNZ   MAXM          ; 若 Z = 0, 则转 MAXM
          STA   2040H         ; 存最大值
          HLT                 ; 暂停
```

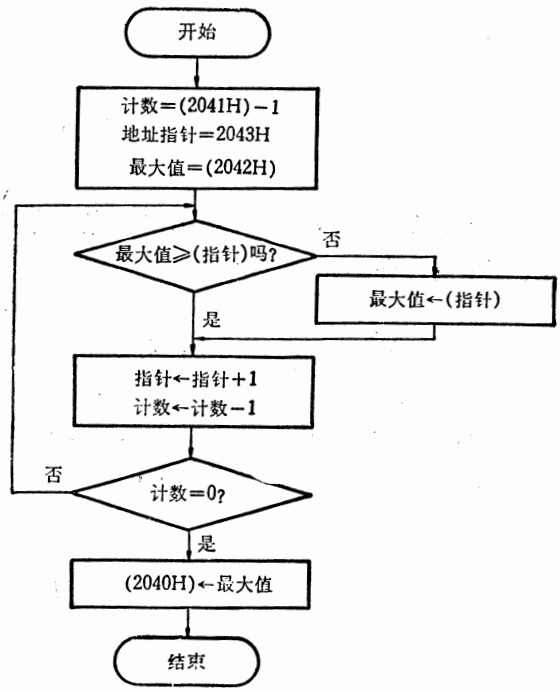


图 9-7(a)　求最大值的程序框图

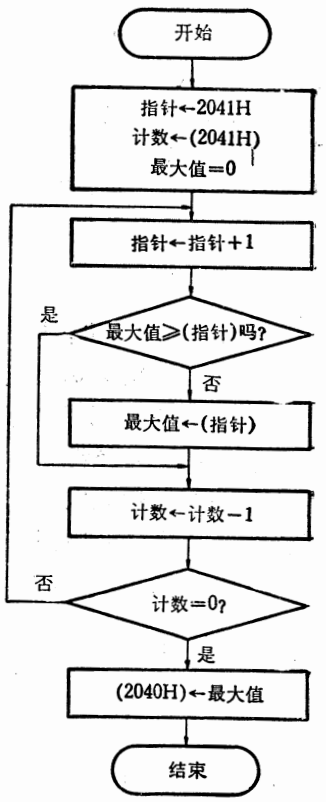


图 9-7(b)　求最大值的程序框图

程序中采用了 CMP M 指令，当 A≥(HL)时，进位标志 CY = 0；当 A<(HL)时，CY = 1。下一条指令 JP NC, NOCHG 就是根据 CY 的状态而决定转向的。

如果我们改变一下算法，并用 Z 80 指令编制程序，画出其程序框图如图 9-7(b)所示。

其源程序如下：

```
标号      指令(助记符)                    注释
          ORG    2000H
          LD     HL,     2041H
          LD     B,      (HL)        ；数据数组长度送入 B
          SUB    A                   ；最大值 = 最小可能值(0)
NEXTE:    INC    HL
          CP     (HL)                ；下一个数据比最大值大吗?
          JR     NC,     DECNT-$     ；否，转 DECNT
          LD     A,      (HL)        ；是，以该数据取代最大值
DECNT:    DJNZ   NEXTE-$
          LD     (2040H),A           ；存最大值
          HALT                       ；暂停
```

本程序利用 8 位不带符号二进制数中最小值为 0 这一事实。假设在进入循环之前将保存最大值的寄存器(本例中为累加器)设置为最小的可能值，则程序一定会使累加器设置为大于 0 的值，除非数组中的全部数据都为 0。

CP (HL) 指令的用法与 8080 A 的 CMP M 相同，即用作在不改变寄存器内容的条件下设置条件标志。

如果是两个带符号数，上述程序就不适用于找最大值了。此时可能出现结果溢出的错误。因而，要将符号标志 S 和溢出标志 P/V 结合起来，才能判断出最大值。Z80 CPU 备有奇偶/溢出标志，它可用来测试补码溢出(即可用 Z80 的 JP PE 或 JP PO 指令来测试)。8080A/8085A CPU 没有设置溢出标志，可用同号异号方法来判断两个带符号数的大小 (请参阅 §6-6 比较指令部分)。

三、算术运算程序

前面曾举例简要地说明了单字节、双字节和多字节的算术运算程序。实际上，在微型机应用方面，大多数算术运算要求多字节运算或者进行比加减法更为复杂的运算。本节将通过实例着重讨论多字节加减法(包括二进制、十进制加减法)、带符号数加法、二进制乘法、除法等常用的算术运算程序。对于这类算术运算的编程，关键是要处理好进位标志位并熟练运用有关的算术运算指令。

1. 多字节加减法运算

(1) 二进制数的加减法运算

如果要进行三字节的无符号数的相加，并且使数据字节的长度(03)放在存贮单元 2060 H 中，两个相加的数分别从存贮单元 2061 H 和 2081 H 开始存放(先存放最低位)，现设：

(CNT) = (2060H) = 03

(FIRST) = (2061H) = 29H

(FIRST + 1) = (2062H) = 0A4H

(FIRST + 2) = (2063H) = 50H

(SECND) = (2081H) = 0FBH

(SECND+1) = (2082H) = 37H

(SECND+2) = (2083H) = 28H

于是，两个 24 位数 50A429H 与 2837FBH 相加的结果如下：

(2061H) = 29H + 0FBH = 24H　产生进位 CY = 1

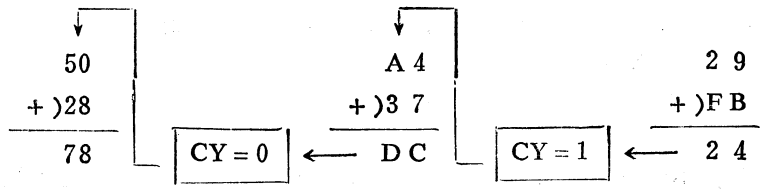
(2062H) = 0A4H + 37H + 进位(1) = 0DCH

(2063H) = 50H + 28H + 进位(0) = 78H

即

$$\begin{array}{r} 50A429H \\ +)\ 2837FBH \\ \hline 78DC24H \end{array}$$

它可以分解为以下三步进行：



此程序也应包括预置、处理和循环控制三部分。这里的关键是计算机在每次求和时，应考虑上一次求和时的进位。

该程序的流程图如图 9-8 所示。

根据流程图，便可得到如下 8080 A 的多字节加法程序：

```
标号      指令(助记符)              注释
          ORG   2000H
          SUB   A                   ; 清进位位
          LXI   H,    2060H         ; 设置数据字节长度计数值
          MOV   B,    M
          LXI   D,    2080H         ; 指出第二个数
MPADD:    INX   D                   ; 修改指针
          INX   H
          LDAX  D                   ; 取出第二个数的一个字节
          ADC   M                   ; 与第一个数的一个字节及
                                    ; 进位位相加
          MOV   M,    A             ; 存和
          DCR   B
          JNZ   MPADD
          HLT
```

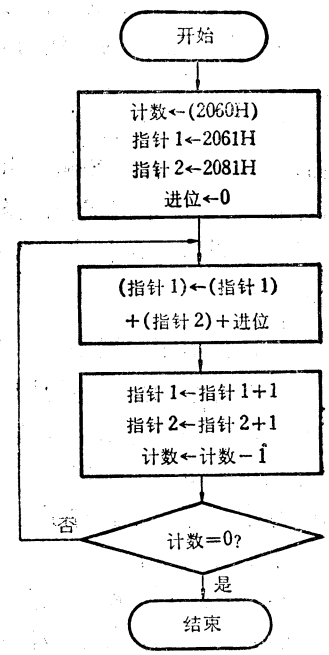


图 9-8　多字节加法程序的框图

程序中用 SUB A 指令对进位位清零（在 8080 A/8085A 指令系统中，没有专门的进位清零指令），从而使低位字节相加时，可用带进位的指令 ADC。当然，指令 XRA A, ANA A, ORA A 也都能起到清除进位标志位的作用。

在上述程序中，只要把指令 ADC 换成 SBB，就可编写出一个多字节减法子程序。

下面我们再写出相应的 Z 80 的多字节加法程序：

对于 Z 80 程序，最好用寄存器 B 作计数器，这样便可用一条 DJNZ 指令来代替 DEC B 和

JR NZ 两条指令，从而使程序缩短。

```
    标号          指令(助记符)                    注释
                  ORG  2000H
                  LD   A,    (CNT)        ; 数的长度送入 B
                  LD   B,    A
                  LD   HL,   FIRST        ; 数 1 首地址存入 HL
                  LD   DE,   SECND        ; 数 2 首地址存入 DE
                  AND  A                  ; 清进位位 CY
LOOP:             LD   A,    (DE)         ; 取数 2 的一个字节
                  ADC  A,    (HL)         ; 加上数 1 的一个字节及进位位
                  LD   (HL), A            ; 存和数
                  INC  DE                 ; 修改指针
                  INC  HL
                  DJNZ LOOP-$             ; B←B-1,若 B≠0, 转至 LOOP; 若 B=0,加法完成,
                                          ; 顺序执行下一条指令
                  HALT                    ; 暂停
2040H CNT:        DEFB 03H
2061H FIRST:      DEFB 29H,  0A4H,  50H
2081H SECND:      DEFB 0FBH, 37H,   28H
                  END                     ; 程序结束
```

对于多字节减法程序，只要把上述程序中的 ADC 换成 SBC，即带进位加换成带借位减就可以把上述程序变成多字节减法程序。

(2) 十进制加法运算

在商业售货终端机、银行终端机、计算器以及导航系统中，十进制数运算应用十分普遍。大多数微处理器都设有十进制数调整指令(如 DAA) 等。这些指令可以将二进制数运算的结果调整成为正确的十进制数形式。这样，微型计算机便能够直接完成十进制数的算术运算。

下面以求两个十进制数 376529 与 224388 的和为例，现设：数据字节长度放在存贮单元 2040 H 中，两个相加的数分别从单元 2041 H 和 2051 H 开始存放；所有数都是二-十进制(BCD 码)的；结果放在第一数的位置中，那末就有：

(CNT) = (2040 H) = 03
(FIRST) = (2041 H) = 29
(FIRST + 1) = (2042 H) = 65
(FIRST + 2) = (2043 H) = 37
(SECND) = (2051 H) = 88
(SECND + 1) = (2052 H) = 43
(SECND + 2) = (2053 H) = 22

结果：

(2041 H) = 17
(2042 H) = 09
(2043 H) = 60

其计算过程如下：

$$
\begin{array}{r}
376529 \\
+)\ 224388 \\
\hline
600917
\end{array}
$$

可分为以下六步进行：

①清进位位 CY，每个数的两个低位相加：

$$
\begin{array}{r}
29 = 00101001 \\
88 = 10001000 \\
+)\ CY = \qquad\quad 0 \\
\hline
10110001\ \text{（二进制码）}
\end{array}
$$

运算结果：CY = 0、H(AC) = 1。

②执行一次 DAA 操作，因 H(AC) = 1，高 4 位又大于 9，故高、低 4 位应分别加 6 修正：

$$
\begin{array}{r}
A = 10110001 \\
+)\qquad\quad 0110 \\
\hline
10110111
\end{array}
$$

$$
\begin{array}{r}
A = 10110111 \\
+)\qquad\quad 0110 \\
\hline
CY = [1]00010111 = 17\text{（BCD 码）}
\end{array}
$$

运算结果：累加器内容为 17，CY = 1，将 A 存入相应存贮单元。

③加其次的两位数，并考虑低位数相加时产生的进位。

$$
\begin{array}{r}
65 = 01100101 \\
43 = 01000011 \\
+)\ CY = \qquad\quad 1 \\
\hline
CY = 0 \quad 10101001
\end{array}
$$

运算结果：CY = 0，H(AC) = 0。

④执行一次 DAA 操作。现 A = A 9 H，高 4 位大于 9，故高 4 位需加 6 修正：

$$
\begin{array}{r}
A = 10101001 \\
+)\qquad\quad 0110 \\
\hline
CY = 1 \quad 00001001 = 09\text{（BCD 码）}
\end{array}
$$

⑤每个数的两个高位相加：

$$
\begin{array}{r}
37 = 00110111 \\
22 = 00100010 \\
+)\ CY = \qquad\quad 1 \\
\hline
01011010
\end{array}
$$

运算结果：CY = 0、H(AC) = 0。

⑥执行一次 DAA 操作。现 A = 5 AH，低 4 位大于 9，故低 4 位需加 6 修正：

$$
\begin{array}{rr}
 & A = 01011010 \\
+) & \quad 0110 \\
\hline
 & 01100000 = 60(\text{BCD 码})
\end{array}
$$

最后得到结果为 600917(BCD 码)。

计算流程图与上例的多字节二进制运算相似。

Z 80 的十进制加法源程序如下：

```
标号      指令(助记符)              注释
          ORG   2000H
START:    LD    A,    (CNT)
          LD    B,    A
          LD    HL,   FIRST
          LD    DE,   SECND
          AND   A
LOOP:     LD    A,    (DE)
          ADC   A,    (HL)
          DAA                     ;十进制调整
          LD    (HL), A
          INC   DE
          INC   HL
          DJNZ  LOOP-$
          HALT
CNT:      DEFB  03
FIRST:    DEFB  29,65,37
SECND:    DEFB  88,43,22
          END
```

由此可知，十进制数的加法程序，可在多字节二进制加法程序中的 ADC A, (HL) (对 8080A 为 ADC M)指令之后，紧接着加入一条 DAA 指令即可获得。

(3) 十进制减法运算

当进行多字节十进制减法时，由于 Z 80 有一个专门的减法运算标志(N)，使得 DAA 指令亦可用于十进制减法运算。即在十进制减法的循环体中减法指令 SBC 后面紧接着加入一条 DAA 指令即可获得十进制减法程序。而 Intel 8080 A/8085 A 微处理器没有 N 标志，它们的 DAA 指令仅能在加法之后才能正确地进行调整。因此，对于 Intel 8080 A/8085 A 微处理器，其十进制减法程序必须通过改变算法来编程，情况要复杂一些。

2．单字节带符号数的加法运算

在计算机中凡是带符号数都用补码形式进行运算。一个单字节带符号数所能表示的数的范围是 +127～−128。由于 8 位数的范围限制，在运算程序中应考虑检验“溢出”标志。当出现溢出(即计算结果超出上述范围)时，就要及时停机或转溢出处理。

关于如何判断溢出的方法，我们在 §2-6 中已经详细介绍过，这里不再重复。简单地说，两个带符号数相加时，若符号相异，则结果不会产生溢出；若符号相同，则结果可能产生溢出。这时，可用前面所述的判断溢出方法进行判断。

根据以上分析，图 9-9 示出了带符号数加法程序的框图。

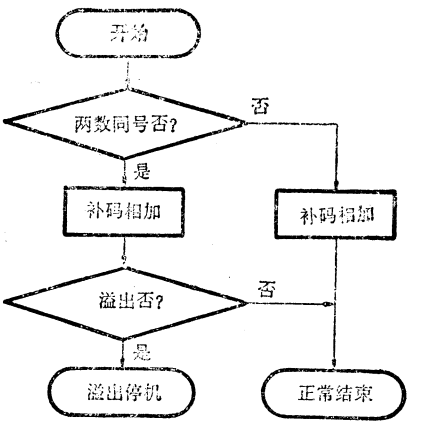


图 9-9 带符号数加法程序的框图

【例】 设两个单字节带符号数已分别放在寄存器 D、E 中，运算结果存放在寄存器 A 中。

对于 Z 80 微型机而言，由于 CPU 中设有溢出标志位，因此，编制程序就比较简单。

其源程序如下：

```
        ORG   2000H
        LD    A,    D
        ADD   A,    E      ; 两数相加
        JP    PE,   ERR    ; 若 P/V = 1,则转 ERR
        HALT               ; 正常停机
ERR:    HALT               ; 溢出停机
```

这里，ERR 表示溢出处理程序的入口。

对于 Intel 8080 A 微处理器而言，由于其 CPU 没有设置溢出标志位，所以必须用其它指令判断溢出。这样，程序就复杂些了。

其源程序如下：

```
标号    指令(助记符)          注释
        ORG   2000H
        MOV   A,    D      ; D 中的数送 A
        XRA   E            ; A 和 E 的两个数相“异或”；如两数符号位相同，则 S = 0，反之
                           ; S = 1 且清除 CY
        MOV   A,    D      ; D 中的数重新送 A
        JP    SAME         ; 符号相同，转 SAME，否则执行下一条
        ADD   E            ; 符号相异的数相加，结果存入 A 中
RIGHT:  HLT                ; 正常结果，和在 A 中
SAME:   ADD   E
        MOV   B,    A      ; 保存结果
        RAR                ; A7←CY
        XRA   B            ; 结果的进位位与符号位相“异或”
        MOV   A,    B      ; 不影响标志
        JP    RIGHT        ; 若两者相同，则表示没有溢出转正常停机
        HLT                ; 若两者相异，则溢出停机
```

3．8 位二进制数乘法运算

两个无符号 8 位二进制数的乘法可用连加法或移位法来完成。连加法就是将被乘数连续相加，相加的次数由乘数决定。连加法的程序简单，但运算时间太长，因而，实际上常采用移位法。移位法原理可用下面的例子来说明。

【例】 设乘数 = 5，被乘数 = 7，即：

111	被乘数
×101	乘数
111	乘数 2^0 位数为 1，第一行部分积是 7
000	乘数 2^1 位数为 0，第二行部分积是 0
+) 111	乘数 2^2 位数为 1，第三行部分积是 7
100011	部分积之和，即最后的乘积

部分积之和是 32 + 2 + 1 = 35，即是 7 和 5 的乘积。在上述二进制乘法中，乘数的各位从低位向高位，每次取一位进行计算。我们也可以由高位向低位反向进行计算，即

111	
×101	
111	被乘数 $\times 2^2 \times 1$
000	被乘数 $\times 2^1 \times 0$
+) 111	被乘数 $\times 2^0 \times 1$
100011	乘积

两种方法所得的结果是相同的。

由上例可知，两个无符号的 n 位数相乘，乘积为 2 n 位，它等于各部分积之和。由乘数从高位向低位逐位去乘被乘数，当乘数的相应位为 1 时，则得到的部分积等于被乘数；当乘数的相应位为 0 时，则得到的部分积为 0。第一个部分积由于最左边的乘数位为 1 而等于“被乘数 $\times 2^{2}$”。它恰好等于被乘数左移 2 位。第二个部分积由于乘数位为 0 而等于 0。第三个部分积由于乘数位为 1 而等于“被乘数 $\times 2^{0}$”，即被乘数位置上的数。

但是，这种运算方法对计算机并不合适。一则必须用 n 个寄存器存放 n 个部分积，二则 n 个部分积相加得到的是 2 n 位的数，这就需要 2 n 位的加法器。所以，必须对这种运算方法加以改进。

从上述运算可知：如果把每次运算得到的部分积逐次累加起来也可得到最后乘积。

采用这种方法时，大致过程如下：

			b_2	b_1	b_0		位号
			1	1	1		被乘数
			× 1	0	1		乘数
			0	0	0		中间结果(初始值清 0)
		0	0	0			中间结果左移一位仍为 0
	+)		1	1	1		$b_2 = 1$，位部分积等于被乘数
		0	1	1	1		中间结果(位部分积)
	0	1	1	1			左移 1 位
	+)		0	0	0		$b_1 = 0$，位部分积等于 0
	0	1	1	1	0		中间结果(位部分积)
0	1	1	1	0			左移 1 位
+)			1	1	1		$b_0 = 1$，位部分积等于被乘数
1	0	0	0	1	1		乘积

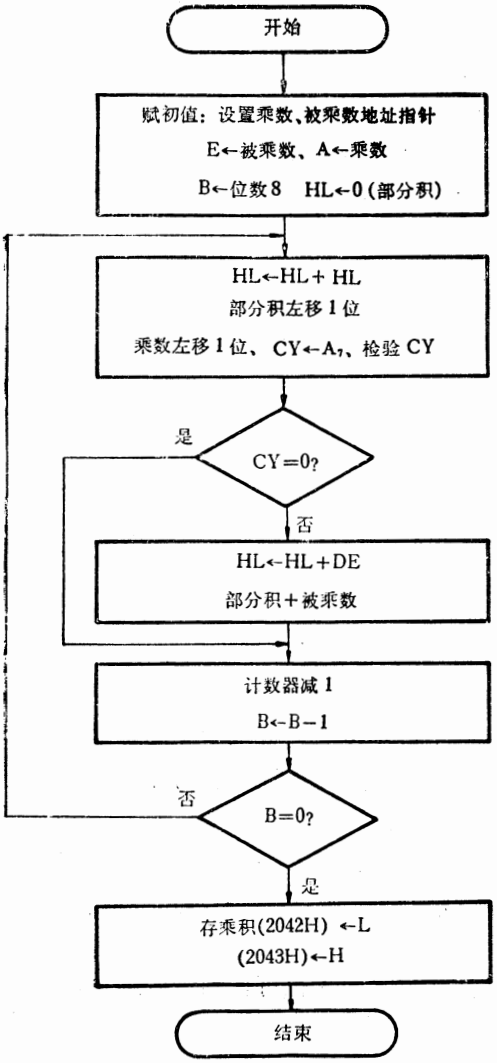


图 9-10　8 位二进制乘法程序的框图

由此可见，采用这种左移移位法计算，每次只有两个部分积相加，所需的寄存器的数目可以节省；另外还发现，在两个部分积相加时，只要虚线右边 n 位相加，而左边的其余部分不再参加运算。这样，只要用 n 位加法器便可实现乘法运算了。

综上所述，可以把两个 8 位二进制数相乘的算法归纳如下：

(1) 初始化：将存放乘积的寄存器对清 0，假定将 HL 寄存器对清零，作为 16 位部分积的累加器(简称 16 位累加器)；同时设置位计数，用来表示乘数的位数。

(2) 从最高位开始，检验乘数的每一位是 0 还是 1，若该位是 1，被乘数就加到部分积上；若该位为 0，就跳过去不加。这可以采用左移乘数一位，使其最高位进入进位位，然后判 CY 值的方法来实现。

(3) 无论哪种情况，部分积都左移 1 位。左移次数等于乘数的位数。

(4) 每次判乘数中的一位，乘数位数计数器就减 1，若计数器 ≠0，则重复步骤 2，否则乘法结束。

【例】 设被乘数为 53 H，存放在 2040 H 内存单元中，乘数为 7 FH，存放在 2041 H 内存单元中，结果为 292 DH，存放在 2042 H 和 2043 H 内存单元中。

8 位二进制乘法程序的框图如图 9-10 所示。

Z 80 的 8 位二进制乘法源程序如下：

```
标号      指令(助记符)
          ORG   2000H              注释
          LD    HL,   2040H   ; 指向被乘数单元
          LD    E,    (HL)    ; 被乘数送寄存器 E
          LD    D,    00      ; 被乘数扩展到 16 位
          INC   HL            ; 指向乘数单元
          LD    A,    (HL)    ; 乘数送累加器 A
          LD    HL,   0000    ; 清部分积寄存器对
          LD    B,    08      ; 设置计数器 = 乘数的位数
MULT:     ADD   HL,   HL      ; 部分积左移一位
          RL    A             ; 乘数带进位左移一位
```

```
        JR    NC,     CHCNT-$  ; 从乘数来的进位=0? 若是,则转 CHCNT
        ADD   HL,     DE       ; 若否,将被乘数加到部分积上
CHCNT:  DJNZ  MULT-$           ; 若 B≠0,则转 MULT 重复
        LD    (2042H), HL      ; 乘积存入指定的内存单元
        HALT
```

下面再写出 8080 A 的 8 位二进制数乘法运算源程序。

```
标号    指令(助记符)
        ORG   2000H
        LXI   H,    2040H
        MOV   E,    M
        MVI   D,    00
        INX   H
        MOV   A,    M
        LXI   H,    0000
        MVI   B,    08
MULT:   DAD   H
        RAL
        JNC   DCRE
        DAD   D
DCRE:   DCR   B
        JNZ   MULT
        SHLD  2042H
        HLT
```

带符号数的相乘，除了必须考虑符号外，其它的和上述原理相同。最常用的方法是采用原码相乘，乘积求补法。即先检查乘数和被乘数的符号，是负数的都变为正数，然后求两个正数的乘积，最后根据乘积的符号 = 被乘数符号⊕乘数符号公式，求出乘积的符号，再对乘积求补码。因此，这种算法是在不带符号数的乘法之前加上变负数为正数的操作，而在它的后面加上对乘积求补码的操作。请读者自编 8 位带符号二进制数的乘法程序。

4. 二进制除法运算

两个无符号二进制数相除，可以用减法和移位法来完成。首先采用试减法判断被除数或余数是否大于或等于除数，如果大(即够减，无借位)，则商为 1，接着作减法；反之，则商为 0，不作减法，然后再进行下一位商的运算。其具体步骤如下：

(1) 初始化：商为 0，计数为 8 (设除数为 8 位)；

(2) 被除数与商左移一位；

(3) 试减，如果没有借位，则商为 1(即被除数的高 8 位大于或等于除数)，反之商为 0；

(4) 位计数器(除数的位数)减 1，判计数器是否为 0? 若不是 0，重复步骤 2，否则结束。

【例】 存放在内存单元 2040 H 和 2041 H 中的 16 位无符号数 (2041 H 存放高 8 位)除以内存单元 2042 H 中的 8 位无符号数。

假设：(1) 被除数的最高位为零，以简化运算，否则在试减前的移位将使被除数的最高位移入进位位中；(2) 除数大于被除数的高 8 位，即商为一个 8 位的数。将商存入内存单元 2043 H 中，而余数存入内存单元 2044 H 中。

现设被除数为 407CH，存放在：(2040 H) = 7CH, (2041 H) = 40 H

除数为 5 AH，存放在：(2042 H) = 5AH

结果：商为 0B 7 H　存放在：(2043 H) = 0B 7 H

余数为 26 H，存放在：(2044 H) = 26 H

即 16508/90 = 183 余数 38。

寄存器分配：HL 存放被除数(或余数和商)

C 存放除数

B 作为位计数器

二进制除法程序的框图如图 9-11 所示。

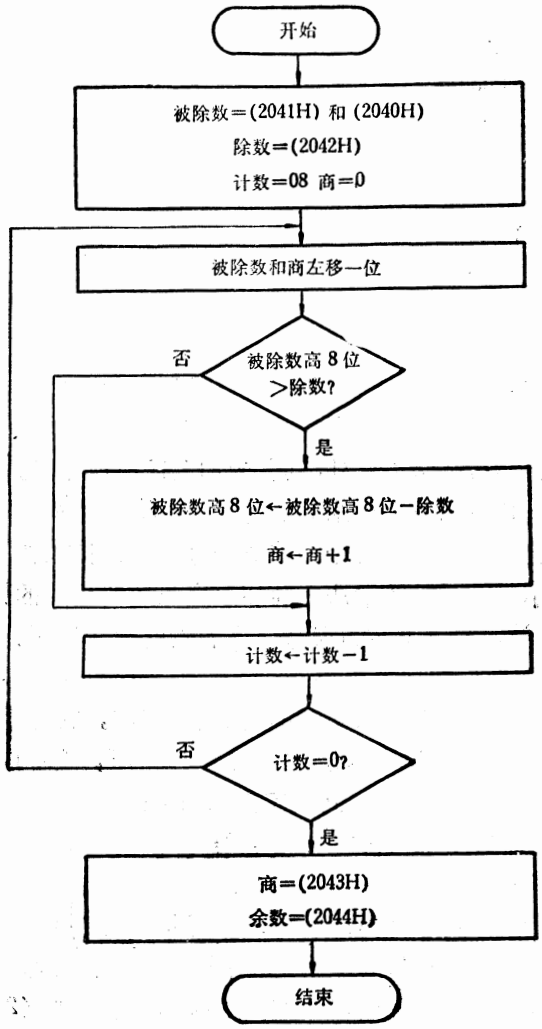


图 9-11　二进制除法程序框图

相应的 Z80 源程序如下：

```
标号      指令(助记符)                    注释
          ORG   2000H
          LD    HL,    (2040H)   ; 取被除数
          LD    A,     (2042H)   ; 取除数
          LD    C,     A
          LD    B,     08H       ; 置计数器初始值
DIV:      ADD   HL,    HL        ; 被除数、商左移一位
          LD    A,     H         ; 取被除数高位
          SUB   C                ; 被除数高8位减除数
          JR    C,     CNT-$     ; 若CY=1(不够减),商为零,跳转CNT
          LD    H,     A         ; 若CY=0(够减)送回差值
          INC   L                ; 商=商+1
CNT:      DJNZ  DIV-$             ; 若计数器减1不为0,则循环
          LD    (2043H),HL       ; 将商和余数存入存贮器
          HALT                   ; 暂停
```

上述程序中，由于巧妙地安排寄存器对 HL 而使程序缩短，执行时间加快。一开始寄存器 HL 存放被除数和商。当被除数逻辑左移时，商就取代寄存器 L 中的被除数。这样，左移 8 次后运算结束时，HL 中的 H 的内容即为余数，而 L 的内容即为商。程序中指令 INC L 是将商的低位置“1”，而 ADD HL, HL已预先将该位清零。

这段程序的安排巧妙，也很简单。它概括了除法程序的基本思想，但是，它有很大的局限性。(1) 因商用一个 8 位寄存器存放，故当被除数的高 8 位大于或等于除数时，将产生溢出错误；(2) 若除数为 0，将产生溢出错误；(3) 因除法流程是在被除数先移位后，再去减除数，因此对于不带符号的数，若被除数最高位为 1，将因移出有用信息而出错。

对字长较长的除法运算，可以使用指令 SBC HL, DE 来实现。

下面再写出相应的 8080 A 二进制数除法源程序：

```
标号      指令(助记符)
          ORG   2000H
          LHLD  2040H
          LDA   2042H
          MOV   C,     A
          MVI   B,     08
DVI:      DAD   H
          MOV   A,     H
          SUB   C
          JC    CONT
          MOV   H,     A
          INR   L
CONT:     DCR   B
          JNZ   DVI
          SHLD  2043H
          HLT
```

5. 乘方表程序

“查表法”是一种可以简化复杂的运算，节省运算时间的重要技巧之一。这种表格应包含该项题目的所有的可能的值。表中的数或符号要按序排列。这样，就把需要做出判断或运算的任务简化为组织表格，使之能够容易地找到正确的数值。直接查表法操作的关键，就是将表首址加偏移量来得到结果地址。当然，为了列出完整的表格，需要花费一定的内存容量。

【例】 利用查表法来计算内存单元 2040 H 中的自然数平方值，并将其存入内存单元 2041 H。

设内存单元 2040 H 中含有在 0～7 之间的一个数（包括 0 和 7），即 0≤(2040 H)≤7。表格从内存单元 2050 H 开始，如表 9-1 所示。

表 9-1 自然数平方表

内存地址 (H)	内容 (16 进制)	内容 (10 进制)
2050	00	0 (0^2)
2051	01	1 (1^2)
2052	04	4 (2^2)
2053	09	9 (3^2)
2054	10	16 (4^2)
2055	19	25 (5^2)
2056	24	36 (6^2)
2057	31	49 (7^2)

编制的 Z 80 源程序如下：

```
标号    指令(助记符)                    注释
        ORG   2000H
        LD    A,    (2040H)   ; 取数据
        LD    L,    A
        LD    H,    00        ; 使数据成为 16 位的索引值(偏移量)
        LD    DE,   SQTAB     ; 取平方表的首地址
        ADD   HL,   DE        ; 求得平方表内的地址
        LD    A,    (HL)      ; 表中的自然数的平方送 A
        LD    (2041H),A       ; 自然数的平方存入 2041 H 单元
        HALT
SQTAB:  DEFB  0,1,4,9,16,25,36,49
```

相应的 Intel 8080A 源程序如下：

```
标号    指令(助记符)
        ORG   2000H
        LDA   2040H
        MOV   L,    A
        MOV   H,    00H
        LXI   D,    SQTAB
        DAD   D
        MOV   A,    M
        STA   2041H
```

```
        HLT
SQTAB:  DB      0,1,4,9,16,25,36,49
```

四、子程序结构

子程序结构程序也是程序设计中的重要技巧之一。任何一个具有标号首地址（或入口地址）和 RET（返回）指令结尾的程序都称为子程序。调用这个子程序的程序称为主程序。主程序是通过调用指令 CALL nn 来调用子程序的。通常，凡是具有相对独立的程序段，或需要多次应用的程序段，都可编制成子程序。这样，既节省了编制程序的工作量，又节省了内存空间。

因而，在程序结构上，应尽量采用循环结构和子程序，这是提高程序效率，节省内存容量的重要手段。当然，在采用子程序时，由于要增加 CALL 和 RET 指令，从而使程序的执行时间有所增加。因此，用子程序的方法来节省内存容量，是以降低一定的速度为代价的。

1．主程序调用子程序的过程

如图 9-12 所示，设子程序 SR_1 的首地址为 SR_1，结尾处有一条返回指令 RET。现在主程序中有两处要调用这个子程序。在每次调用时，当执行调用子程序指令 CALL SR_1 后，则有：

$(SP-1)\leftarrow PC_H$ 程序计数器中的高 8 位地址送至地址为 SP－1 的堆栈中

$(SP-2)\leftarrow PC_L$ 程序计数器中的低 8 位地址送至地址为 SP－2 的堆栈中

$SP\leftarrow SP-2$　　堆栈指示器内容减 2

$PC\leftarrow SR_1$　　子程序的首地址送入程序计数器

这就是说，当执行 CALL 指令时，CPU 将紧跟着 CALL 指令的下一条指令的地址（返回地址）推入堆栈，保护起来（称为记迹，迹是指 PC 的内容），然后，再把该子程序的首地址置入 PC，即转去执行子程序。当子程序执行完最后一条返回指令 RET 时，又有：

$$PC_L\leftarrow(SP)$$

$$PC_H\leftarrow(SP+1)$$

$$SP\leftarrow SP+2$$

也就是说，存在堆栈中的返回地址从堆栈弹出（称为寻迹），从而返回到主程序继续执行。

图 9-12 中示出第一次调用时，返回地址 MR_1+3（因 CALL 指令是三字节指令）先被推入堆栈，执行完子程序后又被弹出堆栈。第二次调用时，返回地址 MR_2+3 也先后被推入堆栈和弹出堆栈。由于记迹和寻迹的作用，因此，子程序每次都能返回到它应返回的地址。

2．子程序的类型及特点

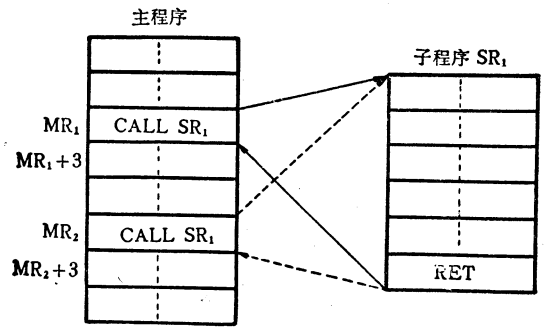


图 9-12　主程序调用子程序的示意图

子程序的类型，通常是根据它从主程序得到参数的方法不同来分类的。这里，参数（参量)是指主程序为子程序所提供的操作数或地址值。

主程序将参数传送给子程序，通常有三种方法：

(1) 通过寄存器传送参数

这类子程序所使用的参数，由主程序在调用转子指令前赋给寄存器，供子程序使用；运算结果存入寄存器而带回主程序。这种方法简单，占用内存单元少，但受到寄存器数量的限制。

(2) 通过存贮器控制区传送参数

这类子程序所使用的参数，由主程序在调用转子指令前，存入存贮器的控制区，然后在子程序中将参数送回有关的寄存器中。它适合于需要传送大量参数给子程序的场合，但其所占用的内存单元较多。

(3) 通过堆栈传送参数

这种方法适合于需要传送大量参数给子程序的场合，与(2) 相比，它所占用的内存单元较少。使用这种方法的关键在于熟练地掌握堆栈指针的应用，并应对堆栈指针的变化了如指掌。堆栈指针应用的要点归纳起来有以下六点：

① 调用转子指令前，应预置堆栈指针的初始值；

② 通过堆栈指针动态存取的，总是包含两个计算机字，而不是一个；

③ 堆栈指针的递增和递减都是自动进行的，数据的入栈或出栈遵循后进先出的原则；

④ 在子程序执行过程中，出现的所有 PUSH 和 POP 指令应成对出现，否则必然使出口处栈中的内容不是返回地址，因而不能保证子程序的正确返回；

⑤ 堆栈区应能容纳得下全部程序所有需要同时进栈的存贮内容；

⑥ 凡子程序中用到的寄存器，都应首先将其中原有的内容保存起来（即保护现场），并在返回主程序前恢复相应寄存的内容(即恢复现场)。

子程序的类型亦可根据使用堆栈的方式分为三种：

① 通过堆栈直接传送参数

【例 1】 主程序

```
        ORG     2000H
        LD      SP,  POINT      ; 设置堆栈指针
        LD      BC,  DATA
        LD      HL,  ADDR
        PUSH    BC              ;
        PUSH    HL              ; 参数入栈
        CALL    SUBR
NEXT:   ……
```

子程序

```
        ORG     2100H
SUBR:   POP     DE              ; 返回地址出栈
        POP     HL              ;
        POP     BC              ; 参数出栈
        PUSH    DE              ; 返回地址回到栈中
        ……
```

TRE

② 通过堆栈传送参数(操作数)的地址

当有一串参数(操作数)需要处理时，可将参数(操作数)串紧跟在CALL指令之后，由CALL指令存入堆栈的PC值是指向CALL指令后的第一个存贮单元的，因而它指到这个参数串的第一个参数(操作数)。换句话说，这种方法从堆栈中取出的PC值是操作数的地址，子程序将以此作为数据指针。取出所需要的操作数。

【例2】 编制求5个数之和的子程序，而主程序按下面的方式调用它，结果放入累加器带回主程序。

主程序

```
        ORG    2000H
START:  LD     SP,     ADDR     ; 置堆栈指针初始值
 MAST:  CALL   TOTAL            ; 调用求和子程序
DOPEY:  DEFB   X1
SNEE2Y: DEFB   X2
HAPPY:  DEFB   X3
  DOC:  DEFB   X4
BASHFL: DEFB   X5
 NEXT:  ……
```

子程序

```
TOAAL:  POP    HL               ; HL 指向参数 X1 的地址
        LD     B,      05H      ; 置计数器初值
        XOR    A                ; 清 A 累加和
 LOOP:  ADD    A,      (HL)     ; 求和
        INC    HL               ; 地址指针加 1
        DJNZ   LOOP-$           ; 若 B≠0, 则转 LOOP
        PUSH   HL               ; 若 B = 0, 则返回地址入栈
        RET                     ; 返回地址弹回 PC
```

本例也可以用两条EX (SP)，HL指令来取代POP HL和PUSH HL指令。

③ 通过堆栈传送参数(操作数的地址)的地址

当主程序只给出操作数的地址，这些地址紧跟在主程序中CALL指令之后，子程序执行完后要求将结果送到主程序CALL指令后面的某些单元中去。对于这类问题，采用本方法比较有效，它使用一些中间存贮单元来存贮操作数，避免了将操作数直接写入子程序中。对于存在ROM中的不能改变的程序，或对于要求跟在CALL指令之后的参数串不允许改变的某些场合，这种技巧是非常有用的。

【例3】 主程序

```
        ORG    2000H
        LD     SP,     POINT
        CALL   SUBR
 MAST:  DEFW   ADDR1
        DEFW   ADDR2
        DEFW   ADDR3
```

```
NEXT:  ……
       子程序
       ORG    2200H
SUBR:  POP    IX              ; IX→第一个参数的地址
       LD     L,    (IX+00H)  ; ADDR1→HL
       LD     H,    (IX+01H)
       LD     C,    (IX+02H)  ; ADDR2→BC
       LD     B,    (IX+03H)
       LD     E,    (IX+04H)  ; ADDR3→DE
       LD     D,    (IX+05H)
       LD     DE,   0006H
       ADD    IX,   DE        ; (IX+0006H)以形成子程序的返回地址
       PUSH   IX              ; 返回地址入栈
       ……
       RET                    ; 返回地址弹回 PC
```

3．子程序的嵌套与递归

子程序在执行过程中，可能又需要调用另一个子程序，即子程序中可以再调用子程序，这种过程叫做子程序嵌套。它的调用过程如图 9-13 所示。

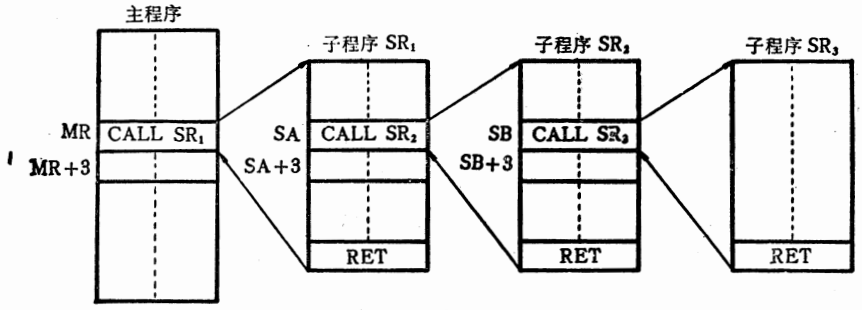


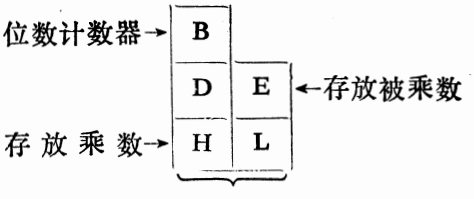
图 9-13　子程序嵌套示意图

在某些情况下子程序还可能需要调用该子程序本身，这称为子程序的递归。通常递归子程序中都设有判断条件，以控制递归的结束。

4．子程序举例：

【例 1】 8 位二进制乘法子程序

假定 Z80 CPU 进行乘法运算，乘法子程序用到的寄存器分配如下：



进行乘法运算前各寄存器的初始状态分别置为 B←8(8 位数)，E←被乘数，D←0，H←乘数，L←0。

乘法运算子程序的框图如图 9-14 所示。

其相应的源程序如下：

```
        ORG  2100H
MULT:   LD   B,   08 H      ; B←8,初始位数
        LD   D,   00        ; 清D寄存器
        LD   L,   D         ; 清L寄存器
LOOP:   ADD  HL,  HL        ; 乘数和16位部分积累加器逻辑左移一位,乘数最高位进入CY
        JR   NC,  DECB-$    ; 若CY=0,跳至DEC B
        ADD  HL,  DE        ; 加被乘数至16位累加器
DECB:   DJNZ LOOP-$         ; B←B-1, B≠0时跳至LOOP重复运算
        RET                 ; B=0时顺序执行乘法子程序,执行完毕返主
```

从上述程序看出，由于这里巧妙地利用了内部寄存器的恰当安排，从而使程序大为缩短。按该程序的算法，最后的乘积就在16位累加器中。因为被乘数是8位，我们可以令16位累加器的低8位初始值为0。16位累加器的高8位开始时还留作它用。这样，运算时可以把乘数放在16位累加器的高8位中，当逻辑左移时，既满足了累加器的左移要求，又达到了乘数的左移需要，一举两得。

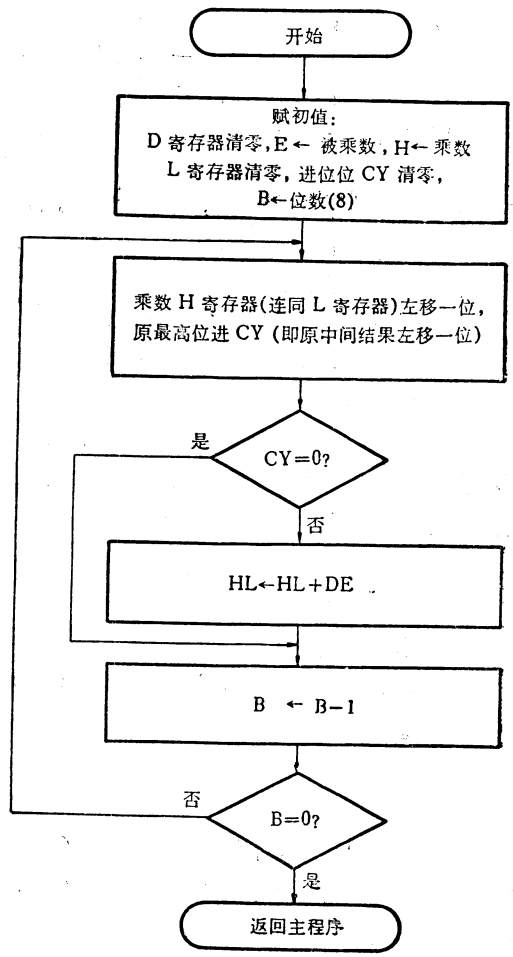
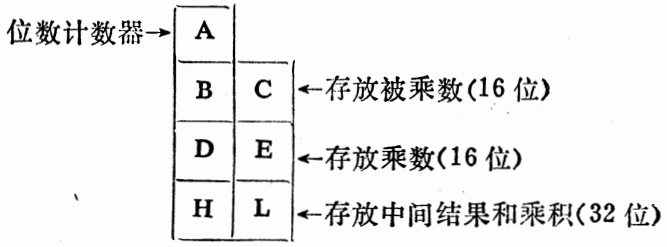


图 9-14　8位乘法子程序的框图

【例 2】 16 位二进制数乘法运算子程序

16 位二进制数相乘时，其乘积为 32 位。它的运算步骤与 8 位乘法运算步骤相似。但寄存器的分配有所不同。16 位乘法操作所需的 CPU 寄存器分配如下：



Z 80 的 16 位乘法子程序如下：

```
标号      指令(助记符)                    注释
          ORG   2200H
MPY:      LD    HL,   0000      ; HL 的初始值置 0
          LD    A,    10H       ; A←16 初始位数
LOOP:     ADD   HL,   HL        ; 中间结果左移一位，最高位进入进位位 CY
          EX    DE,   HL        ; DE 与 HL 内容交换
          ADC   HL,   HL        ; 连进位左移一位，乘数最高位进入 CY
          EX    DE,   HL        ; 乘数放回 DE
          JR    NC,   DECA-$    ; 若 CY = 0，转至 DECA，否则顺序执行
          ADD   HL,   BC        ; 中间结果加上被乘数
          JR    NC,   DECA-$    ; 若 CY = 0，转至 DECA，否则顺序执行
          INC   DE              ; 由于上面 ADD 指令执行时有进位，故应 DE + 1
DECA:     DEC   A               ; 位计数器减 1
          JP    NZ,   LOOP-$    ; 当 A≠0 时，程序跳转到 LOOP 程序
          RET                   ; 乘法子程序结束，返回主程序
```

【例 3】 ASCII 字符串的比较子程序

假设这里有两个字符串进行比较，看它们是否相同。字符串的长度放在累加器中。一个字符串的起始地址放在 HL 寄存器；另一个字符串的起始地址放在 DE 寄存器。经过比较，若两个字符串相等，则清除累加器；否则，把累加器置成 FFH。

例如：

A = 03	DE = 2050H	HL = 2060H
(2050H) = 43, 'C'	(2051H) = 41, 'A'	(2052H) = 54, 'T'
(2060H) = 43, 'C'	(2061H) = 41, 'A'	(2062H) = 54, 'T'

因其比较结果相等，故 A = 0 。

又如：

A = 03	DE = 2050H	HL = 2060H
(2050H) = 52, 'R'	(2051H) = 41, 'A'	(2052H) = 54, 'T'
(2060H) = 43, 'C'	(2061H) = 41, 'A'	(2062H) = 54, 'T'

因其比较结果不等，故 A = FFH。

假定第一和第二个字符串的地址指针分别为 POINTER 1 和 PO NTER 2，字符串长度为

COUNT，则这个程序的流程图如图 9-15 所示。

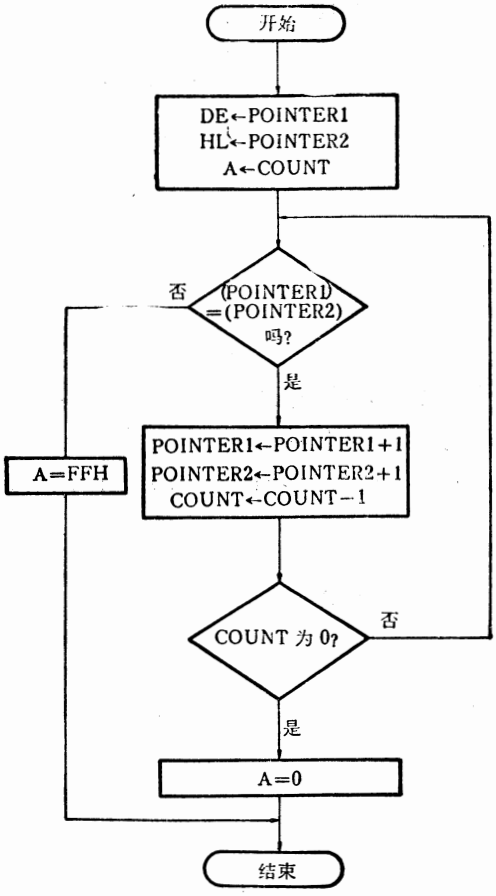


图 9-15　字符串比较程序流程图

根据这个流程图就可以编写出程序。首先把堆栈指示器预置为 2 FC 0 H，两个字符串的起始存贮单元地址分别为 2050 H 和 2060 H，字符串长度放在 2040 H 单元，比较结果放在 2041 H 单元。

为了能返回主程序，在子程序的最后设置一条返回指令 RET。

实现上述功能的字符串比较的 Z 80 源程序如下：

```
标号      指令(助记符)          注释
          ORG     2000H
          LD      SP,     2FC0H     ; 置堆栈指示器为 2FC0H
          LD      DE,     2060H     ; 取字符串 1 的起始地址
          LD      HL,     2050H     ; 取字符串 2 的起始地址
          LD      A,      (2040H)   ; 取字符串长度
          CALL    PMTCH             ; 调用子程序 PMTCH，检验两个字符串是否相等
          LD      (2041H),  A       ; 存比较结果
          HALT
```

以下为子程序：

```
        ORG   2080H
PMTCH:  LD    B,      A          ; 计数=字符串长度
        LD    C,      0FFH       ; 取表示两个字符串不相同的标志=0FFH
CHCAR:  LD    A,      (DE)       ; 从字符串1中取一个字符
        CP    (HL)               ; 比较两个字符串
        JR    NZ,     DONE-$     ; 比较结果不等,转DONE
        INC   DE
        INC   HL
        DJNZ  CHCAR-$
        LD    C,      00         ; 比较结果相等,A=0
DONE:   LD    A,      C
        RET
```

【例4】 编制一个子程序，将2100H单元的内容拆成两段，其高4位送入2141H单元的低4位，其低4位送入2142H单元的低4位，2141H和2142H单元的高4位清零，这三个单元的地址放在主程序中调用指令CALL后面的6个单元中，每个地址的高8位在前。

主程序：

```
        ORG   2000H
START:  LD    SP,     ADDR
        CALL  BREBYT
MAST:   DEFW  2100H
        DEFW  2141H
        DEFW  2142H
NEXT:   ……
```

子程序：

```
        ORG   2200H
BREBYT: POP   IX                 ; 置IX初始值为MAST
        LD    L,    (IX+00H)
        LD    H,    (IX+01H)     ; 2100H→HL
        LD    C,    (IX+02H)
        LD    B,    (IX+03H)     ; 2141H→BC
        LD    E,    (IX+04H)
        LD    D,    (IX+05H)     ; 2142H→DE
        XOR   A                  ; 清除累加器A
        RLD                      ; 取(HL)高4位
        LD    (BC), A            ; 送入2141H单元
        RLD                      ; 取(HL)低4位
        LD    (DE), A            ; 送入2142H单元
        LD    DE,   0006H
        ADD   IX,   DE           ; 形成返回地址
        PUSH  IX                 ; 返回地址←堆栈
        RET
```

本例从堆栈中取出的不是操作数的地址，而是操作数地址的地址。子程序开头的POP指

令和 RET 指令前面 3 条指令，都是为了把返回地址正确地送入堆栈而设置的。

五、排序(Sort)程序

通常把数组的重新排列称为排序。经过排序数按上升或下降顺序排列。最常用的排序方法之一称为“冒泡”(上推)排序法，即模拟水底的气泡逐个地交换位置向上浮起过程的方法。

【例】 若一数组有N项数，现设为 10 个数，数组长度放在 2040H 单元，数组的起始地址为 2041H。设这 10 个数为带符号的数，如表 9-1 所示。现要求编制一个排序程序，使这 10 个数按由小到大顺序排列。

我们采用两两比较法，其要点如下：

(1) 从数组的最后一个数开始[注]，地址由大到小，让某数与它的前一个数相比较；

(2) 若该数较大，则不交换，反之则交换；

(3) 数组的指针减 1，并重复以上过程，直至按小到大的顺序排好为止；

(4) 数组通过第一遍扫描(称大循环)之后，将最小项“漂浮”到顶部；在第一次大循环中，对N项的数组要进行(N－1)次比较，即小循环了(N－1)次；

(5) 此后，每通过一遍扫描，可以比上一遍少比较一次；

(6) 实际上有的数组经过几次扫描后，可能已经排列好了，为避免不必要的再循环，可引入交换标志，如果在一次扫描中，未发生交换，则表示排序结束；

(7) 对N项数组来说，大循环变量 I＝2 至 N；每一个大循环中要进行比较的小循环变量 J＝N 至 I。

现设：寄存器 E←I (大循环变量为 2)；

寄存器 D←J (小循环变量为 N)；

寄存器 C ←交换标志；

变址寄存器 IX←地址指针(2041H)；

COUNT←数组长度(10)。

表 9-1 数组排序程序的扫描过程

内存单元地址(H)	初始状态	第一遍扫描后	第二遍扫描后	第(N－1)遍扫描后(实际为第七遍)
2041	15(0F)	−50(CE)	−50(CE)	−50(CE)
2042	−16(F0)	15(0F)	−16(F0)	−16(F0)
2043	70(46)	−16(F0)	15(0F)	−09(F7)
2044	−05(FB)	70(46)	−09(F7)	−08(F8)
2045	−50(CE)	−05(FB)	70(46)	−05(FB)
2046	−08(F8)	−09(F7)	−05(FB)	02(02)
2047	127(7F)	−08(F8)	−08(F8)	15(0F)
2048	−09(F7)	127(7F)	02(02)	45(2D)
2049	45(2D)	02(02)	127(7F)	70(46)
204A	02(02)	45(2D)	45(2D)	127(7F)

注：括号内为 16 进制补码数

【注】 也可以从数组的第一个数开始，地址由小到大，进行两两比较法。

由于是对两个带符号数进行比较，因此，应采用符号标志S和溢出标志P/V相结合的方法来判断它们的大小。

Z80的排序源程序：

```
START:  LD      E,      02H         ; 置I初始值
LDOP:   LD      IX,     TABLE       ; IX指向数组的起始地址
        LD      C,      COUNT       ; 置交换标志
        LD      D,      COUNT       ; 置小循环变量初始值
        LD      B,      00
        ADD     IX,     BC
        DEC     IX                  ; IX指向数组的最后一个数
LOOP1:  LD      A,      IX          ; eJ→A
        LD      B,      (IX-1)      ; eJ-1→B
        CP      B
        JP      M,      SIGN
        JP      PE,     EXCH
        JR      UNCH
SIGN:   JP      PE,     UNCH
EXCH:   LD      (IX-1), A
        LD      (IX-0), B           ; 交换 eJ←→eJ-1
        LD      C,      D           ; 改变交换标志
UNCH:   DEC     IX                  ; 修改地址指针
        DEC     D                   ; J←J-1
        LD      A,      D
        CP      E                   ; 比较J与I
        JP      P,      LOOP1
        LD      A,      C
        CP      COUNT               ; 检查交换标志
        JR      Z,      DONE        ; 数组已排列好，转停机
        INC     E
        JP      LOOP
DONE:   HALT
        ORG     2041H
TABLE:  DEFB    15,-16,70,-5,-50,-8,127,-9,45,2
COUNT:  EQU     0AH
        END
```

限于篇幅，本章仅介绍汇编语言程序设计中的基本方法和技巧。微型计算机系统通常都配有浮点运算及基本函数运算的子程序库，读者可在学习本章的基础上，结合实际需要，消化现成的子程序库[注]，并将其移植到自己的微型机应用系统中来。

【注】 参考资料：《8080子程序库》电子科学(日文) 1977年9月号，《台式机浮点程序》(日文)小方定修，小林茂男著。

第十章　微型计算机的存贮器及其接口

本章着重介绍微型计算机中最常用的半导体存贮器的特点、结构及其与微处理器的接口设计。

§10-1　半导体存贮器的分类

微型计算机的存贮器有半导体存贮器、磁芯存贮器、磁盘、磁带等多种。

由于半导体存贮器具有速度高、集成度高、体积小、功耗低、成本低、应用方便等优点，近年来已被大量采用。目前微型计算机的主存贮器几乎全部采用半导体存贮器，而且都已经 LSI 化了。现在国外单片容量为 64 K 位的 RAM 已经上市，256 K 位的 RAM 和 EPROM 也已出现。常用的外存贮器有盒式磁带和磁盘。

半导体存贮器的种类很多。从器件原理来分，有双极型和 MOS 型存贮器；从存贮原理来分，有静态和动态存贮器；从存取方式来分，有随机存取和只读存贮器；从信息传送方式来分，有并行（字长的所有位同时存取）和串行（一位一位存取）存贮器。半导体存贮器的分类，如图 10-1 所示。

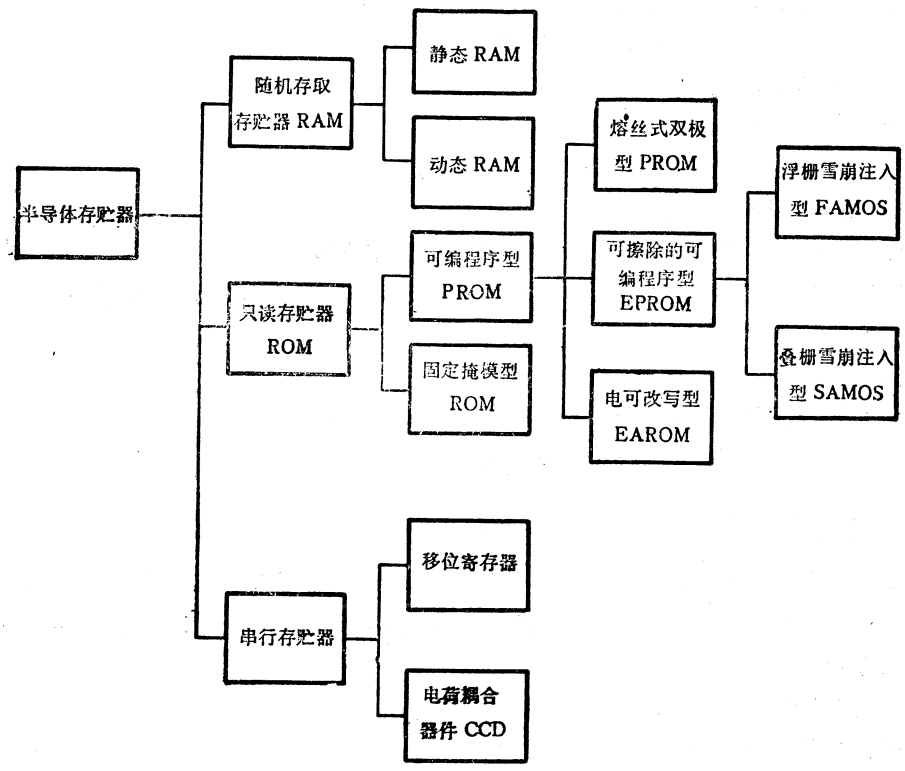


图 10-1　半导体存贮器的分类

下面从存取方式和使用功能方面简要地介绍存贮器的分类：

一、随机存取存贮器 RAM(Random Access Memory)，又称读写存贮器 RWM

RAM 的特点是它的存贮单元的内容能够根据需要读出或写入。由于它是易失性的存贮器，即断电后存贮单元的信息也随之消失。因此，在微型计算机中，它常用来暂时存贮输入/输出数据、中间计算结果或者断电后不影响大局的程序等。

在 RAM 中，又可分双极型(Bipolar)和 MOS 型两类：

1．双极型 RAM

其特点是：

(1) 存取速度比 MOS 型高。射极耦合逻辑 (ECL) 电路的开关速度可达 10 ns；肖特基 (Schottky) TTL 逻辑电路的开关速度可达 25 ns。

(2) 以晶体管的触发器作为基本存贮单元，故管子数目较多。

(3) 集成度较低，功耗大、成本高。

因此，双极型 RAM 主要用在高速的微型机中。

2．MOS 型 RAM

MOS 型 RAM 的主要特点是工艺简单、功耗低、成本低、集成度高。因而，MOS 型 RAM 得到了广泛的应用。它又可分为静态(Static)RAM 和动态(Dynamic)RAM 两种。

(1) 静态 RAM 的特点：

① 一般由 6 管构成的触发器作为基本存贮单元；

② 集成度高于双极型，但低于动态 RAM；

③ 功耗比双极型的低，但比动态 RAM 高；

④ 不需要刷新，可省去刷新逻辑；

⑤ 易于用电池作为备用电源；

⑥ 存取时间为 200～450 ns (如 Intel 2114)。

(2) 动态 RAM 的特点：

① 可用 3 管甚至单管电路组成基本存贮单元(用 MOS 管的栅衬电容存贮电荷)；

② 集成度高，目前高达 64 K 位的动态 RAM 如 Intel 2164 A 已经产品化；

③ 功耗比静态 RAM 更低；

④ 价格比静态 RAM 更便宜；

⑤ 需要刷新，通常要求每隔 2 ms 刷新一遍；

⑥ 存取时间为 150～350 ns (如 Intel 2116)。

二、只读存贮器 ROM(Read Only Memory)

ROM 是一种一旦“编入程序”之后，就只能读出而不能写入的存贮器。它是非易失性的存贮器，即断电后存贮单元的信息能够保留，故常用来存放固定的程序，如微型机的监控程序、汇编程序等，还用来存放各种表格、常数。

按存贮信息的方式 ROM 又可分为以下三种：

1．固定掩模型 ROM(Fixed-mask type ROM)

这种 ROM 是在制作集成电路时，用定做的掩模进行编程的。一旦制造完毕，存贮器内容

就被固定下来，用户是不能修改的。如果要修改存贮器的内容，就只能重新设计掩模。这种ROM在大批量生产时成本很低，但不适宜于科研工作。

2．可编程序的只读存贮器PROM(Programmable ROM)

这种PROM允许用户根据需要来编写ROM中的内容，但是只允许编程一次。一般用于用户自行编程的场合。

常用的一种是熔丝熔断型的PROM。用户用熔断或不熔断熔丝来表示"1"或"0"。这种PROM一旦写入信息后，就具有永久固定的内容，只能读出，不能重写。

以上两类ROM可以用双极型也可以用MOS工艺制作。

3．可擦除的PROM(Erasable Programmable ROM简称EPROM)

这种ROM可以多次改写，它又可分为用紫外线擦除（EPROM）和用电改写(EAROM——Electrically-Alterable ROM)等不同的类型。由于EPROM具有多次改写的性能，因而获得了广泛的应用。但是，写入EPROM的速度较慢，且需要专用的EPROM写入器。此外，价格也较高。

§10-2 只读存贮器

一、固定掩模只读存贮器ROM

这类ROM可由二极管、双极型晶体管或MOS型晶体管构成。每个存贮单元只用一个耦合元件，集成度很高。

ROM主要由地址寄存器、地址译码器、存贮单元矩阵、输出缓冲寄存器以及芯片选择逻辑等部件组成。如图10-2(a)所示。图10-2(b)示出了它的信号引线。

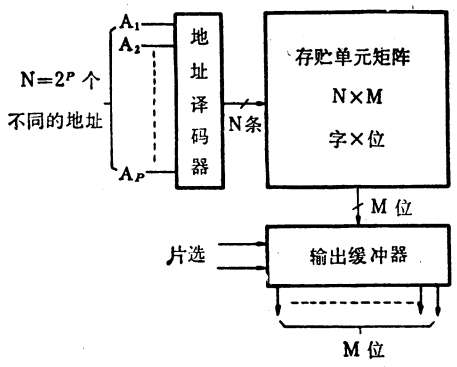


图10-2(a) ROM的基本组成

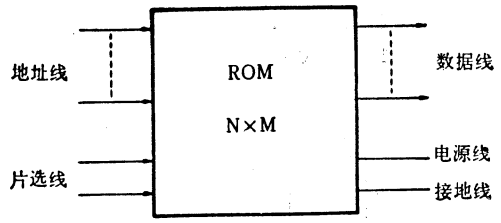


图10-2(b) ROM的信号引线

图中，地址译码器有P个输入端，故能译出$N=2^P$个地址。每个地址单元的字有M位，因此输出缓冲器（用于提供驱动电流）输出M位的内容。片选输入信号用以选择指定的芯片进行操作。

存贮器中的每一个存贮单元，都必须有一个唯一的地址，根据它的地址可以读出这一单元的信息。存贮器采用两种不同的编址方案，即线选法和重合选择法。前者较简单，常用于小容量的ROM中；重合选择法通常用于大容量ROM中。

1．线选法编址

采用线选法编址的ROM，其基本结构如图 10-3 所示。

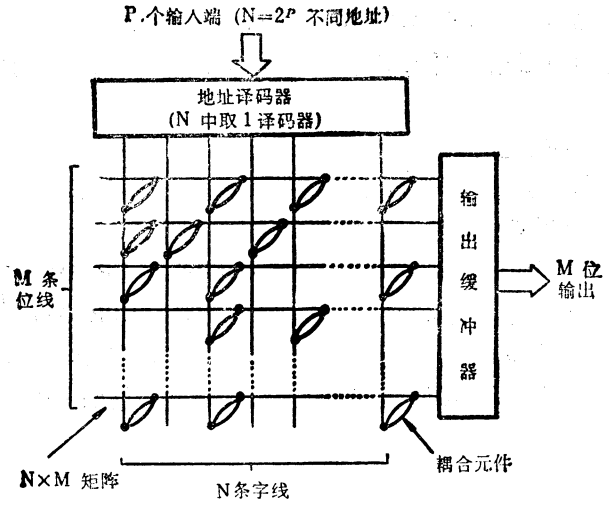


图 10-3　线选法编址的ROM

图中的地址译码器通常用“N 中取一”译码器，译码输出使N条字线中之一有效。每一个字由M位组成，排列成 N×M 的二维(度)存贮矩阵。经地址译码后，相应的N条字线中之一将出现“1”态电平，故凡与这一字线有耦合元件相连的各位都会得到“1”态电平。也就是说，这些位都存有信息“1”，而没有耦合元件与该字线连接的位，即存有信息“0”。

用作线选法 ROM 的耦合元件的种类有电阻、电容、二极管、双极型或 MOS 型晶体管等元件。

图 10-4 是用二极管作耦合元件的 4×4 矩阵例子，图中表示地址输入为 $A_1A_0=10$ 时的情况。此时译码输出使字线 W_2 有效，输出缓冲器的输出为 1011。也可将图 10-4 中的二极管

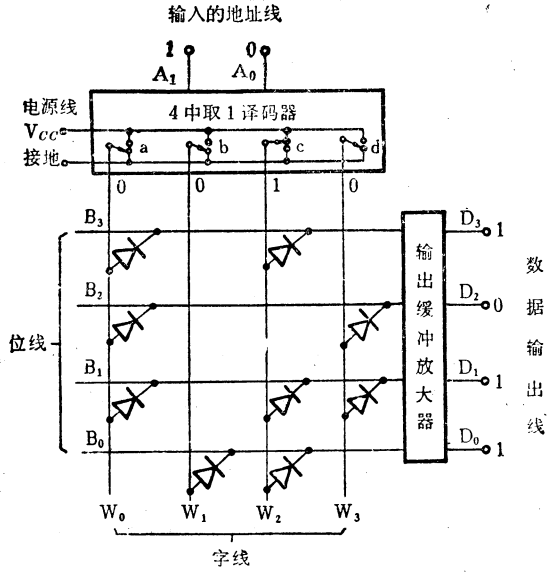


图 10-4　采用线选法的二极管耦合的 ROM

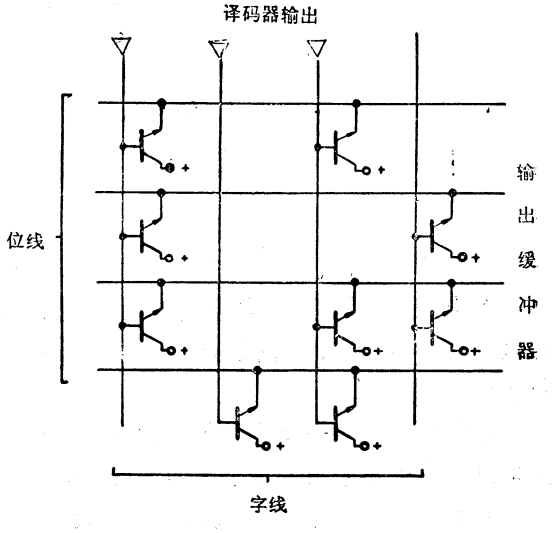


图 10-5　双极型晶体管耦合的存贮矩阵

改用双极型晶体管作耦合元件，如图 10-5 所示。

2．重合选择法编址

对于大容量的 ROM，若采用线选法编址，存贮地址译码器将变成很复杂的逻辑网络。这时宜采用重合选择法编址，简称合选法。合选法编址采用一个N位的 X-Y 矩阵来代替图10-3 中的每一条位线。由于共有 M 条位线，所以需 M 个 X-Y 矩阵。这时，图 10-3 中的地址译码器将变成由X中取 1 及 Y 中取 1 两个译码器组成，而存贮器则由 M 个 X-Y 矩阵组成。对存贮器寻址时，要同时提供X地址和 Y 地址。

图 10-6 是一个用重合选择法编址的 ROM 例子。其中X中取 1 译码器有 q 个输入端，译码后有X条线输出，$X=2^q$。与此相似，Y 译码器有 r 个输入端，译码输出有 Y 条线，$Y=2^r$。所以 $N=X\cdot Y=2^{q+r}$。对照线选法编址，可见 $p=q+r$，即两种编址方法所需输入端的总数是相同的。

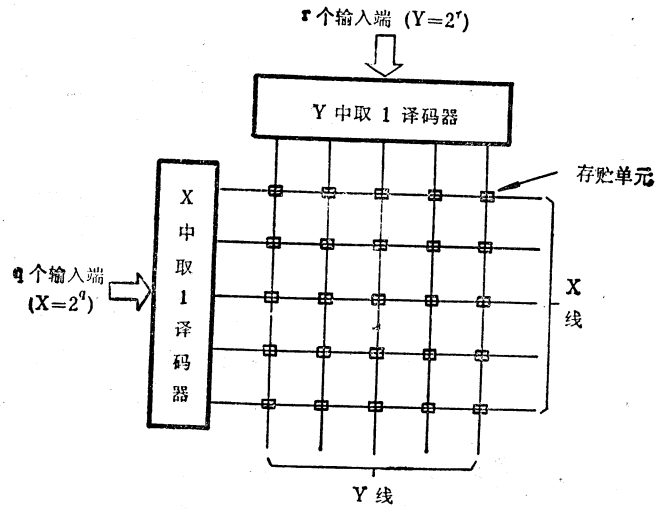


图 10-6　重合选择法编址的ROM中的一个X-Y 阵列

当读取 ROM 中的某存贮字时，可将 q 个输入加至X译码器，由它激发相应的一条X线（或称行选或字选）。同样地，将 r 个输入加至 Y 译码器，也激发相应一条 Y 线（或称列选或位选）。这两条被激发的X线与 Y 线相交于矩阵中的某一个存贮单元上，然后通过读出线将该位的状态送至输出缓冲器，这样便可读出 ROM 矩阵中的一个字的一位。如果这个字是 M 位，要有 M 个 X-Y 阵列并行连接，同时工作。这样，就可实现 M 位的并行操作。若设 M = 8 位，则要有 8 块存贮器板并联起来，用同一方法寻址。每一块板输出一位信息，8 块板共输出 8 位信息，组成一个字的输出，如图 10-7 所示。这 8 块板的组合称为存贮器模块。

图 10-8 所示的重合选择法的 ROM 中，X-Y 存贮阵列用 MOS 管耦合的形式。由栅极连到X线上的管子表示逻辑“0”；栅极断开的表示逻辑“1”。

为了简单起见，图 10-8 中X和 Y 选择线只画了两条。T_{11}、T_{12}、T_{21}、T_{22} 为 MOS 晶体管耦合元件。V_{DD} 为电源电压。

设X选择线 1 和 Y 选择线 1 被选中，它们都是高电平，这时 T_1 管导通，耦合元件 T_{11} 管被选中。由于 T_{11} 的栅极断开，故 T_{11} 管截止，读出线上读出的是高电平，表示逻辑“1”。如果X选择线 1 和 Y 选择线 2 被选中，则 T_{12} 管被选中。由于 T_{12} 的栅极接通为高电平，故 T_{12} 导

通，同时 T_2 管也导通，因而读出线读出的是低电平，表示逻辑“0”。

3．二极管组成的 ROM

图 10-9 是一个 16×1 的二极管组成的 ROM。它有四根地址线 A_0、A_1、A_2、A_3，可以选择 $2^4=16$ 个地址。根据 A_0、A_1 是 00、01、10、11 四种可能中的一种，可通过列选择的 4 中取 1 译码器选中一条线。译码器的四条输出线分别接到四个放大器。如某一条线被选中，则与这一条线相连的放大器的输入和输出就相通。如果没有选中，放大器就处于高阻状态，使输入和输出隔离。这是一种三态控制放大器，它允许四个放大器的输出端连在一起。线 1、2、3、4 称为三态控制线(允许线)。

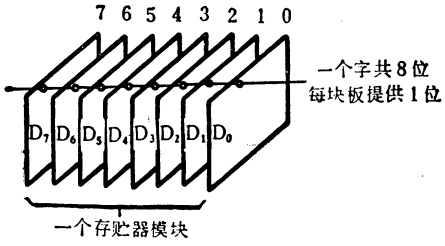


图 10-7　存贮器模块

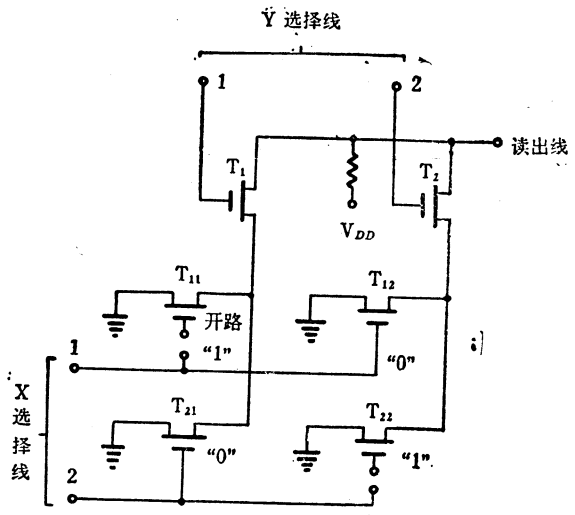


图 10-8　采用 MOS 晶体管耦合的 X-Y 存贮器阵列

如果图中的线 4 和线 5 被选中，则二极管 a 被选中，高电平通过二极管 a 经过放大器和输出驱动器而输出，表示输出逻辑“1”。输出驱动器有一片选控制端 CS。仅当 CS 为“1”电平时，

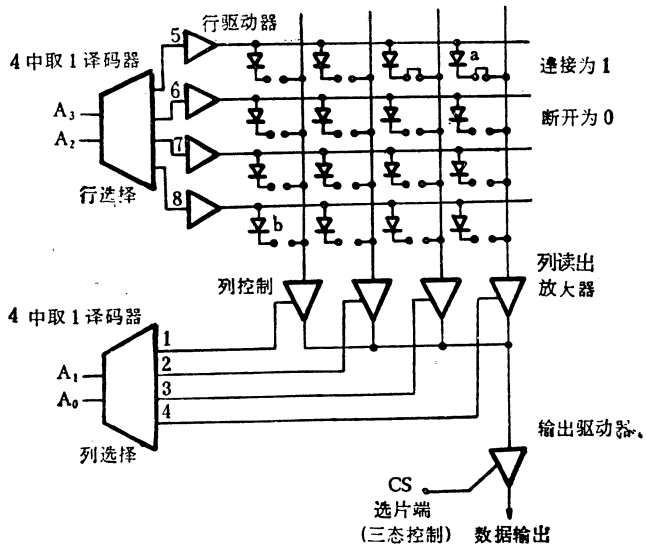


图 10-9　16×1 二极管组成的 ROM

才允许驱动器输出，否则为高阻状态。如果线 1 和线 8 被选中，则二极管 b 被选中，由于它是断开的，故输出为逻辑“0”。

ROM 的存贮单元主要采用 MOS 或双极型工艺制作。为了便于多片存贮器并联共用同一

数据总线，许多半导体存贮器带有三态输出，在不用这一片时，使它处于高阻状态。通常，在这类 ROM 片中备有 CE (片允许)或 CS (片选择)引线，用以选择或隔离芯片。

二、可编程序的只读存贮器 PROM

为了便于用户根据需要来确定 ROM 的存贮内容，生产了 PROM，它可允许用户编程一次。

PROM 常采用可熔金属丝连接存贮单元的发射极。出厂时所有管子的熔丝都是连着的，在制造厂完成后可作一次性修改。例如，用镍-铬(Ni-Cr)丝做成的连接线，由外部通以足够大的电流即能把所选定回路的熔丝熔断(断开)，以存取信息。图 10-10 示出熔丝式 PROM 单元。

若行线被选中，则 T_{XY} 管应该导通。如果连接的熔丝未熔断，则列线应被拉到“V_{CC}(+5V)”电平；如果这条熔丝已先被熔断，则列线仍然断开，保持“0”电平。

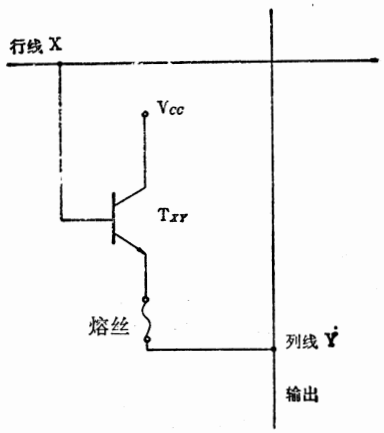


图 10-10　熔丝式 PROM 单元

三、可擦除的可编程序只读存贮器 EPROM

EPROM 的特点是可由用户根据需要进行编程，而且可以加以反复修改，因而获得了广泛的应用。

实现 EPROM 的技术，常用的有浮栅技术和迭栅技术，即所谓浮栅雪崩注入式 MOS——Floating gate Avalanche injection MOS (简称 FAMOS) 和迭栅雪崩注入式 MOS——Avalanche injection MOS(简称 SAMOS)。这两种器件的工作原理大致相同，其中以 FAMOS Stacked gate 器件在微型计算机中用得较多。这里仅介绍 FAMOS 器件的工作原理。

1.器件结构和工作原理

FAMOS 器件的结构如图 10-11(a)所示。

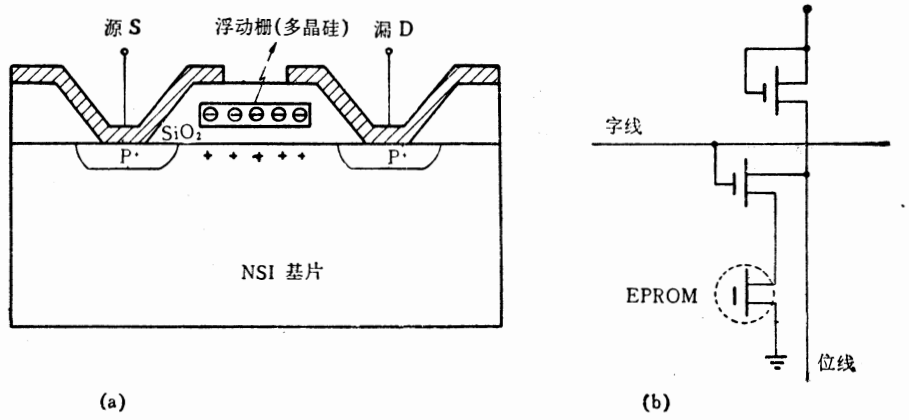


图 10-11　P 沟道 EPROM 结构示意图

FAMOS 器件与普通的 P 沟道 MOS 电路相似。用N型硅为衬底，在N型表面上，用掩蔽扩散的方法做成两个高掺杂的 P^+ 型扩散区，从它上面分别引出源电极(S)和漏电极 (D)。在

S 和 D 之间有一个由多晶硅制成的栅极浮置在绝缘的 SiO_2 层中，与四周没有电气的接触，故称为浮动栅。在 FAMOS 器件没有编程以前，由于浮动栅上没有电荷，则管子内没有导电沟道。P 型的源扩散区和漏扩散区之间隔着 N 型区，这样，相当于“背靠背”地连在一起的二极管。这时，即使在 S 和 D 之间加上电压，也不会导通。

当把 FAMOS 器件用于存贮矩阵时，一个基本存贮电路如图 10-11(b)所示。

这种 FAMOS 器件是以浮动栅是否带了电荷来区分“1”与“0”两种状态的。如以带电时为“1”，不带电为“0”。浮栅带电的原理是源极接地，漏极加以较高的负电压(对 PMOS管)，如 −25～−30V，另外再加上编程脉冲(其宽度为 50 ms)，则所选中的单元在这个电源作用下，导致漏区 P-N 结发生雪崩击穿。由雪崩击穿所形成的大量热电子，就会穿过薄氧化层 (SiO_2) 而注入浮栅，使浮栅充电。当高压电源去除后，因为浮栅被绝缘层包围，无放电回路，故浮栅上积累了负电荷。于是，浮栅上的负电荷就在氧化层下面产生一个电场，把硅表面附近的电子排斥走，产生一个有大量空穴的薄层。这个薄层是一个能导电的 P 型层，称为反应层。这个反应层成为导电沟道，把该晶体管的源扩散区与漏扩散区沟通起来。当漏极与源极之间加上一定的电压时，就会有电流流过沟道。如果浮栅中没有保存电子电荷，则该晶体管源扩散区与漏扩散区之间就不会形成导电沟道，而处于截止状态。这样，便可以区分出该晶体管所存的信息是“1”还是“0”。利用这种方法，我们就可以根据需要，对 EPROM 进行编程。

当我们需要对 EPROM 中的信息进行修改时，最常见的方法是采用紫外线照射 EPROM 芯片上的石英玻璃窗口，使浮栅中的电子获得高能量，从而形成光电流从浮栅流入基片。这实际上就是为浮栅造成一个泄放电流的通路，使存贮在浮栅中的电子电荷泄漏掉而恢复电路的初始状态。这样，经过紫外线照射后的 EPROM 就可以重新编程。现在市场上已有供应带有紫外线光源的 EPROM 擦除器，可以迅速、方便地擦除 EPROM 中的信息。

如果晶体管采用 NMOS 型工艺，例如在 Intel 2716 中，若浮栅上积存电子电荷，则浮栅下面感应的正电荷，将使该晶体更不易导电(表示信息“0”)；若浮栅上没有积存电子电荷，则该晶体管易于导通(表示信息“1”)。

2. EPROM 的实例——Intel 2716

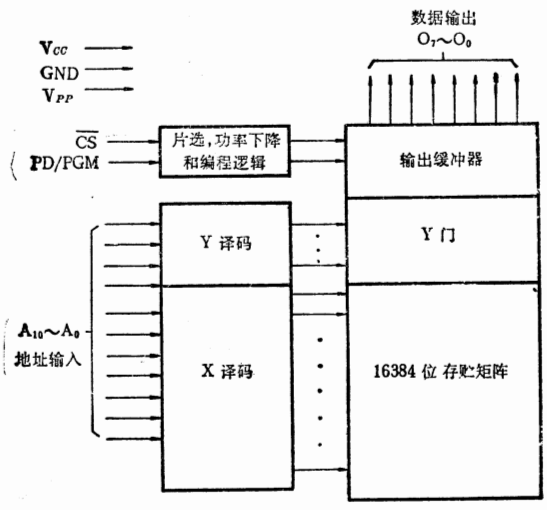


图 10-12　2716 的内部结构方框图

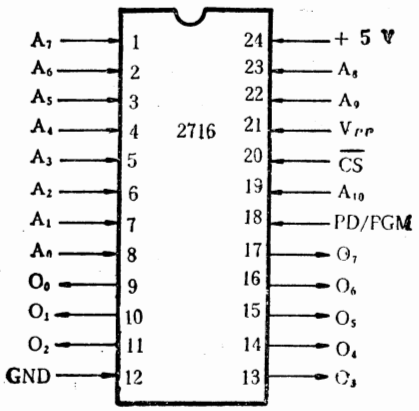


图 10-13　2716 的引线图

(1) 2716 的结构及引线

Intel 2716 是一个 16 K 位(2K×8 位)的 EPROM，采用 NMOS 工艺。其内部方框图及引线图如图 10-12，图 10-13 所示。

2716 引 线 名

A_7～A_0	地　　址
PD/PGM(CE/PGM)	功率下降/编程(芯片允许/编程)
$\overline{CS}$(OE)	片选(输出允许)
O_7～O_0	输出

在上面引线名表中，括号内的引线名表示该引线的另一种名称。如 PD/PGM (Power Down/Program)，又称 $\overline{CE}$/PGM(Chip Enable/Program)；$\overline{CS}$(Chip Select)，又称 $\overline{OE}$(Output Enable)。

因为 2716 的存贮容量是 2K×8 位，所以用 11 条地址线。其中，7 条用于 X 译码，以选择 128 行中的一行，4 条用于 Y 译码，以选择所寻址的列。8 位数据都通过缓冲器输出。它只要求单一的 +5V 电源，仅当需要编程(即写入程序)方式时，才用到电源 V_{PP} = +25 V；而在正常读数方式时，V_{PP} = +5 V。

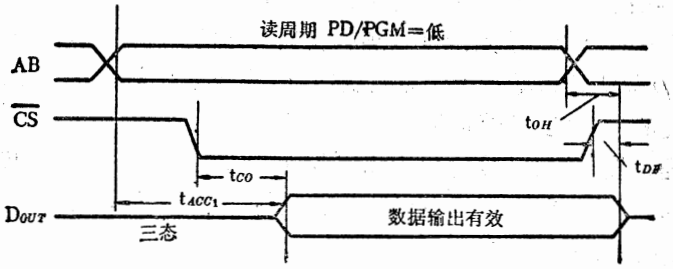
(2) 2716 的工作方式

Intel 2716 的工作方式如表 10-1 所示。

表 10-1　2716 的工作方式选择

方　式	引线				
	PD/PGM (18)	CS (20)	V_{PP}(V) (21)	V_{CC}(V) (24)	输出 (9～11,13～17)
读	V_{IL}	V_{IL}	+5	+5	D_{OUT}
未选中	/	V_{IH}	+5	+5	三态
功率下降	V_{IH}	/	+5	+5	三态
编程	V_{IL} 到 V_{IH} 50 ms 由低到高的脉冲	V_{IH}	+25	+5	D_{IN}
程序检验	V_{IL}	V_{IL}	+25	+5	D_{OUT}
程序禁止	V_{IL}	V_{IH}	+25	+5	三态

下面简要介绍这几种工作方式：



符　号	参　数	极　限			单　位	测　试　条　件
		min	典　型	max		
t_{ACC1}	地址到输出延迟		250	450	ns	PD/PGM=$\overline{CS}$=V_{IL}
t_{ACC2}	PD/PGM 到输出延迟		280	450	ns	$\overline{CS}$=V_{IL}
t_{CO}	片选到输出延迟			120	ns	PD/PGM=V_{IL}
t_{PF}	PD/PGM 到输出浮空	0		100	ns	$\overline{CS}$=V_{IL}
t_{DF}	片选无效到输出浮空	0		100	ns	PD/PGM=V_{IL}
t_{OH}	地址断开到输出保持	0			ns	PD/PGM=$\overline{CS}$=V_{IL}

图 10-14　2716 读周期定时波形图

① 读数方式

2716 的读周期定时波形图如图 10-14 所示。它从地址有效开始，经过一个存取时间(≤450 ns)后，数据即可由存贮单元读出。但能否输出至外部数据总线，还取决于片选信号 $\overline{CS}$。即片选信号必须在数据有效前保证有效。

② 未选中(禁止输出)

这时，$\overline{CS}$=1 ($\overline{OE}$=1)，故数据输出线呈高阻状态，即该芯片不起任何作用。

③ 功率下降方式 PD——POWER DOWN

当 PD/PGM=1($\overline{CE}$/PGM=1)时，2716 芯片处于功率下降方式(后备方式)。这种情况与未选中状态相似，区别仅在于此时的功率仅为最大运行功率的 1/4。即 2716 的功率可由 525 mW 下降为 132 mW，下降 75%。这可由在 PD/PGM 输入端输入一个 TTL 的高电平信号来实现，此时 EPROM 的输出端处于高阻状态。其定时波形图如图 10-15 所示。

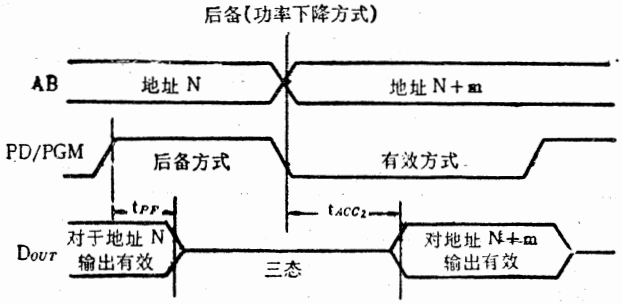


图 10-15　2716 功率下降方式的时序图

通常情况下 $\overline{CS}$ 线与 PD/PGM 线是连在一起的。这样，未选中的芯片就工作在功率下降方式，以减低功耗。

④ 编程方式

2716 在每次擦除信息后，所有存贮单元都为“1”信息。要编程序时，V_{PP} 为 +25V；$\overline{CS}$ 在整个写入期间应为高电平；CPU 给定地址以选择所要写入的单元，接着再供给所要写入的 8 位数据。然后，在 PD/PGM 输入端加上一定宽度(45～55 ms)的 TTL 高电平脉冲，即可实现“写入”。加过脉冲单元的信息变为“0”信息。

对 2716 的编程是按单元分别进行的。对 2K 个单元的写入，编程总共约需 100 秒。如果需要把 2716 单元的“0”改成“1”，则只有用紫外线照射整个 EPROM 芯片，使之全部单元变为“1”信息之后，才能重新编程。

对 2716 编程也是从地址有效开始的，只有地址线稳定以后，才能输入 PD/PGM 脉冲，并且必须保证在 PD/PGM 有效前输入数据。其定时波形如图 10-16 所示。

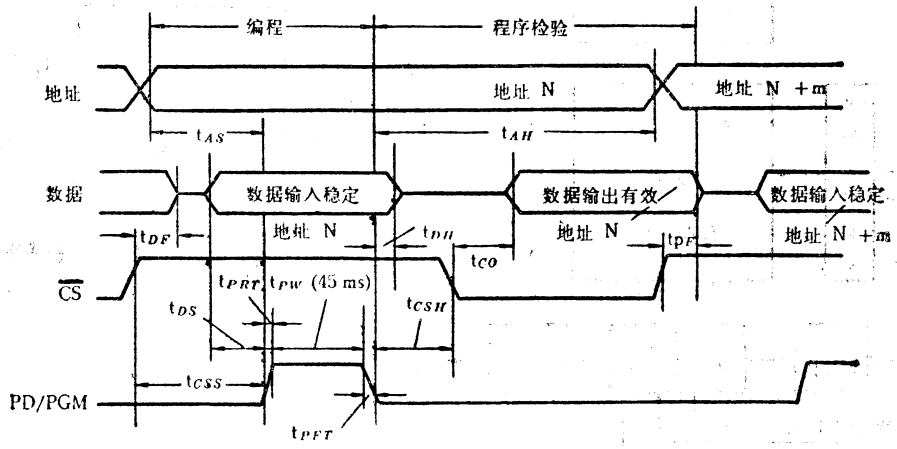


图 10-16　2716 编程写入时序图

⑤ 程序检验方式

在编程完毕后，将其中的内容进行一次读出，并与应写入的信息进行比较以检验编程是否正确。

⑥ 程序禁止方式

此时，禁止将数据线上的内容写入 EPROM。

四、电可改写的 PROM——EAROM

上面介绍的 EPROM 比固定掩模型 ROM 和只能进行一次编程的 PROM 在性能方面优越得多。但美中不足的是在擦除信息时，不管芯片中的存贮内容哪些合用，哪些不合用，一律都擦除干净。为此，采用金属-氮-氧化物-硅 (MNOS) 之类的工艺可以进一步构成电可改写的 PROM，即 EAROM。这种 PROM 可以让电流只通入指定的存贮单元，把其中某一个字擦除重写，其余未通入电流的单元中的信息仍然保留。不过，要完成这种改写手续常需要较复杂的设备。

美国 Nitron 公司供应的 NC7051 芯片是 EAROM 中有代表性的一种。这种 1K 位存贮片几乎可以改写内容 100 万次，改写电压 20～30 V，改写程序的时间约需 0.1～0.5 秒。信息保留时间可达 10 年，存取时间 1～5μs。从这里可以看出，虽然 EAROM 使用方便，但存取时间太慢，目前还不适宜于在微型机中应用。现在正在发展高密度、高速度的 EAROM 技术。

§10-3　随机存取存贮器

随机存取存贮器 RAM 又称为读写存贮器。它在一片数平方毫米的硅片上集成几百、几千个存贮单元，并将它们按矩阵形式排列，通常采用重合选择方式由 X(行)、Y (列)地址译码器来选择某一存贮单元进行信息的写入或读出，其结构如图 10-17 所示。

图中 A_{11}～A_0 是地址输入线，其数目取决于存贮器的存贮容量。现为 12 条地址输入线，

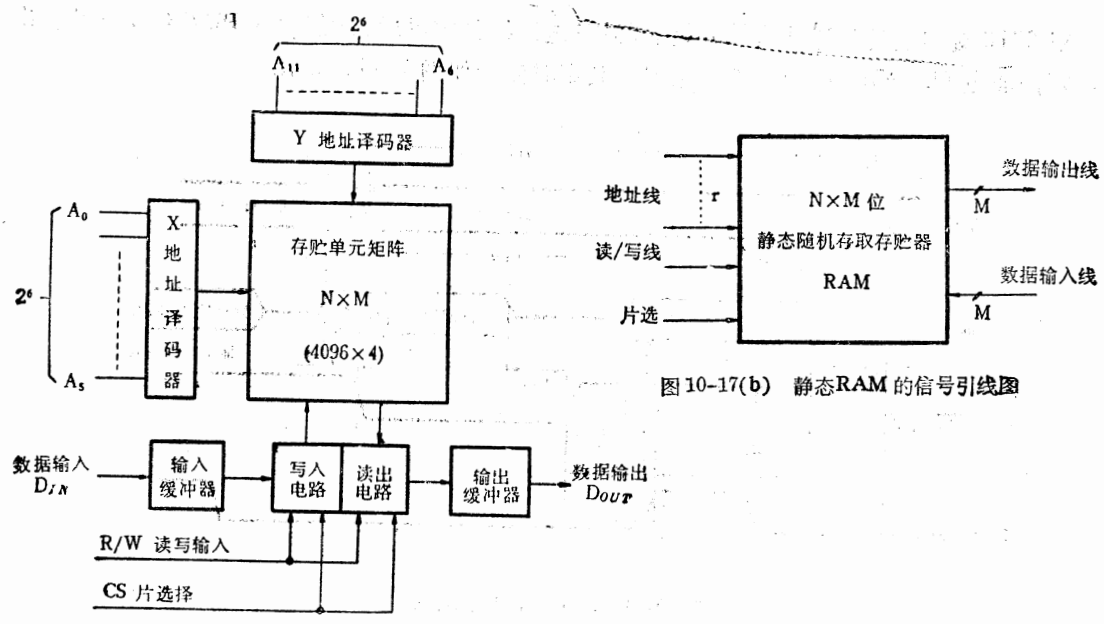


图 10-17(a)　静态 RAM 的内部结构框图

图 10-17(b)　静态RAM 的信号引线图

其容量为 $2^{12}=4096$，即 4 K。片选 (CS) 信号用作控制整个芯片是否允许进行读出或写入操作。读写(R/W)线用以传送读、写的命令。数据输入(D_{IN})和数据输出(D_{OUT})线用以传送输入和输出数据，其数目取决于存贮器的位数。若字长为 4 位，则 D_{IN} 和 D_{OUT} 线分别有 4 条。

MOS 型的 RAM 按其存贮单元的结构可分为静态 RAM 和动态 RAM。一般小容量存贮器常用静态 RAM，大容量存贮器常用动态 RAM。

一、静态 RAM

静态 RAM 采用触发器构成存贮单元，每个单元存贮信息的一个数位。每个静态存贮单元一般约需 6～8 个晶体管。静态 RAM 的信号线一般包括地址线、数据线、读/写线和片选线，图 10-17 (b)表示了某静态 RAM 的信号引线。下面着重说明静态 RAM 存贮单元的基本结构。

1. 静态 RAM 的存贮单元

图 10-18 所示是典型的静态 RAM 存贮单元。图中 T_1、T_2 组成触发器，是存贮单元的基础部分。T_3、T_4 分别是 T_1、T_2 的负载管。T_5、T_6 与 T_7、T_8 用作开关管。它们分别由X地址选择线和 Y 地址选择线控制。

当读出时，由X地址选择线与 Y 地址选择线重合而选中某一单元，即在两地址选择线上同时出现控制脉冲时，使 T_5～T_8 都处于导通状态，于是触发器的状态就经过 T_6、T_8 以及读出放大器传送至输出端。当写入时，除 X、Y 地址选择线重合而选中某一单元外。还需根据所要写入的“1”信息或“0”信息，相应地在写入端 3 加上适当电位，经 T_5、T_7 和 T_6、T_8 送至触发器，将触发器置成所需要的状态。

读或写操作是由 CPU 发出的读或写信号来控制的，读操作不改变触发器的状态。

2. 静态 RAM 的实例——Intel 2114

Intel 2114 是一种 4096 位(即 1024×4 位)静态随机存贮器，采用 N-沟硅栅 MOS 工艺。其

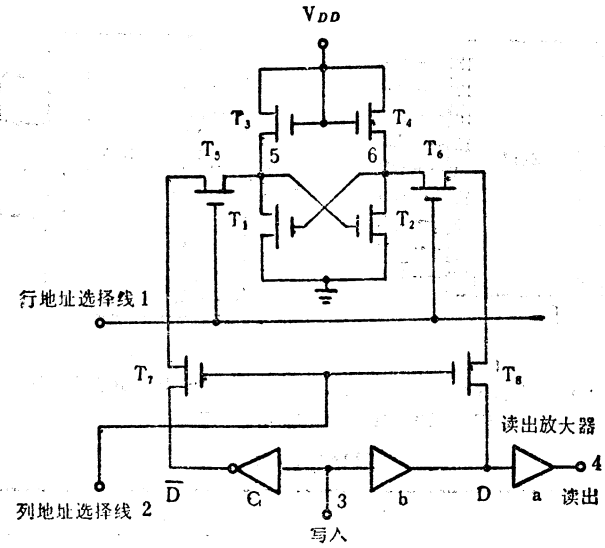


图 10-18　典型的 MOS 静态存贮单元

内部结构框图见图 10-19。芯片的引线和逻辑符号如图 10-20(a)、(b)所示。

2114 的存贮容量为 1 K×4 位，故芯片上共 4096 个六管存贮单元，它们排列成 64×64 的矩阵。因为有 1 K 字，故地址线需要 10 条即 A_9～A_0。其中 6 条即 A_8～A_3 用于行译码，产生 64 条行选择线。4 条 A_0、A_1、A_2、A_9 用于列译码，以产生 16 条列选择线，每条线同时接至 4 位。

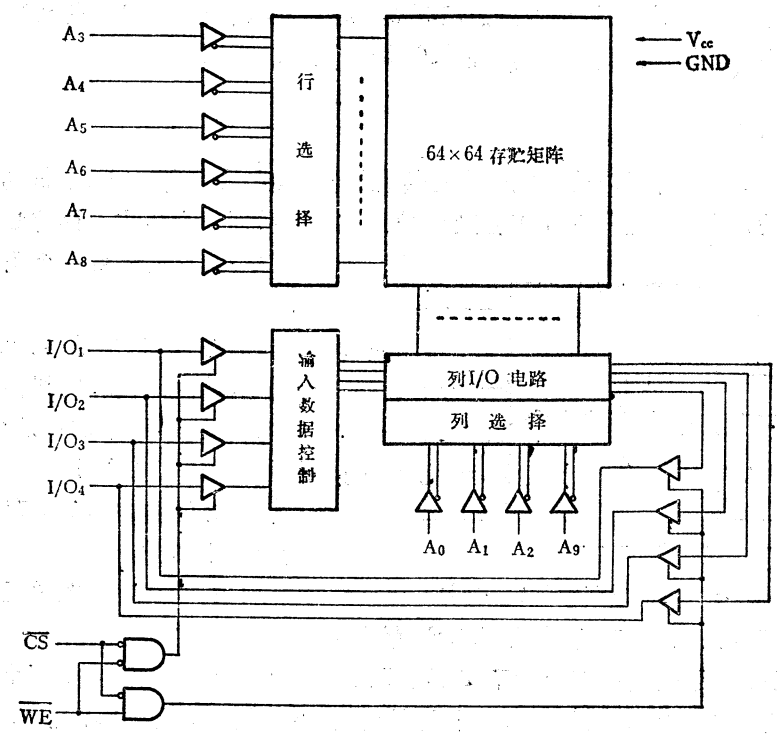
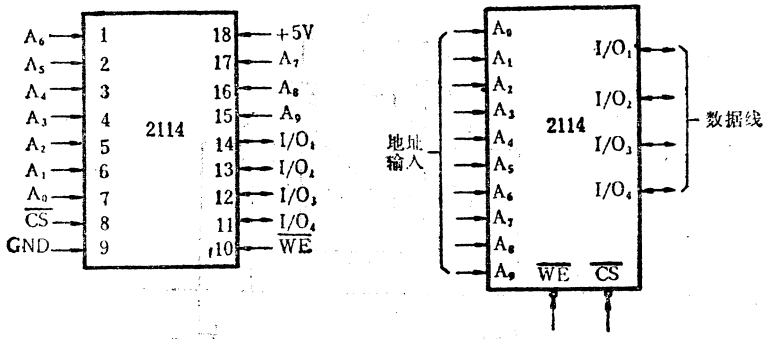


图 10-19　2114 的内部结构方框图



(a) 2114 引线图　　(b) 2114 逻辑符号图

图 10-20　2114 的芯片引线及逻辑符号

2114 引线名

$A_9 \sim A_0$	地　址　输　入
$\overline{WE}$	写允许
$\overline{CS}$	片选
$I/O_4 \sim I/O_1$	数据输入/输出

2114 的数据为 4 位，故数据线需要 4 条，即 $I/O_4 \sim I/O_1$。2114 内部数据通过 I/O 电路以及输入和输出的三态门与数据总线相连。由片选信号 $\overline{CS}$ 和写允许信号 $\overline{WE}$ 一起控制这些三态门。当 $\overline{CS}$ 有效(低电平)时，若 $\overline{WE}$ 为低电平，则输入三态门导通，信息沿数据总线写入存贮器；若 $\overline{WE}$ 为高电平时，则输出的三态门开放，从存贮器读出的信息送往数据总线。

关于 2114 读、写周期的时序，放在 §10-4 中作为实例分析。

二、动态 RAM

动态 RAM 是一种以电荷形式来存贮信息的器件，它具有高集成度、低功耗、快速和廉价的优点。但是，由于动态 RAM 中存贮的信息在几毫秒后便会消失，因此必须周期性地刷新(再生)，这就需要一个附加的刷新逻辑电路。

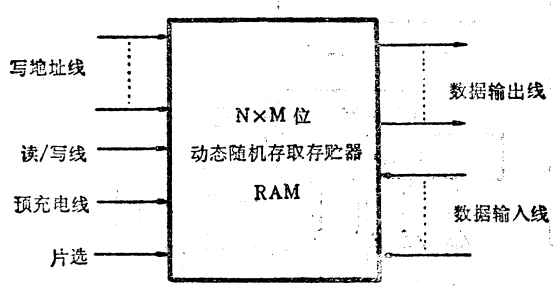


图 10-21　典型的 N×M 动态 RAM

动态 RAM 的信号线除了与静态 RAM 一样有地址线、读/写线、片选和数据输入、输出线外，还要附加一条“预充电”引线，其引线图如图 10-21 所示。

常见的动态 RAM 存贮单元的内部结构有以下两种：

1. 三管动态存贮单元

图 10-22 所示的是三管动态 RAM 存贮单元。

由图可见，除了需要用三个管子 T、T_2、T_3 做基本单元外，还需要第四个管子 T_4 (一列公用)，以控制对输出电容 C_D 预充电。对 T_2 管的栅-衬底的分布电容 C_g 充电，就相当于在该位存入了信息“1”。要读这一位时，必须先在“预充电”线上加一脉冲，使 T_4 导通，于是输出寄生电容 C_D 就预充电至 V_{DD}，然后启动读选择线，使 T_3 导通，给 T_2 提供一电压。若该位为

"0"(即 C_g 已放电，或未充电)，则 T_2 截止，C_D 上的电荷仍保留，读数据线上为"1"电平；若该位为"1"(C_g 已充电)，则 T_2 导通，C_D 将通过 T_3、T_2 放电，使读数据线上为"0"电平。即输出信息实际上恰好与所存信息反相，这只要经过读出放大器再反相就可以正确输出存贮信息了。

当写操作时，在写数据线上加一要写入信息的相应电平，然后在写选择线上通以脉冲，使 T_1 导通，则 C_g 将充电到写数据线上的值。

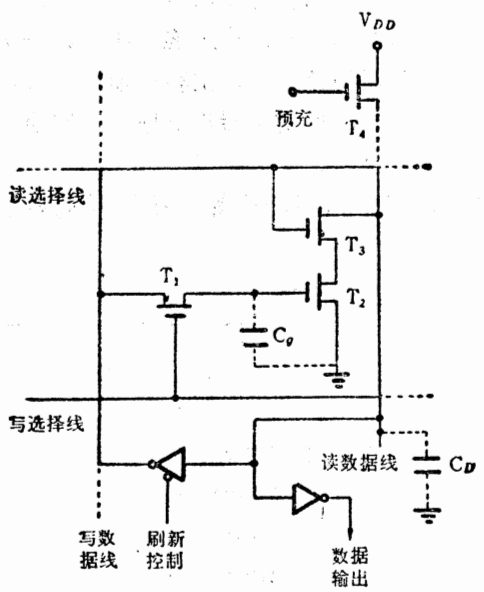


图 10-22　三管动态 RAM 存贮单元

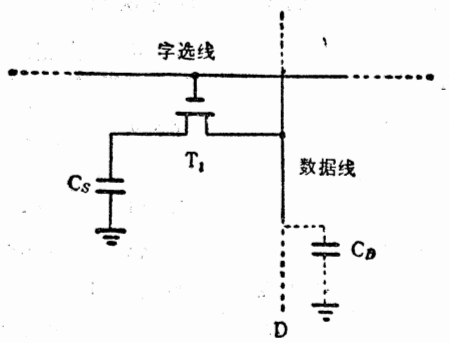


图 10-23　单管动态 RAM 存贮单元

动态 RAM 电路需要较高的输入阻抗，以防止 C_g 快速放电，所以通常这些电路都采用 MOS 工艺。采用 MOS 器件需要周期性地刷新存贮单元（即对 C_g 再充电）。刷新的时间间隔是温度的函数，在 0～55℃ 范围内，一般为 1～3 毫秒。

2．单管动态 RAM 存贮单元

单管动态 RAM 存贮单元的内部结构如图 10-23 所示。

它是由一个管子 T_1 和一个电容 C_s 构成的。写入时，字选择线为"1"，T_1 管导通。写入信号由位线(数据线)存入电容 C_s 中，即使得 C_s 充电到位线的电平("1"或"0")，此时，如果字选择线恢复为"0"电平，则 T_1 管截止，而 C_s 仍将保持已充电的电平，这就是写入过程。

在读出时，字选择线为"1"，使 T_1 管导通，存贮在电容 C_s 上的电荷通过 T_1 输出到数据线上，通过读出放大器即可得到存贮信息。

为了节省面积，这种单管存贮单元的电容 C_s 一般都比数据线上的分布电容 C_D 小得多，故在位线上只能得到微弱的读出信号，且读"1"和"0"时，数据线上的差别很小。因此，必须采用放大器放大后再输出。这样，外围电路就显得复杂化了。由于 $C_s \ll C_D$，所以每次读出后，存贮内容就被破坏，要保存原先的信息，必须采取恢复措施，即读出后再重写。

3．动态 RAM 的刷新(再生)

一种动态 RAM 的刷新电路如图 10-24 所示。

刷新过程是这样的：先将要刷新单元的信息读出，并将其信息传送给写数据线，然后在写选择线上通以写选择脉冲，则所读出的信息就可写回到该单元中。设图中 K 列的一位被读出(对照图 10-22)，若该位为"1"，则读数据线上为"0"(反相)，这使 T_a 截止。若 P 端和控制端均为"1"电平，那么写数据线将处于 V_{DD} 电位("1"态)。这时，给写选择线加一脉冲，电容 C_g 便

再充电。读者可验证，如 C_s 处于已放电状态，那么它也将保持这一状态。

每一列都有一刷新放大器，故一行中的所有单元都可在一周期内同时刷新。在刷新操作期间，必须将行地址送至译码器。由于一行的所有元件是同时刷新的，故没有必要再提供列地址。存贮器所有各行必须在 2 毫秒内刷新一次，可以一周期刷新一行，连续逐行刷新，一次完成。也可以分散同各正常读/写周期一起进行。在正常读/写周期期间，由存贮器地址寄存器提供地址。在刷新周期期间，由地址多路转换器转接至刷新地址发生器提供地址。刷新地址发生器还要有加 1 计数的功能，以便更换行地址，不断进行刷新。

在微型计算机中，一般要使用多种存贮器如 RAM、ROM、EPROM 等。表 10-2 列出了常用存贮器芯片的型号及性能说明。

表 10-2　微型计算机常用的 RAM、ROM、EPROM

种　类	型　号	字×位	存取时间 ns(max)	工　艺	电　源 V	引　线	特　点
静态 RAM	2101A	256×4	350～450	N-MOS	+5	DIP-22	可以用备用电池
	5101	256×4	450～800	C-MOS	+5	DIP-22	
	2111A	256×4	250～450	N-MOS	+5	DIP-18	
	2102A	1024×1	250～450	N-MOS	+5	DIP-16	
	2114	1024×4	200～450	N-MOS	+5	DIP-18	
	2115	1024×1	45～95	N-MOS	+5	DIP-16	
	2147	4096×1	60～90	N-MOS	+5	DIP-18	
	2148H	1024×4	45～70	H-MOS	+5	DIP-18	
动态 RAM	2104A	4096×1	150～300	N-MOS	±5, +12	DIP-16	
	2107B	4096×1	200～450	N-MOS	+5, +12	DIP-22	
	2108A	8192×1	200～300	N-MOS	±5, +12	DIP-16	
	2116	16K×1	200～300	N-MOS	±5, +12	DIP-16	
	2118	16K×1	100～150			DIP-16	
	2164A	64K×1	150～200			DIP-16	
EPROM	2708	1024×8	350～450	N-MOS	±5, +12	DIP-24	紫外线擦除
	2716	2048×8	300～450	N-MOS	+5	DIP-24	
	2732A	4096×8	200～450	N-MOS	+5	DIP-24	
	2764	8K×8	200～450			DIP-28	
	27128	16K×8	250～450			DIP-28	
	27256	32K×8	250～450			DIP-28	
	μPD458	1024×8	450	N-MOS	+5, +12	DIP-28	电可改写
ROM	8308	1024×8	450	N-MOS	±5, +12	DIP-24	与 2708 有互换性
	8316	2048×8	850	N-MOS	+5	DIP-24	与 2716 有互换性
	8316E	2048×8	450	N-MOS	+5	DIP-24	

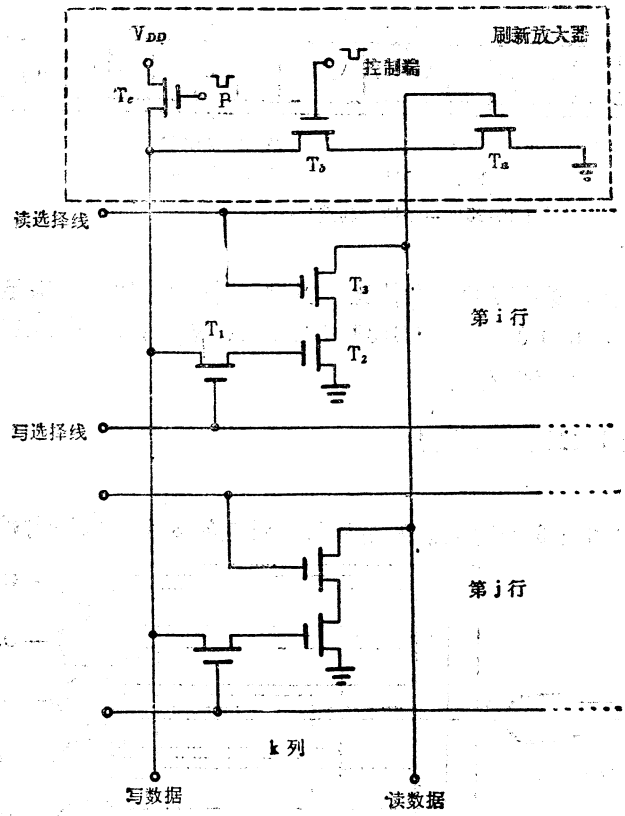


图 10-24　动态 RAM 的刷新电路

§10-4　存贮器与 CPU 的连接

一、存贮器的寻址方法

微型计算机系统的地址总线是用来选择存贮器内某一存贮单元（或 I/O 部件内某一寄存器）的。要完成这种功能，必须进行两种选择：一是必须选择出这一存贮器芯片（或 I/O 接口芯片），称为片选；二是必须选择出该芯片中的某一存贮单元（或某一寄存器），称为字选。通常，这些任务由地址译码电路来完成。地址译码电路的任务就是当微处理器发出一个需要访问的存贮器的地址之后，通过地址译码电路能很快地寻找到该存贮器芯片及相应的存贮单元。

下面介绍两种存贮器的寻址方法。

1．线性选择

在线性选择中，地址总线高位中的某一条线可直接用来作为选择某一存贮器的片选信号线。如图 10-25 所示。

这种寻址方法常用于规模较小的微型计算机系统。它的优点是不需要采用地址译码器，缺点是可寻址的器件数目受到很大的限制，而且地址空间可能是不连续的，这会给编程带来困难。设某系统中有一个 2KB ROM 和一个 2KB RAM 的存贮器。我们知道，要能够在该存贮器中选择 2K 个地址，必须用 11 条地址线来进行字选，其它高 5 位可用于芯片选择。其中 A_{11} 位用来选择 RAM 或 ROM 芯片剩下高 4 位可用来选择 I/O 端口。

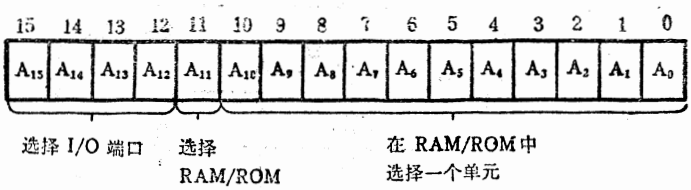


图 10-25　一条地址线选择一个器件的线性选择

2. 地址译码选择

当微型计算机系统需要的存贮器器件比较多时，线性选择便不能满足要求，必须用地址译码器。常用的译码器有 Intel 8205、74LS138、74LS 156 等。下面介绍 8205(74 LS 138 的引线与 8205 的引线相容)的功能和应用。

二、8205"3 中取 1"二进制译码器

1. 8205 的结构

每一片 8205 译码器有 3 个地址输入端和与其对应的 8 个译码输出端。另外还有 3 个允

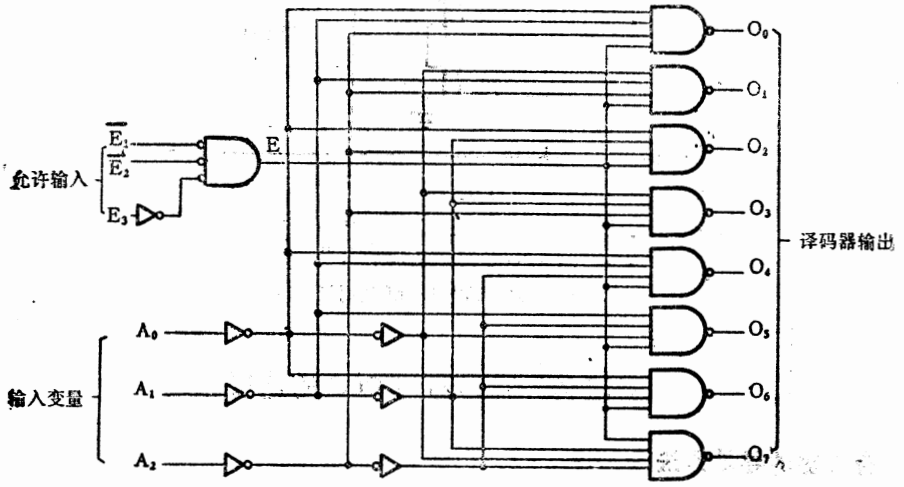


图 10-26　8205 内部线路图

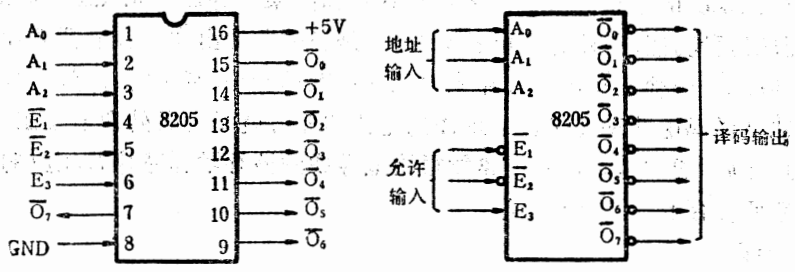


图 10-27　Intel 8205 的引线及逻辑符号图

引线名

$A_2 \sim A_0$	地　址　输　入
$\overline{E}_1\overline{E}_2E_3$	允许输入
$\overline{O}_7 \sim \overline{O}_0$	译码器输出
V_{CC}, GND	电源(+5V)，地

许工作输入端，一个电源及一个接地端。它采用高速 STTL 工艺并能同 TTL 逻辑电路直接相容。它的外部引线 16 条，采用双列直插式陶瓷或塑料封装。

8205 内部线路图如图 10-26 所示，其引线及逻辑符号如图 10-27 所示。

从图 10-26 可知，8205 是一个具有允许工作输入端的 3-8 译码器，只有当 $\overline{E}_1 \cdot \overline{E}_2 \cdot E_3 = 1$ 时，才能启动译码器，产生有效的低电平输出信号。

2. 8205 的功能

8205 是一个“8 中取 1”的二进制译码器，又称 3-8 译码器，可用作存贮器或 I/O 端口的地址译码器。8205 接受 3 位二进制码，用控制这些输入地址的方法来产生一个对应于该输入码值的唯一的输出信号。例如，若地址输入线 A_0、A_1、A_2 上输入二进制码 101，当器件允许时，则在输出端 O_5 上出现一个有效的低电平信号，而其它的输出端都为高电平，因此这一有效的输出信号是“唯一”的。这个译码器输出可作为芯片 ROM 或 RAM 或 I/O 端口的片选信号。对所有其它的输入地址来说，译码器输出也将以同样的方式按表 10-3 所示的真值表进行工作。

表 10-3　8205 译码器真值表

地址			允许工作			输出							
A_0	A_1	A_2	$\overline{E}_1$	$\overline{E}_2$	E_3	0	1	2	3	4	5	6	7
L	L	L	L	L	H	L	H	H	H	H	H	H	H
H	L	L	L	L	H	H	L	H	H	H	H	H	H
L	H	L	L	L	H	H	H	L	H	H	H	H	H
H	H	L	L	L	H	H	H	H	L	H	H	H	H
L	L	H	L	L	H	H	H	H	H	L	H	H	H
H	L	H	L	L	H	H	H	H	H	H	L	H	H
L	H	H	L	L	H	H	H	H	H	H	H	L	H
H	H	H	L	L	H	H	H	H	H	H	H	H	L
×	×	×	L	L	L	H	H	H	H	H	H	H	H
×	×	×	H	L	L	H	H	H	H	H	H	H	H
×	×	×	L	H	L	H	H	H	H	H	H	H	H
×	×	×	H	H	L	H	H	H	H	H	H	H	H
×	×	×	H	L	H	H	H	H	H	H	H	H	H
×	×	×	L	H	H	H	H	H	H	H	H	H	H
×	×	×	H	H	H	H	H	H	H	H	H	H	H

注：表中“L”表示低电平，“H”表示高电平。

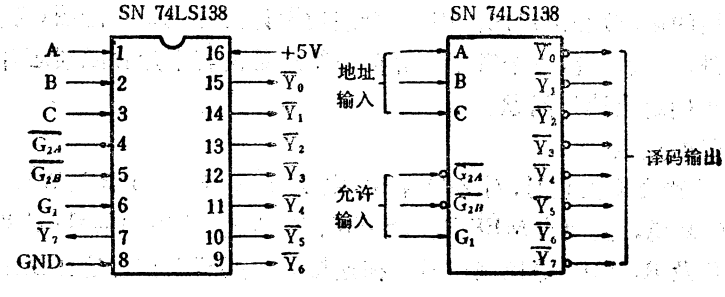


图 10-28　74LS138 的引线及逻辑符号图

3．74 LS 138 3-8 译码器

74 LS 138 3-8 译码器与 Intel 8205 的性能、引线都相同，图 10-28 示出了它的引线及逻辑符号图，其内部结构框图见图 10-26 所示。

三、RAM 与 CPU 的连接

在微型计算机中，CPU 对存贮器进行读写操作，首先要由地址总线给出地址信号，然后要发出相应的读或写的控制信号，最后才能在数据总线上进行数据交换。RAM 与 CPU 的连接主要有以下三个部分：

(1) 地址线的连接；

(2) 数据线的连接；

(3) 控制线的连接。

1．RAM 与 CPU 连接时应考虑的问题

(1) CPU 总线的负载能力

通常，CPU 总线的直流负载能力为能带一个标准 TTL 负载。由于 MOS 存贮器的直流负载很小，主要的负载是电容负载，故在小型系统中，CPU 可以与存贮器直接相连。对于较大的存贮系统，就应该考虑增加总线的驱动能力。常用加缓冲器或总线驱动器等方法来实现。

(2) CPU 的时序与存贮器的存取速度之间的配合问题

① 最大存取时间

从地址线开始有地址信息到数据总线上出现有效的数据信息，这一最大的延迟时间称为存贮器的最大存取时间。在选用存贮器时，应尽可能选用最大存取时间小于 CPU 系统存取时间的存贮器。否则就要考虑是否需要 T_W 时态，以及如何实现的问题。

② 某些存贮器与 CPU 相连时，由于它们的电平不同而不能直接相连。这时要加缓冲器，它可以变换电平，增加驱动电流。但增加缓冲器后，信息通过时会产生延迟。我们希望缓冲器的延迟时间越短越好。为此，大多数微型机采用 TTL 缓冲器，它的延迟时间为 20～50 ns。

(3) 存贮器的地址分配和选片问题

通常，微型机的内存分为 RAM 和ROM 两大部分，而 RAM 又分为系统区(即机器监控程序或操作系统应用的区域)和用户区。因而，合理地设计内存地址分配图(简称内存配址图)是十分重要的。另外，一般地说，一个微型计算机系统总是由若干片存贮器芯片组成一个存贮系统的。这就要求正确解决选片问题。

(4) 控制信号的连接

CPU 在与存贮器交换信息时，对 Z80 来说，主要有 $\overline{M_1}$、$\overline{MREQ}$、$\overline{RD}$、$\overline{WR}$ 以及 $\overline{WAIT}$ 等控制信号；对 8080A 来说，主要有 $\overline{MEMR}$、$\overline{MEMW}$、READY 等控制信号；对 8085A 来说，主要有 IO/$\overline{M}$、$\overline{RD}$、$\overline{WR}$ 及 READY 等控制信号。在连接时，要特别注意这些控制信号的正确接法。

2．静态 RAM 与 CPU 的连接

如果用 Intel 2114 1K × 4 位的 RAM 芯片构成一个 8KB 的 RAM 系统(存贮空间为0000～1FFFH)，若 CPU 为 Z80，并采用 Intel 8205(或 74LS 138)3-8 译码器对 RAM 进行选片，其连接如图 10-29 所示。

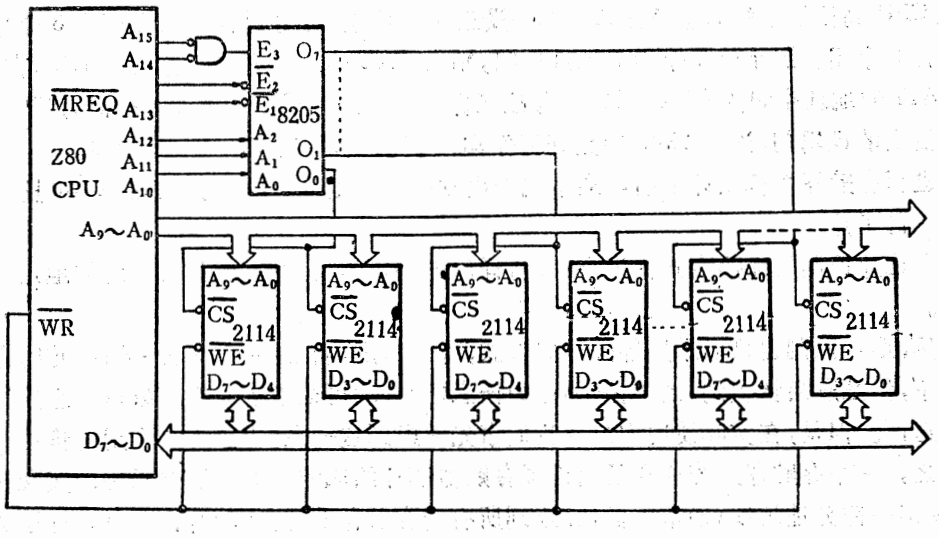
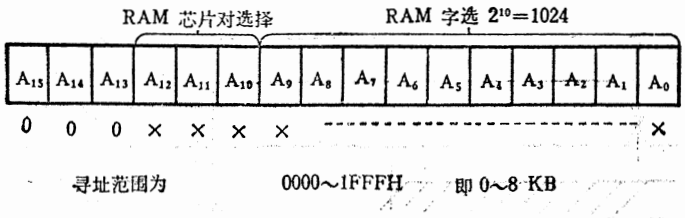


图 10-29　8KB RAM 与 CPU 连接框图

每片 2114 为 1K×4 位，故 8KB RAM 共需 16 片。每片有 10 条地址线，直接连至 CPU 的地址总线 A_9～A_0，可寻址 1K 个存贮单元，这就是解决字选问题。系统总共为 8KB RAM，



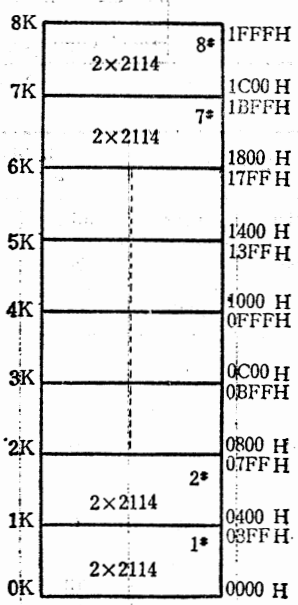


图 10-30　存贮空间地址分配图

可看成 8 组。如何正确地区别和选择这不同的 8 组，就得解决片选问题。现采用 8205 3-8 译码器从 CPU 高位地址线 A_{10}、A_{11}、A_{12} 上接受三条地址输入，经过译码后可产生 8 个片选信号用来选择 8 个组。至于这 8 个组的地址空间，则由 8205 的三个允许工作端来确定。在本例中，要求 RAM 的地址空间为 0000～1FFFH，故应保证高位地址线 A_{13}、A_{14}、A_{15} 都为"0"电平，然后经过恰当的逻辑门，连至 8205 的允许工作端 E_1、E_2、E_3，使得 $\overline{E_1} \cdot \overline{E_2} \cdot E_3 = 1$。

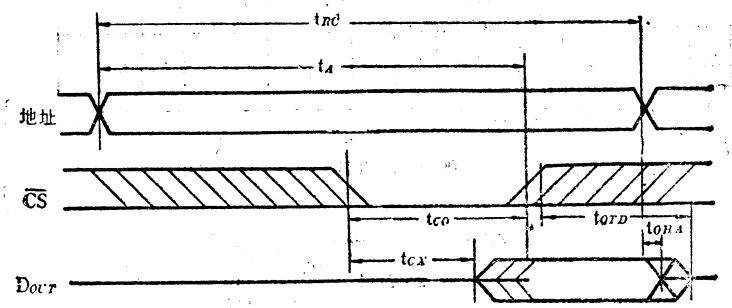
在进行存贮器系统设计之前，通常先画出存贮空间地址分配图。如图 10-30 所示。这样可使人一目了然。

以上选片控制的译码方式称为全译码，译码电路略为复杂些，但是每一组的地址都是唯一确定的。

当然，选片控制也可以采用部分译码方式，即不考虑高位地址线 A_{15}～A_{13}，这时译码电路比较简单，但是应当认识到，当 A_{15}～A_{13} 为任意值时，均可选中这 8 组。故每一组的地址都有很大重迭区。但是，在实际使用中只要我们事先考虑到这一点，仍然有其实用的价值。

总之，一个存贮器系统通常是由许多存贮器芯片组成的。为了能够正确地实现存贮器寻址，必须用一部分地址线（通常是低位）连到所有的存贮器芯片，实现片内寻址，即字选。另外一些地址线（通常是高位）或单独使用（线选），或组成译码方式（部分译码或全译码），其输出作为芯片的片选信号（实际上的片选信号还要考虑到 CPU 的控制信号，例如 Z80 的 $\overline{MREQ}$ 等），以实现组的寻址。在连接时应注意它们的地址分布和重迭区。

3. CPU 的时序和存贮器的存取速度之间的配合



符号	参数	2114-2		2114-3 2114L-3		2114 2114L		单位
		min	max	min	max	min	max	
t_{RC}	读周期时间	200		300		450		ns
t_A	读取时间		200		300		450	ns
t_{CO}	片选有效到输出稳定时间（数据有效时间）		70		100		100	ns
t_{CX}	片选有效到数据出现的时间	0		0		0		ns
t_{OTD}	从断开片选到输出变为三态的时间	0	40	0	80	0	100	ns
t_{OHA}	地址改变后的持续时间	10		10		10		ns

图 10-31　2114 读周期定时波形图

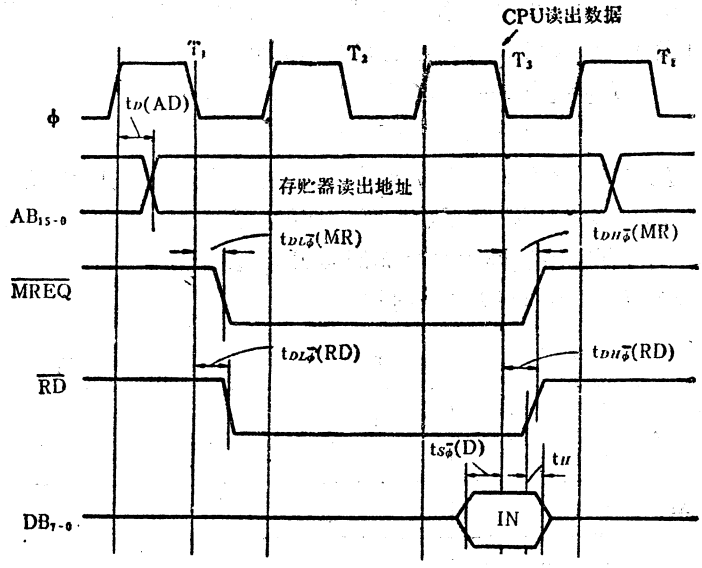
下面我们以 Intel 2114 的读、写周期与 CPU 的时序配合为例，详细分析一下存贮器与 CPU 连接中的这一十分重要而又有实用价值的问题。

(1) 读周期

Intel 2114 读周期定时波形图及其主要参数如图 10-31 所示。

由图 10-31 可知，要实现 2114 读操作，首先要求在读周期 t_{RC} 时间内，地址码要稳定。读周期是指该芯片进行两次连续的读操作必须间隔的时间。其次，在数据输出前，应保证片选信号 $\overline{CS}$ 有效(低电平)，以打开输出三态门，同时还应使 $\overline{WE}$ 为高电平，以便进行读操作。

从图 10-31 还知，自地址码送至 2114 起，经过一段时间 t_A 后，数据就可能输出，故称 t_A 为读取时间。通常，$t_A \leqslant t_{RC}$。但是，数据能否送往外部数据总线，取决于片选信号 $\overline{CS}$ 是否有效。只有 $\overline{CS}$ 有效，开放三态门，输出数据经输出放大器延时才能送到数据总线，从 $\overline{CS}$ 有效到数据在数据线上稳定，需要经过 t_{CO} 时间，称之为数据有效时间。因此，对 2114 来说，若从地址码有效时间算起，要求经过 t_A 时间读出的数据能在外部数据总线上稳定，那末，就得要求 $\overline{CS}$ 信号至迟在地址有效后的 $(t_A - t_{CO})$ 之内有效，否则，即使经过了 t_A 时间，因为三态门仍关闭，数据仍然不能送往外部数据总线。换言之，在设计电路时，若已选定芯片，则可根据 t_A 和 t_{CO}



符　号	参　数	min	max	单　位
t_D(AD)	地址输出延迟		145	ns
$t_{DL\phi}$(MR)	从时钟的下降沿到 MREQ 变"低"的延迟		100	ns
$t_{DH\phi}$(MR)	从时钟的下降沿到 $\overline{MREQ}$ 变"高"的延迟		100	ns
$t_{DL\phi}$(RD)	从时钟的下降沿到 $\overline{RD}$ 变"低"的延时		130	ns
$t_{DH\phi}$(RD)	从时钟的下降沿到 $\overline{RD}$ 变"高"的延迟		110	ns
$t_{s\phi}$(D)	数据建立时间到时钟的下降沿	60		ns
t_H	输入保持时间	0		ns

图 10-32　Z80 CPU 存贮器读周期时序波形图

两个时间参数倒推上去，确定 $\overline{CS}$ 信号应在何时有效。

此外，图 10-31 中，还有两个保持时间：t_{OTD} 表示片选信号失效后，数据仍能保持的时间；t_{OHA} 表示地址已经改变后，数据还能保持的时间。

现设某系统存贮器选用 2114-2，与 Z80 CPU 连接，下面分析一下它们之间的时序配合问题。

图 10-32 是 Z80 CPU 的存贮器读周期的时序波形图及其主要参数。

① 从地址有效到数据有效时间的配合

若 Z80 CPU 的时钟频率为 2.5 MHz，即时钟周期 T = 400 ns，从 T_1 时态经过 $t_D(AD)$后，地址信息有效，到 T_3 下降沿，CPU 采样数据信息，其间的时间间隔为：T_1 时态尚剩下的时间 $200+[200-t_D(AD)]$(ns)，整个 T_2 时态以及 T_3 的半个时态。但它要求数据信息在 T_3 下降沿前，$t_{S\overline{\phi}}(D)$时间有效。故总的时间间隔为：

$$T_{RD}=200+[200-t_D(AD)]+400+200-t_{S\overline{\phi}}(D)$$
$$=200+[200-145]+400+200-60=795\ ns$$

而 2114－2 的读取时间 $t_A=200$ ns，故 $T_{RD}>t_A$。

② 从 $\overline{MREQ}$ 有效到数据有效时间的配合

通常，以 $\overline{MREQ}$ 与地址信息配合作为片选信号，故 $\overline{MREQ}$ 有效，即为 $\overline{CS}$ 有效时间。从 $\overline{MREQ}$ 有效到 T_3 下降沿前，输出数据稳定之间的时间间隔为：

$$T_{CS}=[200-t_{DL\overline{\phi}}(MR)]+400+200-t_{S\overline{\phi}}(D)$$
$$=200-100+400+200-60=640\ ns$$

而 2114-2 所提供的从 $\overline{MREQ}$（即 $\overline{CS}$）有效到数据有效时间 $t_{CO}=70$ ns，故 $T_{CS}>t_{CO}$。

因此，2114-2 完全能满足 Z80 CPU 的存贮器读周期的时序要求。

下面再来分析一下，存贮器实际上从什么时刻开始能把数据输出。

图 10-33 表示 Z80 CPU 与 2114-2 的时序配合。

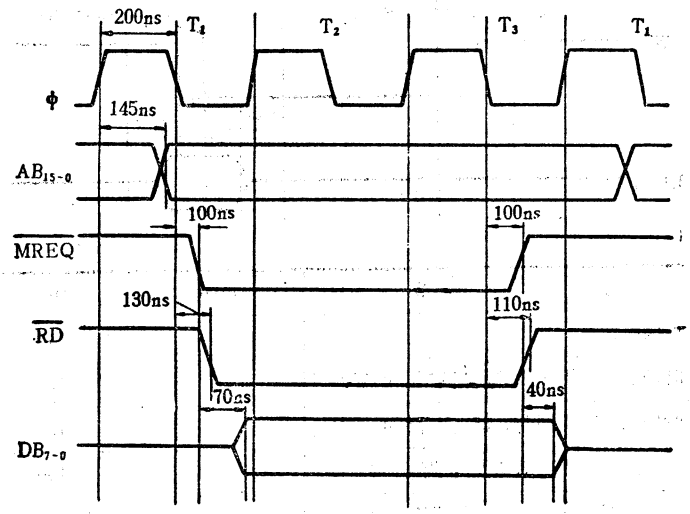
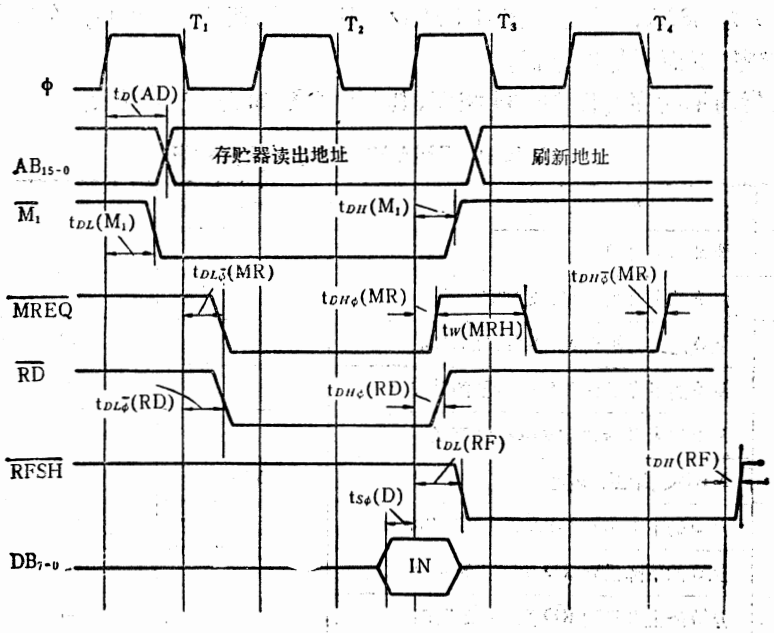


图 10-33　Z80 CPU 与 2114-2 的时序配合

从图中看出，从 T_1 上升沿起，经过 $t_D(AD)$ 和 t_A，即自 T_1 上升沿起经过 145 + 200 = 345 ns 以后，所选择的单元的信息从该存贮单元读出至存贮器内部数据总线上。但是，能否输出还要

取决于片选信号，现以 $\overline{MREQ}$ 与地址译码输出作为片选信号，则自 T_1 上升沿起经过 T/2 + $t_{DL\overline{\phi}}(MR) + t_{CO}$ 即 200 + 100 + 70 = 370 ns 以后，数据能通过三态缓冲门稳定输出到 CPU 的数据总线上。现在是后一个时间长，故自 T_1 上升沿起经过 370 ns（即在 T_1 时态末尾），选中的单元的数据可能读出到 CPU 的数据总线上了。而 CPU 是在 T_3 时态的下降沿时才采样数据总线，因而完全能满足要求，而不需要插入 T_W 时态，可把 Z80 CPU 的 $\overline{WAIT}$ 线始终接至高电平。

Z80 对存贮器要求较高的是取指周期，此时 CPU 是在 T_3 时态的 ϕ 上升沿采样数据线，其时序图及主要参数如图 10-34 所示。



符　号	参　数	min	max	单　位
$t_{D}(AD)$	地址输出延迟时间		145	ns
$t_{DL}(M_1)$	从 ϕ 的上升沿到 $\overline{M_1}$ 变“低”的时间		130	ns
$t_{DH}(M_1)$	从 ϕ 的上升沿到 $\overline{M_1}$ 变“高”的时间		130	ns
$t_{DL\overline{\phi}}(MR)$	从 ϕ 的下降沿到 $\overline{MREQ}$ 变“低”的时间		100	ns
$t_{DH\overline{\phi}}(MR)$	从 ϕ 的下降沿到 $\overline{MREQ}$ 变“高”的时间		100	ns
$t_{DH\phi}(MR)$	从 ϕ 的上升沿到 $\overline{MREQ}$ 变“高”的时间		100	ns
$t_{W}(MRH)$	$\overline{MREQ}$ 高电平宽度	170		ns
$t_{DL\overline{\phi}}(RD)$	从 ϕ 的下降沿到 $\overline{RD}$ 变“低”的时间		130	ns
$t_{DH\phi}(RD)$	从 ϕ 的上升沿到 $\overline{RD}$ 变“高”的时间		100	ns
$t_{DL}(RF)$	从 ϕ 的上升沿到 RF 变“低”的时间		180	ns
$t_{DH}(RF)$	从 ϕ 的上升沿到 RF 变“高”的时间		150	ns
$t_{S\phi}(D)$	从 ϕ 的上升沿到数据建立时间	50		ns

图 10-34　CPU 取指周期时序波形图

由于取指可视为一种特殊的存贮器读操作，故仍用 $\overline{MREQ}$ 与地址信息组成片选信号。从片选信号有效到 CPU 要求数据稳定(在 T_3 的前沿 $t_{S\phi}(D)$ 时刻)之间的时间间隔为：

$$T_{CS} = 400 + [200 - t_{DL\bar{\phi}}(MR)] - t_{S\phi}(D) = 450(ns)$$

故 $t_{CS} > t_{CO}$(2114-2 片选到输出稳定时间 70 ns)，仍然满足要求。

另外，从地址有效到 CPU 要求的数据稳定之间的时间间隔为：

$$T_{RD} = 400 + 200 + [200 - t_D(AD)] - t_{S\phi}(D) = 605(ns)$$

也大于 2114-2 的读取时间 $t_A = 200$ ns 的要求。

根据以上计算，2114-2 与 Z80 CPU 配合时，在时序上完全能满足要求。

但若 CPU 为 Z80A，其时钟频率 4 MHz，即时钟周期 T = 250 ns。在这种情况下 2114-2 能否满足要求呢？下面再加以分析。

Z80A 取指周期时序与图 10-34 相同，其主要参数如表 10-4 所示。

表 10-4 Z80A 取指周期时间参数表

符 号	参 数	min	max	单 位
$t_D(AD)$	地址延迟时间		110	ns
$t_{DL}(M_1)$	从 ϕ 的上升沿到 $\overline{M_1}$ 变“低”的时间		100	ns
$t_{DH}(M_1)$	从 ϕ 的上升沿到 $\overline{M_1}$ 变“高”的时间		100	ns
$t_{DL\bar{\phi}}(MR)$	从 ϕ 的下降沿到 MR 变“低”的时间		85	ns
$t_{DH\phi}(MR)$	从 ϕ 的上升沿到 $\overline{MREQ}$ 变“高”的时间		85	ns
$t_W(MRH)$	$\overline{MREQ}$ 高电平宽度	110		ns
$t_{DH\bar{\phi}}(MR)$	从 ϕ 的下降沿到 $\overline{MREQ}$ 变“高”的时间		85	ns
$t_{DL\bar{\phi}}(RD)$	从 ϕ 的下降沿到 $\overline{RD}$ 变“低”的时间		95	ns
$t_{DH\phi}(RD)$	从 ϕ 的上升沿到 $\overline{RD}$ 变“高”的时间		85	ns
$t_{DL}(RF)$	从时钟的上升沿到 $\overline{RFSH}$ 变“低”的延迟		130	ns
$t_{DH}(RF)$	从时钟的上升沿到 $\overline{RFSH}$ 变“高”的延迟		120	ns
$t_{S\phi}(D)$	在 M_1 周期数据建立与 ϕ 上升沿之间的时间	50		ns

从表中可知，这时从地址有效到 CPU 要求的数据稳定(T_3 上升沿前 $t_{S\phi}(D)$ 时刻)之间的时间间隔为：

$$T_{RD} = 250 + 125 + 125 - t_D(AD) - t_{S\phi}(D) = 340(ns)$$

故 $T_{RD} > t_A(= 200ns)$。

从 $\overline{MREQ}$ 有效到 CPU 要求数据稳定之间的时间间隔为：

$$T_{CS} = 250 + 125 - t_{DL\bar{\phi}}(MR) - t_{S\phi}(D) = 240\ ns$$

也仍然大于 2114-2 的 $t_{CO} = 70$ ns。

根据以上计算，2114-2 与 Z80A CPU 配合时，在时序上也完全可以满足要求。

如果选用常用的 2114 型 RAM，能否满足 Z80A CPU 高速存取的要求呢？下面加以分析。

2114 型RAM，其 $t_A = 450$ ns，$t_{CO} = 100$ ns。

在取指周期时，$T_{CS} = 240ns > t_{CO} = 100$ ns，

但 $T_{RD} = 340$ ns $< t_A = 450$ ns，故不能满足要求，这时应采用等待电路，插入一个 T_W。

但在存贮器读周期时，由于 CPU 是在 T_3 下降沿采样数据总线，故

$$T_{RD} = 250 + 125 + 125 - t_D(AD) - t_{S\phi}(D) = 465\ ns$$

即 $T_{RD} > t_A$，因此能够满足要求。

由上可知，若选用 2114 与 Z80A CPU 相配合，仅需要在取指周期(M_1 周期)，插入一个 T_W 时态。

在 M_1 周期插入一个 T_W 时态的电路，如图 10-35(a)所示。其工作过程如图 10-35(b)所示。

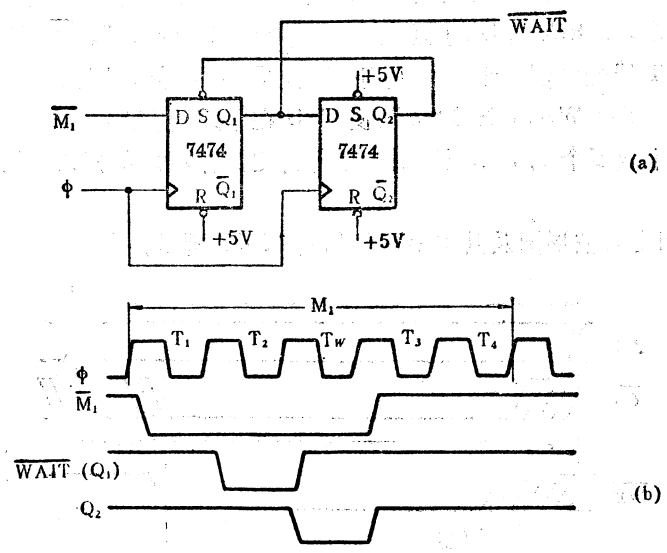


图 10-35　在 M_1 周期插入一个 T_W 的电路

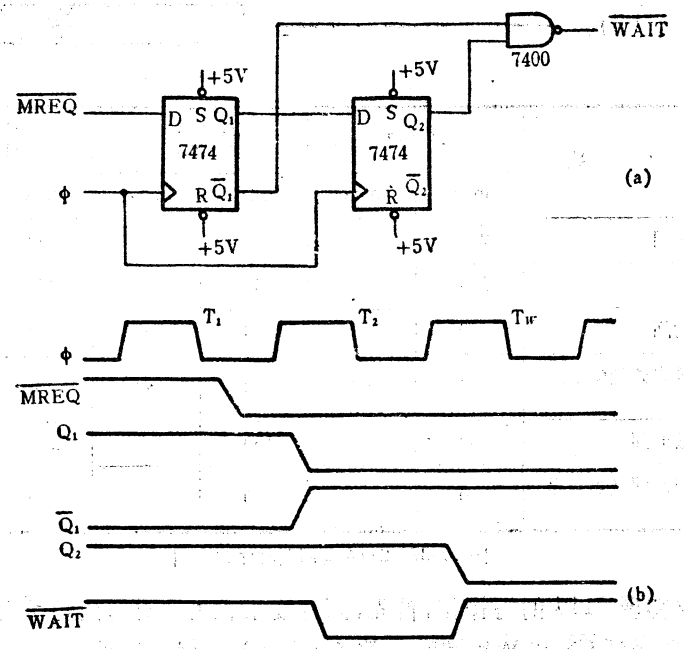


图 10-36　在存贮器读周期插入一个 T_W 的电路

现简要说明如下：

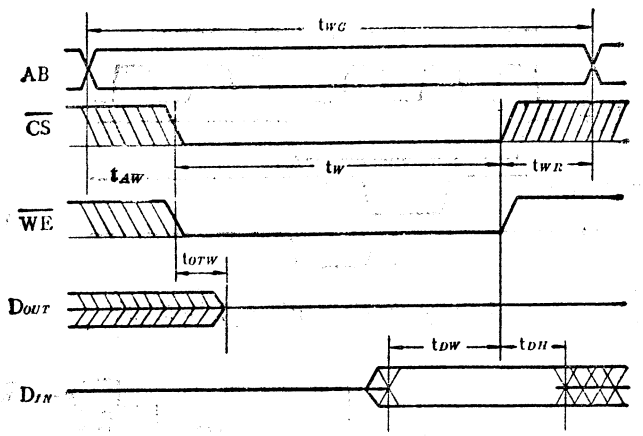
取指周期 T_1 上升沿使 $\overline{M_1}$ 有效，于是在 T_2 上升沿，使 Q_1 变“低，”即 $\overline{WAIT}$ 线变“低”（有效），故当 CPU 在 T_2 下降沿查询 $\overline{WAIT}$ 信号时，因其有效（低电平），即插入一个 T_W 时态。而在 T_W 上升沿，使 Q_2 变“低”，又将 Q_1 置为高电平。故在 T_W 下降沿，CPU 查询 $\overline{WAIT}$ 信号时，因其已变“高”，所以不再插入新的 T_W，而进入 T_3。从而实现插入一个 T_W 的要求。

如果需要在存贮器读周期时，也插入一个 T_W，则可用如图 10-36(a)所示的等待电路。此时，用 $\overline{MREQ}$ 代替 $\overline{M_1}$，$\overline{WAIT}$ 信号是用 $\overline{Q_1}$ 与 Q_2 的“与”来产生的。其工作过程如图 10-36(b)所示。在 T_1 下降沿，$\overline{MREQ}$ 有效，于是在 T_2 上升沿使 Q_1 变“低”，$\overline{Q_1}$ 变“高”。由于此时 Q_2 也为“高”，故 $\overline{WAIT}$ 线变“低”。于是在 T_2 下降沿，CPU 采样时，即插入一个 T_W。但在 T_W 上升沿，使 Q_2 变“低”，于是 $\overline{WAIT}$ 线变“高”，使 CPU 在 T_W 下降沿采样时转入 T_3。

如果要求插入两个或多个 T_W，则只要对上述等待电路作相应的修改即可。

(2) 写周期

Intel 2114 写周期定时波形图及其主要参数如图 10-37 所示。



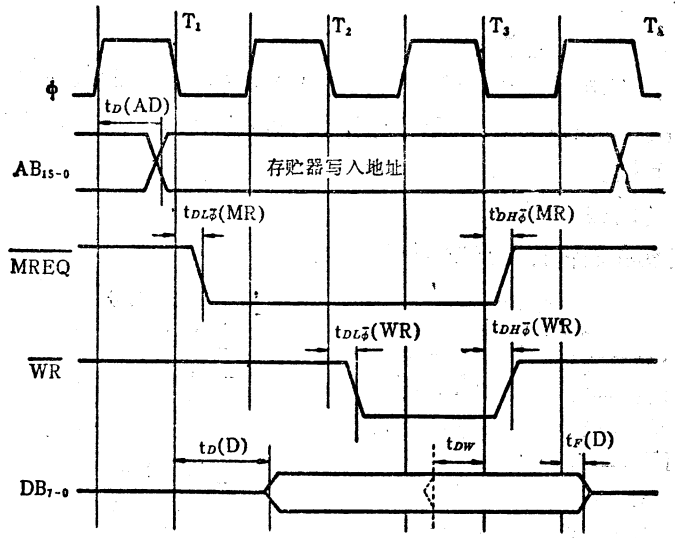
符号	参数	2114-2		2114-3 2114L$_3$		2114 2114L		单位
		min	max	min	max	min	max	
t_{WC}	写周期时间	200		300		450		ns
t_W	写时间	100		150		200		ns
t_{WR}	写恢复时间	0		0		0		ns
t_{OTW}	从写信号有效到输出三态时间	0	40	0	80	0	100	ns
t_{DW}	数据有效时间	100		150		200		ns
t_{DH}	数据保持时间（写信号无效后）	0		0		0		ns

图 10-37 2114 写周期时序波形图

由图可知，要实现 2114 的写操作，首先，同样要求在 t_{WC} 时间内地址码要稳定。其次，在数据写入时，必须保证 $\overline{CS}$ 和 $\overline{WE}$ 同时有效（低电平）。但在地址信息改变期间 $\overline{WE}$ 必须为“高”，否则可能造成误写入。

为了保证在 $\overline{WE}$ (和 $\overline{CS}$)变为无效前，将数据可靠地写入存贮器，被写数据必须在 t_{DW} 前有效并稳定。

下面我们分析 Z80、Z80A 与 2114-2、2114 连接时的写时序配合问题。Z80 CPU 的存贮器写周期时序及其主要时间参数如图 10-38 所示。



符　号	参　数	min	max	单　位
t_{D}(AD)	地址输出延迟		145	ns
$t_{DL\phi}$(MR)	从 ϕ 的下降沿到 $\overline{MREQ}$ 变低的延迟		100	ns
$t_{DH\phi}$(MR)	从 ϕ 的下降沿到 $\overline{MREQ}$ 变"高"的延迟		100	ns
$t_{DL\phi}$(WR)	从 ϕ 的下降沿到 $\overline{WR}$ 变"低"的延迟		90	ns
$t_{DH\phi}$(WR)	从 ϕ 的下降沿到 $\overline{WR}$ 变"高"的延迟		100	ns
t_{D}(D)	数据输出延迟		230	ns
t_{F}(D)	在写周期数据到浮动(高阻态)的延迟		90	ns

图 10-38　Z80 CPU 的存贮器写周期时序波形图

Z80A CPU 的存贮器写周期主要时间参数如表 10-5 所示。

表 10-5　Z80A 写周期时间参数

符　号	min	max	单　位
t_{D}(AD)		110	ns
$t_{DL\phi}$(MR)		85	ns
$t_{DH\phi}$(MR)		85	ns
$t_{DL\phi}$(WR)		80	ns
$t_{DH\phi}$(WR)		80	ns
t_{D}(D)		150	ns
t_{F}(D)		90	ns

由于 Z80 CPU 写周期存取信息时间是从 T_1 下降沿到 T_3 下降沿，而取指周期是从 T_1 下

降沿到 T_3 上升沿，其时间比读周期短。因此，若 2114 能满足取指周期的速度要求，就一定可以满足读写周期的工作。

参照存贮器读周期的分析方法，可以发现不仅高速的 2114-2 能与 Z80、Z80ACPU 写周期的时序匹配，而且当选用低档的 2114 时，也能满足 Z80A 写周期的时序要求。

综上所述，从存取速度来看，Z80 CPU 存贮读、特别是取指周期对速度要求很高，而存贮器写周期则较易满足要求。

4．动态 RAM 与 CPU 的连接

在大容量的微型计算机系统中，16 K 的动态 RAM（如 Intel 2116 型或 4116 型）正在迅速地获得广泛的应用。下面以 Intel 2116 为例，说明动态 RAM 与 Z80 或与 Z80A CPU 连接时，所要考虑的主要问题。

(1) Intel 2116 的结构

Intel 2116 的引线和逻辑符号如图 10-39 所示。

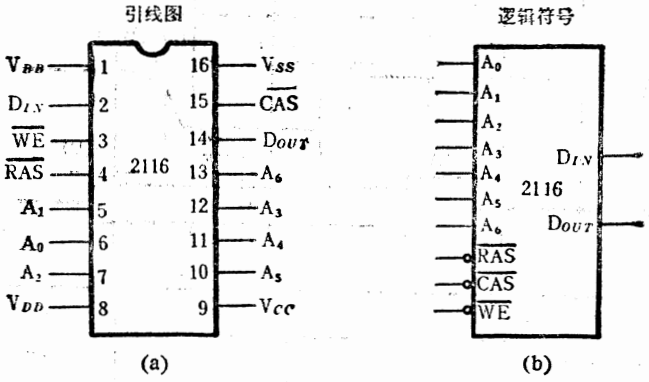


图 10-39 Intel 2116 的引线及逻辑符号图

引 线 表

$A_6 \sim A_0$	地址输入	$\overline{WE}$	写允许
$\overline{CAS}$	列地址选通	V_{BB}	电源(−5V)
D_{IN}	数据输入	V_{CC}	电源(+5V)
D_{OUT}	数据输出	V_{DD}	电源(+12V)
$\overline{RAS}$	行地址选通	V_{SS}	地

每片 Intel 2116 的容量为 16K×1 位，需 14 条地址线。但 2116 引线只有 16 条。为此，把地址线分成行地址与列地址两部分，存贮单元排列成 128×128 的矩阵。这样，地址线只有 7 条，内部设有地址锁存器，由行地址选通信号 $\overline{RAS}$(Row Address Strobe)，把第一个时钟脉冲出现的 7 位地址($A_6 \sim A_0$)，送至行地址锁存器，2116 没有片选信号，$\overline{RAS}$ 同时还起片选信号作用。接着出现列地址选通信号 $\overline{CAS}$(Colnmu Address Strobe) 把第二个时钟脉冲出现的 7 位列地址($A_{13} \sim A_7$) 送至列地址锁存器。同时，7 条地址线也用作刷新地址。

7 位行地址经过译码，产生 128 条选择线，分别选择 128 行；7 位列地址经过译码，也产生 128 条选择线，分别选择 128 列。

2116 的内部结构框图如图 10-40 所示。

当某一行被选中时，则这一行的 128 个存贮单元都被选通到读出放大器去。但列译码器，只选通 128 个放大器中的一个，把它送至锁存器和缓冲器中去。

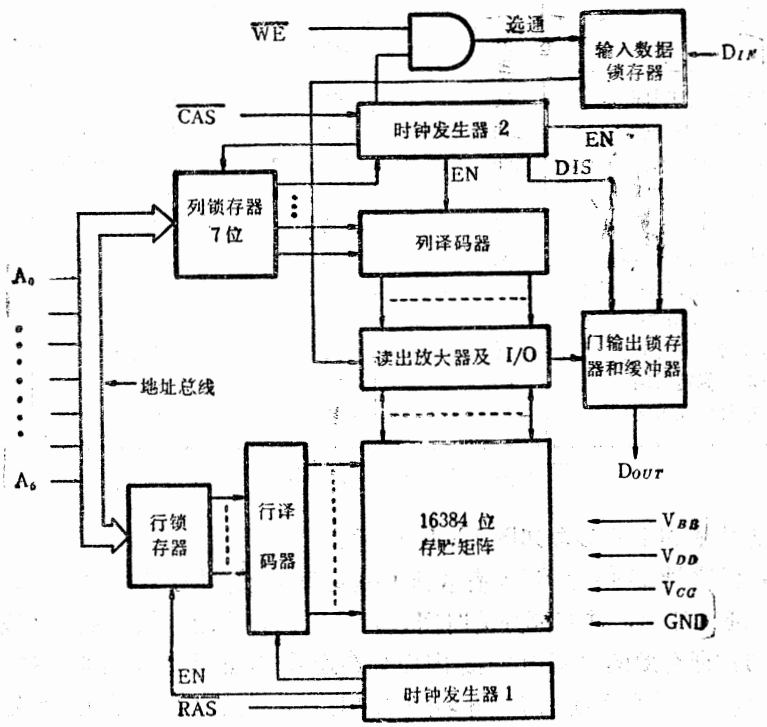


图 10-40　2116 内部结构框图

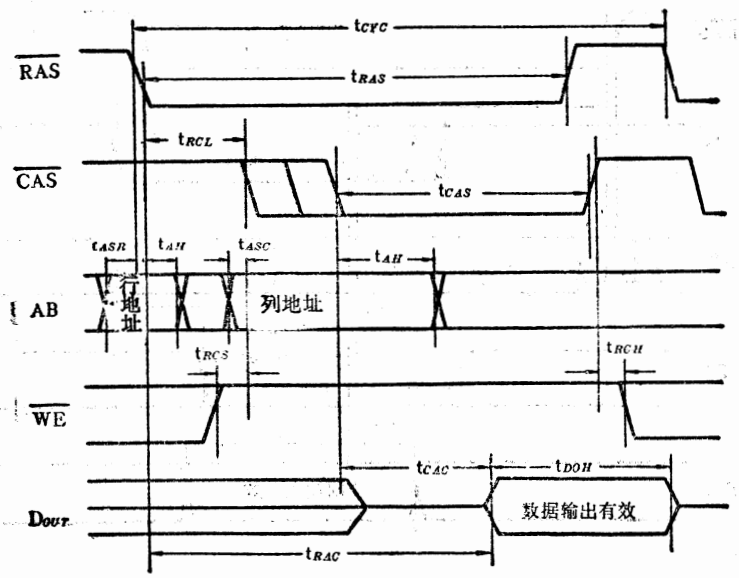
2116 的数据线只有一位，电路的输入(D_{IN})和输出(D_{OUT})端是分开的，有各自的锁存器。

2116 有一读写控制信号线 $\overline{WE}$，当其为低电平时表示写，高电平时表示读。

2116 需要 3 个电源：$V_{DD}=+12V$、$V_{CC}=+5V$、$V_{BB}=-5V$、V_{SS} 接地。

(2) 2116 特性

① 读周期



符　号	参　　数	2116-2		2116-3		2116-4		单　位
		min	max	min	max	min	max	
t_{CYC}	随机读周期时间	350		375		425		ns
t_{RAS}	$\overline{RAS}$ 脉冲宽度	275	32000	300	32000	330	32000	ns
t_{CAS}	$\overline{CAS}$ 脉冲宽度	125	10000	150	10000	190	10000	ns
t_{RCH}	读命令保持时间	20		20		20		ns
t_{RCS}	读命令建立时间	0		0		0		ns
t_{DOH}	数据输出保持时间	32		32		32		ns
t_{RAC}	$\overline{RAS}$ 有效到数据有效		200		250		300	ns
t_{CAC}	$\overline{CAS}$ 有效到数据有效		125		150		190	ns
t_{ASR}	行地址建立时间	0		0		0		ns
t_{AH}	行地址保持时间	45		50		60		ns
t_{ASC}	列地址建立时间	−10		−10		−10		ns

图 10-41　2116 读周期时序波形图

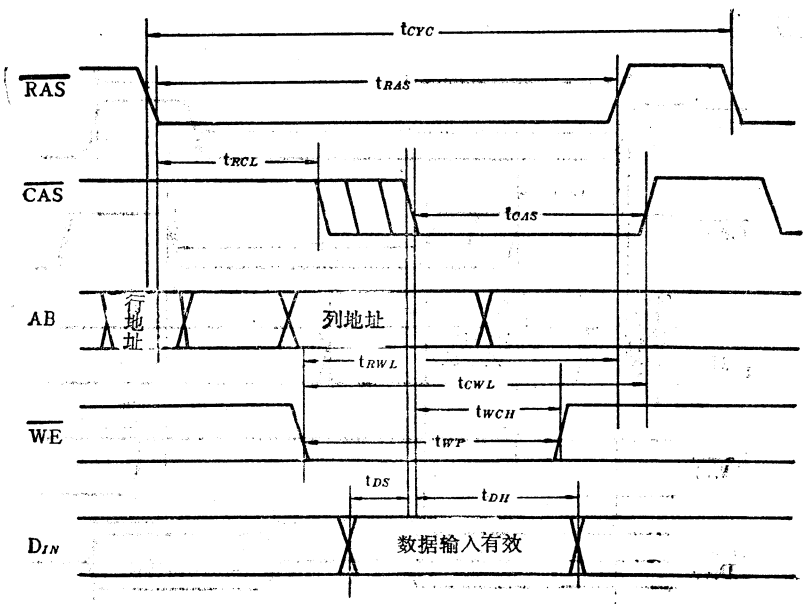
2116 读周期的波形和主要参数如图 10-41 所示。

2116 读周期：行地址必须在 $\overline{RAS}$ 有效前 t_{ASR} 时间有效，并在 $\overline{RAS}$ 有效后保持 t_{AH} 时间，以保证正确选通行地址，并有充分的时间进行译码。行选通有效到数据有效的时间为 t_{RAC}，列选通有效到数据有效的时间为 t_{CAC}，因而列选通 $\overline{CAS}$ 必须在行选通后（$t_{RAC}-t_{CAC}$）时间内有效，否则数据有效时间将取决于 $\overline{CAS}$ 有效加上 t_{CAC}。列地址必须比列选通提 t_{ASC} 有效，以保证正确选通列地址及有充分时间进行译码。

由于 2116 芯片中有地址锁存器，因此在所要求的列地址保持时间以后，在读、写周期完成以前，外部的地址信息可以改变，这与 2114 RAM 不同。

② 写周期

2116 写周期的时序及其时间参数如图 10-42 所示。



符号	参数	2116-2		2116-3		2116-4		单位
		min	max	min	max	min	max	
t_{CYC}	随机写周期时间	350		375		425		ns
t_{RAS}	$\overline{RAS}$ 脉冲宽度	275	32000	300	32000	330	32000	ns
t_{CAS}	$\overline{CAS}$ 脉冲宽度	125	10000	150	10000	190	10000	ns
t_{WCH}	写命令保持时间	75		100		100		ns
t_{WP}	写命令脉冲宽度	50		100		100		ns
t_{RWL}	从$\overline{RAS}$无效到写命令开始的时间	125		200		200		ns
t_{CWL}	从$\overline{CAS}$无效到写命令开始的时间	100		150		160		ns
t_{DS}	输入数据建立时间	0		0		0		ns
t_{DH}	输入数据保持时间	100		100		125		ns

图 10-42　2116 写周期时序波形图

在 2116 写周期中，$\overline{RAS}$ 和 $\overline{CAS}$ 信号以及它们与地址信号之间的关系与读周期相同。

为保证在 t_{RAS} 时间内，能把外部的数据可靠地写入存贮器，要求 $\overline{WE}$ 信号必须在 $\overline{RAS}$ 变高前 t_{RWL} 时间有效或 $\overline{CAS}$ 变高前 t_{CWL} 时间有效。并且写命令信号必须大于宽度 t_{WP}。同时，还要求数据在选通信号（$\overline{CAS}$ 信号与 $\overline{WE}$ 信号的较迟出现者）有效前 t_{DS} 已经稳定，且应保持 t_{DH} 时间。

(3) 刷新周期

如前所述，动态存贮器是靠 MOS 管的栅-衬电容来存贮信息的，因而必须周期性(2ms)地刷新。

2116 在每次读或写周期时，由 7 位行地址所选中的整个一行（在选中的片上）被刷新。要刷新整个 16K RAM，必须在每 2 ms 时间内，对 128 个行地址的每一行的存贮单元，执行一次 $\overline{RAS}$ 的存贮周期（对于刷新序列来说 $\overline{CAS}$ 是不需要的），共 128 个 $\overline{RAS}$ 刷新周期。

为了控制刷新，往往要求增加外部刷新逻辑。而 Z80 CPU 本身具有刷新的逻辑功能，因而，刷新问题较易解决。

动态 RAM 刷新周期的波形如图 10-43 所示。

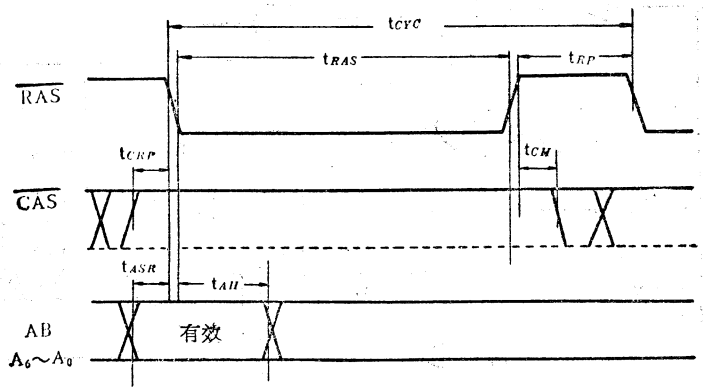


图 10-43　动态 RAM 刷新周期时序波形图

刷新时，$\overline{RAS}$ 为“低”而 $\overline{CAS}$ 为“高”。而且，$\overline{RAS}$ 的宽度必须大于 t_{RAS}。

刷新地址必须在 $\overline{RAS}$ 有效前有效，且要保持一段时间。在刷新周期应将输出断开。

(4) 动态 RAM 与 Z80 的接口

由于 2116 RAM 用 $\overline{RAS}$ 和 $\overline{CAS}$ 来分别选通和锁存行地址和列地址，且在行地址中，又需要能接入刷新地址，因而需要多路转换器。其接口电路的方框图如图 10-44 所示。

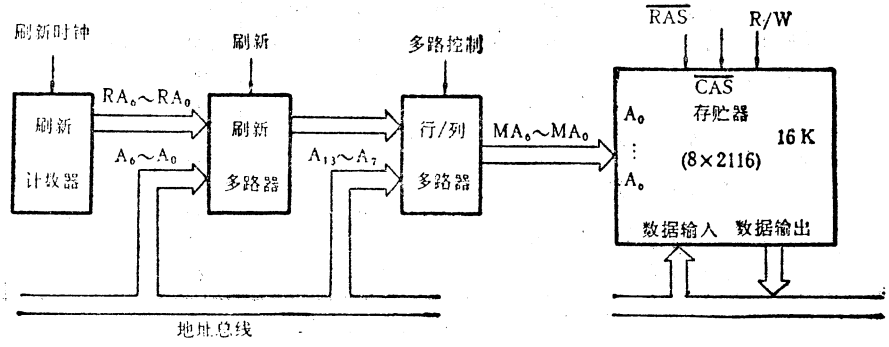


图 10-44 动态 RAM 与 Z80 的接口电路

由图可知，行地址——地址总线上的 A_6～A_0 和刷新计数器（Z 80 中的 R 寄存器）的 7 位地址通过一个多路转换器输出至行/列多路转换器，只有在刷新时，才为刷新地址，否则为行地址。列地址——地址总线上的 A_{13}～A_7 也送到行/列多路转换器，由多路转换器控制输出的 7 位地址是行地址还是列地址，然后送至存贮器的地址线，由选通信号 $\overline{RAS}$ 和 $\overline{CAS}$ 选通送至各自的地址锁存器，以实现读或写。

Z80 的刷新是在取指周期(M_1)内进行的。在 M_1 周期的 T_3 前沿，指令已经选通至 IR，接着在 T_3、T_4 时态执行指令译码以及 CPU 的内部操作，在这段时间内不使用地址总线。故 T_3、T_4 时态 R 寄存器把刷新地址送至地址总线的低 7 位($A_7=0$)，且 CPU 发出 $\overline{RFSH}$ 信号以控制刷新。此时，$\overline{MREQ}$ 为有效，用以选择存贮器，而 $\overline{RD}$ 无效，保证断开数据总线。

(5) 选通信号的产生

在动态 RAM 中，需要用外部电路产生 $\overline{RAS}$ 和 $\overline{CAS}$，产生的方法很多。图 10-45 所示的电路就是其中一种。

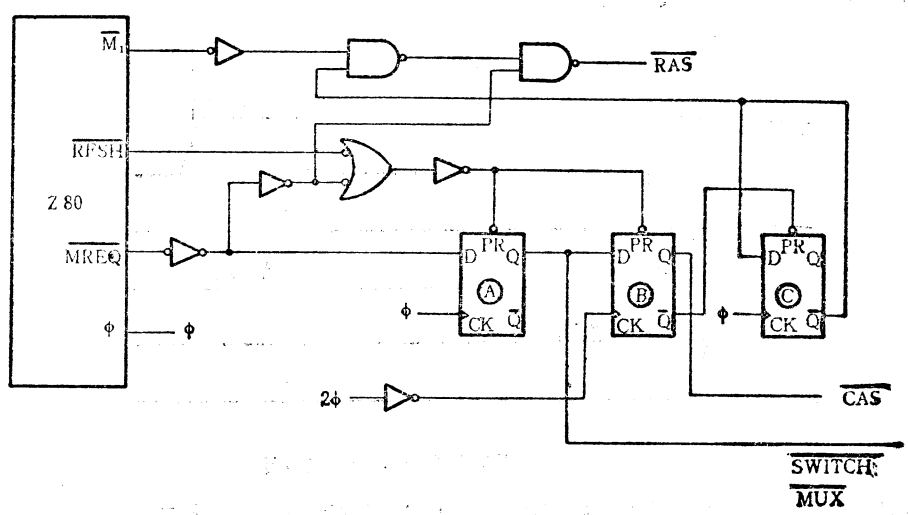


图 10-45 产生选通信号的一种电路

此电路是从一个 8 MHz 的振荡源分频产生 CPU 的 4 MHz 的时钟脉冲。其工作过程的波形图如图 10-46 所示。

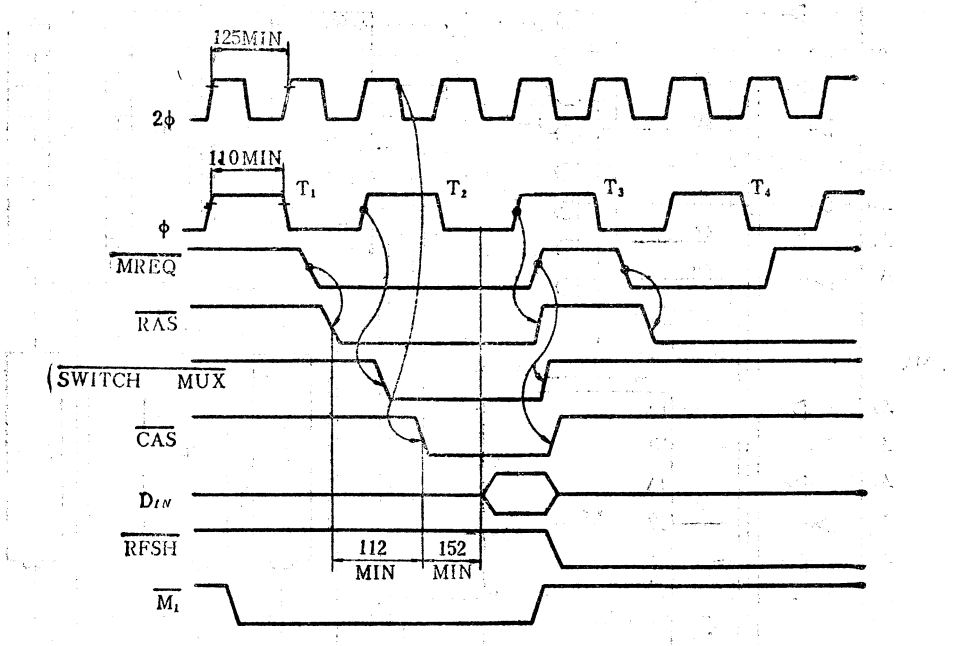


图 10-46 选通信号电路的工作波形图

在开始工作前，触发器 A、B、C 都置为“1”。即 Q_A = “1”，Q_B = “1”，$\overline{Q_C}$ = “0”。在 M_1 周期，T_1 的 ϕ 的后沿使 $\overline{MREQ}$ 有效（低电平），它经过几个门的延迟时间后，使 $\overline{RAS}$ 变“低”。且 $\overline{MREQ}$ 信号控制触发器 A 的 D 端，故在下一个时钟（T_2）的上升沿使 Q_A 变“低”，即 $\overline{SWITCH\ MUX}$ 变“低”，以用作将多路转换器从行地址转接到列地址上的控制信号。而 Q_A 又控制触发器 B 的 D 端，故在下一个 2ϕ 脉冲的下降沿使 $\overline{CAS}$ 变“低”（有效）。同时，$\overline{Q_B}$ = “1”，释放了触发器 C 的置位端，则 C 工作在计数状态。在下一时钟（T_3）的上升沿使 C 翻转，$\overline{Q_C}$ = 1，则它使 $\overline{RAS}$ 变“高”。当 $\overline{MREQ}$ 变“高”后，它经过几个门的延迟时间后，使触发器 A、B 的置位端为“低”。同时使 $\overline{SWITCH\ MUX}$ 和 $\overline{CAS}$ 变“高”。

由波形图可知，$\overline{RAS}$ 的宽度大于一个 T 时钟周期（若 CPU 为 Z80 A，则为 250 ns，Z80 为 400 ns）；$\overline{CAS}$ 的宽度大于半个 T 时钟周期。在图中表示自 $\overline{RAS}$ 有效至 $\overline{CAS}$ 有效的时间至少为 112 ns，而且 $\overline{CAS}$ 有效至数据稳定的时间至少为 152 ns。

因而，若 CPU 为 Z 80 A，则选用 2116-2 完全可满足取指周期（也必然是整个读写周期）的时序要求；若 CPU 为 Z80，则可选用 2116-4。

(6) 动态 RAM 与 Z 80 A 连接的例子

一个 16K × 8 位的动态 RAM 与 CPU 的接口电路，如图 10-47 所示。

产生 $\overline{RAS}$、$\overline{SWITCH\ MUX}$ 和 $\overline{CAS}$ 信号的电路与图 10-45 相同。7 位行地址与 7 位列地址的选通由两片 4 位 2 选 1 数据选择器 LS 157 来实现，选通信号用 $\overline{SWITCH\ MUX}$。当 $\overline{SWITCH\ MUX}$ = “1”时，它把行地址送动态存贮器的地址输入端，由 $\overline{RAS}$ 信号把它们选通至行地址锁存器；当 $\overline{SWITCH\ MUX}$ = “0”时，则把列地址送存贮器，由 CAS 信号选通至列地址锁存器。而刷新时的地址 $RA_7 \sim RA_0$ 由 CPU 在刷新周期自动送至地址总线的低 7 位，作

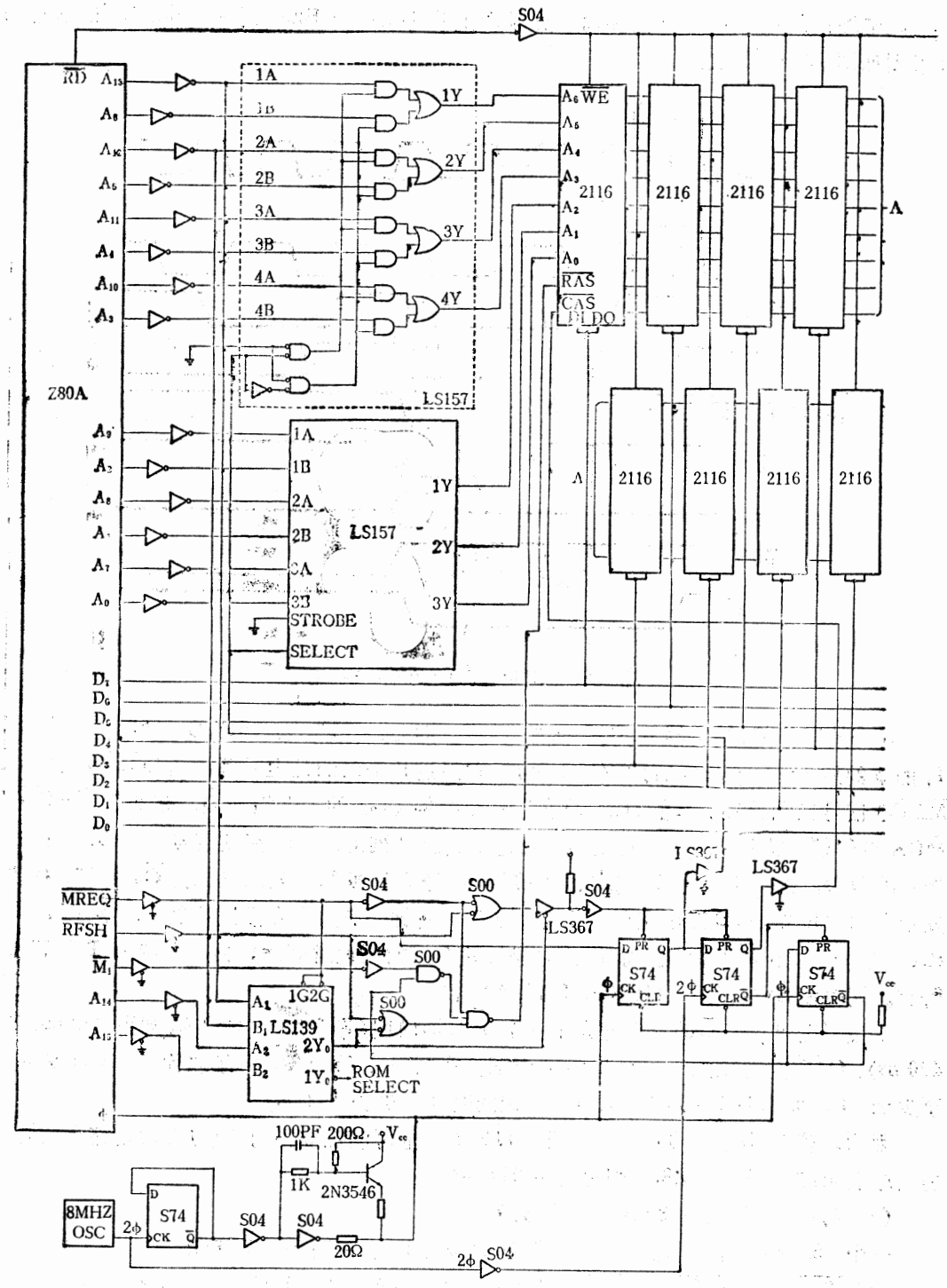


图 10-47　动态 RAM 与 Z80A 连接框图

为行地址送至动态存贮器。

由于在系统中除了 RAM 以外，一般还有 ROM 或 EPROM，因此，要把它们的地址与 16K

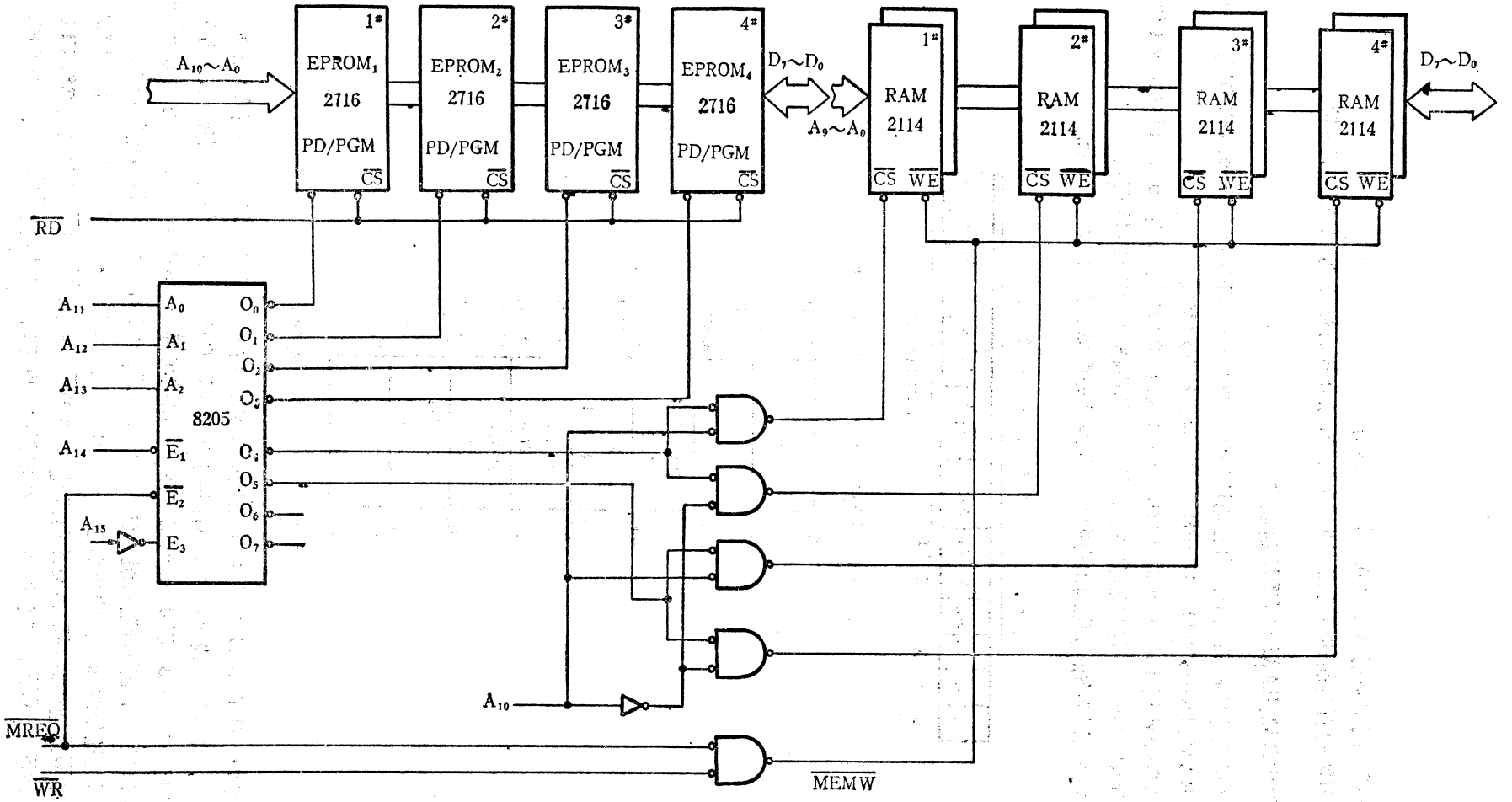


图 10-48　存贮器与 CPU 之间的连接

RAM 区分开。图中是用 A_{15} 和 A_{14} 来加以区分的。

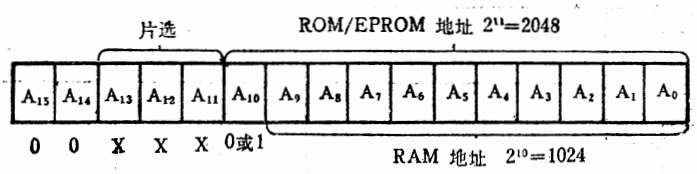
由 8 MHz 的石英晶体振荡器产生 2ϕ 信号，经分频后得到 ϕ，作为系统时钟。

四、EPROM、RAM 与 CPU 的连接

设 Z80 微型计算机系统中，内存地址空间：0000～1 FFFH (8 KB) 为 EPROM；2000～2FFFH 为 RAM(4 KB)；3000～3 FFFH(4 KB)为待扩存贮空间。

其中，EPROM 用 4 片 Intel 2716(2K×8 位)构成；RAM 用 16 片 Intel 2114(1K×4 位)构成。采用 Intel 8205 3-8 译码器实现选片控制。其连接图如图 10-48 所示。

设计该存贮器系统时，先画出存贮器空间地址分配图。如图 10-49 所示。



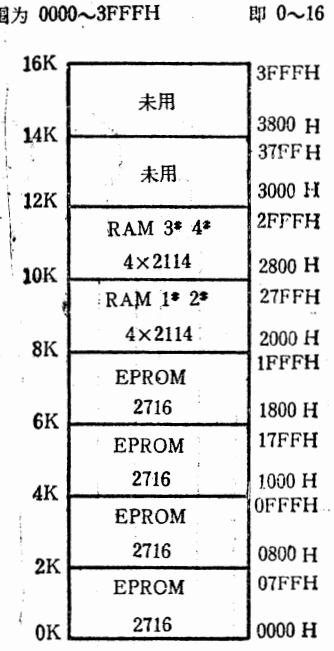


图 10-49 存贮空间地址分配图

由图可知，对于 2716 需要用 11 条地址线实现片内字选；对 2114 需要用 10 条地址线实现片内字选。高位地址线 A_{13}、A_{12}、A_{11} 用于选择 EPROM 和 RAM 的芯片(片选)。由于 2114 RAM 存贮容量为 1 KB，而 8205 的每条译码输出端可寻址 2 KB 存贮空间，因此必须用 A_{10} 与 8205 的译码输出端进行逻辑组合(即二次译码)后，才能对 RAM(2114)进行片选。

存贮器的地址线和数据线分别连接至 CPU 的地址总线和数据总线上。存贮器 RAM 的写入允许端 $\overline{WE}$ 连接到存贮器写控制信号 $\overline{MEMW}$。这里，$\overline{MEMW}$ 信号是由 Z80 CPU 的控制信号 $\overline{MREQ}$ 与 $\overline{WR}$ 相组合得到的。在本例中，也可以把 2114 的 $\overline{WE}$ 直接接至 CPU 的控

制信号线 $\overline{WR}$ 上。

应当指出，这里，把译码器的译码输出端分别连接到各片 2716 的 $\overline{CE}$ /PGM(PD/PGM)端上，而把 CPU 的 $\overline{RD}$ 信号连接到各片 2716 的 $\overline{OE}$($\overline{CS}$) 端上。这样安排，除了地址选中的 2716 芯片 $\overline{CE}$/PGM 端为低电平、由 $\overline{RD}$ 信号控制读出外，其余的 2716 芯片的 $\overline{CE}$/PGM 端全为高电平，即未工作的芯片都处于“功率下降”方式，这样可以减少功耗。

第十一章　微型计算机的输入和输出

§11-1　概　　述

输入/输出(I/O)是指微型计算机与外界之间的信息交换，即通信 (Communication)。

微型计算机与外界的通信，是通过输入输出设备进行的，通常有两种通信方式，即并行通信(几位同时传送)和串行通信(一位一位地传送)。常用的 I/O 设备有键盘 (Key Board)、纸带输入机、电传打字机、行式打印机、CRT 显示器、盒式磁带机、磁盘机以及模/数转换器 (A/D Converter)、数/模转换器(D/A Converter)等。通常一种 I/O 设备与微型机连接，就需要一个连接电路，我们称之为 I/O 接口。存贮器也可以看作是一种标准化的 I/O 设备。它们的种类不多，传送速率差异不大，而且各种存贮器与微处理器交换数据的方式大都相似。但是，其它 I/O 设备种类繁多，可以是机械式、机电式、电子式以及其它形式的。它的输入信号也不尽相同，可以是数字量，也可以是模拟量(模拟式的电流和电压)。而且，传送数据的速率也相差很大，可以是手动的键盘输入(每个字符输入的速度为秒级)，也可以是软磁盘输入，在找到所需的磁道以后，它能以 250000 位/秒的速率传输数据。由于 I/O 设备的多样性，因此使得 I/O 接口部分成为微型计算机系统设计中最为复杂的一部分。好在目前有不少 I/O 接口逻辑电路已采用大规模集成电路并已系列化、标准化，而且许多 I/O 接口芯片具有可编程序的功能，有很好的灵活性，为输入/输出(I/O)创造了良好的物质条件。

如前所述(见 § 4-4)，I/O 接口有以下五种功能：

1．地址译码；

2．数据缓冲；

3．信息转换；

4．提供命令译码和状态信息；

5．定时和控制(定序)。

本章着重讨论输入/输出的寻址方式，I/O 指令及其时序、I/O 设备与 CPU 数据传送的方式等内容。

§11-2　输入/输出的寻址方式

微处理器进行 I/O 操作时，对 I/O 接口的寻址方式与前所提及的存贮器寻址方式相似。即必须完成两种选择：一是选择出所选中的 I/O 接口芯片(称为片选)；二是选择出该芯片中的某一寄存器(称为字选)。

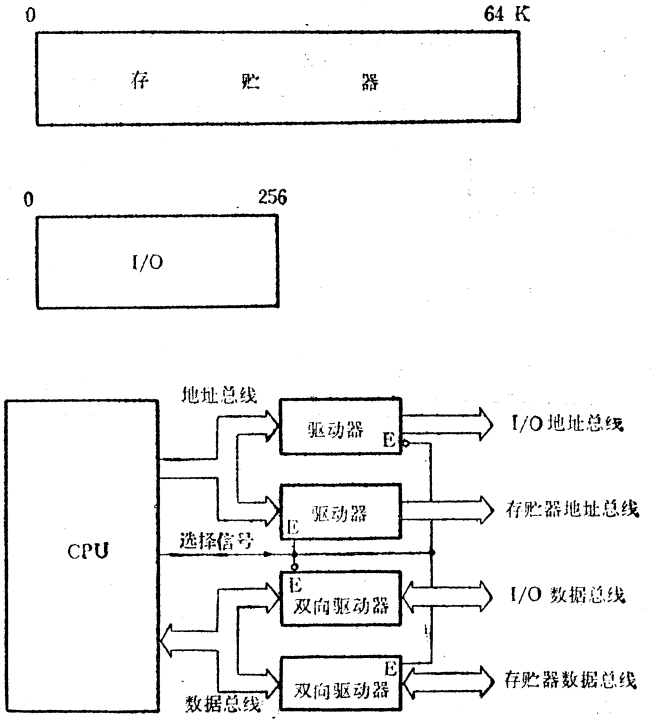
通常有两种 I/O 接口结构：一种是标准的 I/O 结构；另一种是存贮器映象 I/O 结构(Memory Mapped I/O)。与之对应的有两种 I/O 结构寻址方式。

一、标准的 I/O 寻址方式

标准的 I/O 寻址方式也称为独立的 I/O 寻址方式或称为端口(Port)寻址方式。它有以下

三个特点：

1．I/O 设备的地址空间和存贮器地址空间是独立的、分开的。也即 I/O 接口地址不占用存贮器的地址空间。如图 11-1 所示，它通常由一个“部分选择”信号启动一组总线驱动器之一，从而区分存贮器和 I/O 设备。



“部分选择”信号为“1”电平表示存贮器周期；为“0”电平则表示是 I/O 周期

图 11-1 标准的 I/O 接口结构

一般说来，CPU 用地址总线的低 8 位对 I/O 设备寻址，因此其寻址范围为 $2^8=256$。但是，8080 A/8085 A 微处理器可用地址总线的低 8 位或高 8 位对 I/O 进行寻址。

2．微处理器对 I/O 设备的管理是利用专门的 IN（输入）和 OUT（输出）指令来实现数据传送的。

3．CPU 对 I/O 设备的定时控制是用 I/O 读写控制信号（$\overline{\text{I/OR}}$, $\overline{\text{I/OW}}$）。

采用标准的 I/O 寻址方式的微处理器有 Intel 8080 A/8085 A、Zilog Z 80 等。其中又可分两种寻址方法：即线性选择法和地址译码器选择法。

应当指出，标准的 I/O 寻址方式是以端口（Port）作为地址的单元，因为一个外设往往不仅有数据寄存器，还有状态寄存器和控制寄存器，它们各需要一个端口才能区分，故一个外设常需要若干个端口地址。

二、存贮器映象 I/O 寻址方式

存贮器映象 I/O 寻址又称为存贮器对应 I/O 寻址方式，它也有三个特点：

1．I/O 接口与存贮器共用同一地址空间。即在系统设计时，指定存贮器地址空间内的一个区域供 I/O 设备使用，故每一个 I/O 设备占用存贮空间的一个地址。这时，存贮器与 I/O 设

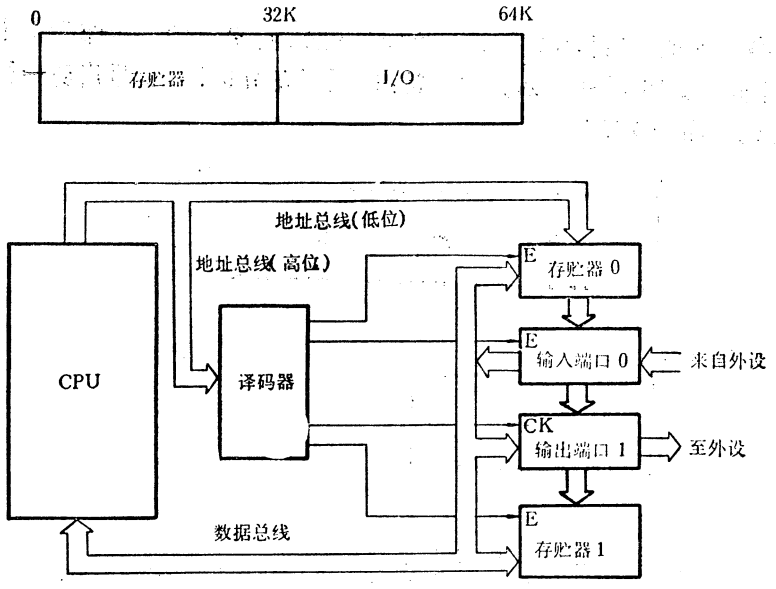


图 11-2 存贮器映象 I/O 接口结构

备之间的唯一区别是其所占用的地址不同。如图 11-2 所示。

一般设定 I/O 端口占用地址线 A_{15} 为 "1" 的 32 K 地址空间(存贮地址占用 A_{15} 为 "0" 的 32 K 地址空间)。显然,当 A_{15} = "1" 时,则 I/O 接口动作;当 A_{15} = "0" 时,则存贮器动作。

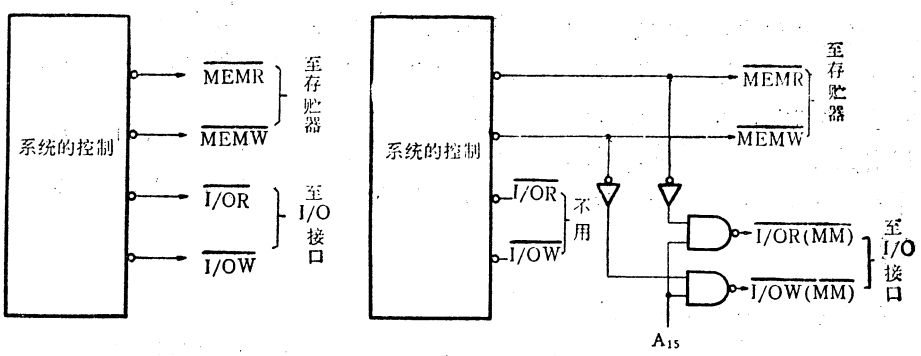
这里选用 A_{15} 作为 I/O "标志",因为 A_{15} 是地址最高位,用软件能较容易地控制。同时,它仍允许存贮器的寻址空间达 32K。

2. 微处理器对 I/O 设备的管理,是利用对存贮器的存贮单元进行操作的指令来实现数据传送的。

3. CPU 用存贮器读写控制信号 $\overline{(MEMR)}$ $\overline{(MEMW)}$,并经 A_{15} 状态控制来确定是存贮器还是 I/O 设备。

MC 6800 微处理器是具有存贮器映象 I/O 结构的典型例子。当然,Intel 8080 A/8085 A 和 Z 80 微处理器也可使用存贮器映象 I/O 寻址方式。

图 11-3 示出了标准的 I/O 结构和存贮器映象 I/O 结构的控制信号图。



(a) 标准的 I/O结构　　(b) 存贮器映象 I/O 结构

图 11-3 两种 I/O 结构之比较

三、两种 I/O 结构及寻址方式之比较

标准 I/O 结构寻址的优点是：

1．I/O 设备不占用存贮器地址空间；

2．程序设计时，易把 I/O 指令与存贮器指令分开；

3．通常，I/O 指令只需两个字节，所以寻址速度较快。

存贮器映象 I/O 结构寻址的优点是：

1．扩大了管理 I/O 设备的寻址范围(可超过 256 个 I/O 端口)；

2．可利用更多的存贮器指令访问 I/O 端口，使用灵活，方便。

此方式的缺点是存贮器地址空间被 I/O 设备占用了一半。

§11-3 输入/输出指令及其时序

一、Intel 8080A/8085A 的 I/O 指令

Intel 8080A/8085A CPU 仅有两条标准的 I/O 指令、即输入(IN)指令和输出(OUT)指令。它们的第一字节是操作码，第二字节指出 I/O 端口的地址。应当记住，Intel 8080A/8085A CPU 在执行 I/O 指令时，把所要寻址的 I/O 端口地址同时复制到地址总线的高 8 位(A_{15}～A_8)和低 8 位(A_7～A_0)上。因而，在 8080A/8085A 系统中，I/O 接口芯片的地址线可以挂在 CPU 地址总线的高 8 位或低 8 位上。通常，地址总线低 8 位上总挂着存贮器的地址线。有时，为了合理地分担总线上的负载，往往把 I/O 接口电路的地址线挂在 CPU 地址总线的高 8 位上。

此外还应指出，Intel 8080A/8085A 标准的 I/O 指令，只能允许指定的 I/O 端口与 CPU 中的累加器 A 直接交换信息，故又称为累加器型的 I/O 指令。

二、Z80 的 I/O 指令

Z80 CPU 具有丰富的 I/O 指令组，它除含有 Intel 8080A/8085A 两条标准的 I/O 指令外，还增加了两类功能很强的 I/O 指令：

(1) 寄存器 I/O 指令

这类 I/O 指令允许指定的 I/O 端口直接与 CPU 中的任一内部寄存器之间交换信息。而 I/O 端口的地址则由寄存器 C 来确定。

(2) 数据块传送 I/O 指令

这类指令能够在由寄存器 C 指出地址的 I/O 端口和由 HL 寄存器对指出地址的存贮单元之间传送数据块。

Z80 的 I/O 指令组详见附录 1 表 A 1-11 和附录 2 表 A 2-11。

下面分别予以说明。

1．直接寻址的 I/O 指令

指令格式：

```
IN    A,   (n)   ; A←(n)
OUT   (n),  A    ; (n)←A
```

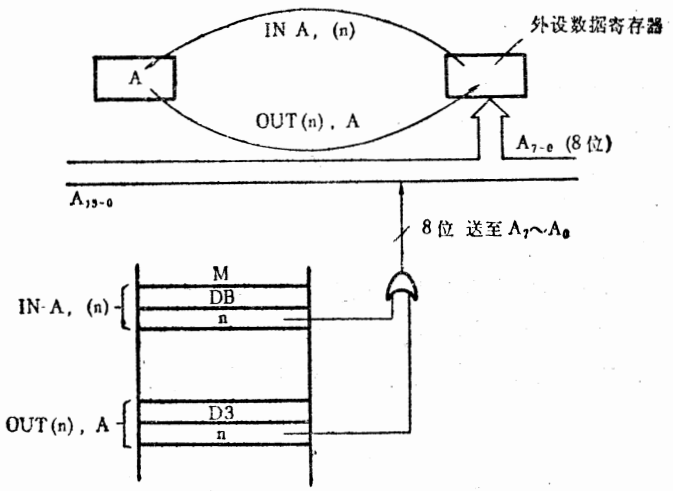


图 11-4　直接寻址 I/O 指令示意图

I/O 指令执行的示意图如图 11-4 所示。

其中 n 为所要寻址的一字节的端口地址。执行时，端口地址出现在地址总线的低 8 位（A_7～A_0）上（累加器 A 内容送至 A_{15}～A_8）。输入指令的功能是把地址为 n 的端口的数据或状态寄存器的内容送至累加器 A；输出指令的功能是把累加器 A 的内容送至地址为 n 的端口的数据或控制寄存器中去。执行标准 I/O 指令不影响标志位的状态位。

2．用寄存器 C 间接寻址的 I/O 指令

指令格式：

```
IN    r,    (C)    ;r←(C)
OUT   (C),  r      ;(C)←r
```

指令执行的示意图如图 11-5 所示。

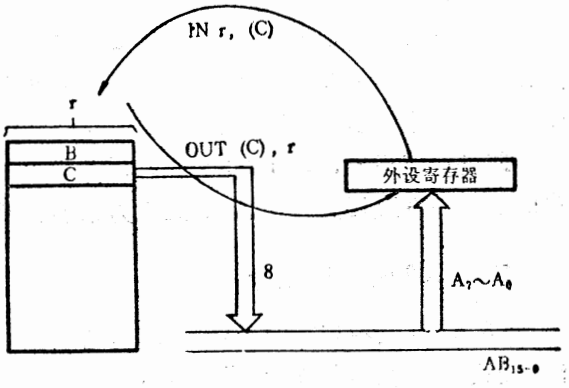


图 11-5　间接寻址方式 I/O 指令示意图

这类指令的功能是允许由寄存器 C 指出地址的 I/O 端口直接与 CPU 中任何内部寄存器 A、B、C、D、E、H 或 L 交换信息。

其操作特点是：

(1) I/O 端口地址是由寄存器 C 的内容确定的（最多为 256 个），故执行该类指令前，必须事先预置 C 寄存器的内容。例如：

```
LD    C, PORTN    ;把端口地址送寄存器 C
IN    E, (C)      ;从 PORTN 端口把数据输入到寄存器 E 中
```

(2) 执行 IN r, (C)指令，将影响标志寄存器中的 S、Z、P/V 的状态。若输入数据为负，则 S = "1"；若输入数据为 0，则 Z = "1"；若输入数据奇偶性为偶，则 P/V = "1"。不符合上述条件时，则 S、Z 和 P/V 分别置 "0"。对于进位标志 C，则不受其影响，而半进位标志 H 和减法标志 N 均置 "0"。

(3) 执行 OUT (C), r 时，不影响标志位的状态。

(4) 地址低 8 位（A_7～A_0）放 C 寄存器内容，高 8 位（A_{15}～A_8）放 B 寄存器内容。

3．数据块输入输出指令

(1) 单个的数据 I/O 指令

指令格式：

INI ；(HL)←(C)，　HL←HL+1，　B←B−1

IND ；(HL)←(C)，　HL←HL−1，　B←B−1

OUTI；(C)←(HL)，　HL←HL+1，　B←B−1

OUTD；(C)←(HL)，　HL←HL−1，　B←B−1

指令执行的示意图如图 11-6 所示。

这类指令是在由寄存器对 HL 所指定的存贮单元和由寄存器 C 所选定的 I/O 端口之间传送数据。而用寄存器 B 作为数据字节计数器。在执行一次输入/输出数据以后，寄存器 B 的内容

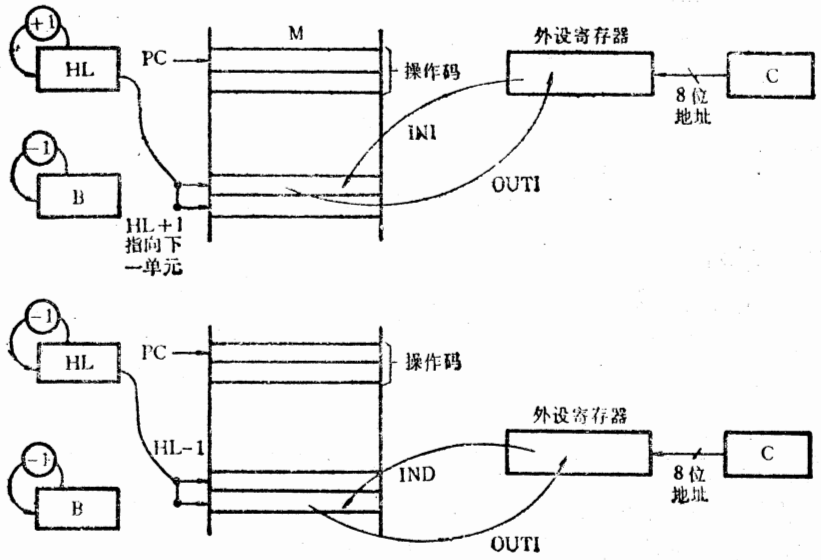


图 11-6　数据块 I/O 指令示意图

减量，而 HL 寄存器对中的存贮单元的地址自身增量(或减量)。执行这类指令时，若 B←B−1，B = 0，则零标志 Z = 1，否则 Z = 0。因为计数器 B 是一个 8 位寄存器，故其最多只能传送 256 个字节。

(2) 成组的数据块 I/O 指令

指令格式：

INIR ；(HL)←(C)，　HL←HL+1，　B←B−1，　直至 B = 0

INDR ；(HL)←(C)，　HL←HL−1，　B←B−1，　直至 B = 0

OTIR ；(C)←(HL)，　HL←HL+1，　B←B−1，　直至 B = 0

OTDR；(C)←(HL)，　HL←HL−1，　B←B−1，　直至 B = 0

其功能与单个传送的情况相似，只是执行这类指令时，若 B 减 1 的结果不为零，则自动重复执行数据块传送直至 B 减 1 结果为零而结束。

应当指出，上述指令的数据块最大容量不能超出 256 个字节。这点与数据块传送或数据块搜索指令是有区别的。后者是用 BC 寄存器对作为 16 位计数器，因而，数据块传送的最大容量可达 64K 字节。

4. Z 80 CPUI/O 时序

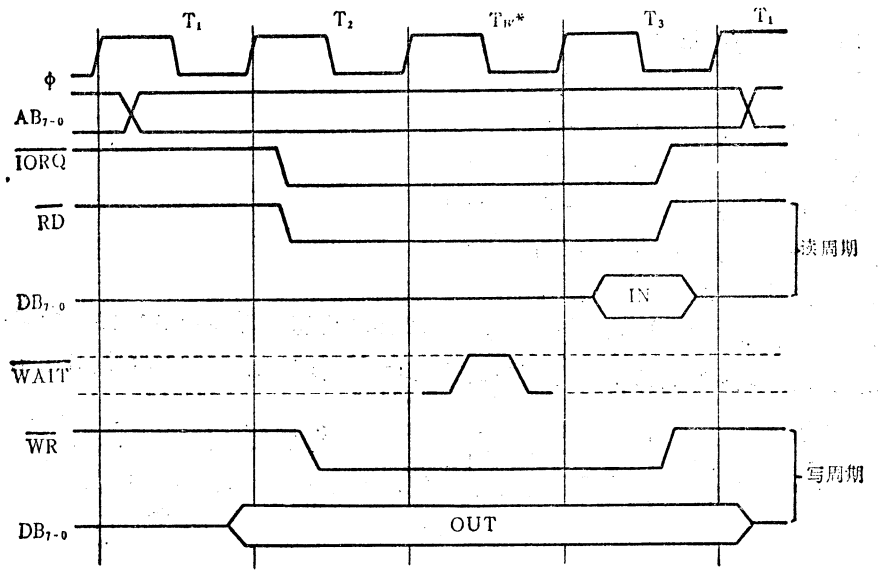


图 11-7　Z 80 CPU I/O 时序波形图

Z 80 CPU 的 I/O 时序如图 11-7 所示。

Z 80 CPU 在执行 I/O 指令期间产生了 I/O 周期。由图 11-7 可知，I/O 过程是从地址有效开始的，在 T_1 时态的上升沿后，所选择的 I/O 端口地址有效(出现在地址总线低 8 位 A_7～A_0)。在 T_2 时态的上升沿后，$\overline{IORQ}$ 和 $\overline{RD}$ 或 $\overline{WR}$ 信号都变为有效。应当注意，在 I/O 周期内，在 T_2 时态结束后，将自动地插入一个等待时态 T_W^*，这是因为在 I/O 操作期间，从 T_2 上升沿开始启动 $\overline{IORQ}$ 信号有效，直到 CPU 对 I/O 设备所发出的等待信号 $\overline{WAIT}$ 进行查询，这期间的时间间隔不应太短，否则将没有充足的时间对 I/O 端口的地址进行译码寻址。

CPU 在 T_3 时态的下降沿采样数据总线，获取由外设输入的数据。利用 T_3 的下降沿，使 $\overline{IORQ}$ 和 $\overline{RD}$ 同时变为无效(恢复高电平)，输入周期即告结束。

当进入输出周期时，CPU 要输出的数据，自 T_1 时态的下降沿后就出现在数据总线上，并且在整个输出周期维持不变。在 T_2 时态上升沿之后，$\overline{IORQ}$ 和 $\overline{WR}$ 信号同时有效 (低电平)，表示允许进行 I/O 写操作。同样，T_2 时态结束后，自动插入一个 T_W^* 时态。当外设控制器完成了对端口地址的译码后，将在 T_3 时态的前半周结束前，把数据总线上保持的数据输出到指定的外设中去。

应当指出，单独用 $\overline{RD}$ 和 $\overline{WR}$ 信号还不能作为 I/O 的选通信号，因为在存贮器读写时，$\overline{RD}$ 和 $\overline{WR}$ 信号也同样有效。所以必须同时用 $\overline{MREQ}$ 或 $\overline{IORQ}$ 来区别是存贮器的读写操作，还是 I/O 的读写操作。

如果 I/O 周期在 T_2 时态结束后，自动插入一个 T_W^* 时态还不能使外设做好接收或发送数据的必要准备，那末，外设将向 CPU 发出 $\overline{WAIT}$ 信号。CPU 利用 T_W^* 的下降沿查询该信号。当发现请求等待信号 $\overline{WAIT}$ 有效时，就在 T_W^* 结束之后不进入 T_3 时态，而是延长一个等待时态 T_W。随后利用 T_W 的下降沿，再对 $\overline{WAIT}$ 信号进行查询，若它仍为有效，则再延长一个 T_W，直到查询后发现 $\overline{WAIT}$ 为无效时为止，并在下一个 T 时态转入 T_3 时态。这就是具有等待时态的 I/O 周期；它与无等待时态的 I/O 周期的区别仅在于根据需要插入了 T_W 时态，而使该机器周期延长。

§11-4 输入/输出传送方式

一、CPU 与 I/O 之间的接口信号

CPU 与外设之间交换信息，如图 11-8 所示，通常需要以下接口信号：

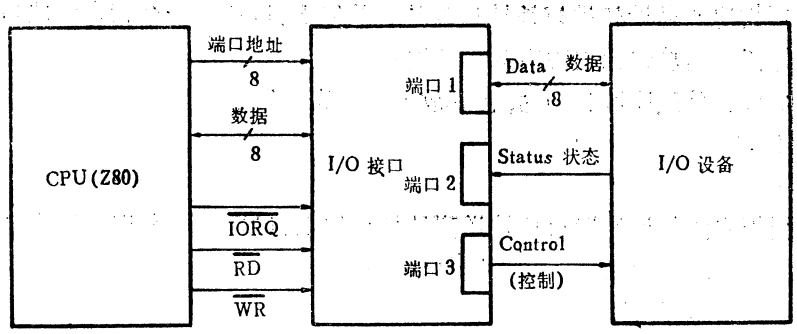


图 11-8 CPU 与 I/O 之间的接口

1．数据

在微型计算机与 I/O 接口之间传送的数据，通常为 8 位或 16 位的数字量。而从 I/O 设备发出或接收的数据物理量可分为三种类型：

(1) 数字量

这是指以二进制形式表示的数或以 ASCII 码表示的数或字符。它们可通过输入设备直接送入计算机。

(2) 开关量

这是表示一个开关状态的量。例如，阀门的打开或关闭，电动机的运转或停止、开关的合或断等等。这些量只要用一位二进制数即可表示。数字量的每一个二进制位，都可以表示一个开关量。

(3) 模拟量

在生产过程中，许多连续变化的信息量都是模拟量。例如温度、压力、流量、速度等等。这些非电量模拟信息经过传感器转换为电量，并经放大即得模拟电压或电流。然后，再经过 A/D 转换器转换为数字量才能输入计算机；计算机的控制输出也必须经过 D/A 转换器转换为模拟量，才能去控制执行机构。

2．状态信息

这是指由 I/O 接口输入微型机的反映 I/O 设备工作状态的信息。如输入设备的信息是否准备好(Ready)，输出设备是否有空(Empty)，以及输出设备正在工作，即“忙”(Busy)等等。

3．控制信息

这是指由 CPU 向 I/O 接口输出的用以识别所要进行的操作性质的控制信息。如，选通(Strobe)控制信号等。

如图 11-8 所示。以上三种不同性质的信息分别由不同地址的端口进行传送。

确切地说，CPU 对外部设备寻址应是对该外设的 I/O 接口器件的端口的寻址。一个端口的寄存器是 8 位的，正好用于传送 8 位数据。而状态或控制端口只用其中的一位或几位，因而不同的外设的状态或控制信号可以共用一个端口。

二、输入/输出的操作过程

根据 Z80 的 I/O 周期的时序分析，我们可把 I/O 操作过程归纳如下：

1．输入操作过程

其步骤如下：

(1) CPU 把指定的外设的端口地址送到地址总线上，以选择一个指定的端口；

(2) CPU 等待数据总线上出现有效的数据；

(3) CPU 从数据总线读取数据，并将其放入一个寄存器。

2．输出操作过程

其步骤如下：

(1) CPU 把指定的外设的端口地址送到地址总线上，以选择一个指定的端口；

(2) CPU 将其欲输出的数据放到数据总线上；

(3) CPU 等待数据传送完成的回答。

由上可知，关键在于如何协调 CPU 的定时与 I/O 设备的定时之间的差异，使 CPU 与 I/O 设备之间的数据交换取得同步。

三、I/O 传送方式

通常，为了实现 CPU 与 I/O 设备之间的数据传送，采用以下三种 I/O 传送方式：

1．程序传送方式

在这种传送方式中，微型机与外部设备之间的数据传送由程序来控制，这是最简单的传送方式。

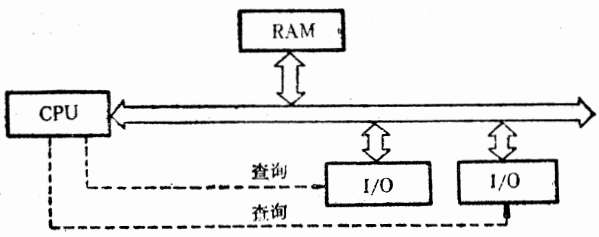
2．中断传送方式

这种传送方式是由外部设备发出中断请求，迫使微型机暂停正在执行的程序，转而为外围设备服务的一种数据传送方式。实质上，这种方式也是由微型机的程序控制的，因而也可以认为是程序控制输入/输出方式的一种。

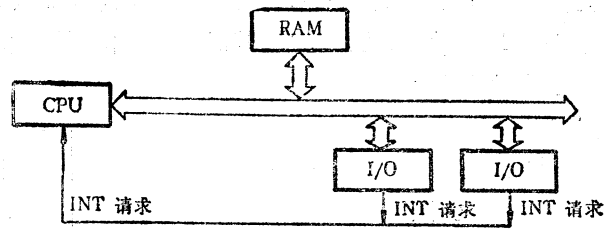
3．直接存贮器存取(DMA)方式

这种传送方式是由外围设备(或 DMA 控制器)来控制的。它能直接在外围设备与存贮器之间进行数据传送，而不需要 CPU 的干预。它适用于进行大量的高速的数据传送。

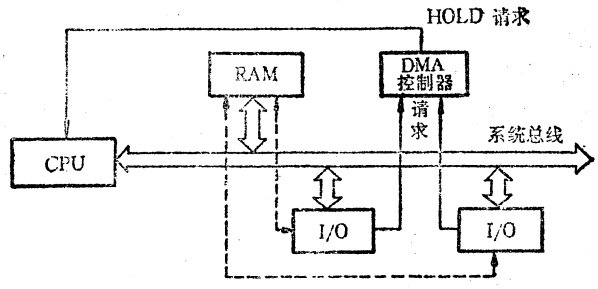
以上三种 I/O 传送方式可用图 11-9 (a)、(b)、(c) 示意。下面将分别予以讨论。其中，有关中断传送方式将归并在第十二章中断系统详细介绍。



(a) 程序传送方式(查询)



(b) 中断传送方式



(c) DMA 传送方式

图 11-9 输入输出的控制方式

四、程序传送方式

程序传送方式又可分为两种情况：即无条件传送方式和条件传送方式。

1. 无条件传送方式

无条件传送方式是指在外部控制过程的定时是固定的或是已知的条件下，进行数据传送的方式。在这种传送方式中，外设总是被认为处于"待命"状态，可以根据其固定的或已知的定时，将 I/O 指令插在程序中。当程序执行至该 I/O 指令时，就开始发送或接收数据。无条件传送是所有传送方式中最简单的一种，它所需要的硬件和软件都是最节省的。但这种方式必须已知并且确信外设已经准备就绪的情况下才能够应用，否则就会出错。

现举一个简单的系统为例。该系统接有 1 号和 2 号两个外围设备（或端口）。1 号设备是一个 8 位 A/D 转换器，2 号设备是一个 8 位 D/A 转换器。A/D 转换器的数据输入经 CPU 处理后，再由累加器输出给 D/A 转换器。这一过程可通过在某一主程序中插入数据传送的指令来实现。现用 8080 A 的汇编语言写出 I/O 控制程序如下：

```
    ⋮
IN   1    ;读入 1 号设备的数据至累加器 A
    ⋮
OUT  2    ;将累加器中的数据输出给 2 号设备
```

在此程序中，IN 1 指令将 1 号设备的数据缓冲器的内容经由数据总线送入累加器 A。发出这条指令时，CPU 不必考虑 1 号设备的状态，因为已假定该设备将输入数据准备就绪了，也即假定外设与 CPU 总能保持同步；同样，指令 OUT 2 是把累加器的内容送入 2 号设备，它也不必去测试该设备状态。换言之，CPU 可以不必关心外围设备 1# 和 2# 的状态，而无条件地执行 I/O 传送，因而称为无条件传送方式。

图 11-10 是实现这种数据交换的基本电路。

输入数据时，先从存贮器取出输入指令 IN1，然后将“设备地址”(例中是 0001)送至译码器输入端。该译码器能接通 16 种 I/O 设备。当启动脉冲加至译码器的相应允许端时，便使设备选择线 1 有效，然后在“输入选通”脉冲的作用下，经与门 3 输出，使三态缓冲器 1 开放，于是 1 号设备的数据便可通过数据总线输入至累加器。其定时波形图如图 11-11 所示。

输出数据时，先从存贮器取出输出指令 OUT 2，并将累加器的内容置于数据总线上，然后把“设备地址”(例中是 0010) 和启动脉冲送至译码器上，经译码使设备选择线 2 有效，三态缓冲器 2 开放，从而使数据总线上的信息送至锁存电路(图中的 D 触发器)的 D 输入端，然后在“输出选通”脉冲的作用下，数据进入该锁存电路，直到 CPU 给该设备发出第二个命令为止。在这里，用锁存器来协调 CPU 与 D/A 转换器在定时上的差异。其定时波形图见图 11-11。

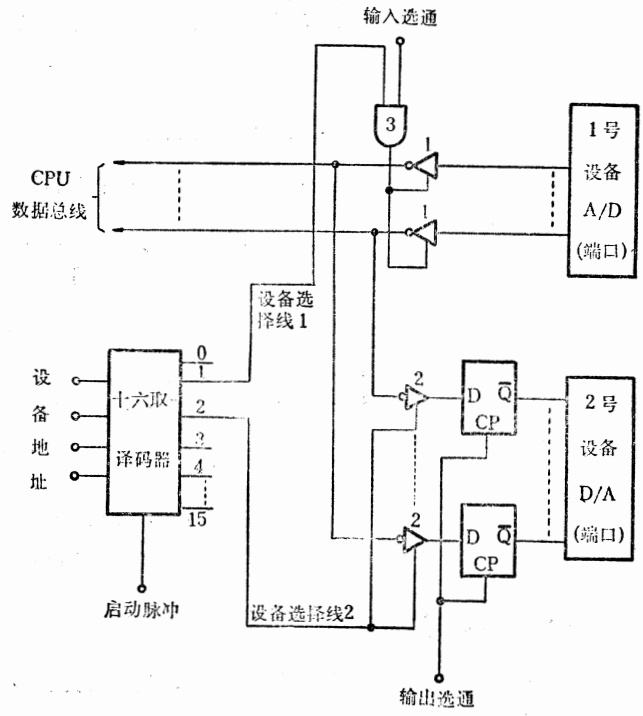


图 11-10　无条件传送电路框图

图 11-10 中的控制信号“输入选通”、“输出选通”、“启动脉冲”、“设备地址”等都是由微处理器配上适当外部逻辑电路而产生的。外部逻辑电路的结构是根据具体的微处理器来确定的。

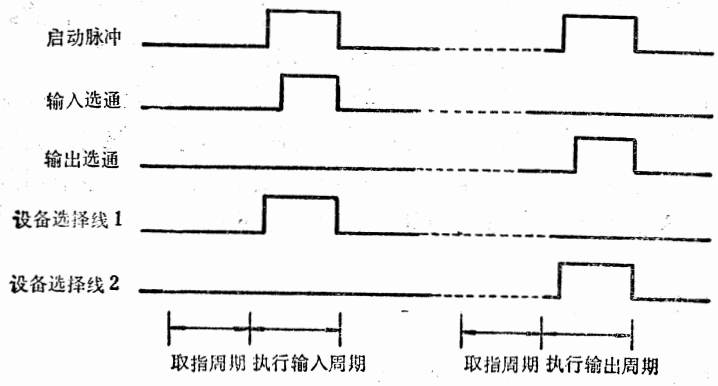


图 11-11　无条件传送的定时波形图

“设备地址”通常在 I/O 指令的执行周期内发至地址总线。此时，可以通过与门或由启动脉冲控制的三态缓冲器传送至译码器。

在上例中，我们假定输入指令在执行时，A/D 转换器已做好传送数据的准备。这就要求在执行输入指令之前，必须先启动设备，并保证有足够的时间，以便完成 A/D 转换。通过向另一个端口(如端口 3) 执行一条输出指令，该端口上的数据线就可以用来启动 A/D 转换器。

2．条件传送方式或称异步传送(Asynchronous transfer)方式、或称查询(Polling)传送方式

条件传送也称为“信息交换I/O”方式，是微型计算机系统中常用的传送方式。采用这种方式时，微型机在执行一个I/O操作之前，必须先对外围设备的状态进行测试。完成一次数据传送的步骤如下：

(1) 测试所选择的外围设备的现行状态；

(2) 根据该设备的状态进行条件转移，如该设备处于“忙”状态，则程序转至重复测试设备状态循环等待；若该设备已准备“就绪”，便发出一条I/O指令。

3．执行数据传送(输入或输出)

4．数据传送结束后，使该设备暂停

这种传送方式的流程图如图11-12所示。

为了说明查询传送方式，现举一个微型计算机系统处理8位A/D转换器输入数据的例子。设A/D转换器作为该系统的1号端口，它的状态触发器作为2号端口。2号端口与8位数据总线的第4位线相接。电路图见图11-13。

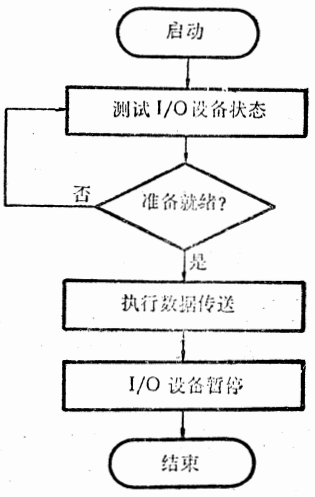


图11-12 条件传送方式的流程图

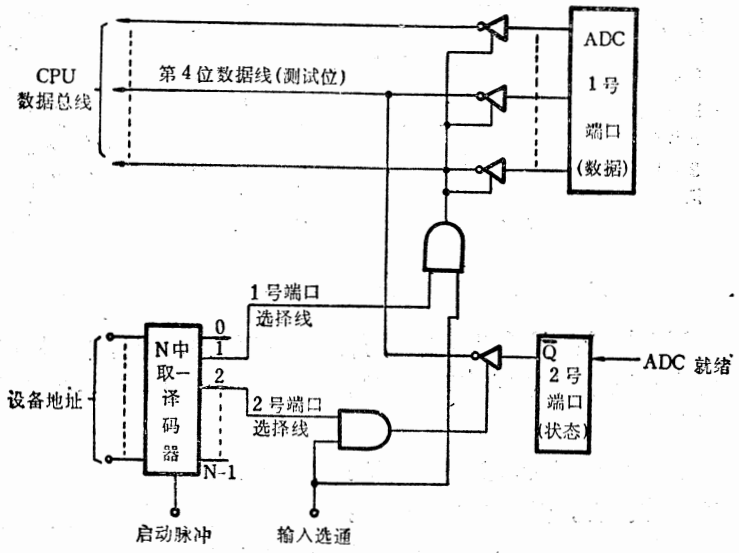


图11-13 条件传送电路图

下面用8080A汇编语言写出其数据传送的程序：

```
        ⋮
TEST:   IN      2     ；2号端口的内容送累加器A(即读入A/D转换器的状态)
        ANI     10H   ；为析取第4位而屏蔽其余各位
        JZ      TEST  ；若“忙”,则转向TEST,循环测试
        IN      1     ；否则,1号端口数据输入至累加器A
        ⋮
```

在这一段程序中，指令IN 2是将2号端口的内容(即A/D转换器的状态)送入累加器A。通过指令ANI 10H将第4位以外的各位屏蔽掉。如设备准备就绪，则第4位即为“1”，就执行IN 1指令，将1号端口的数据送入累加器A中；如设备未准备就绪，则第4位为“0”，就将微处理器的零标志位Z置“1”，然后执行JZ TEST指令，使程序循环转向TEST，又从指令IN 2

开始重复执行，直至设备准备就绪为止。随后执行 IN 1 指令，把由 A/D 转换器输入的数据送入累加器 A。

如图 11-13 所示，当发出测试设备状态的输入指令时，就将设备地址加至 N 中取 1 译码器上，该译码器允许寻找 N 个 I/O 端口。当启动脉冲送至译码器后，就使 2 号端口选择线有效，在“输入选通”脉冲作用下，将状态触发器（ADC 的状态）的状态经第 4 位数据线送入累加器。若设备状态为忙 ($\overline{Q}=1$)，则重复此测试程序；若设备状态变为“就绪” ($\overline{Q}=0$)，就执行输入指令 IN 1。此时的设备地址使 1 号设备选择线有效，“输入选通”脉冲将 1 号设备的数据经数据总线送入累加器，然后在“启动”脉冲的末尾使 1 号设备暂停，其定时波形如图 11-14。

至于 A/D 转换器的启动信号，既可由 CPU 产生，也可由外部电路发出。如果由 CPU 产生信号，则必须通过一专用输出端口发出输出命令。

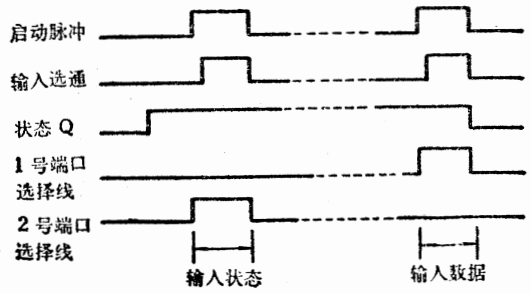


图 11-14　条件传送的定时波形图

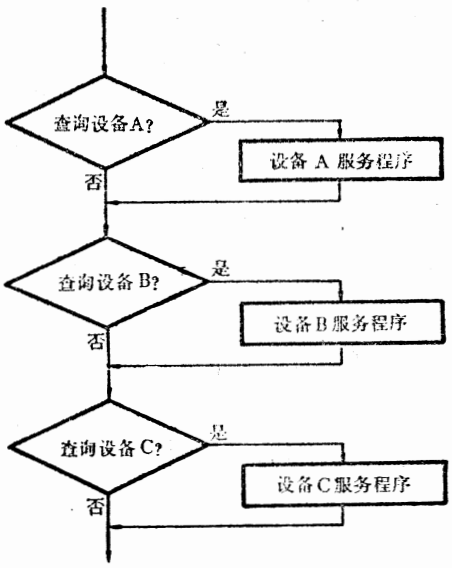


图 11-15　查询三个 I/O 设备的流程图

以上讨论的是 CPU 和一个外围设备通信的情况。如果有多个 I/O 设备，则 CPU 就要周期地依次逐个查询 I/O 设备，测试 I/O 设备接口的状态位是否准备就绪。其程序流程如图 11-15 所示。

条件传送方式的主要优点是：能较好地协调外围设备与 CPU 之间的定时差异，且用于接口的硬件较省。但其主要缺点是：CPU 必须循环等待，以不断测试外围设备的状态，直至外围设备为传送数据准备就绪为止。这样循环等待，不但浪费时间，而且在许多控制过程中是行不通的。例如，对于要求在等待传送的同时还必须维持各控制信号的场合，就不宜采用查询传送方式。为此，可采用程序中断传送方式。

五、中断传送方式

中断概念开始是为使计算机具有实时处理能力而引入的。在实时控制系统中，生产过程的信息变化是随机的，而且要求快速响应和处理，这就导致了中断处理技术的产生。中断控制方式的特点是允许 CPU 和 I/O 设备并行工作，而仅当 I/O 设备数据准备就绪之后，才向 CPU 发出中断请求的信号。此时，CPU 才暂停执行主程序，而转去执行为外围设备服务的中断服务程序，待处理完毕之后，又返回到被中断了的主程序继续执行。这就充分发挥了微型计算机的高速效能。这种原理和调用子程序相仿。不过，这里要求转移到中断处理程序的请求是由外界发出的。

有关中断传送的内容详见第十二章中断系统。

六、直接存贮器存取(DMA传送)方式

利用程序中断传送方式,在一定程度上提高了微型计算机的效率。例如,某一外设一秒钟能传送100个字节。若用查询方式输入,则在这一秒钟内,CPU全部用于查询和传送;若用中断方式,CPU传送一个字节的服务程序需100μs,则传送100个字节,CPU只需用10ms,即只占1秒的1/100。在另外的99/100的时间内,CPU仍可用于执行主程序。

但是,中断传送仍然是由CPU通过程序来传送数据的。每次传送还需要保护断点,保护现场,需要用许多指令。因而,通常传送一个字节大约需要几十μs到几百μs。这对于高速I/O设备,以及成组地交换数据的场合(例如,对于软磁盘、CRT显示终端等),高速外设与内存交换信息就显得速度太慢了。

直接存贮器存取(DMA)方式或称为数据通道方式是一种由硬件执行I/O交换的传送方式。DMA控制器从CPU处接受对系统总线的控制权后,使得存贮器和I/O设备之间能够直接进行数据交换,而不需要CPU的干预,从而大大加快了传送的速度。

DMA实际上也是采用请求-响应的方式工作的,与中断传送方式相比,DMA的优先级高于程序中断的优先级别。

目前,已有LSI的DMA控制器,尽管其复杂程度相当于一般的CPU,但使用起来还是相当方便的。

1. DMA控制器的功能

DMA控制器是在I/O设备与存贮器之间直接传送数据的过程中,代替CPU起控制作用的器件。它应具有如下的功能:

(1) 能接受从外围设备发出的DMA请求,并向CPU发出HOLD(或$\overline{\text{BUSRQ}}$)信号;

(2) 当CPU响应请求,发出HLDA(或$\overline{\text{BUSAK}}$)信号后,能接管对总线的控制,进入DMA操作方式;

(3) 能发出存贮器地址,确定数据传送的地址单元,并能自动修改地址指针;

(4) 能识别传送数据的方向,发出读或写等控制信号;

(5) 能确定传送数据的字节数,并判断DMA传送是否结束;

(6) 发出DMA操作的结束信号。

典型的DMA传送数据的流程图如图11-16所示。

能实现上述功能的DMA控制器的方框图如图11-17所示,其工作过程波形图见图11-18。

2. DMA操作的步骤

(1) DMA控制器的预置(初始化)

DMA控制器操作之前,必须给它预置如下一些信息:一是数据传送方向,即指定I/O设备要对存贮器"读"还是"写",这就是要指定其控制/状态寄存器中相应控制位(例如,设为第0位)的值;二是数据应传送至何处,这就是要指定其地址寄存器的初值(即首地址);三是有多少数据(字数)需要传送,这就是要指定其字计数寄存器的初值。

预置DMA控制器初值的工作,在DMA数据交换操作开始以前由CPU来完成。CPU可以把DMA控制器中这些寄存器看作是I/O端口,或是可寻址的存贮器。然后用适当的指令,

例如，“存数”指令或输出指令，存入所需的值。

(2) DMA 数据传送

DMA 数据传送按如下步骤进行：

① 外围设备发出选通脉冲，使输入数据送入缓冲寄存器，并使 DMA 请求触发器置“1”；

② DMA 请求触发器向控制/状态端口发出准备就绪信号，同时向 DMA 控制器发出 DMA 请求信号；

③ DMA 控制器向 CPU 发出 HOLD (或$\overline{\text{BUSRQ}}$) 请求；

④ CPU 在完成现行机器周期后，即响应 DMA 请求，发出 HLDA 信号，并由 DMA 控制器发出 DMA 响应信号，使 DMA 请求触发器复位，此时，CPU 浮动它的总线，使之进入高阻状态，于是由 DMA 控制器或相应的外围设备接管系统总线；

⑤ DMA 控制器控制系统总线，发出存贮器地址，并在数据总线上给出数据，随后在其 R/W 线上发出“读”或“写”的命令；

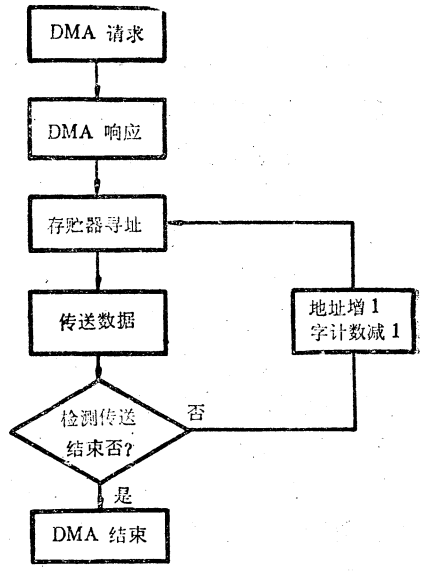


图 11-16 DMA 传送数据的流程图

⑥ 数据沿着数据总线直接在 I/O 设备与存贮器之间传送，本例是输入操作，故在存贮器写命令控制下，将 I/O 数据写入存贮器；

⑦ 每传送一个字，DMA 控制器的地址寄存器加 1，从而得到下一个地址，字计数寄存器则减 1，如此循环，直至字计数器之值为 0，数据传送完毕。

(3) DMA 结束

DMA 传送完毕时，可利用字计数寄存器为 0 的信号，由 DMA 控制器发一信号去封住微处理器用作 DMA 操作的引出端(8080 A 的 HOLD 端，Z 80 的 $\overline{\text{BUSRQ}}$ 端)，从而结束 DMA 操作。

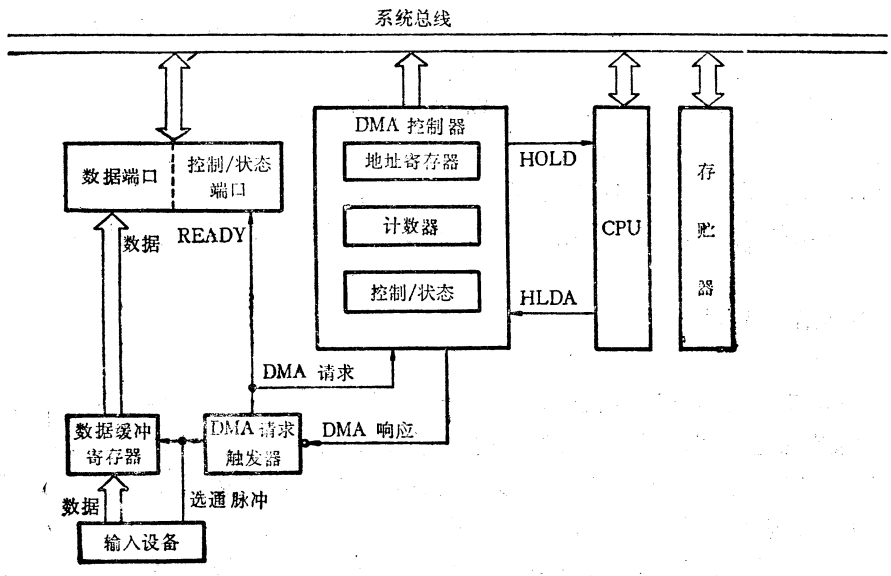


图 11-17 DMA 系统方框图

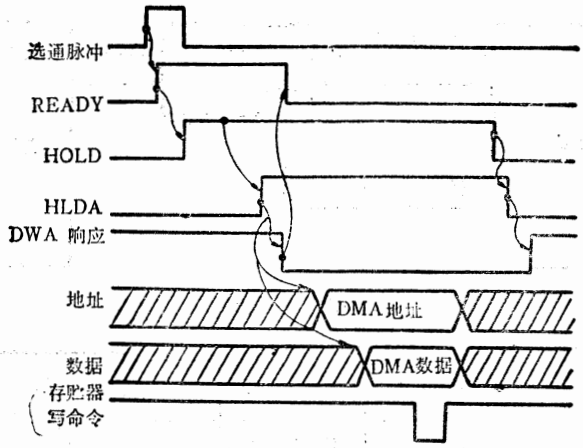


图 11-18 DMA 工作过程波形图

§11-5 输入/输出端口结构

输入/输出端口是输入/输出部分的主要组成部件。

一、单输入端口

如果外围设备的数据变化缓慢以至不需要定时或控制信号(例如来自键盘、开关、传感器的信号)，那末 CPU 和输入端口之间不必设置“就绪”、“应答”等控制信号线，而只要用数据线连接起来即可。因为只有一个输入端口，所以可不用地址线，如图 11-19 所示。

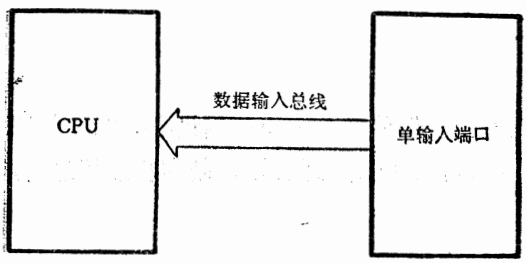


图 11-19 单输入端口

若 CPU 的字长为 8 位，而外围设备每次传送 CPU 的数据位数比 CPU 的短，假如数据位数只有 4 位，那末只要连接 4 条数据线，其余 4 条数据线可以空着不接。这可以利用指令 ANI 屏蔽无用的位。

若外围设备每次传送给 CPU 的数据位数比 CPU 的长，则需要几个端口分时传送。

二、单输出端口

输出端口与输入端口的不同点是，输出端口需要保存 CPU 输出的数据。为此，可用锁存器来锁存数据。当时钟脉冲启动锁存器时，锁存器输出的数据就等于这次时钟启动时所锁存的数据。当只有一个输出口时，CPU、锁存器、外围设备之间的连接如图 11-20 所示。

由图可知，因为单输出端口只有一个输出口，所以也可不用地址线，仅用 CPU 的写脉冲作为锁存器的时钟脉冲。CPU 通过数据输出线把数据送到锁存器，再通过锁存器输出到外围设备。

外围设备的数据位可以等于或少于 CPU 的数据线。如果少于的话，则多余的数据线不起作用。CPU 输出的 8 位数据线也可以通过锁存器接到多个外围设备上，如图 11-21 所示。

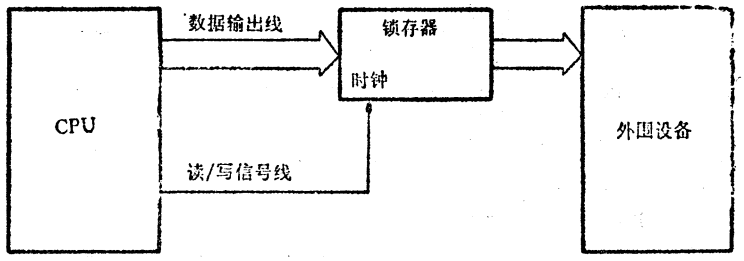


图 11-20　单输出端口

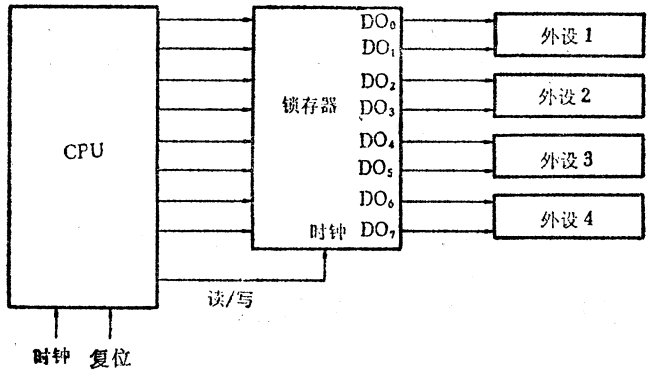


图 11-21　多个外设与单个输出端口的连接

这时，每次输出操作都是向 4 台外围设备发送 2 位数据。第 0 位和第 1 位发送给 1 号外设，其余依次类推。

三、多端口传送

在许多情况下，输入/输出部分都有若干个 I/O 端口。即使只有一个外围设备，如果外围设备的字长(比如有 12 位)超过 CPU 的字长(比如 8 位)，或需要分开传送控制和状态信息时，就需要几个 I/O 端口。

1. 多输入端口的连接

如果有几个输入端口，它们和数据总线相连时，应根据输入端口的地址，由 CPU 发出控制信号，通过地址译码使选中的输入端口和数据总线接通。图 11-22 表示多个输入端口和总线的连接方法。

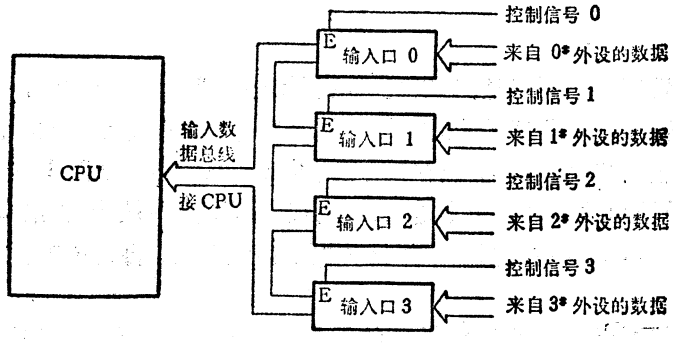


图 11-22　多个三态输入端口与总线的连接

由图可知，所有挂在数据总线上的输入端口都应通过三态门与总线相连接。当有控制信号加到启动端E时，外围设备的数据就传送到输入数据总线。当撤消控制信号时，输入端口便呈现高阻抗状态。当某一输入端口被选中时，它与CPU数据总线接通，这时其余的输入端口都处于高阻抗状态。

2．多输出端口的连接

多输出端口的连接方法如图11-23所示。输出端口应能锁存CPU输出的数据。当选中某一输出端口时，CPU输出的地址信息经译码后使该输出端口的控制信号端为高电平；其余的输出口都没有选中，它们的控制信号都为低电平。这样，当写脉冲有效(高电平)时，就可以通过被选中的控制与门到达锁存器的时钟输入端CK。此CK信号用来锁存CPU输出的数据信息，以供随时输出到外围设备中去。

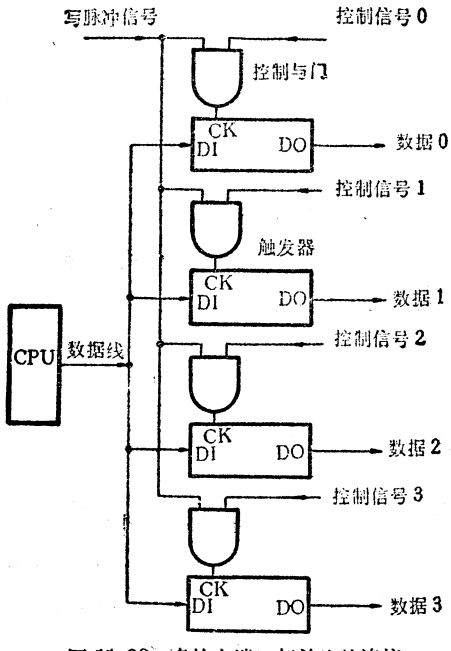


图11-23　多输出端口与总线的连接

3．多I/O端口的数据传送

下面通过两个例子来说明多个I/O端口的数据传送过程。

【例1】 设一个慢速的外围设备的数据字长为16位，CPU的字长为8位，它与CPU之间的数据传送采用无条件传送方式。

为此，CPU需要通过两个端口分时传送。CPU先通过第一个端口输入或输出高8位数据，再通过第二个端口输入或输出低8位数据，即分两次完成数据传送。

如果要求同时把16位数据输出给外围设备，则需要增加一个控制端口，如图11-24所示。

在这种情况下，CPU分两次将16位数据输出到输出端口1和2的数据锁存器，然后通过输出端口3，给三态输出缓冲器的控制端E，送出一个控制信号（本例为高电平有效）。于是16位数据就同时传送给外围设备。

【例2】 某外围设备有数据端口、数据就绪端口、输入应答端口等三个端口，采用异步传送方式传送数据，如图11-25所示。

当进行输入操作时，先由外围设备送出数据和“数据就绪”信号，并将“数据就绪”信号锁存

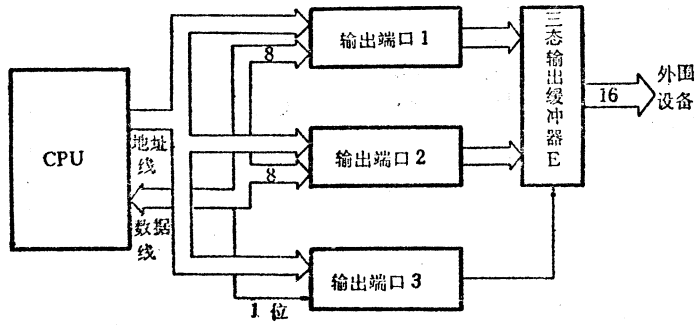


图11-24　16位数据同时输出给外围设备

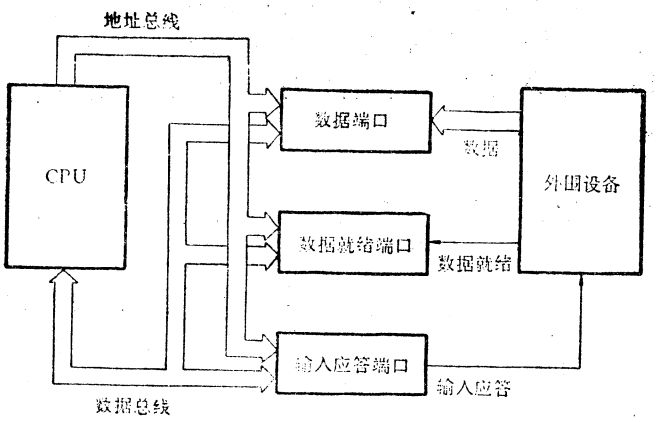


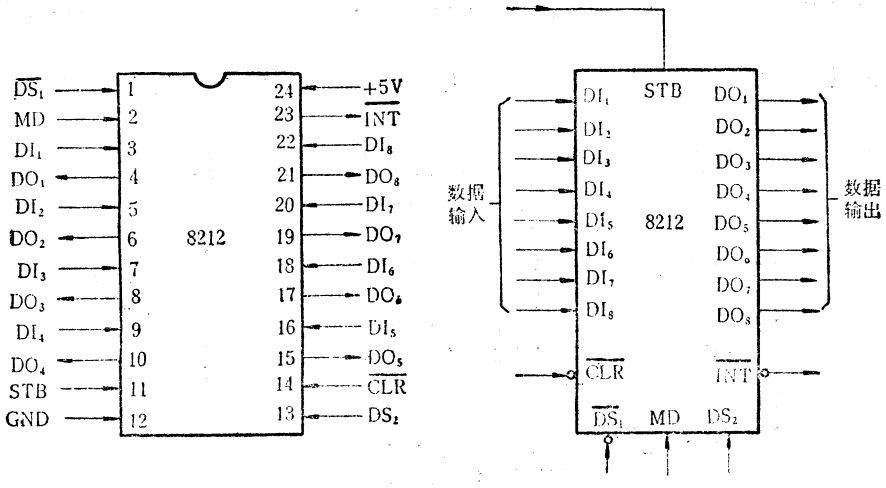
图 11-25　具有控制和状态的 I/O 端口

到数据就绪端口的锁存器中。当 CPU 查询到“数据就绪信号”时，就从数据端口读入数据，然后，CPU 送出“输入应答”信号，通知外围设备，可以送一个新的数据。在 CPU 读数据或输出“输入应答”信号时，应清除“数据就绪”信号。当外围设备准备好下一个数据时再送出“数据就绪”信号。随后重复上述输入过程。

§11-6　Intel 8212——8 位通用 I/O 接口电路

Intel 8212 是一个 8 位通用 I/O 接口芯片，用作 CPU 与 I/O 设备的接口电路。它采用 STTL 工艺，由一个具有三态输出的 8 位锁存器、一个中断请求触发器及控制逻辑电路等组成。可以用来实现微型机系统中几乎所有的 I/O 设备与 CPU 的接口电路。与后面将要介绍的可编程序 I/O 接口芯片对照，8212 是一个以硬布线逻辑来完成各种功能的接口芯片。

图 11-26(a)，(b) 分别为 8212 引线图和引线逻辑图。



(a) 8212 引线图　　(b) 8212 逻辑符号图

图 11-26　8212 的引线和逻辑符号图

引　线　名

$DI_1 \sim DI_8$	数据输入
$DO_1 \sim DO_8$	数据输出
$\overline{DS_1} \cdot DS_2$	设备选择 (Device Select)
MD	方式 (Mode)
STB	选通 (Strobe)
$\overline{INT}$	中断(低电平有效)
$\overline{CLR}$	清除(低电平有效)

一、8212 的内部结构和功能说明

8212 的内部结构逻辑框图如图 11-27 所示。

其各组成部分及信号线功能说明如下：

1．数据锁存器

它由 8 个 D 型触发器构成。当时钟输入端(C)变为高电平时，触发器的输出(Q)，跟随着数据输入(D)而变化。当时钟(C)返回低电平时，保持锁存作用，即 Q 端的信息保持不变。

数据锁存器可由一个异步的复位输入 $\overline{CLR}$ (低电平有效)来清除。

2．输出缓冲器

数据锁存器的输出 (Q) 连接到三态输出缓冲器。缓冲器有一条公共的控制线 (允许线 EN)，这条控制线既可允许缓冲器发送来自数据锁存器输出端 (Q) 的数据，也可以封锁缓冲器，迫使其输出端处于高阻状态。因此，可以把 8212 直接连接到微处理器的双向数据总线上。

3．控制逻辑

8212 具有控制输入 $\overline{DS_1}$、DS_2、MD 及 STB，这些输入信号用于控制器件的选择、数据锁存、确定输出缓冲器和服务请求触发器的状态。

(1) $\overline{DS_1}$ 和 DS_2 (器件选择)

这两个输入信号用于选择器件。当 $\overline{DS_1}$ 为低电平，而 DS_2 为高电平时，器件被选中。在选中状态时，输出缓冲器允许工作，而中断请求触发器(SR)则异步地置位。

(2) MD (方式)

MD 信号用来控制输出缓冲器的状态，并决定通到数据锁存器的时钟端输入(C)的来源。

当 MD 是高电平(输出方式)时，输出缓冲器允许工作，此时通向数据锁存器的时钟端(C)来源是器件选择逻辑($\overline{DS_1} \cdot DS_2$)。当 MD 是低电平(输入方式)时，输出缓冲器的状态是由器件选择逻辑($\overline{DS_1} \cdot DS_2$)来决定的，而通向数据锁存器的时钟端(C)来源则是 STB (选通) 输入。

(3) STB (选通)

当输入方式时(MD = 0)，此输入端用作通向数据锁存器的时钟端(C)，并同步地使中断请求触发器(SR)复位，从而使 $\overline{INT}$ 端发出中断请求。

应当指出，SR 触发器是下降沿触发的，所以 $\overline{INT}$ 信号在 STB 下降沿时变为有效 (低电平)。

4．中断请求触发器(SR)

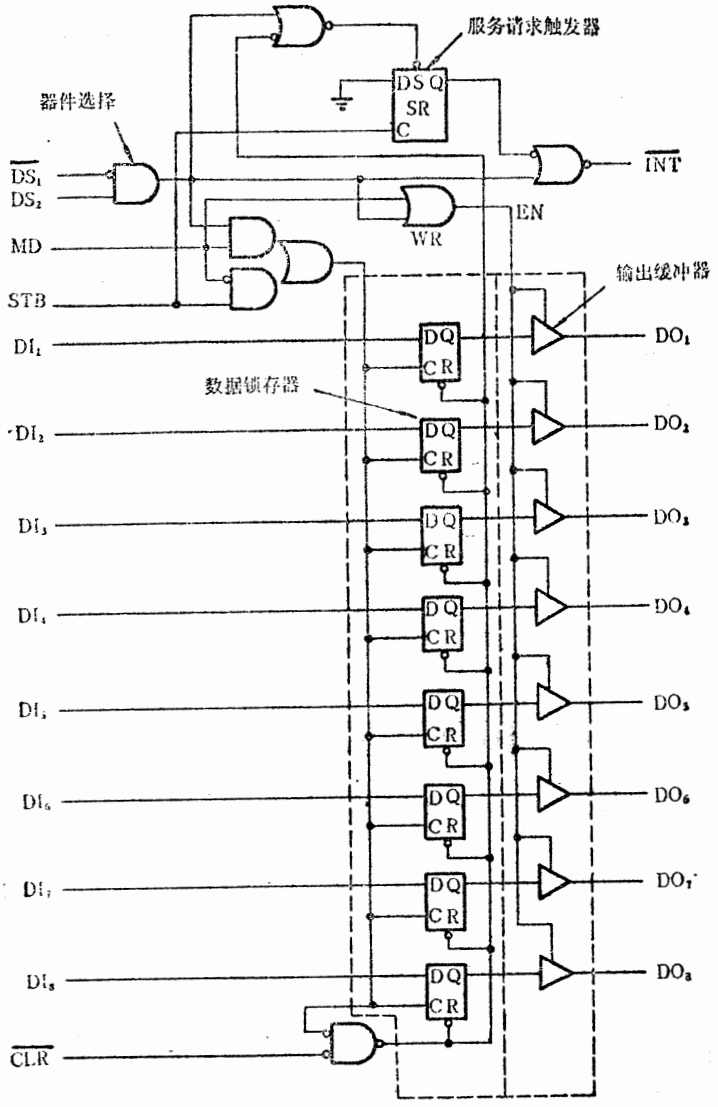


图 11-27　8212 内部结构框图

表 11-1　8212 控制信号的功能

STB	MD	$(\overline{DS_1} \cdot DS_2)$	数据输出等于
0	0	0	3 态
1	0	0	3 态
0	1	0	数据锁存器
1	1	0	数据锁存器
0	0	1	数据锁存器
1	0	1	数据输入
0	1	1	数据输入
1	1	1	数据输入

SR 触发器在微型计算机系统中用来产生和控制中断。它由 $\overline{CLR}$ 输入(低电平有效) 异步地置位。当这个 SR 触发器置 "1" 时，它处在非中断的状态。

SR 触发器的输出端(Q)连接到一个或非门(NOR)的反相输入端，或非门的另一个输入端是不反相的，它连接到器件选择逻辑($\overline{DS_1}\cdot DS_2$)。或非门的输出端 $\overline{INT}$ 为低电平有效(中断请求)，它连接到中断优先权控制电路的输入端。

上述这些控制信号的状态的不同组合及其对数据输出的影响，见表 11-1。

注意：$\overline{CLR}$ 使数据锁存器复位，SR 触发器置位，对缓冲器没有影响。

二、8212 的典型应用举例

8212 是 8 位输入/输出接口电路，并可产生中断请求($\overline{INT}$)信号，因此它在微型机系统中应用很广，现仅列举几例：

1．门控缓冲器(三态)

8212 最简单的应用是作为门控缓冲器。当方式信号为低电平(即 MD = 0)时，若选通输入为高电平(STB = 1)，则数据锁存器作用就像一个直通门。然后由器件选择逻辑 $\overline{DS_1}$ 及 DS_2 使输出缓冲器工作。门控缓冲器的逻辑图如图 11-28 所示。

当($\overline{DS_1}\cdot DS_2$) = 0 时，输出为三态；

当($\overline{DS_1}\cdot DS_2$) = 1 时，允许从系统来的输入数据直接传送给输出端。

8212 输入数据时的负载仅 250 μA，输出时的最大拉电流可为 15 mA，输出高电平的最小值为 3.65 V。

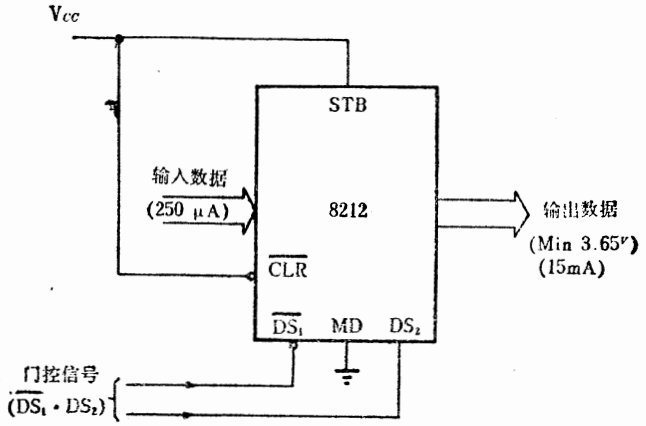


图 11-28　门控缓冲器逻辑图

2．中断输入口

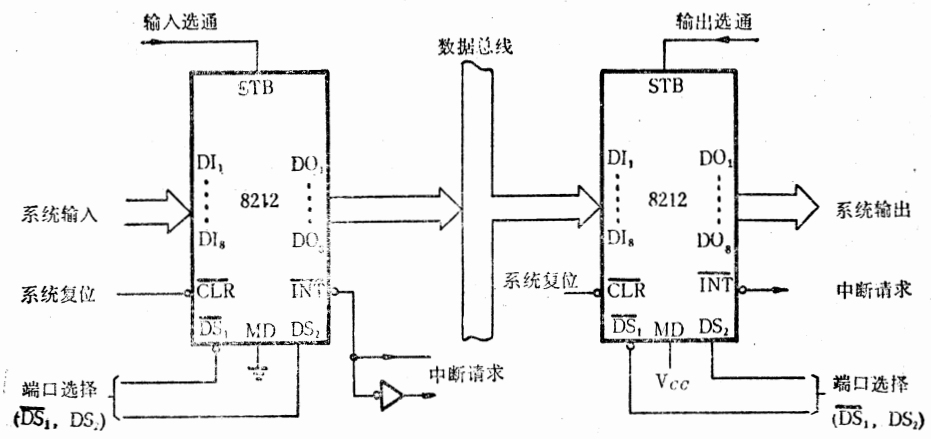
当 8212 作为输入方式时，如图 11-29(a) 所示。

此时，MD 端接地，三态输出缓冲器的状态由 $\overline{DS_1}\cdot DS_2$ 确定，STB 选通信号用作数据锁存器的时钟脉冲，如果在选通端加上输入选通脉冲信号，那末 DI_8～DI_1 上的数据将被送入锁存器。同时，这个 STB 信号通过 SR 触发器(下降沿触发)在 $\overline{INT}$ 端产生一个中断请求信号 $\overline{INT}$ (低电平有效)。如果微处理器响应这一中断请求，就转入执行中断服务程序，以形成它所要求的端口选择($\overline{DS_1}$ = 0, DS_2 = 1)，并通过数据总线从输入口读取数据。

3．输出口(有联络信号)

当 8212 作为输出方式时，如图 11-29(b)所示。

此时，MD 端接 V_{CC}(+5 V 电源或高电平)，三态缓冲器总是处在允许输出状态。数据锁



(a) 中断输入口　　(b) 输出口(带联络信号)

图 11-29　中断输入口及输出口(带联络信号)

存器的时钟信号由端口选择 $\overline{DS_1}\cdot DS_2$ 确定。若外围设备向微处理器请求输出数据，则可在 STB 端给出输出选通信号，从而在 $\overline{INT}$ 端发出一个中断请求。如果 CPU 响应中断，则转入执行中断服务程序，以形成所要求的端口选择($\overline{DS_1}=0, DS_2=1$)，并在由 $\overline{DS_1}\cdot DS_2$ 产生的时钟脉冲作用下，将数据总线上的数据锁入锁存器，再经由三态缓冲器输出给外围设备。

第十二章　微型计算机的中断系统

§12-1　概　　述

中断技术是提高微型计算机工作效率的一种重要手段。尤其是在计算机实时控制系统中，中断更是一种不可缺少的功能。本章将详细阐述微型计算机中断系统，并以 8080A/8085A 和 Z80 CPU 的中断系统为例加以说明。

一、问题的提出

早期的计算机没有中断系统。在没有中断系统的情况下，CPU 和外设之间的信息交换，是通过程序传送方式进行的。但是，常用的查询式程序传送方式，妨碍计算机高速性能的充分发挥。这样就存在着一个快速的 CPU 和慢速的外设之间的矛盾。为此，一方面要提高外设的工作速度；另一方面就产生了中断概念。

确切地说，中断概念开始是为使计算机具有实时处理的功能而引入的。在计算机实时控制系统中，现场的各种参数、信息，变化是随机的，而且要求 CPU 自动快速地响应和处理。特别是当生产过程发生紧急情况时，更必须立即申请中断，及时地采取适当的措施。这类的实时处理在采用查询传送方式下是不可能实现的。于是，就发展了中断技术。

二、中断和中断系统的概念

中断是指计算机的 CPU 暂时中止它正在执行的主程序，转而去执行请求中断的那个外设或事件的中断服务(处理)程序，待处理完毕之后，又返回到被暂时中止了的程序的这样一个过程。

计算机所具有的上述功能，就称之为中断功能。为了实现中断功能而设置的各种硬件和软件，统称为中断系统。

引起中断的事件称为中断源(Interrupt source)，一般有以下几种：

1．由硬件故障引起的，例如电源掉电；

2．由外围设备等外部要求引起的；

3．由外部事件引起的，例如实时时钟中断、控制台置某个控制开关或多机系统中的它机中断等。

计算机响应中断的过程与执行“转子”指令的过程很相似。其唯一的区别是：转子指令是程序员事先在程序中安排好的，只有执行到“转子”指令时，才转去执行子程序；而中断请求则是由外围设备随机地提出的，不可能事先安排好。因此，必须由机器自动地随机处理。此外，还可能发生几个中断源同时请求中断的情况，这就要求计算机能鉴别出它们的轻重缓急，按优先级高低分别进行处理。

三、中断的用途

1．用中断方式实现输入/输出操作

假设打印机打印一个字符需要 0.1 秒，而计算机利用指令把一个字符的信息送至打印机，只需大约十几 μs。如果没有中断功能，那么计算机送出一个字符后，只好空等着，直到打印机打完一个字符后，再送第二个字符，这样，计算机的使用效率很低。采用中断措施后，可以实现 CPU 和打印机并行工作，仅当打印机打完一个字符后，才向 CPU 发出中断请求的信号。此时，CPU 暂停执行主程序，转而执行为打印机服务的中断服务程序，待服务完毕后，又返回被中断了的主程序继续执行。这就充分发挥了计算机的高速效能。

2．传感器、开关或比较器等利用中断方式产生报警输入信号。虽然报警状态出现的机会不多，但是其要求响应的时间应当尽可能短。

3．电源发生故障(掉电)

当电压下降到正常电压以下某一值时，应立即发出中断请求信号。由于电压下降有一段时间，这段时间虽然很短，但是对于计算机来说，足够执行许多条指令，这样可以在电源掉电前，把一些重要的数据，保存在利用备用电池供电的存贮器中，或者把备用电池接通。电源掉电关系到整个系统的正常工作。因而，它应安排为中断的最高优先级。

4．控制台或人工干预

利用这种中断可以方便地对系统进行外部控制，以便进行现场维护、修复、测试和调试。例如，单步、断点等操作。

5．实时时钟

在工业控制中，常遇到时间控制。若用程序延时的方法，则这段时间内 CPU 不能处理其它工作，降低了 CPU 的利用率，因而常用外部实时时钟电路。利用实时时钟，每隔一定时间发出一次中断请求信号，执行某种操作。例如，在巡回检测系统中要求每隔一定时间检测一次温度、压力等参数。

6．多处理机系统中的各处理机之间的协调

7．控制直接存贮器存取(DMA)操作

总之，在确定中断用途时，当响应时间比事件发生之间的平均时间短得多的情况下，采用中断技术是合适的。但是，如果输入输出的数据速率较高，则中断的局限性就明显地表现出来。其主要原因是中断仍然需要在程序控制之下通过 CPU 来传送数据，其中取出指令并对指令进行译码都需要时间。而且中断本身也要求执行一些相应的操作。因此，对于要求高速数据传送的场合，就应当采用 DMA 方式传送数据。

§12-2　中断的处理过程

一、中断系统应解决的问题

1．CPU 何时检测中断请求信号？

2．CPU 在什么情况下可以响应中断？如何响应中断？

3．CPU 响应中断时，如何保存程序计数器(PC)的地址(断点地址)，以便当中断服务程序执行完毕后可返回主程序。

4．CPU如何识别中断源？

5．CPU如何转而执行中断服务程序以及从中断服务程序返回到主程序？

6．CPU如何开放和关闭中断？

7．如何保护现场和恢复现场？

8．当几个外设同时请求中断时，如何根据轻重缓急决定中断的优先权排队？

9．如何处理优先级较高的中断源去中断正在执行的优先级较低的中断服务程序(即多重中断)的问题？

以上几个问题就是中断处理过程中所要解决的基本问题。下面分别加以说明。

二、CPU 响应中断的条件

1．设置中断请求触发器

由于每个中断源向 CPU 发出中断请求信号是随机的，而大多数 CPU 都是在现行指令周期结束时，才检测有无中断请求信号发生。故在现行指令执行期间，必须把随机输入的中断信号锁存起来，并保持至 CPU 响应这个中断请求后，才可以清除中断请求。因此，要求每一个中断源有一个中断请求触发器，如图 12-1 中Ⓐ所示。

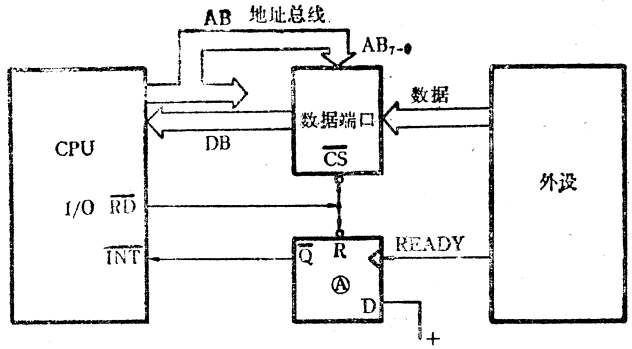


图 12-1 设置中断请求触发器

2．设置中断屏蔽触发器

为了在多个中断源请求中断的情况下，能增加控制的灵活性，常要求在每一个外设的接口电路中，设置一个中断屏蔽触发器，只有当此触发器为“1”时，外设的中断请求才能被送到 CPU，如图 12-2 中Ⓑ所示。可把 8 个外设的中断屏蔽触发器组成一个中断屏蔽寄存器端口，用输出指令来控制它们的状态。

3．设置中断允许触发器的状态

在 CPU 内部有一个中断允许触发器。只有当其为“1”时(即中断开放时)，CPU 才能响应中断；若其为“0”(即中断关闭时)，即使中断请求线上有中断请求，CPU 也不响应。可用 EI (允许中断)和 DI (禁止中断)指令来设置中断允许触发器的状态。当 CPU 复位时，中断允许触发器也复位为“0”，即关中断，故必须用 EI 指令来开中断。当中断响应后，CPU 就自动关闭中断，以禁止接受另一个新的中断(否则要处理多重中断)，因而通常在中断服务程序结束时，必须有两条指令，即允许中断指令 EI 和返回指令 RET。由于中断请求是随机的，因此，是否可能在 EI 和 RET 指令之间正好又有一个新的中断请求，而使它回不到主程序？这是不会的。因为在设计 EI 指令时，已经考虑到只有当 EI 指令的下一条指令执行之后，才能允许中断，也

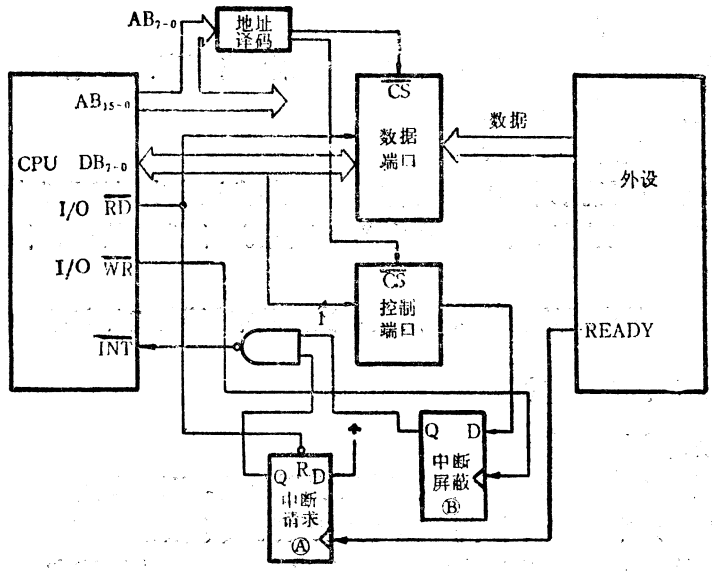


图 12-2 具有中断屏蔽的接口电路

就是说，只有在执行了 RET 指令，使其回到主程序后，才可接受新的中断请求。

4．CPU 在现行指令结束时响应中断

在满足上述条件下，如果外设有中断请求，CPU 也不是就立即响应，而是在现行指令周期的最后一个机器周期的最后一个 T 时态时，CPU 才采样中断请求线。若发现有中断请求，则把 CPU 内部的中断请求触发器置“1”，产生被 CPU 同步了的中断请求信号。然后，下一个机器周期就不进入取指周期，而转入中断周期。其时序流程如图 12-3 所示。

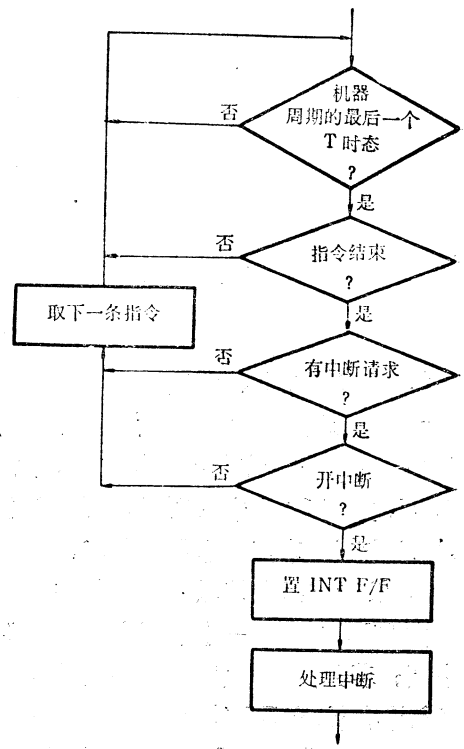


图 12-3 中断时序流程图

三、CPU 响应中断时的情况

当满足上述条件后，CPU 就响应中断，转入中断周期。中断响应的顺序如下：

1．在现行指令执行完毕时，CPU 向请求中断的外设发出中断响应信号。如 8080 A/8085 A 发出中断响应信号 $\overline{\text{INTA}}$；Z80 发出中断响应信号 $\overline{M_1}\cdot\overline{\text{ORQ}}$。

2．在响应中断后，CPU 在发出中断响应信号的同时，自动关闭中断允许触发器，以禁止接受另外的中断请求。

3．保护断点，即保存程序计数器 PC 的内容（断点地址）。CPU 响应中断后，封锁 PC＋1，并且把 PC 推入堆栈保存起来，以便中断处理完毕后，能返回主程序继续执行。

4．给出中断入口地址（即中断服务程序的起始地址）。程序转移到相应的中断服务程序。

8080A是由中断源提供的RST指令形成中断入口地址的；Z80有三种不同的中断方式，详细情况在后面介绍。

5．保护现场。为了使中断服务程序不影响主程序的运行，必须把断点处的各有关寄存器的内容和标志寄存器的状态，推入堆栈保护起来。8080A/8085A和Z80都是由软件（即在中断服务程序中）把需要保护的寄存器的内容用PUSH指令推入堆栈。MC6800在响应中断时，保护现场和恢复现场都是由CPU自动进行的，不需要在中断服务程序中编写这方面的程序。

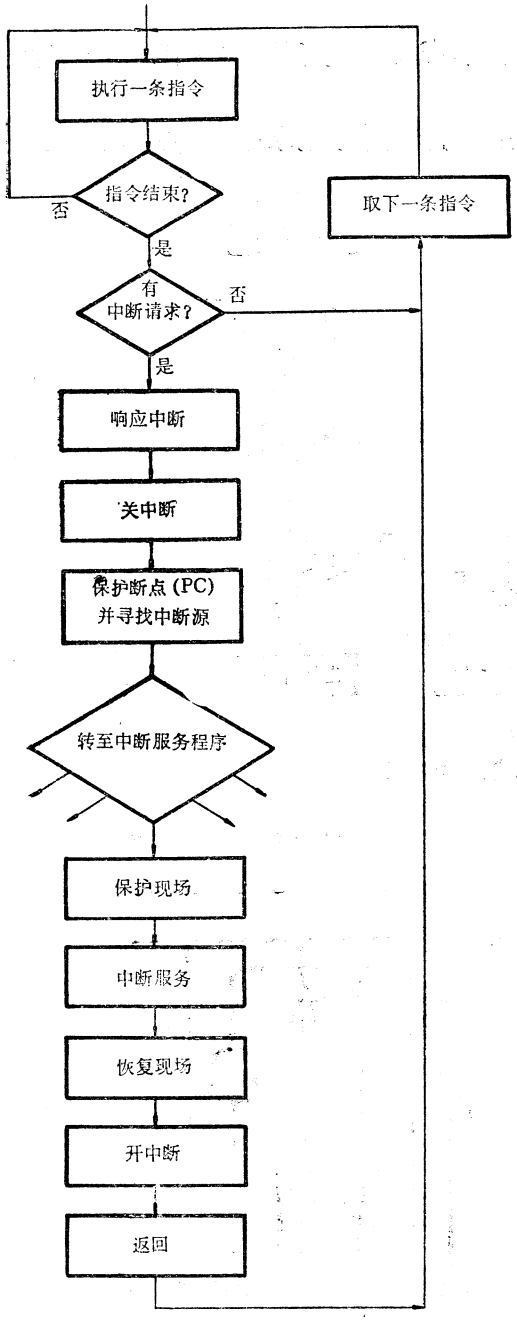


图 12-4　中断传送的流程图

6．执行中断服务程序，即在软件控制下，实现CPU和外设的数据交换。在中断服务程序完成后，还要做下述的操作。

7．恢复现场，即把所保存的各个寄存器的内容和标志寄存器F的状态，从堆栈中弹出，送回CPU中相应的寄存器。对于8080A/8085A和Z80，这个操作是在中断服务程序中用POP指令来完成的。

8．开中断与返回。在中断服务程序的末尾，通常要安排两条指令即EI和RET。EI指令用来开中断，以便CPU能响应新的中断请求；利用RET指令将堆栈内保存的PC的内容弹出，送回到PC，这样，程序就返回原来的断点处，继续执行下去。

上述中断的全过程可用图12-4来表示。

四、中断源的识别

当CPU响应中断时，在关中断和保存断点后，就要解决如何寻找中断源，找到相应的中断服务程序的入口地址。

识别中断源有两种方法：即查询中断和矢量中断。下面分别予以说明。

1．查询中断

应当指出，前面介绍的程序传送方式的等待循环“查询”与现在介绍的“查询中断”是不同的。前者CPU要花费很多时间不断地循环询问，以等待设备的“就绪”信号；后者是一种中断处理方式，在设备没有提出中断请求时，CPU照常执行主程序，只有在接收到外围设备的中断请求后，CPU才去“查询”它们，以寻找申请中断的设备，因此它不会在查询循环中浪费大量时间。查询中断主要采用软件查询方法。

软件查询方法是用程序查询接在该中断线上的每一个外围设备。查询程序依次读出每一设备的中断状态位(标志位),并通过测试这一状态位来判断它是否曾发出了中断请求,直至检测到中断状态位为"1"时,就找到了请求中断的外设。然后程序转向相应的中断服务程序,在程序控制下进行数据交换。一个管理三个外设的简单询问测试程序的流程图及示意图如图12-5所示。

假如三次测试都未发现有中断请求,则表示中断输入线的信号是由于错误引起的,这时便转向出错出口。

图12-6是实现软件询问的一种电路。图中各个外设的中断请求都可以通过图下方的或

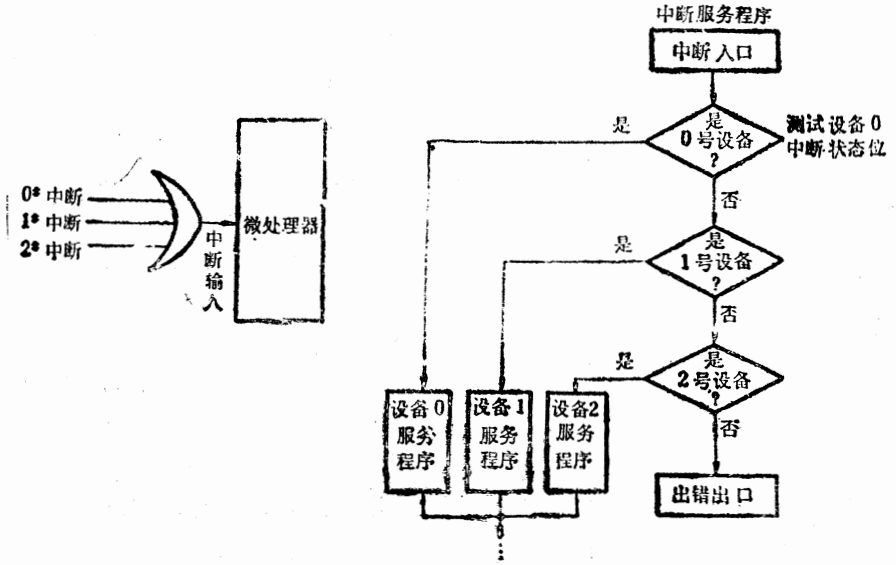


图12-5 软件查询式识别中断源

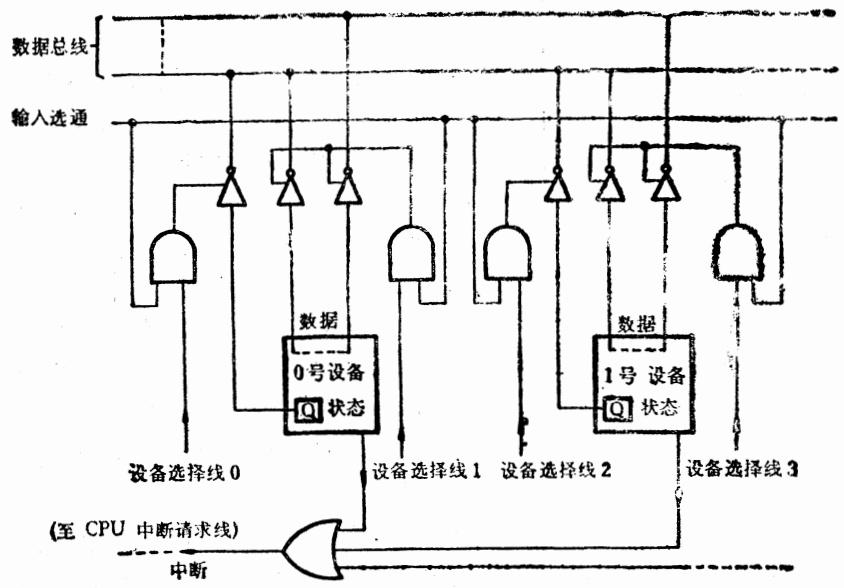


图12-6 软件查询中断的一种电路

门送至 CPU 的中断请求线上。CPU 响应中断后，自动保存 PC 的内容及 CPU 的状态，然后用查询法寻找中断源。它首先查询 0 号设备，在地址总线上发出 0 号设备状态寄存器地址，经地址译码(图中未画)后，使设备选择线 0 有效。在输入选通信号作用下，0 号设备的中断状态位经数据总线送 CPU 进行测试。若测试结果为“1”，则 CPU 知道 0 号设备有中断请求，它将转到 0 号设备的中断服务程序中去进行数据交换。此时，CPU 将发出 0 号设备的数据寄存器地址，并使设备选择线 1 有效。在 CPU 发“读”选通信号时，就可将 0 号设备的数据经数据总线读入 CPU。若 0 号设备中断状态位测试结果为“0”，表示它没有请求中断，则往下查询 1 号设备，CPU 在地址总线上发出新的地址，使设备选择线 2 有效，同样又读入其中断状态位进行测试，照此顺序，直至找到请求中断设备为止，然后转入中断服务。这里，每一设备的优先权是由在查询程序中其所处的位置决定的，显然设备“0”的优先权最高。

软件查询方法仅适用于中断源较少的系统，否则，识别中断源将花费 CPU 许多时间。运用软件技巧可以先查询出现频度较高的中断源或者每次查询一组中断源，这样可以减少软件开销。进一步的改善措施是采用矢量中断方法。

2. 矢量中断(Vectored Interrupt)

在具有矢量中断的微型计算机系统中，每个 I/O 设备都预先指定一些不同的设备中断地址或中断矢量(注意这个设备中断地址与该设备地址是不同的)。当 CPU 识别出某个 I/O 设备请求中断并予以响应时，控制逻辑就将该 I/O 设备的设备中断地址送入 CPU，以自动提供

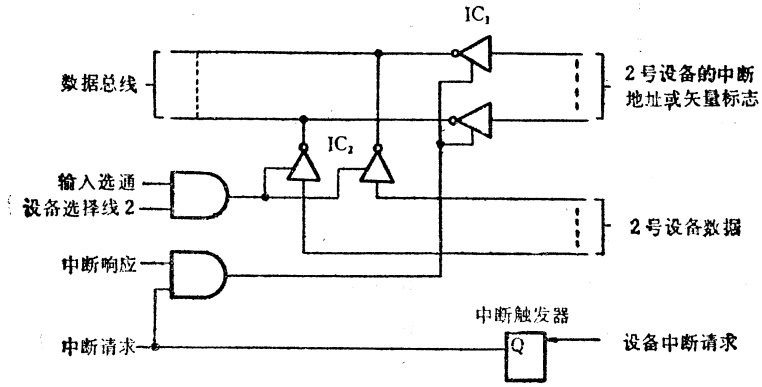


图 12-7　矢量中断传送电路

相应的中断服务程序的入口地址，转入中断服务。如果说查询中断确定中断源的方法主要是用软件实现的话，那末矢量中断确定中断源的方法则主要是用硬件来实现的。对矢量中断言，当发生中断时，它不必去查询中断，而是直接转到相应的固定的地址单元，执行相应的中断服务程序。因而，矢量中断比查询中断能更快地响应中断。

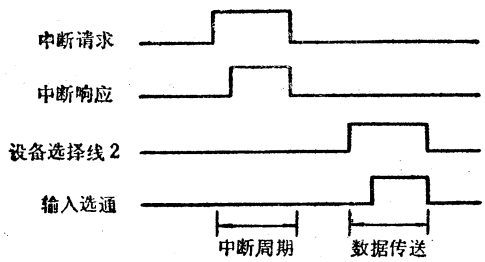


图 12-8　中断传送的定时波形图

图 12-7 是实现矢量中断的一种基本电路。

假定2号设备作为请求中断的设备，在响应中断请求时，“中断响应”信号使三态缓冲器控制门 IC_1 开放，将 2 号设备的中断地址或称矢量标志送往数据总线。这样就确定了请求中断的设备 2，并使程序转移至相应的中断服务程序。执行这一服务程序，可使设备选择线 2 有效，同时再

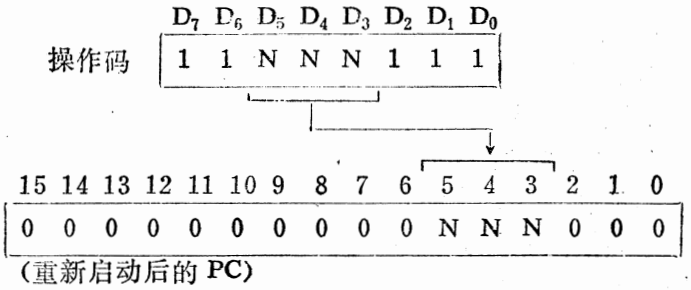
用输入选通信号("读"命令)将三态缓冲控制门 IC_2 开放，使该设备的数据经数据总线送入累加器 A，从而完成了数据传送。该电路传送信息的定时波形如图 12-8 所示。

3．重新启动指令 RST(Restart)

Intel 8080 A 和 Z 80 的中断方式 0 都是采用矢量中断方式。它们是利用特殊的单字节的 RST 指令，自动地提供相应的中断服务程序的入口地址，转入相应的中断服务程序。

(1) RST 指令格式

RST p



其中 NNN 为三位二进制码，其范围为 000～111，故 RST 指令形式有 8 种组合。

RST 指令的功能如下：

$(SP-1)\leftarrow PC_H$ ；返回地址高 8 位送 SP－1 存贮单元保存

$(SP-2)\leftarrow PC_L$ ；返回地址低 8 位送 SP－2 存贮单元保存

$SP\leftarrow SP-2$ ；堆栈指针指向 SP－2 的栈顶

$PC\leftarrow 8\times NNN$ ；控制转移到零页地址

因此，对应于 RST 指令的 8 条例行子程序的入口地址分别为

RST p	NNN	入口地址 (H)
RST 00 H	0 0 0	0 0 0 0
RST 08 H	0 0 1	0 0 0 8
RST 10 H	0 1 0	0 0 1 0
RST 18 H	0 1 1	0 0 1 8
RST 20 H	1 0 0	0 0 2 0
RST 28 H	1 0 1	0 0 2 8
RST 30 H	1 1 0	0 0 3 0
RST 38 H	1 1 1	0 0 3 8

由此可见，重新启动指令 RST 能调用位于存贮器前 64 个字节的 8 个例行子程序中的任意一个。每一个入口有 8 个单元，如果中断服务程序较长，则可在 8 个入口处各放一条转移指令，以转至中断服务程序的入口。

(2) 重新启动指令 RST 转移流程举例，如图 12-9 所示

(3) RST 指令的时序

当CPU执行主程序时，若中断是开放的并且接收到了中断请求，则在现行指令最后一个机器周期的最后一个 T 时态，采样到 INT 信号，于是 CPU 发出中断响应信号 $\overline{INTA}$，然后在下一机器周期进入中断周期，其过程如下：

M_1，T_1：$AB_{15\sim0}\leftarrow PC$ ；程序计数器的内容被锁存到 CPU 的地址总线上。

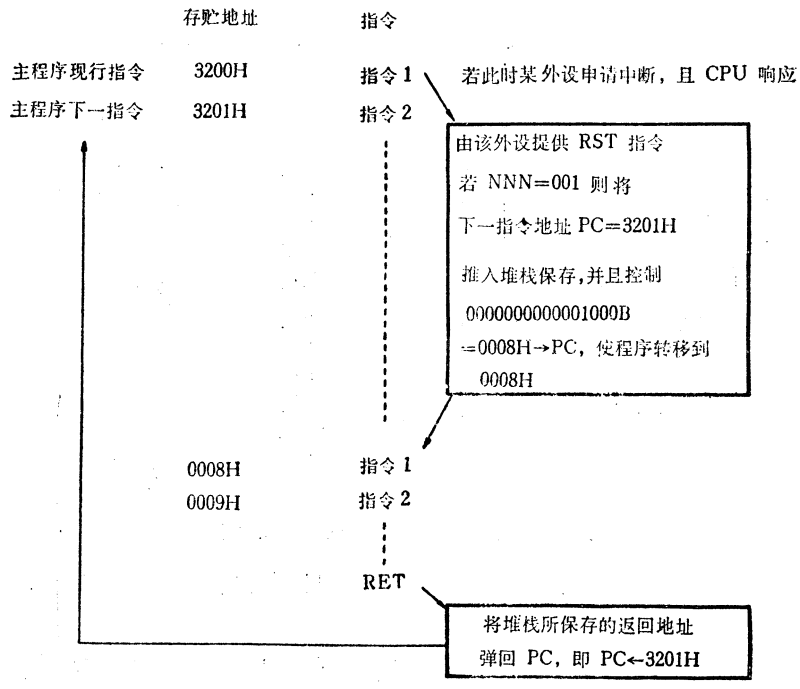


图 12-9 RST p 指令的转移流程图

M_1, T_2: 封锁 PC+1 的操作 ; 程序计数器保持中断前的状态，即中断返回地址。

M_1, T_3: W←0, IR←INST ; CPU 从数据总线上读取中断矢量。

在中断周期内存贮器读信号被封锁，不执行存贮器读操作。但是 CPU 仍在 T_3 时态采样数据总线以读取指令。这时数据总线上的指令不是由存贮器读出的，而是由请求中断的中断源在 CPU 响应中断时，将一条 RST p 指令插入 $DB_{7\sim0}$，在 T_3 时 CPU 将其送至指令寄存器 IR，然后经过译码，转入 RST 指令的执行周期。

M_1, T_4、T_5: SP←SP−1 ; 为把 PC_H 推入堆栈提供地址。

M_2, T_1: $AB_{15\sim0}$←SP ; 给出堆栈指针，实现堆栈写周期。

M_2, T_2: SP←SP−1

M_2, T_3: $DB_{7\sim0}$←PC_H ; 将 PC_H 作为数据推入堆栈保存。

M_3, T_1: $AB_{15\sim0}$←SP

M_3, T_2: Z←TMP = 00NNN000 ; 在 WZ 中形成中断矢量入口地址。

M_3, T_3: $DB_{7\sim0}$←PC_L ; 将 PC_L 作为数据推入堆栈保存。

此时，暂存器 WZ 的内容为：

W = 00000000, Z = 00 NNN 000

在下一条指令的

M_1, T_1: $AB_{15\sim0}$←WZ ; 进入取指周期。

M_1, T_2: PC←WZ+1 ; 为保证从中断入口继续执行中断服务程序中的第二条指令作好准备。

M_1, T_3: 存贮器读 ; 按中断入口地址读出中断服务程序第一条指令的第一个字节。

(4) RST 指令的形成

当 CPU 响应中断后，不是根据 PC 指出的地址到存贮器中去取指令，而是由申请中断的

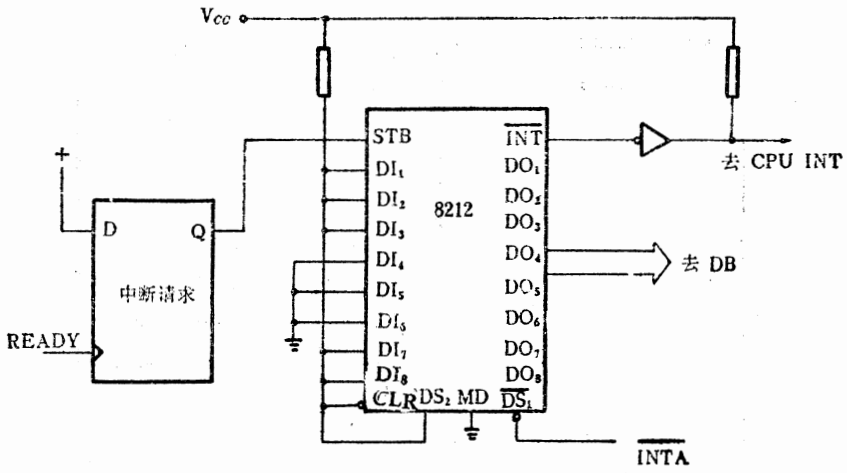


图 12-10　形成 RST 指令的硬件逻辑电路

外设用硬件逻辑来提供 RST 指令，经数据总线输入到指令寄存器 IR 中去，其硬件逻辑如图 12-10 所示。

图中形成的是 RST 0 指令，只要改变 8212 的 DI_4、DI_5、DI_6 的接法(接 V_{CC} 或接地)就可以得到 RST 0～RST 7 中的任一种。显然 NNN 值可用一个 8-3 编码器来获得。

五、中断的优先权

在实际系统中，常常遇到多个中断源同时请求中断的情况，这时 CPU 必须确定首先为哪一个中断源服务，以及服务的次序。解决的方法是用中断优先排队的处理方法。这就是根据中断源要求的轻重缓急，排好中断处理的优先次序，即优先权(Priority)，先响应优先权级别最高的中断请求。有的微处理器有两条或更多的中断请求线，而且已经安排好中断的优先权，如 Intel 8085 A、Z80、M 6800 等。但有的微处理器如 Intel 8080 A 只有一条中断请求线。凡是遇到中断源的数目多于 CPU 的中断请求线的情况时，就需要采取适当的方法来解决中断优先权的问题。

另外，当CPU正在处理中断时，也要能响应优先权更高的中断请求，而屏蔽掉同级或较低级的中断请求，即所谓多重中断的问题。

通常，解决中断的优先权的方法有以下几种：

1．软件查询优先方式

软件查询优先方式是最简单的中断优先处理方式。图 12-11 示出用软件查询方式的接口电路，把 8 个外设的中断请求触发器组合为一个端口，并赋以设备号；然后，把各个外设的中断请求信号相“或”后，作为 INT 信号，故

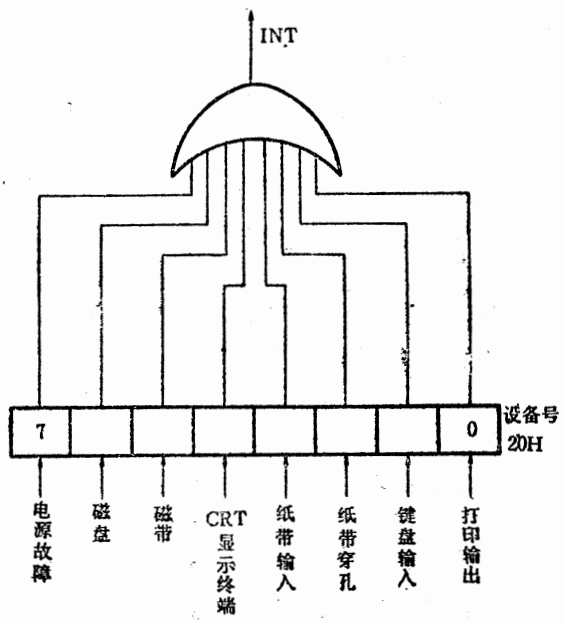


图 12-11　软件查询优先方式的接口电路

当任一外设有中断请求时，都可以向 CPU 送出 INT 信号。当 CPU 响应中断后，把中断寄存器的状态，作为一个外设读入 CPU，然后逐位检测它们的状态。若发现有中断请求就转至相应的中断服务程序的入口。这样，当查询程序用条件转移指令顺序测试各中断源的中断状态位，以寻找中断源的同时，就自然解决了中断的优先权问题。优先权的级别取决于查询程序的顺序，先测试的中断源的优先权比后测试的中断源的级别高。对于图 12-5 的情况，显然，三个设备的中断优先权的顺序是：0 号设备优先权最高；1 号设备次之；2 号设备最低。

软件查询优先方式的优点是硬件简单，不需要有判断和确定优先权的硬件排队电路。但是，在中断源较多的情况下，软件查询的时间较长。

查询程序有两种方法：

(1) 屏蔽法

```
IN    A, (20H)    ; 输入中断请求触发器的状态
AND   80H         ; 查询 D7 是否为 1
JP    NZ, PWF     ; 是，则转至电源故障服务程序入口
IN    A, (20H)
AND   40H         ; 查询 D6 是否为 1
JP    NZ, DISS    ; 是，则转至磁盘服务程序
⋮
```

(2) 移位法

```
XOR   A           ; 清 A 及进位位
IN    A, (20H)
RLA               ; D7 左移至 CY
JP    C, PWF      ; D7 = 1, 转至电源故障服务程序
RLA               ; D6 左移至 CY
JP    C, DISS     ; D6 = 1, 转至磁盘服务程序
⋮
```

图 12-12 是一个用软件来实现优先中断的基本优先管理电路。它可管理 8 个中断(如图中的 INT_0～INT_7)。其内部有一个 8 位的中断屏蔽寄存器，仅当其中的某一位为“1”时，才允许相应的中断请求信号通过。如果要使用所有的中断线时，可使中断屏蔽寄存器中的各位都置“1”。当有中断请求时，未被屏蔽的中断寄存器的相应位将被置“1”。中断寄存器的内容可由 IN 指令读至数据总线(见图上部的相应位)。由中断寄存器的各条输出线相“或”所得到的中断请求信号(见图左边)被送至微处理器的中断请求端。

如果几个设备同时请求中断服务，它们的优先权可采用软件查询的方法确定。

2．硬件查询方式

常用的硬件查询技术称为优先级中断链(daisy-chaining)。使用这种方法时，可用硬件逻辑查询来代替上述的软件查询，某框图如图 12-13 所示。

当中断请求得到响应时，“中断响应”信号就传送到优先权最高的 I/O 设备，并按串行方式往下传送。如某设备有中断请求，“中断响应”信号就不再往下传送，而使之终止在该设备上，从而允许该设备使用总线与微处理器交换信息。

能实现这种逻辑要求的一种优先级中断链电路如图 12-14 所示。

来自 CPU 的“中断响应”信号从 0 号设备开始串行地往下传送。若 0 号设备有中断请求，则“中断响应”信号将在门 2 处被封锁，于是它就不能再往下传送而终止在 0 号设备上，这使得

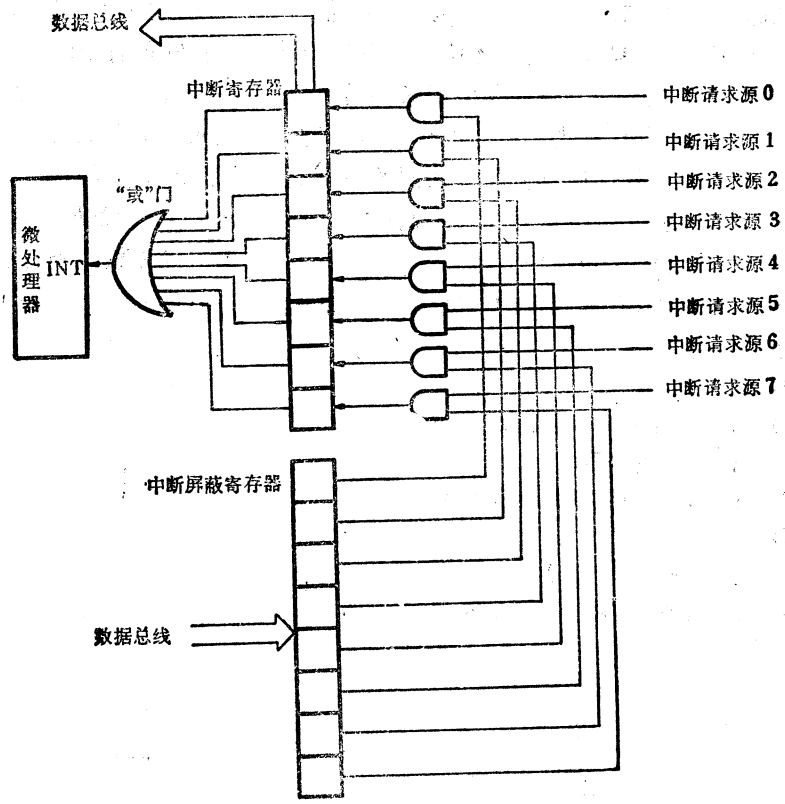


图 12-12　中断优先权管理电路

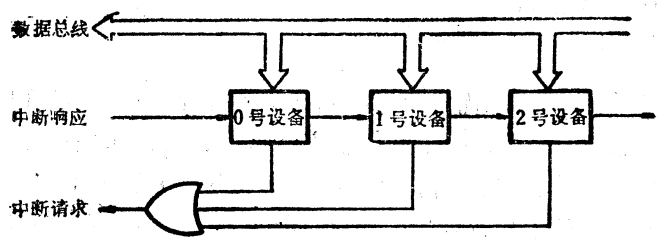


图 12-13　优先级中断链查询系统框图

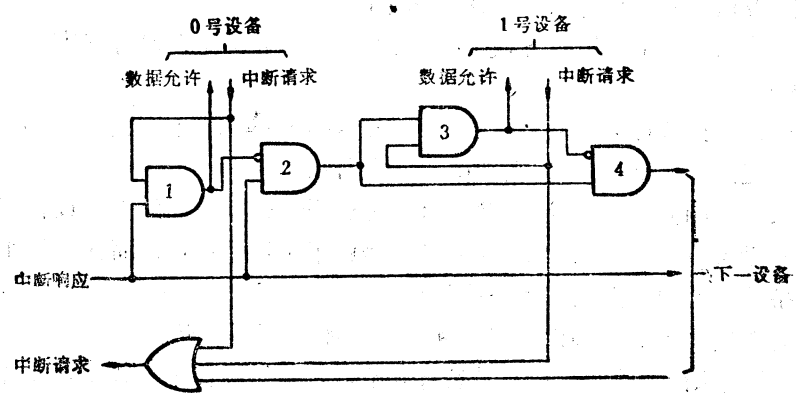


图 12-14　优先级中断链电路图

在它后面的设备如1号设备等都因得不到CPU的“中断响应”信号而被屏蔽。同时0号设备的“数据允许”线变为高电平，从而允许该设备能够使用数据总线，将其设备的中断标志放在数据总线上并送入CPU。由此，CPU就找到请求中断的设备，转到执行为它服务的中断服务程序中去。若0号设备没有中断请求，这时“中断响应”信号可通过门1、2传给下一个设备，即1号设备，若1号设备也没有中断请求，则“中断响应”信号又往下传送，直至传到有中断请求的设备为止。该设备就是微处理器所寻找的中断源。

由于优先级中断链本身就能识别信号，因此实际上它是一种矢量中断。

优先级中断链又称为链式优先权排队电路。如图12-14所示，当中断响应信号串行通过所有I/O设备时，它们的中断优先权由其在链式排队电路中的先后顺序来决定。显然，在链式排队电路中，排在链的最前面的，即最靠近CPU的设备的优先权最高。

3．中断优先权编码电路

由硬件编码器和比较器组成的中断优先权排队电路如图12-15所示。

设有8个中断源，当任何一个有中断请求时，通过“或”门，即可产生一个中断请求信号，但它能否送至CPU的中断请求线，还必须受比较器的控制。

8条中断输入线的任一条，经过编码器可以产生三位二进制优先权编码$A_2A_1A_0$，优先权最高的中断输入线的编码为111，优先权最低的中断输入线的编码为000。而且若有多个中断输入线同时输入，则编码器只输出优先权最高的编码。

正在进行中断处理的外设的优先权编码，由CPU通过软件，经数据总线送至优先权寄存器，

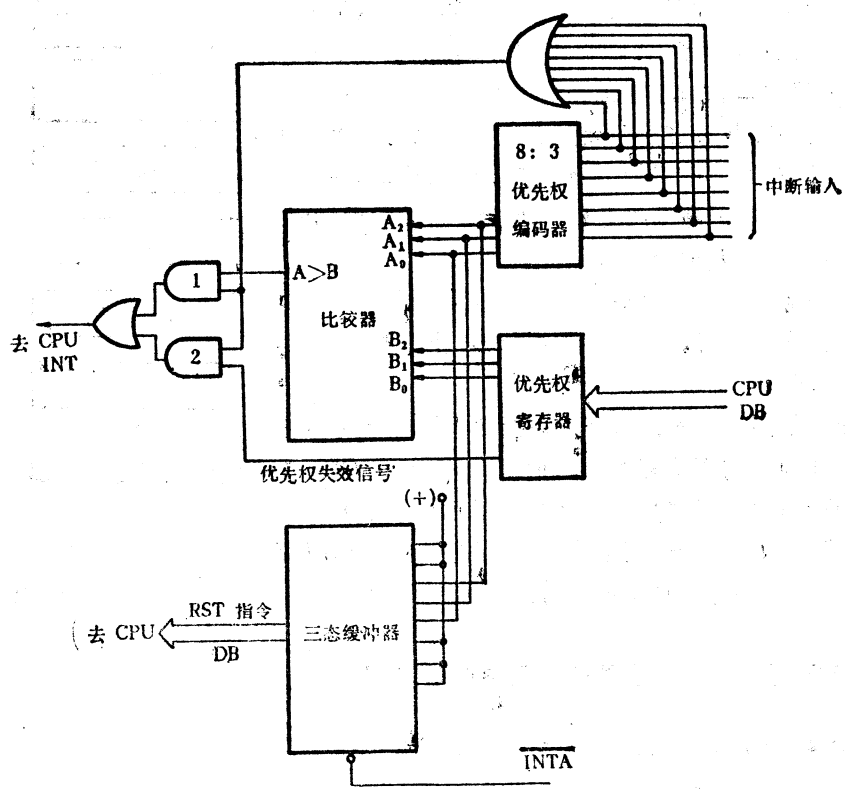


图12-15　编码器和比较器的优先权排队电路

然后输出编码 $B_2B_1B_0$ 至比较器。

比较器对编码 $A_2A_1A_0$ 与 $B_2B_1B_0$ 的大小进行比较，若 A≤B，则“A＞B”端输出低电平，封锁与门 1，禁止向 CPU 发出新的中断请求；只有当 A＞B 时，比较器输出端才为高电平，打开与门 1，将中断请求信号送至 CPU 的 INT 输入端；当 CPU 响应中断后，就中断正在进行的中断服务程序，转去执行优先权更高的中断服务程序。

若 CPU 不在执行中断服务程序时(即在执行主程序)，则优先权失效信号为高电平，此时，如有任一中断源请求中断，都能通过与门 2，向 CPU 发出 INT 信号。

当外设的个数≤8 时，它们可以公用一个产生 RST 指令的电路，RST 指令码中的 NNN 由编码器的编码 $A_2A_1A_0$ 提供，以实现对于不同的编码转入不同的入口地址。

4．中断的多级嵌套

当 CPU 执行优先权级别较低的中断服务程序时，允许响应比它优先权级别高的中断源请求中断，而挂起正在处理的中断，这就是中断嵌套或称多重中断。此时，CPU 将暂时中断正在进行着的级别较低的中断服务程序，优先为级别高的中断源服务，待优先权高的中断服务结束后，再返回到刚才被中断的较低的那一级，继续为它进行中断服务。

多重中断流程的编排与单级中断的区别有以下几点：

(1) 加入屏蔽本级和较低级中断请求的环节。这是为了防止在进行中断处理时不致受到来自本级和较低级中断的干扰，并允许优先权比它高的中断源进行中断。

(2) 在进行中断服务之前，要开放中断。因为如果中断仍然处于禁止状态，则将阻碍较高级中断的中断请求与响应，所以必须在保护现场、屏蔽本级及较低级中断完成之后，开放中断，以便允许进行中断嵌套。

(3) 中断服务程序结束之后，为了使恢复现场过程不致受到任何中断请求的干扰，CPU 必须执行 DI 指令，将中断关闭，才能恢复现场。

(4) 恢复现场后，应该执行开中断指令 EI，重新开放中断，以便允许任何其它等待着的中断请求有可能被 CPU 响应。应当指出只有在执行了紧跟在 EI 指令后面的一条指令以后，CPU 才重新开放中断。一般紧跟在 EI 指令后的是返回指令RET，它将把原来被中断的服务程序的断点地址弹回 PC，然后 CPU 才能开放中断，响应新的中断请求。

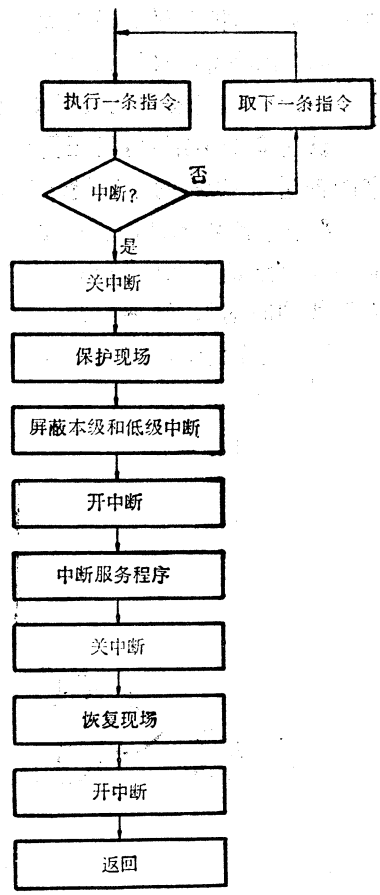


图 12-16 多个中断源，单一中断请求线的流程图

多个中断源、单一中断请求线的流程图如图 12-16 所示。

六．Intel 8214 优先权中断控制器及其应用

1．8214 的结构和原理

8214 是一个 8 级优先权中断控制器，它的方框图和引线图如图 12-17 和图 12-18 所示。

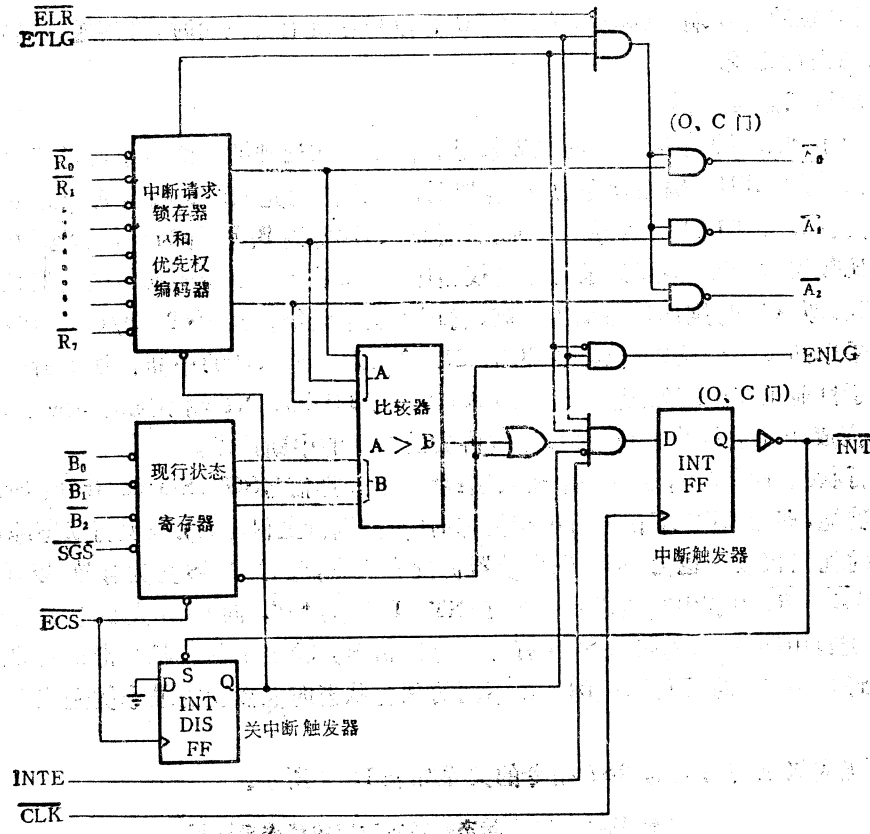


图 12-17 8214 的方框图

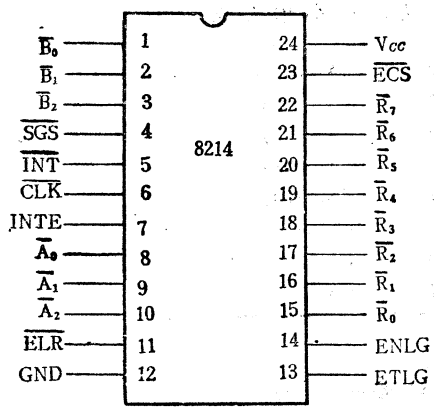


图 12-18 8214 的引线图

图中：$\overline{R_7}$～$\overline{R_0}$——中断请求级别（$\overline{R_7}$ 优先权最高）

$\overline{B_2}$～$\overline{B_0}$——现行状态

SGS——状态组选择

$\overline{ECS}$——允许现行状态

INTE——中断允许

$\overline{CLK}$——时钟(INT F/F)

$\overline{ELR}$——允许级别读

ETLG——允许这个级别组

$\overline{INT}$——中断请求

$\overline{A_2}$～$\overline{A_0}$——请求级别

ENLG——允许下一个级别组

从图 12-17 可知，8214 主要由中断请求锁存器和优先权编码器、现行状态寄存器以及优先权比较器等三个部分组成。

(1) 中断请求锁存器和优先权编码器

它可以接受多至 8 个的中断请求(低电平有效)，$\overline{R_7}$ 优先权最高，$\overline{R_0}$ 优先权最低。

编码器可把每一个中断请求输入编为 3 位二进制数的相应的优先权编码 000～111。当有两个或两个以上的中断请求同时输入时，编码器将把其中具有最高优先权级别的中断请求编码成相应的 3 位二进制码(A_2、A_1、A_0)经 O.C 门输出，作为重新启动指令 RST p 中的 NNN

值，并通过 8 位的 I/O 端口(8212)送到 CPU 的数据总线上去。同时，将其送至优先权比较器，与现行状态进行比较。

(2) 现行状态寄存器

这是为了判断新的中断请求的优先权是否高于正在处理的中断请求而设置的。它是一个 4 位锁存器，可由CPU用输出指令的方式，把现行状态的优先权编码输至这个寄存器(但CPU不能读它的内容)，当其$\overline{ECS}$输入端为低时，接受输入数据。$B_2\ B_1\ B_0$ 即为现行状态的优先权编码，它被送至比较器，与新的中断请求的优先权相比较。如果新的中断请求的优先权级别比现行中断的级别高，即 A>B，则比较器输出为高，它可以使中断触发器 INT F/F 置"1"(触发器需要由时钟信号来同步)，经反相后送至CPU的$\overline{INT}$输入端，请求新的中断。如果新的中断的级别等于或低于目前正在中断处理的级别，即 A≤B，则 8214 的$\overline{INT}$端不发出新的中断请求，而让 CPU 继续完成正在进行的中断处理，然后再去处理新的中断请求。

在现行状态寄存器中，还有一个输入端：状态组选择信号$\overline{SGS}$(Status Group Select)，它有一种特殊的功能，即当 CPU 不在进行中断处理时(即在执行主程序时)，则现行状态编码为 000，若中断请求的优先权级别也为 000，则比较器仍没有输出，此时，$\overline{SGS}$变为有效，以代替 A>B 的作用，只要$\overline{R_7}\sim\overline{R_0}$中有中断请求，就可使 INT F/F 为"1"，向 CPU 发出中断请求。因而当 CPU 未在进行中断处理时，应使$\overline{SGS}$有效。应当指出，输出至现行状态寄存器的是现行状态优先权编码 $B_2\ B_1\ B_0$ 的反码$\overline{B_2}\overline{B_1}\overline{B_0}$，在不同的现行状态时能接受的最小优先权如表 12-1 所示。

8214 优先级别与对应的 RST 指令的关系如表 12-2 所示。

表 12-1 8214 状态寄存器对应的优先权级别

状态寄存器 $\overline{SGS}$	$\overline{B_2}$	$\overline{B_1}$	$\overline{B_0}$	可接受的最小优先权	
1	1	1	1	0	(A=000)
0	1	1	1	1	(A=001)
0	1	1	0	2	(A=010)
0	1	0	1	3	(A=011)
0	1	0	0	4	(A=100)
0	0	1	1	5	(A=101)
0	0	1	0	6	(A=110)
0	0	0	1	7	(A=111)

注：其中 7 级的中断级别最高。

表 12-2 8214 优先级别与对应 RST 指令关系表

优先权申请		RST	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
			1	1	A_2	A_1	A_0	1	1	1
低	0	7 (38H)	1	1	1	1	1	1	1	1
	1	6 (30H)	1	1	1	1	0	1	1	1
	2	5 (28H)	1	1	1	0	1	1	1	1
	3	4 (20H)	1	1	1	0	0	1	1	1
	4	3 (18H)	1	1	0	1	1	1	1	1
	5	2 (10H)	1	1	0	1	0	1	1	1
	6	1 (08H)	1	1	0	0	1	1	1	1
高	7	0*(00H)	1	1	0	0	0	1	1	1

* RST 0 将引导程序计数器转向 00H 单元。

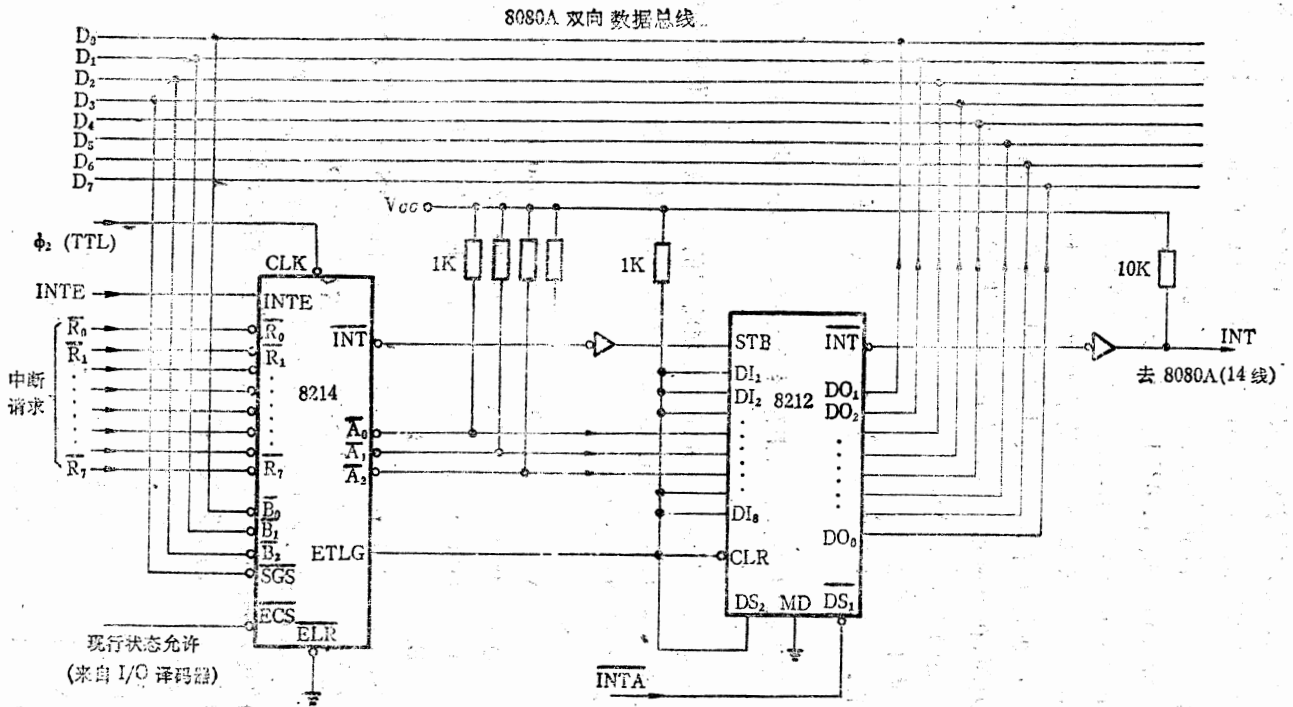


图 12-19 用 8214 组成的 8 级优先权中断系统

$\overline{ELR}$、ETLG、ENLG 这三个信号使 8214 可以进行级连，以扩大中断请求的级别。

2．用 8214 组成 8 级优先权中断系统

8214 最普遍的应用是作为 8080 A 微型计算机系统的一个 8 级优先权中断控制器。

用 8214 和 8212 连接，可以组成 8 级优先权中断系统，如图 12-19 所示。它能接受 8 个中断请求，确定优先权的级别，并将最高优先权的中断请求经 8212 送至 8080 A 的 INT 端。同时，将$\overline{A}_2$、$\overline{A}_1$、$\overline{A}_0$ 送至 8212 以形成相应的 RST指令，并在中断响应信号 $\overline{INTA}$选通时送到 CPU 的数据总线上，由此输入至 CPU 的指令寄存器 IR，经过译码执行 RST 指令后，控制程序转移到相应的中断服务程序的入口。此外，8214 还能管理现行状态，并与新的中断请求进行比较，结果将屏蔽本级或以下级别的中断请求，而响应优先权更高级别的中断请求。

图中的 8214 的 $\overline{R}_7$～$\overline{R}_0$ 接至 8 个中断请求源，$\overline{R}_7$ 优先权最高。现行状态的编码 $\overline{B}_2$ $\overline{B}_1$ $\overline{B}_0$ 接至数据总线，可由 CPU 执行一条输出指令来赋给现行状态。选通信号 $\overline{ECS}$是由这个端口的地址译码器提供的。因而，在中断允许的情况下，8214 就能进行优先权排队与比较，输出 $\overline{INT}$ 信号，以及中断级别的编码 $\overline{A}_2$、$\overline{A}_1$、$\overline{A}_0$。

使用 8214 时应注意下列各点：

(1) 开放中断以前，主程序应使现行状态寄存器为全1(包括 $\overline{SGS}=1$)，此时可接受 $\overline{R}_7$～R_0 中任一个中断请求；

(2) 由于 CPU 不能读 8214 现行状态寄存器的内容，故程序应将现行优先权复制到 RAM 中；

(3) 在重新开放中断前，每个中断服务程序必须将旧的优先级推入堆栈保护，而将新的优先级送到 8214 的状态寄存器中，返回到主程序前要恢复以前的优先权。

初始化 8214 状态寄存器的程序是：

```
8080A/8085A        Z 80
MVI   A, 0FH       LD   A        , 0FH     ; 使 A=00001111B(D3置1),取消比较控制
OUT   STATUS       OUT (STATUS), A         ; 输出至8214现行状态寄存器
STA   PRTY         LD   (PRTY)   , A       ; 复制到 PRTY 单元
EI                 EI
```

8212 用来形成 RST 指令，并且在 CPU 发出中断响应信号 $\overline{INTA}$ 时将 RST 指令送至 CPU 的数据总线上，随后经过译码执行 RST 指令，从而转至相应的中断服务程序入口。

CPU 转至相应的中断服务后，首先要用一连串的 PUSH 指令来保护现场。然后在保留副本后，再把现行状态的优先编码送至 A，用输出指令把它输出给 8214 保存。

下面是一个中断级别为 4 的中断服务程序：

```
          8080 A/8085 A     Z 80
IN4:      PUSH   PSW        PUSH  AF        ;
          PUSH   B          PUSH  BC        ;  保护现场
          PUSH   D          PUSH  DE        ;
          PUSH   H          PUSH  HL        ;
          JMP    SERVR4     JP    SERVR4    ; 转地址 SERVR4 继续
SERVR4:   LDA    PRTY       LD    A,(PRTY); 取状态寄存器副本
          PUSH   PSW        PUSH  AF        ; 保留旧的优先权
          MVI    A,03H      LD    A, 03H    ; 取新的优先级(100的反码为011) 及允许比
                                             ; 较控制位
```

```
OUT     STATUS     OUT    (STATUS),A;     将A的内容送到8214的状态寄存器
STA     PRTY       LD     (PRTY),  A;     存现行优先级的副本到 RAM 中
EI                 EI
:                  :                ;}   中断服务程序
DI                 DI               ;    关中断,准备恢复
POP     PSW        POP    AF        ;    将旧的优先级取回至A
STA     PRTY       LD     (PRTY),  A;     保存旧的优先级到 RAM
OUT     STATUS     OUT    (STATUS),A;     将旧的优先级送 8214 状态寄存器
POP     H          POP    HL        ;}
POP     D          POP    DE        ;}   恢复现场
POP     B          POP    BC        ;}
POP     PSW        POP    AF        ;}
EI                 EI               ;    开中断
RET                RET              ;    返回被中断程序
```

3．当中断源超过 8 个时，可以采用几个 8214 级联的方式来满足要求。为了进行级联，8214 设置了 $\overline{ELR}$、ETLG、ENLG 控制信号。把一个（优先权级别高的）8214 的输出 ENLG 接至下一个（优先权级别较低）8214 的输入 ETLG 端，可依此一直级联下去。优先权级别最高的 8214 的 ETLG 输入端接高电平，当此片的中断输入有中断请求时，它的 ENLG 输出为低电平，屏蔽所有级别低的 8214 片子；只有当这一级的 8 个中断输入全无中断请求时，它的 ENLG 输出才为高电平，使下一级 8214 的中断开放，以后的情况也照此处理。

七、可编程序中断控制器Intel 8259（简称PIC——Programmable Interrupt Controller）

Intel 8259 PIC 电路可用来管理中断优先权、中断屏蔽和自动转移中断矢量。

前已提及，Intel 8080 A 只有一条中断输入线，因此，在多个外部设备同时请求中断时，需要采取适当的方法来解决中断优先权问题。8259 可以管理八级中断。若经过多级级联，则可

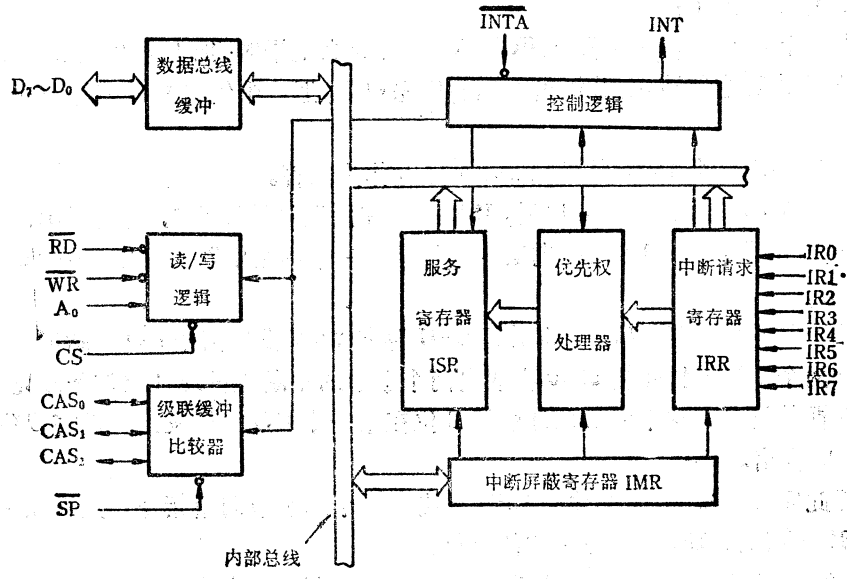


图 12-20　8259 的结构框图

以将其扩充到 64 级。

1．8259 的方框图及引线

8259 的方框图和引线布置图如图 12-20 和 12-21 所示。

在 8259 中，中断请求寄存器 IRR 用来存贮所有中断请求输入信号 IR7～IR0。服务寄存器 ISR 用来存贮所有待处理的中断级别。屏蔽寄存器用来存贮需要屏蔽的那些中断的各位。优先权处理器决定在 IRR 中各位的优先权。优先权最高的那位，在 $\overline{INTA}$ 脉冲作用时，被送到 ISR 相应的位置上去。

INT信号线用于中断请求，接到Intel 8080 A 中断请求输入端 INT。而 $\overline{INTA}$ 信号使 8259 把一个三字节的调用中断服务子程序的指令放到数据总线上去。

读写控制逻辑将从 CPU 接受 OUT 指令发出的信息。它包括预置命令字寄存器（ICW）和操作命令字寄存器（OCW）。这些寄存器既能存贮 I/O 设备操作的不同控制方式，也能把 8259 的状态送到数据总线上去。A_0 信号和读（$\overline{RD}$）、写（$\overline{WR}$）信号一起，既可把命令写入各种命令寄存器，也可从各种状态寄存器读出。

级联缓冲比较器用来把几个 8259 级联扩充成 64 级中断。

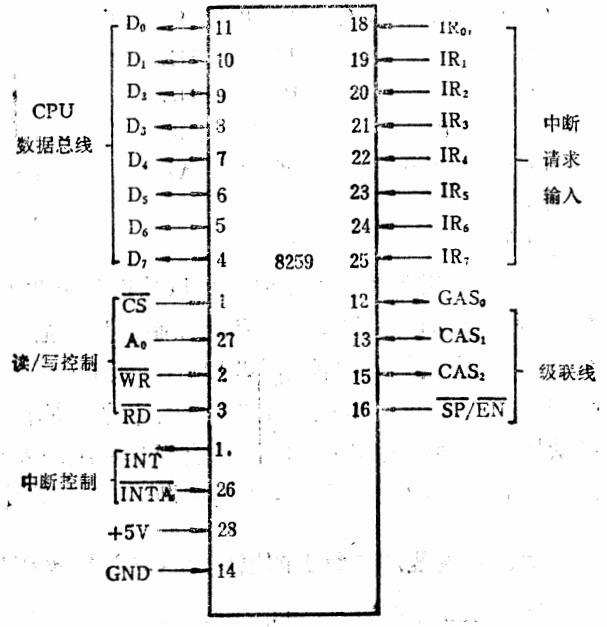


图 12-21 8259 的引线逻辑图

2．8259 的操作说明

8259 具有可编程序的特点，并可利用调用指令自动转移到存贮器的任意地址。

8259 工作过程如下：

(1) 当一个或多个中断请求线(IR 0～IR 7)变为高电平时，把 IRR 相应位置数。

(2) 8259 接受这些请求，分析出它们的优先权后，把一个优先权最高的中断信号(INT)送往 CPU。

(3) CPU 响应这个 INT，并发出 $\overline{INTA}$ 脉冲作为回答。

(4) 在接收 CPU 来的 $\overline{INTA}$ 以后，ISR 的最高优先权位置“1”，并使 IRR 中相应的一位复位。8259 器件通过 D_7～D_0 引线向数据总线送出一个调用中断子程序指令（CALL）操作码（11001101B）。

(5) 这个调用指令(CALL)还将引起 CPU 再向 8259 发出两个 $\overline{INTA}$ 脉冲。

(6) 这两个 $\overline{INTA}$ 脉冲使 8259 再放出两个字节的中断矢量地址（这是预先由程序编好的），放在数据总线上。这个矢量地址的低 8 位在第一个 $\overline{INTA}$ 时发出，而高 8 位地址则在第二个 $\overline{INTA}$ 时发出。

(7) 至此完成了由 8259 器件向 CPU 送出的一个三字节调用指令。只有在一个中断结束命令发给 8259 以后，ISR 的相应位才能复位。

此外，8259 还可以执行中断的屏蔽。如果八个中断级别中有任一个需要禁止，那末，只要

在相应的屏蔽位中置“0”即可。8259 还可自动禁止比某个中断级别优先权较低的中断请求。这里选中的中断级别先存入“服务寄存器”中。如果外面还有另外的中断请求，则借助于比较器对其优先权和“服务寄存器”中的中断优先权作出比较。如果前者较低，则自动被禁止。

关于 Intel 8259 器件的编程请参阅 MCS-85 微型计算机用户手册。

3. 8259 与标准系统总线的连接

8259 与标准系统的连接如图 12-22 所示。

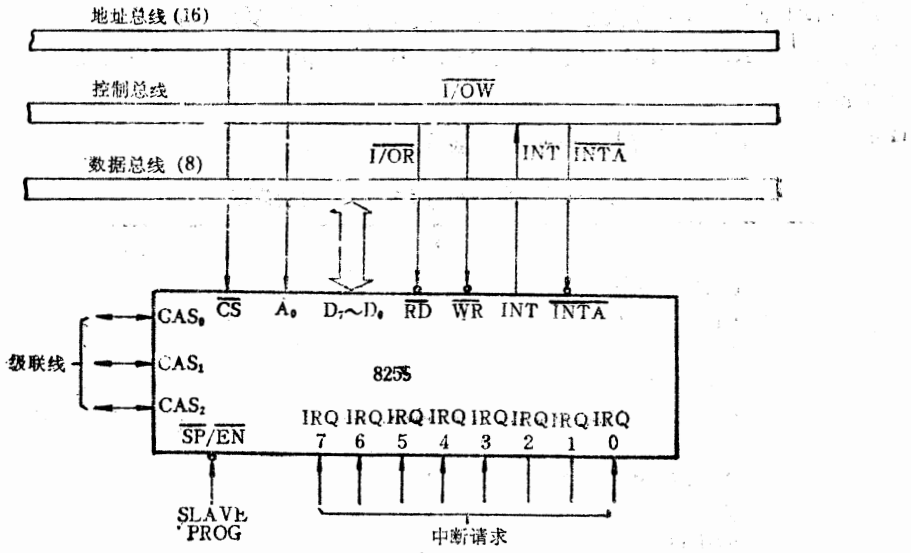


图 12-22 8259 与标准总线连接图

§12-3 Z 80 的中断系统

一、Z 80中断系统的组成

Z 80 中断系统主要包括两个部分：

1. 具有不可屏蔽的中断输入线 $\overline{NMI}$ (Non-Maskable Interrupt) 和可屏蔽中断输入线 $\overline{INT}$，两者都为低电平有效；

2. 两个中断允许触发器 IFF_1 和 IFF_2。

IFF_1	IFF_2
用于控制开中断或关中断	用于暂存 IFF_1 的状态

当 IFF_1 为“1”时，就允许中断；当 FF_1 为“0”时，就禁止中断。而 IFF_1 的状态可由程序员用允许中断指令(EI)或禁止中断指令(DI)来置“1”或置“0”。

当 CPU 复位时，将迫使 IFF_1 和 IFF_2 都处于“0”状态，即禁止中断状态。但是程序员可在任何时候用一条 EI 指令来开放中断。在执行 EI 指令后，一直到执行完紧跟着 EI 指令后面的一条指令之前，CPU 不接受任何中断请求。这是因为：通常在 EI 指令后面是一条返回指令，为了保证断点的正确返回，必须在返回指令执行完成以后，才允许接受新的中断。

EI 指令使 IFF_1 和 IFF_2 两者都处于允许中断状态。当 CPU 接受中断后，IFF_1 和 IFF_2 自动地被置“0”，禁止新的中断请求，直到程序员用新的 EI 指令来开放中断为止。

在上述情况下，IFF_1 和 IFF_2 的状态始终是相同的。

当 Z80 CPU 进行不可屏蔽中断($\overline{NMI}$)处理时，IFF_2 用来保存 IFF_1 的状态。当 $\overline{NMI}$ 请求被接受时，为了防止发生新的中断，IFF_1 将被置“0”，直到程序员重新用 EI 指令来开放中断为止。但是，如果在 $\overline{NMI}$ 发生以前，CPU 正在处理可屏蔽中断，此时，IFF_1 的状态是由中断服务程序中是否已经开中断来决定的。当 CPU 响应 $\overline{NMI}$ 请求时，将迫使 IFF_1 置“0”，因而，为了保存 CPU 响应 $\overline{NMI}$ 以前 IFF_1 的状态，就设置 IFF_2 来暂存 IFF_1 的状态，待 CPU 处理 $\overline{NMI}$ 完毕之后，自动地将 IFF_2 的状态送回 IFF_1，使中断系统恢复原来的状态。

当执行 LD A, I 和 LD A, R 指令时，IFF_2 的状态被复写到奇偶标志位 P/V 中，以供存贮和检测。

IFF_1 和 IFF_2 在各种情况下的状态如表 12-3 所示。

表 12-3 IFF_1 和 IFF_2 的状态表

作　　用	IFF_1	IFF_2
CPU 复位	0	0
DI 指令	0	0
EI 指令	1	1
LD A, I	·	· $IFF_2 \longrightarrow P/V$
LD A, R	·	· $IFF_2 \longrightarrow P/V$
接受 $\overline{NMI}$	0	·
RETN($\overline{NMI}$ 返回)	IFF_2	·
接受 $\overline{INT}$	0	0
RETI($\overline{INT}$ 返回)	·	·

注：·表示不变

二、Z80 所具有的供中断系统使用的专用指令

1. EI (允许中断)指令：它通过将中断触发器置“1”来开放可屏蔽中断。

2. DI (禁止中断)指令：它通过将中断触发器置“0”来封锁可屏蔽中断。

3. RST (重新启动)指令：它是一个单字节的调用指令。前面已述及。

4. RETI(从中断返回)指令：它与RET相似，所不同的是 Z80 的 I/O 接口芯片(P.O、CTC、SIO)可以识别 RETI 指令，并可用它来通知：现行中断服务程序已经结束。

5. RETN(从不可屏蔽中断返回)指令：它与 RET 也很相似，所不同的是：RETN 将 IFF_2 的内容送入 IFF_1，使中断系统恢复原来的状态。

6. LD A, I 指令：它将 I(中断矢量)寄存器的内容送入累加器。这条指令(和 LD A, R)也将 IFF_2 的内容放入标志寄存器的 P/V 位。因此，该标志位可用于测试或保存在堆栈中。

7. LD I, A 指令：它将累加器的内容送入 I (中断矢量)寄存器。

8. IM (设置中断方式)指令：它确定中断服务方式。有三种中断方式可任选，即 0、1 或 2。这些内容将在后面说明。

三、Z80 中断处理的流程

Z80 CPU 中断处理的流程图，如图 12-23 所示。

Z80 CPU 按照如下的优先权响应外部请求：

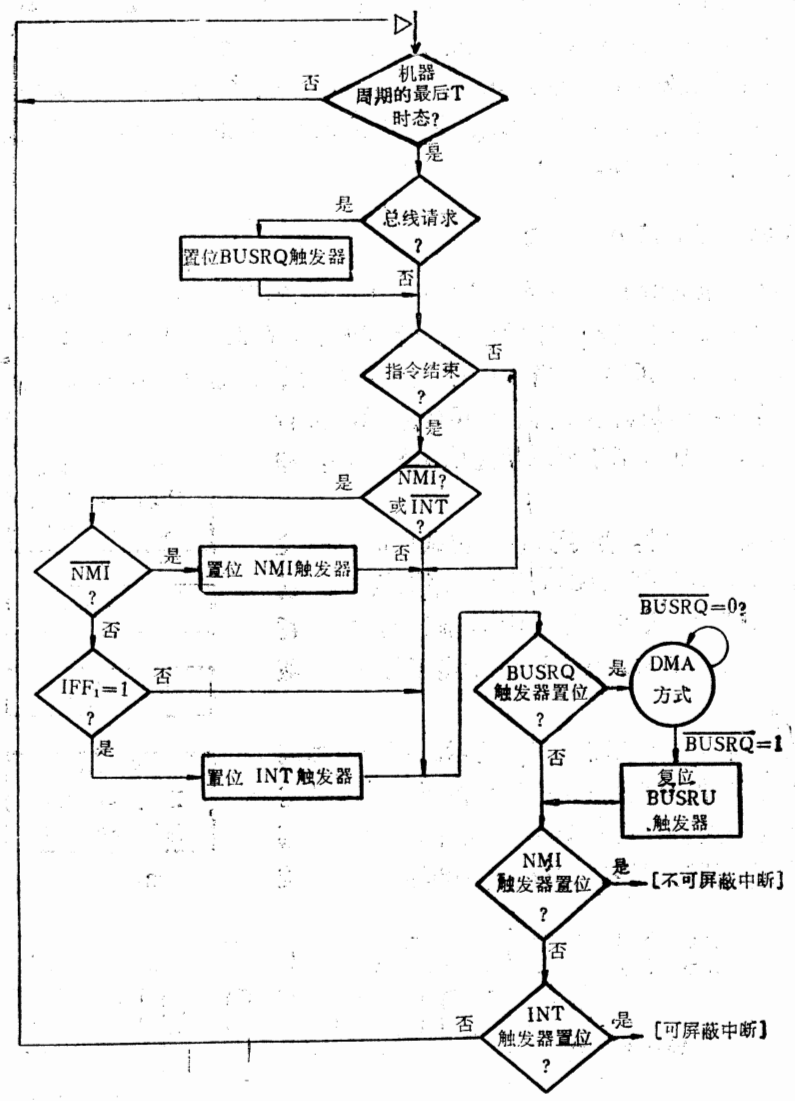


图 12-23　Z80 CPU 中断处理的流程图

1．总线请求 ($\overline{\text{BUSRQ}}$)；

2．不可屏蔽中断($\overline{\text{NMI}}$)；

3．可屏蔽中断($\overline{\text{INT}}$)。

当 CPU 开始执行指令时，第一个机器周期总是取指周期。取出的指令经译码后，就可确定执行这条指令有几个不同类型的机器周期。当一条指令执行完后，就进入下一条指令的取指周期。但在每一个机器周期的末尾，CPU 采样 $\overline{\text{BUSRQ}}$ 线，若发现有总线请求，则置位总线请求触发器，进入 DMA 方式。在每一条指令的最后一个机器周期的最后一个 T 时态，CPU 采样 $\overline{\text{NMI}}$ 与 $\overline{\text{INT}}$ 线，若有 $\overline{\text{NMI}}$ 请求，则置位 NMI 触发器，从下一个机器周期开始，进入不可屏蔽中断响应周期，转至相应的中断服务程序，如图 12-24 所示。

若没有$\overline{\text{NMI}}$请求，CPU 将检测可屏蔽中断触发器(INT F/F)的状态；若有 $\overline{\text{INT}}$ 请求，则置位 INT触发器。如果此时中断是开放的，则从下一个机器周期开始，进入可屏蔽中断响应周期，

转至相应的可屏蔽中断服务程序，否则 CPU 返回主程序继续执行，如图 12-26 所示。

因此，总线请求的优先权最高，总线请求的响应时间不超过一个机器周期。其次是$\overline{NMI}$请求，它不受中断允许触发器的影响，其响应时间不超过一个指令周期。再次是 $\overline{INT}$ 请求，如果 CPU 的中断系统是开放的，那末其响应时间也不超过一个指令周期。如果没有上述请求，CPU 就进入下一个指令周期。

四、Z80 的不可屏蔽中断方式

在没有总线请求的情况下，Z 80 CPU 在任何时候都能够接受不可屏蔽中断请求。CPU在执行现行指令的最后一个机器周期的最后一个 T 时态，查询 $\overline{NMI}$，若有效，则 CPU 立即予以响应。在 $\overline{NMI}$ 响应周期，CPU 相当于执行一条 RST 指令，但是其转向的入口地址不是 RST p 指令所指定的地址，而是自动转向 0066 H 单元，执行 $\overline{NMI}$ 的服务程序。它通常用于要求快速响应的事件，如电源故障等。

当 CPU 响应 $\overline{NMI}$ 时，用于开放可屏蔽中断的触发器(IFF_1)被置“0”，以封锁在 $\overline{NMI}$ 服务程序期间产生新的中断。不过，IFF_1 的状态在进入 $\overline{NMI}$ 前已经放到暂存触发器 IFF_2 中保存起来。图 12-24 示出了不可屏蔽中断的流程图。

不可屏蔽中断的时序如图 12-25 所示。第一机器周期类似于取指周期，但是数据总线至 IR 被封锁，代替它的是内部产生的一条转移到 0066 H 的指令。第二和第三机器周期是存贮器写周期。它分别将 PC_H 和 PC_L 推

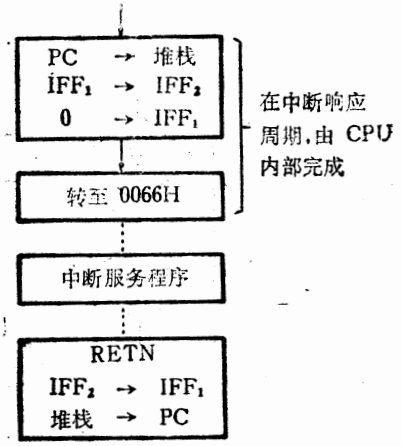


图 12-24 不可屏蔽中断的流程图

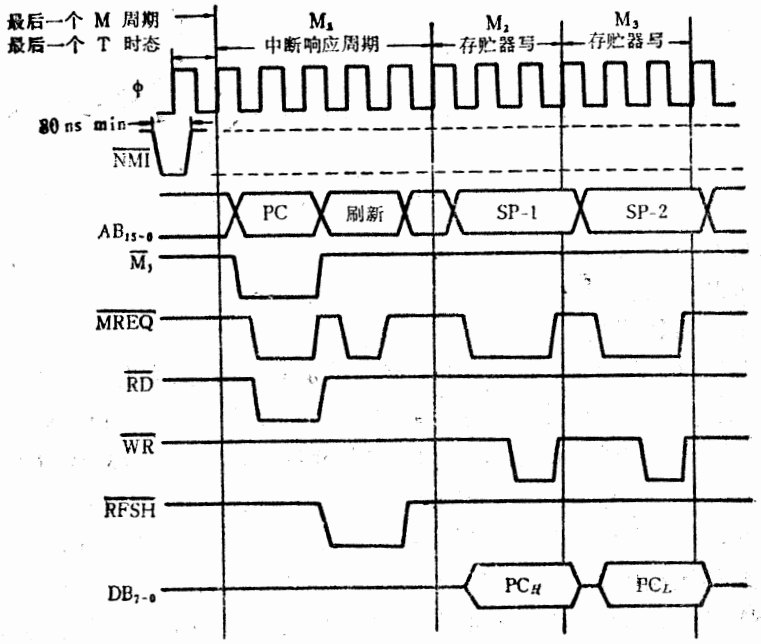


图 12-25 不可屏蔽中断的时序

入堆栈保存。在执行了不可屏蔽中断服务程序之后，就执行一条 RETN 指令。于是 IFF_2 的内容又重新送回到 IFF_1 中，从而自动恢复了可屏蔽中断允许触发器的状态。

上述操作都是由 CPU 内部的硬件逻辑自动完成的，不需要请求中断的外设提供 RST 指令。同样，执行 RETN 指令的操作也是由硬件完成的。

五、Z80 的可屏蔽中断方式

可屏蔽中断($\overline{INT}$)可以由程序员用软件 (EI 或 DI 指令) 来确定开中断或关中断。开中断指令(EI)把中断允许触发器 IFF_1 和 IFF_2 都置 "1"，其后再执行一条指令，即可响应任何一个可屏蔽中断。关中断指令(DI)则把 IFF_1 和 IFF_2 都置 "0"，从而禁止响应中断。应当指出：在执行 EI 或 DI 指令周期间，可屏蔽中断将被禁止。Z 80 的可屏蔽中断有三种方式：方式 0、1 2，这可用中断方式指令 IM 0、IM 1、IM 2 来选择。CPU 复位时，将自动地把中断方式置为方式 0。

1．可屏蔽中断方式 0

当 CPU 复位时，自动处于中断方式 0。可屏蔽中断方式 0 也可用一条中断方式指令 IM 0 来设置。当 CPU 响应方式 0 中断时，由请求中断的外设提供一条 RST 指令送至 CPU 数据总线上，经 CPU 译码、执行后，自动转向位于内存前 64 个字节的 8 个中断服务程序的入口之一。它与 Intel 8080 A 中断方式相同。

应当指出，外设也可以向 CPU 提供其它指令。在中断响应周期中，若读到的是多字节指令中的第一个字节，则其后面的字节是用正常的内存读出时序读取(但在这些周期中，都必须封锁 PC←PC＋1 的操作，以保证 PC 为中断前的值)。当 CPU 响应中断后，就自动禁止中断(IFF_1 和 IFF_2 置 0)。若系统要求能在中断处理过程中响应更高级的中断，则可在保护现场后的任何时刻，用 EI 指令来开放中断。中断方式 0 时，在中断响应周期内，CPU 从数据总线上取得外设提供的 RST 指令，然后就转入执行 RST 指令的时序，用两个机器周期实现堆栈写操作(保护断点)，并将 WZ 置成 8×NNN，从而转入相应的中断服务程序。

在中断响应周期中，CPU 还自动插入两个等待时态，以便有充足的时间来实现外部链形优先权中断控制。

中断方式 0 的流程图如图 12-26(a)所示。

2．可屏蔽中断方式 1

可屏蔽中断方式 1 可用一条中断方式指令 IM1 来设置，它除了使 CPU 自动地转向 0038H 单元(即执行 RST 38H 指令的入口地址)而不是转向 0066H 外，与不可屏蔽中断 $\overline{NMI}$ 很相似。CPU 同样自动地把 PC 推入堆栈，如图 12-26(b) 所示。

它的优点是不需要由外设接口提供一条 RST 指令，而是自动地转向 0038 H 单元，同时没有附加的 T_W^* 时态，因此，执行 IM 1 达到中断处理的目的所需要的时态数，比正常完成 RST 指令所需要的时态数少。当采用这种中断方式时，如果有两个或两个以上中断源，必须在 0038H 开始的单元内存放一个程序，用查询方式来识别中断源以及确定它们的优先权，转去执行优先权等级最高的中断服务程序。

3．可屏蔽中断方式 2

方式 2 是 3 种可屏蔽中断方式中功能最强、最灵活的一种。其中断服务流程图见图 12-26(c)。

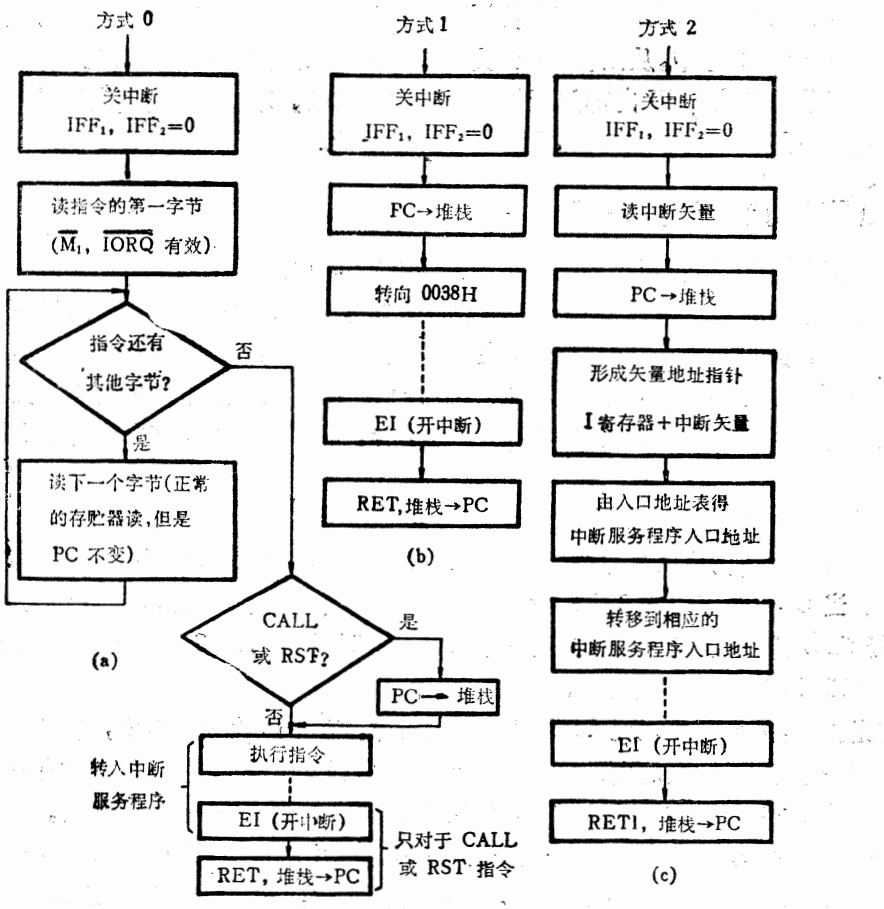


图 12-26　可屏蔽中断的流程图

在方式 0 中，当有两个或两个以上中断源时，必须外加中断控制逻辑电路以提供相应的 RST 指令。而且，当中断源多于 8 个时，这种外加的中断控制逻辑电路就更加复杂了。在方式 1 中，当中断源较多时，用软件查询方式识别中断源的时间显得太长。

方式 2 充分利用 Z80 CPU 和 Z80 系列的 I/O 接口电路所设置的矢量中断结构，使用户能够方便地对内存贮器的任何位置进行间接访问，而且可允许的中断源数目多达 128 个，即有 128 个中断服务程序的入口地址。可把这些入口地址编成 16 位的入口地址表，如图 12-27 所示。此表占用 256 个单元，即一页。可由用户确定放在内存的任何一页。在中断响应时，应形成 16 位的中断矢量地址指针，以便从入口地址表中获得所需要的中断服务程序的入口地址。这一指针的低 8 位地址，由请求中断的外设提供一个中断矢量（由于中断矢量地

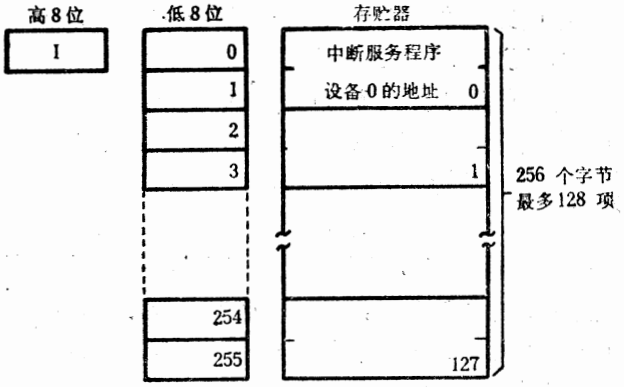


图 12-27　中断方式 2 的 16 位入口地址表

址指针最低位总为0，因此中断设备只需提供7位即可)；高8位地址由Z80 CPU中的I寄存器提供。I寄存器的内容，应在中断以前由用户用以下的指令来设置：

```
LD  A, n
LD  I, A
```

外设提供的中断矢量的低8位地址，通常是由Z80系列的I/O接口电路(如CTC、PIO、SIO等)，在程序初始化时，用指令来规定的。而在中断响应周期，由外设把规定的低8位字节送至CPU数据总线上。当然，也可用前节所提到的8212(或其它三态缓冲器)产生RST指令类似的方法来产生。

方式2时，转至中断服务程序入口的步骤如图12-28所示。

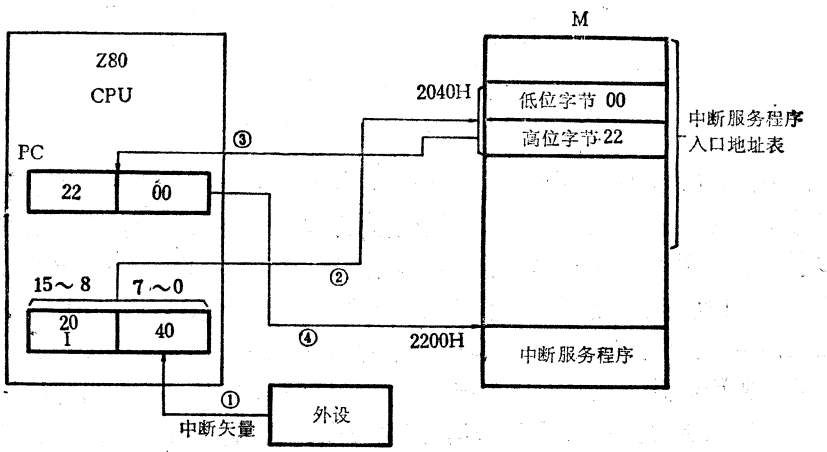


图12-28 方式2转至中断服务程序入口的步骤

在CPU响应中断后，首先由请求中断的外设通过数据总线提供中断矢量低8位地址，在CPU内与I寄存器的内容组合成16位中断矢量地址指针(注意：它并不是中断服务程序的入口地址)，以指向中断服务程序入口地址表。从该指针所指明的单元(存放入口地址的低8位)，以及相邻的单元(存放入口地址的高8位)中取出的内容，即为中断服务程序的入口地址。CPU将此入口地址送至PC，以控制程序转向相应的中断服务程序。

Z80中断方式2的中断响应与服务过程可归纳如下：

(1) 在初始化程序中，用中断方式指令IM2设置中断方式2。当满足中断响应条件后，转入中断响应周期；

(2) 由请求中断服务的I/O设备的接口电路提供16位中断矢量指针的低8位地址；

(3) 将断点地址PC保存在堆栈中，以便中断服务结束时返回断点处继续执行；

(4) 用CPU中的I寄存器内容(8位)作为中断矢量地址指针的高8位，与申请中断的外设所提供的低8位中断矢量相结合，形成16位中断矢量作为中断服务程序入口地址表的指针；

(5) 从中断服务程序入口地址表中，找出与第(4)步中所形成的16位中断矢量地址指针相对应的相邻两个单元的内容，即为所要求的中断服务程序的入口地址；

(6) 将所提供的中断服务程序入口地址送入PC，转至中断服务程序运行；

(7) 当中断服务程序结束时，执行RETI指令，把保存在堆栈中的断点地址弹出，返回到原来被中断的程序后继续执行下去。

六、Z80 的中断优先权和中断嵌套

1．链形中断优先权结构

由于 Z80 系统中设置有链形中断优先权结构，因此使它处理多级中断的内部实时操作大为减少。在 Z80 CPU 和外设（指 Z80 系列中的 CTC、PIO、SIO 等）间，不要求附加的逻辑，中断结构就能从几个同时请求服务的外设中间选择优先权最高的外设。

优先权的高低是根据外设在链形中断结构中所处的位置决定的。每个外设的中断请求都连到 $\overline{INT}$ 线上。对于每个外设来说，它能否向 $\overline{INT}$ 线发出中断请求，不仅取决于这个外设是否有中断请求，而且还取决于它的 IEI 是否有效。每一种外设都设有中断允许输入（IEI）和中

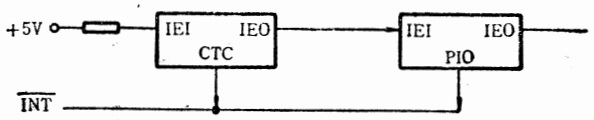


图 12-29　Z80 的链形优先权电路

断允许输出（IEO）两条线。图 12-29 表示了 Z80 CTC 和 Z80 P.O 连接的一种典型接法。CTC的 IEI 接到 +5 伏，表明它有最高的优先权，即中断允许输入始终为“高”，表示始终开放中断。而 CTC 的 IEO 输出接至 PIO 的 IEI 输入，当 CTC 有中断请求时，IEO 输出为“低”，屏蔽了 PIO的中断请求（即 P.O 即使有中断请求，也不能向 $\overline{INT}$ 发出中断请求），而且 PIO 的 IEO 输出也为“低”，屏蔽了以下的所有外设。只有当 CTC没有中断请求时，它的 IEO 输出为“高”，方才允许 P.O 请求中断。故 PIO 具有次高的优先权。若把 PIO 的 IEO 输出接至另外的 I/O 电路的 IEI 端，则另外的 I/O 电路的优先权低于 PIO，依此类推。所以任何给定外设的 IEO 线将满足下列关系：

$$IEO = IEI \cdot \overline{HELP}$$

式中 HELP 表示该外设需要服务。此时，HELP = 1，$\overline{HELP} = 0$，使 IEO = 0，封锁低级的中断请求。

这种链形优先权结构，是 Z80 系列的独特的优点。它适用于任何一种可屏蔽中断方式。

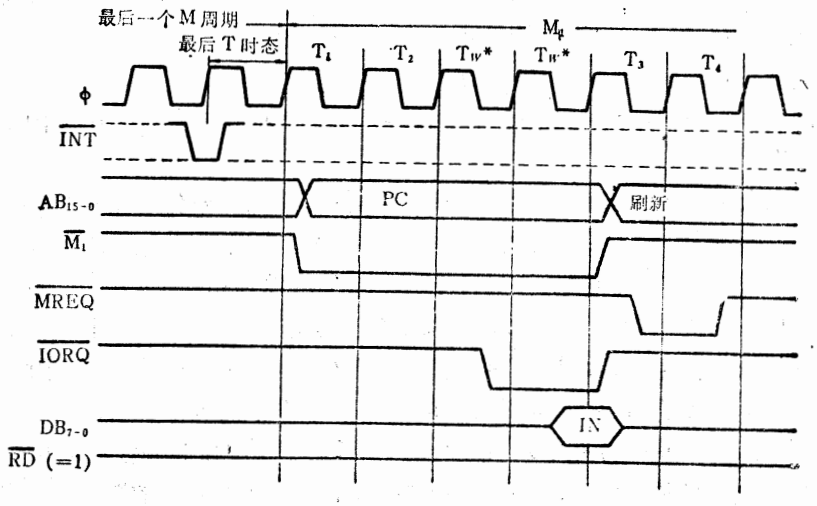


图 12-30　可屏蔽中断时序

2．屏蔽中断时序

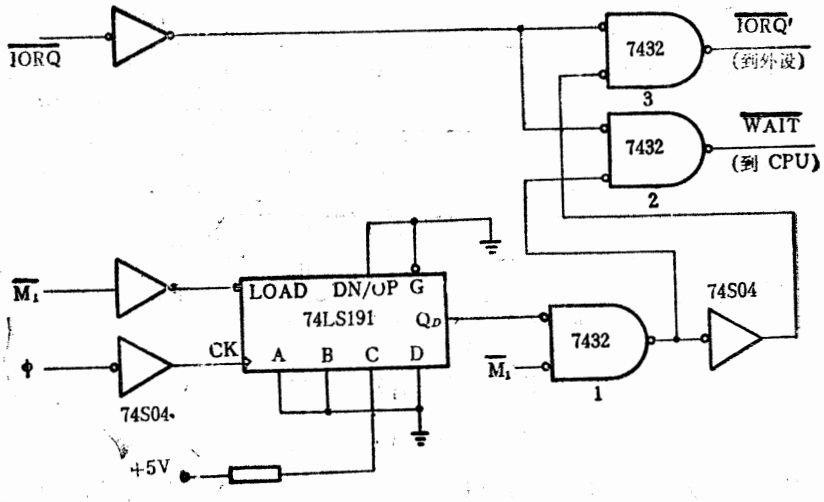
CPU 在中断响应期间，在 $\overline{M_1}$ 有效到 $\overline{IORQ}$ 信号有效之间的时间间隔用来识别链形结构的中断优先权，故在屏蔽中断的中断周期中，CPU 自动插入了两个 T_W^* 时态，以增加这个时间间隔。可屏蔽中断的中断时序如图 12-30 所示。

$\overline{M_1}$ 信号与取指周期一样，在 T_1 上升沿后有效，但 $\overline{IORQ}$ 信号直至第一个 T_W^* 下降沿后有效，保证有大于两个 T 时态的时间，用以确定链形结构中提出中断请求的优先权最高的外设。

在不增加外部逻辑时，这段时间最多可以为 4 个 Z80 外设服务。

当设计的系统超过 4 个 I/O 外设时，用户可以用延长中断响应周期（即利用 $\overline{WAIT}$ 线插入 T_W 时态），或减少优先权排队的门延时间的方法来解决。

延长中断响应周期采用的技术是控制发往外设的 $\overline{IORQ}$ 脉冲。图 12-31 示出用 $\overline{WAIT}$ 线以增加 T_W 时态，推迟发向外设的 $\overline{IORQ'}$ 的电路。



74LS191

计　数	输　出　Q			
	D	C	B	A
起始状态	0	1	0	0
1	0	1	0	1
2	0	1	1	0
3	0	1	1	1
4	1	0	0	0

图 12-31　用等待状态延长中断响应周期

图中，74LS191是一个四位的二进制同步计数器，允许输入端 G 置于允许计数状态，加/减计数输入端(DN/UP)置于加法计数状态。当 $\overline{M_1}$ 有效后，计数器就在时钟脉冲的下降沿工作。A、B、C、D 端是计数器的预置输入端。在图 12-31 所示的输入预置状态下，Q_D 起始为低电平。故当 $\overline{M_1}$ 有效后，与门 1 输出为“低”；当有 $\overline{IORQ}$ 信号后，$\overline{WAIT}$ 有效（低电平），使在 T_W^* 时态后，插入一个 T_W 时态。因与门 1 输出为“低”，经反相器输出为“高”，故与门 3 的输出，即输出给外设的与 $\overline{M_1}$ 一起作为中断响应的 $\overline{IORQ'}$ 信号为高电平。

时钟脉冲使同步计数器计数，当 Q_D 由 0→1 时，与门 1 的输出就变“高”，从而使 $\overline{WAIT}$ 线变“高” 同时使 $\overline{IORQ'}$ 有效，使外设得到中断响应信号，把中断矢量经过数据总线输入 CPU。所以，图 12-31 电路可以延长输出至外设的 $\overline{IORQ'}$ 时间，增加了识别链形优先权的时间。

究竟插入几个 T_W 时态，取决于同步计数器的预置输入。在图 12-31 所示的预置输入情况下，可插入一个 T_W 时态，其时序图如图 12-32 所示。

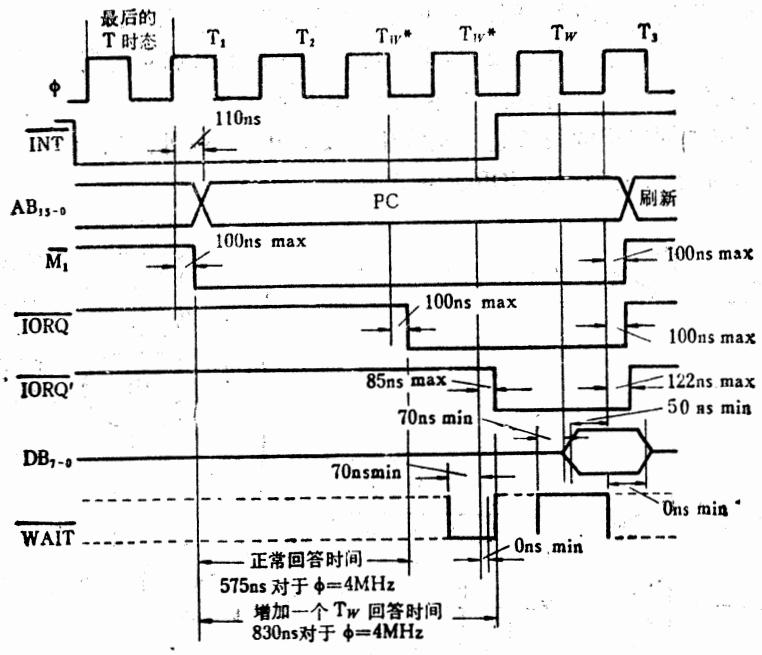


图 12-32 插入一个 T_W 时态的时序图

在这种情况下，要求计数 4 次后 $Q_D=1$，在 $\overline{M_1}$ 有效后，$\overline{WAIT}$ 线有效(变为“低”)，且计数器在接收时钟脉冲的下降沿开始计数，于是 T_1、T_2、T_{W1}^*、T_{W2}^* 下降沿都使计数器计数，T_{W2}^* 下降沿采样 $\overline{WAIT}$ 线，此时 $\overline{WAIT}$ 线为“低”，故插入一个 T_W 时态，而采样后，经过门的延迟时间，$\overline{WAIT}$ 线就变“高”(因为 Q_D 输出为高电平)，而 $\overline{IORQ'}$ 变为有效。当在 T_W 下降沿采样 $\overline{WAIT}$ 线时，它已为高电平，故不再插入 T_W 时态，而转为 T_3 时态。

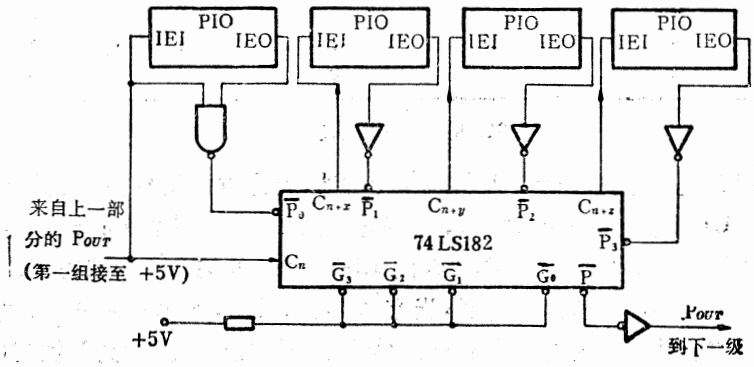


图 12-33 先行进位的链式电路

此外，还可以采用减少门延时间的方法，图 12-33 电路就是一个用先行进位的方式，以减少链形优先权电路的门延时间。

74S182是一个先行进位发生器，它可以用于一组最多不超过 4 个的外设。这种电路能串接起来，以联接更多的外设。在电路中任何一个 IEO 到 P 输出端的最大门延为 25 ns。

关于 74LS191 和 74S182 的详细功能，可参阅美国德克萨斯仪器(TEXAS INSTRUMENTS)公司 TTL 集成电路特性应用手册。

3．中断嵌套

如果在一个外设的中断服务程序执行期间开放中断，就可以使优先权比它高的外设，能够中断现行的服务程序，即暂时挂起优先权较低的外设，转而为优先权较高的外设服务。然后，程序自动返回继续执行优先权较低的中断服务程序。这种过程称为中断嵌套或多重中断。

Z 80 CPU 利用与之配套的 I/O 接口电路的链形优先权电路，不用外加逻辑电路，便可以实现中断嵌套。只要在中断时序开始后，用一条 EI 指令来开放中断就可以了。

为说明中断嵌套结构，我们仍以图 12-29 所示的链形中断优先权电路为例加以说明。

如果 CTC 和 PIO 在链形中断中的位置如图 12-29 所示。若一开始 CTC 没有中断请求，则它的 IEO 输出为高，当 PIO 有中断请求时，就使 $\overline{INT}$ (向 CPU 发出的中断请求信号)和 IEO 变低。设此时中断是开放着的，Z80CPU 在完成了现行指令后，发出中断响应信号($\overline{M_1}$ 和 $\overline{IORQ}$ 同时为低电平)，读取中断矢量，把 PC 内容推入堆栈，转至 PIO 中断服务程序。Z 80 CPU 在每次响应中断请求后，便自动关闭中断允许触发器 ($IFF_1 = IFF_2 = 0$)。如果要求实现中断嵌套的话，就必须在 PIO 中断服务程序中，在保护了现场后即用 EI 指令再开放中断。

如果在执行 PIO 服务程序中，CTC 发出中断请求，由于它的优先权高于 PIO，故可发出新的中断请求信号 $\overline{INT}$ 给 CPU，同时使 CTC 的 $\overline{HELP}$、IEO 变为低电平，亦即使 PIO 的 IEI 变为低电平，封锁了后面的任何外设。Z 80 CPU 在完成了 PIO 服务程序中的现行指令后，响应 CTC 的中断请求，读取中断矢量，把现行的 PC 推入堆栈，再次关闭中断允许触发器 ($IFF_1 = IFF_2 = 0$)。然后，转至 CTC 的服务程序。此时，堆栈中的内容如图 12-34 所示。

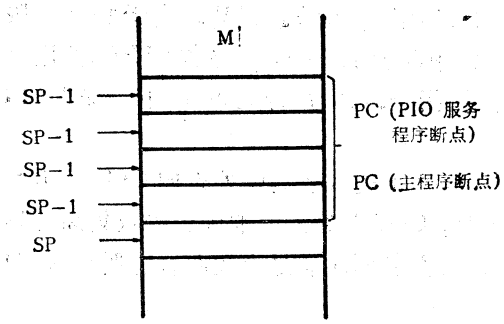


图 12-34　中断嵌套的堆栈示意图

在完成了 CTC 服务程序后，RETI (中断返回)指令把 PC 的内容恢复成 PIO 服务程序的断点地址，此时，CTC 的 IEO 变为高电平，PIO 的 IEI 也变为高电平，表示 PIO 能继续执行它的中断服务程序。其时序如图 12-35 所示。

执行完 PIO 服务程序后，另一条 RETI 指令(ED4D)使 PIO 的 IEO 变为有效，从而开放后面的外设。同时，将主程序的断点地址弹回 PC，继续执行主程序。

但是，若在执行 PIO 服务程序期间，没有及时开放中断，而只是在 PIO 的中断处理结束并恢复现场后才开放中断。这样，虽然 CTC 的中断级别比 PIO 高，但在 PIO 处理过程中，当 CTC 有中断请求时，CPU 也不会响应。直至 PIO 服务至开放中断后，CPU 才会响应。其时序如图 12-36 所示。

由于一开始是 PIO 先有中断请求，而 CTC 尚未中断请求，因此 IEI(PIO)即 IEO (CTC)为“高”，于是 PIO 可以发出中断请求。如果 CPU 的中断是开放的，则 CPU 响应中断，转至 PIO

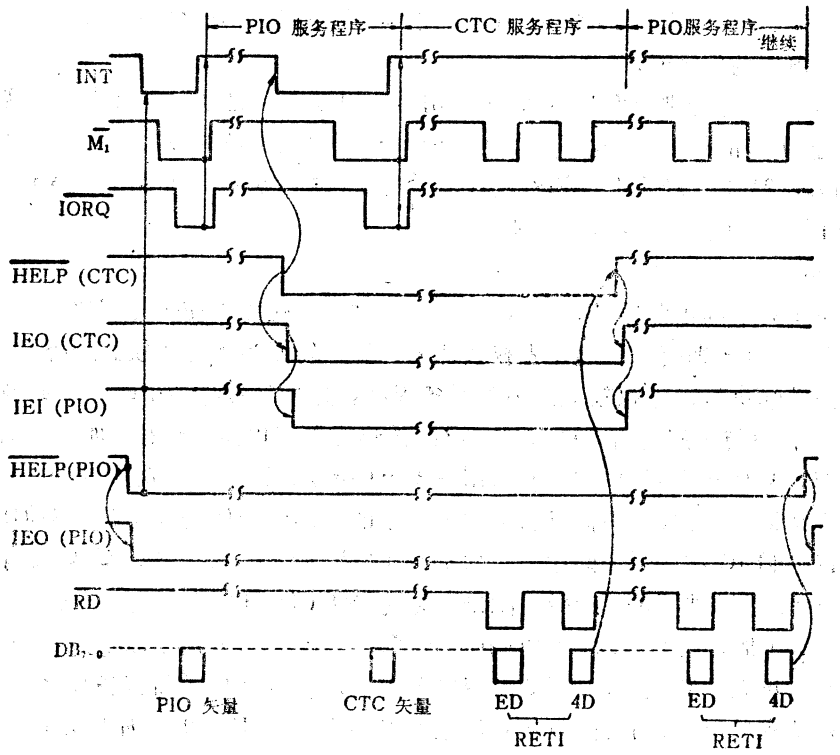


图 12-35 低级中断处理过程中开放中断的中断嵌套时序图

服务程序。若此后 CTC 有中断请求，这时因为它的 IEI 为高，故 CTC 可以发出中断请求，并且它使 CTC 的 IEO 和 PIO 的 IEI 变低(封锁低级外设提出中断请求)。但是，由于在 PIO 服务程序中，没有及时开放中断，故 CPU 不能响应 CTC 的中断请求，一直继续执行 PIO 的服务程序，直至 PIO 服务程序处理完毕，恢复现场后，执行 EI 指令，且在执行完 EI 的下一条 RETI 指令后，CPU 才能响应 CTC 的中断请求。

比较上述两种情况可知：链形优先权排队电路，能保证优先权较高的外设向 CPU 发出中断请求，屏蔽较低级的外设向 CPU 发出中断请求。但是，CPU 是否能够响应优先权较高的外设的中断请求，还取决于 CPU 的中断是否开放。换言之，必须同时满足上述两者的要求，CPU 才能响应优先权级别高的外设的中断请求。

但是，在执行 RETI 指令期间，CTC 将允许它的 IEO 在一个 $\overline{M_1}$ 周期中变为高电平。也就是说，如果优先权高的外设有中断请求，但未被响应，则在 ED(RETI 指令的第一个字节)被译码后，它的 IEO 将在一个 $\overline{M_1}$ 周期中强制再次变为高电平，以使后面的外设对 RETI 译码，使已被响应的外设恢复起始状态。

4．Z80 中断控制逻辑

如上所述，Z80 系列的 I/O 接口芯片的中断控制逻辑，具有下列功能：

(1) 当外设有中断请求，并且在它的 IEI 为高电平时，就能向 CPU 发出中断请求；

(2) 具有链形优先权中断排队电路的输入 IEI 和输出 IEO；

(3) 在 CPU 发出中断响应时，能通过数据总线输出 8 位中断矢量地址；

(4) 在中断返回时，能恢复初始状态。

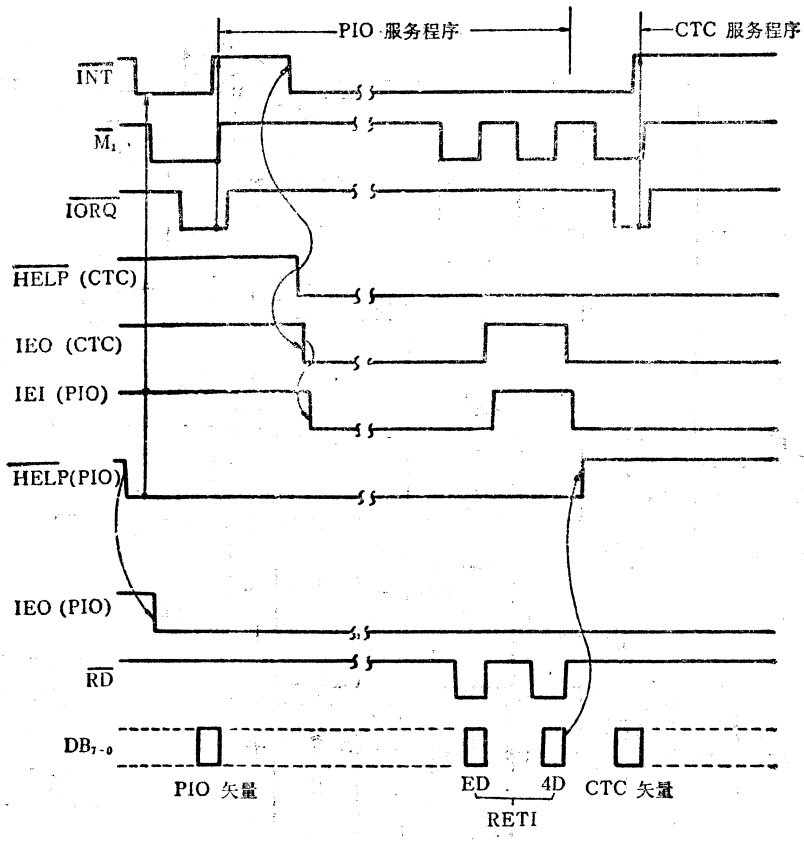


图 12-36 低级中断处理过程中未开放中断的时序图

能实现上述功能的中断控制逻辑电路如图 12-37 所示。

其工作过程如下：当外设要求中断服务时，它使中断请求触发器Ⓐ (FFA) 置“1”。当 IEI = “1”，且又不处在中断服务(用触发器Ⓑ表示) 时，即可输出 $\overline{INT}$ 信号，并保持到CPU响应为止。这就保证了对优先权结构的系统中的外设的响应和服务。

当输入 IEI = “0” 或本装置有中断请求时，则 IEO = “0”，封锁较低级的中断请求。

当 CPU 响应中断请求时，就使 $\overline{M_1}$ 和 $\overline{IORQ}$ 同时有效，作为中断响应信号 $\overline{INTA}$。它使中断请求触发器Ⓐ复位，而使Ⓑ触发器置位。此时，IEO 保持低电平，以保证在中断服务结束前仍封锁较低级的中断请求。

如果 IEI 为高电平，且触发器Ⓑ置“1”，则 INTA 还允许把本装置的中断矢量地址送至数据总线，以便与 Z 80 CPU 的 I 寄存器的内容组合，形成 16 位中断矢量地址指针(在方式 2 中断时)；若为方式 0 中断，则这 8 位中断矢量即为一个 RST 指令。于是，CPU 就转向相应的中断服务程序。

当中断服务完毕恢复现场后，返回指令 RETI 应清除触发器Ⓑ，开放 IEO，此时由 256×4 ROM 辨别数据总线上的指令码，将 “ED”(RETI 指令的第一个字节) 和 “4D” (RETI 指令的第二字节) 译码，在 $\overline{M_1}$ 和 $\overline{RD}$ 有效(即取指周期) 的条件下，在译码后的 “4D” 信号打入Ⓒ触发器的同时，已译码的 “ED” 信号也被打入Ⓓ触发器中，它们经一个三输入与非门后使触发器Ⓑ复位，结束整个中断周期。

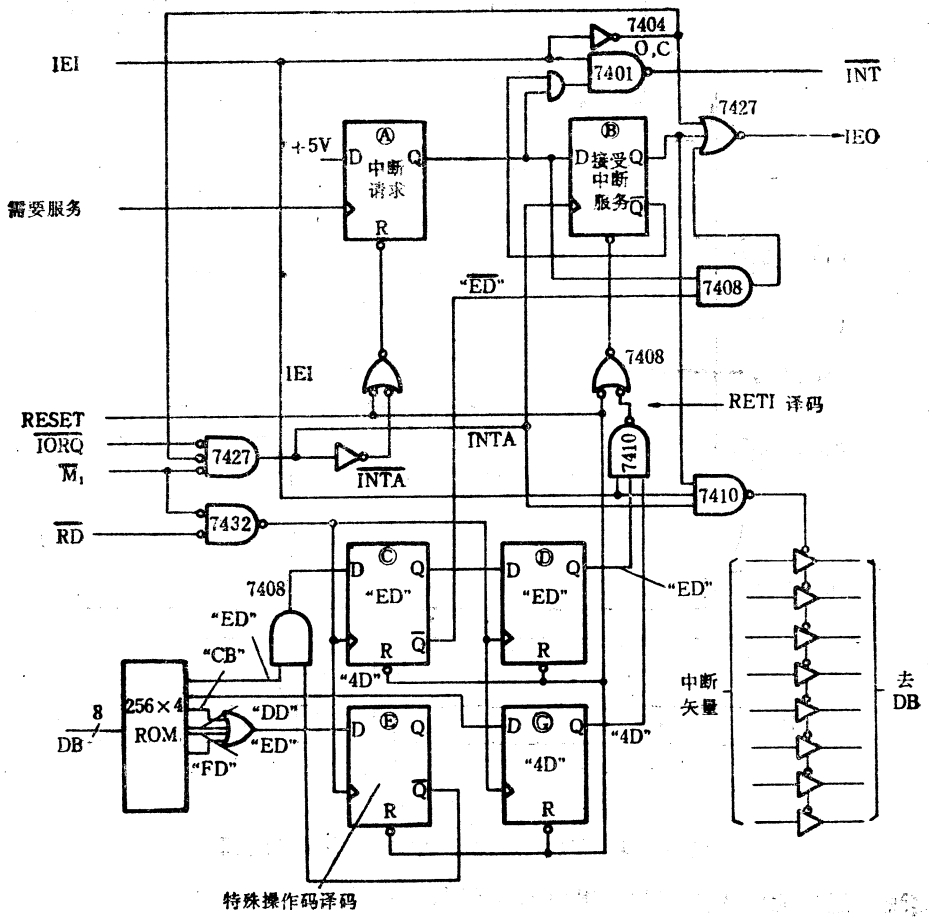


图 12-37　Z80 中断控制逻辑图

但是，在 Z80 中有些指令的字节中有 ED 或 4D，如：

```
SET    5,  L    CB
                ED
LD     C,  L    4D
```

在上述程序中，虽然 ED 后面跟着 4D，但不是 RETI 指令，为了避免在这种情况下被控制电路误认为是返回指令，就对可能产生这种混淆的某些特殊指令进行译码，若检测到 CB，就使触发器Ⓔ置"1"，其反端封锁与门，从而阻止把后一字节的 ED 指令打入触发器Ⓒ。

如果在本装置有中断请求前，较低级中断源已申请中断，且处在中断处理过程中，但 CPU 未再开中断。此时，本装置即使有中断请求，但 CPU 不响应，IEO 输出已变"低"，使较低级的 IEI 线变低。当中断处理进行到执行 RETI 指令时，较低级外设中的"ED"译码器Ⓓ和"4D"译码器Ⓖ已准备好，但由于它的 IEI 被较高级中断封锁了，就无法输出去清除它的中断服务触发器Ⓑ。因而，要求已提出中断请求而尚未被响应的较高级中断源能在此时把它的 IEO 输出暂时变高，以便使较低级中断能够得以恢复初始状态。这个要求是这样来实现的：当数据总线上出现了 RETI 的第一个字节"ED"时，被打入Ⓒ触发器，则它的反端 $\overline{ED}$ 为"低"，此时由于中断未被响应，故 Q_B = "0"，IEI 经反相后为 0，于是 IEO 输出就暂时变高，实现了上述要求。

第十三章　可编程并行 I/O 接口电路

CPU 与外围设备之间的信息交换是通过 I/O 接口电路来实现的。按数据传送方式来分，有并行 I/O 接口电路和串行 I/O 接口电路。而且，为了方便用户，把 I/O 接口电路制成为通用的可编程序的 I/O 接口电路。所谓可编程序 I/O 接口电路，是指利用编程序的方法，使一个 I/O 接口电路片能按几种不同的方式工作。可编程序 I/O 接口芯片种类繁多，表 13-1 和表 13-2 分别列出 Intel 8080 A/8085 A 和 Z 80 微型计算机系统中常用的 I/O 接口电路。本章仅介绍可编程序并行 I/O 接口电路。

表 13-1　8080 A/8085 A 微型计算机 I/O 接口芯片

	分　　类	芯　片　名　称
通用接口芯片	串行接口	8251 A 可编程序通信接口
	并行接口	8212 8 位通用 I/O 接口 8255 A 可编程序 I/O 接口(PPI)
	定时控制	8253 A 可编程序定时器
	DMA 控制	8257 可编程序 DMA 控制器
	中断控制	8214 优先权中断控制器 8259 可编程序中断控制器
专用接口芯片	软盘控制	8271 可编程序软盘控制器
	CRT 控制	8275 可编程序 CRT 控制器
	键盘显示控制	8279 可编程序键盘显示控制器

表 13-2　Z80 微型计算机 I/O 接口芯片

分　　类	芯　片　名　称
定时控制	Z 80 CTC 可编程序计数器定时器
并行接口	Z 80 PIO 可编程序并行 I/O 接口
串行接口	Z 80 SIO 可编程序串行 I/O 接口
DMA 控制	Z 80 DMA 可编程序 DMA 控制器

§13-1　可编程序并行 I/O 接口电路 Intel 8255 A

Intel 8255 A 可编程序 I/O 接口电路(PPI——Programmable Peripheral Interface)是一种通用的 8 位并行 I/O 接口芯片。它采用 40 条引线的双列直插式封装。它有 24 条 I/O 线，分成三个端口(A、B 和 C)。每个端口的功能可用程序进行设定。它与外围设备连接时，通常不需要附加逻辑电路。

8255 A 的引线图和逻辑符号图如图 13-1 所示。

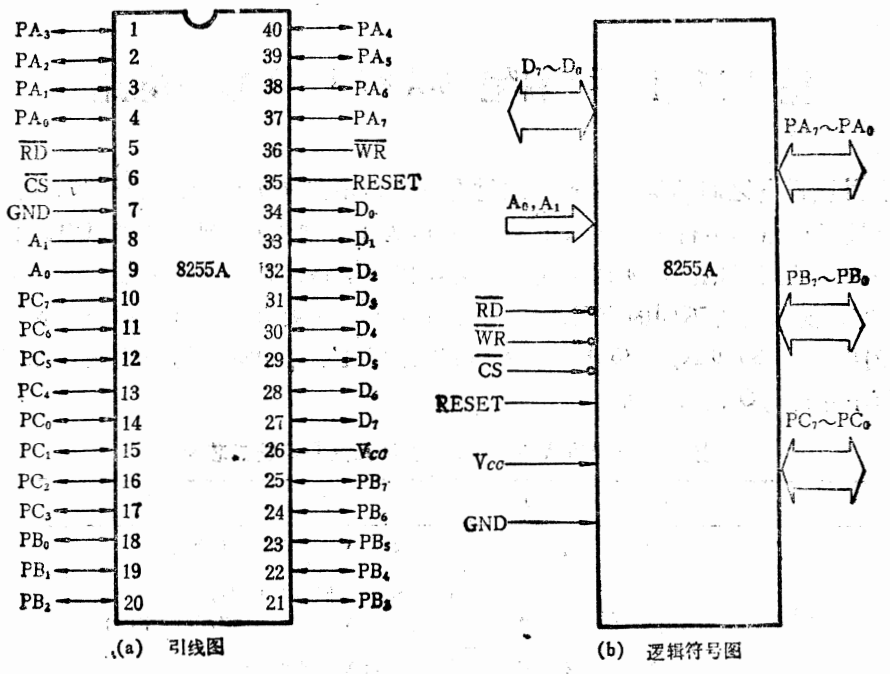


图 13-1 8255A 的引线图和逻辑符号图

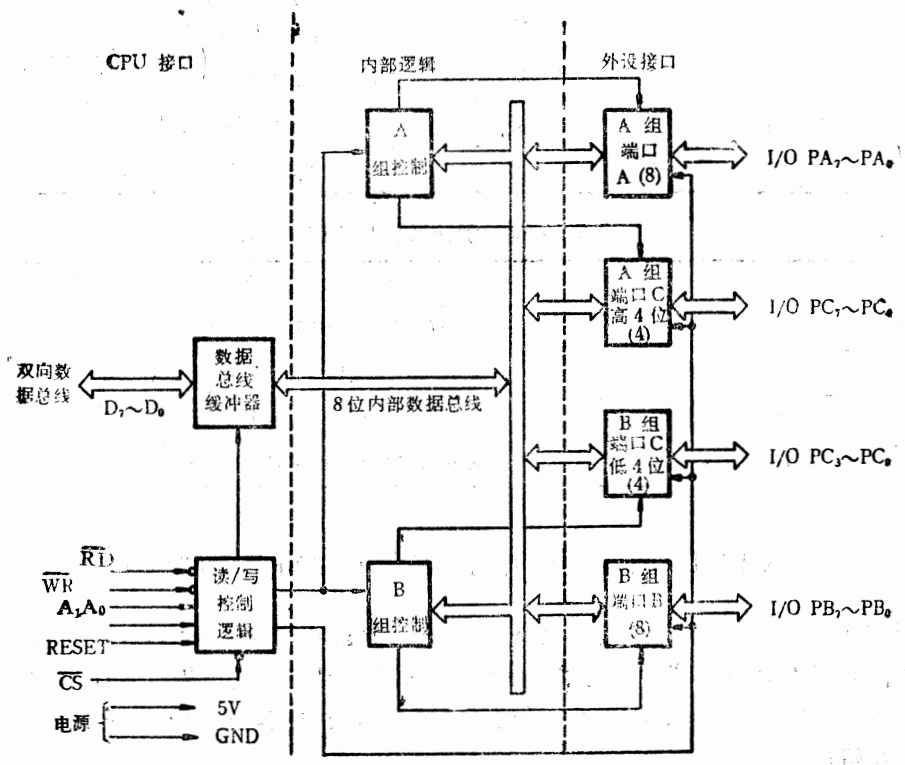


图 13-2 8255 A 方框图

8255A 引线名称表

D_7～D_0	双向数据总线	A_1, A_0	端口地址
RESET	复位输入	PA_7～PA_0	端口 A
$\overline{CS}$	片选	PB_7～PB_0	端口 B
$\overline{RD}$	读命令	PC_7～PC_0	端口 C
$\overline{WR}$	写命令	V_{CC} GND	+5V 0V

一、8255 A 的组成和功能

8255 A 由 A、B、C 三个双向 8 位 I/O 端口(其中 C 端口分成高 4 位和低 4 位) 以及数据总线缓冲器和控制逻辑等部分组成,如图 13-2 所示。

1. 数据总线缓冲器

它是一个三态、双向、8 位的缓冲器,用作 8255 A 与系统数据总线相连接的缓冲部件。缓冲器发送或接收数据是靠 CPU 执行输入或输出指令来实现的。8255 A 的控制字和状态字也是通过数据总线缓冲器传送的。

2. 读/写和控制逻辑

该部件的功能是管理数据和控制字(或状态字)的内部和外部的传送。它接收来自 CPU 地址总线的 A_1、A_0 和控制总线的有关信号($\overline{RD}$、$\overline{WR}$、RESET 等),然后向 8255 A 的 A、B 两组控制部件发送命令。

(1) $\overline{CS}$(Chip Select)片选信号

当"片选"输入线处于低电平时,允许 8255 A 和 CPU 之间进行信息交换。

(2) $\overline{RD}$ 读命令

当这个输入线处于低电平时,允许 8255 A 通过数据总线向 CPU 发送数据或状态信息。实际上,该引线使 CPU 从 8255 A 读取信息。

(3) $\overline{WR}$ 写命令

当这个输入线处于低电平时,允许 CPU 把数据或控制字写入 8255 A 中去。

(4) RESET 复位信号

当该输入线处于高电平时,所有内部寄存器(包括控制寄存器)都被清除,而端口 A、B、C 则被设置为输入方式。

(5) 端口寻址 A_1、A_0

这两个输入端用来选择 8255A 中三个端口之一,或选择控制字寄存器。它们通常与地址总线的最低有效位(A_1 和 A_0)相连接。

A_1、A_0 与 $\overline{CS}$、$\overline{RD}$ 及 $\overline{WR}$ 组合,所实现的各种功能见表 13-3。

3. A 组和 B 组的控制电路

这两组控制电路是根据 CPU 的命令字来控制 8255 A 工作方式的。也就是说,CPU 必须先向 8255 A 输出一个控制字。控制字含有诸如"工作方式"、输入/输出、"置位"、"复位"等命令信息。这些命令决定 8255 A 各端口的工作方式。

每个控制组(A 组和 B 组)都接收来自读/写控制逻辑的"命令",接收来自内部数据总线的"控制字",并向与其相连的端口发出适当的控制信号。

表 13-3　8255 A 的基本操作表

A_1	A_0	RD	WR	CS	输 入 操 作 (读 出)
0	0	0	1	0	端口 A→数据总线
0	1	0	1	0	端口 B→数据总线
1	0	0	1	0	端口 C→数据总线
					输出操作(写入)
0	0	1	0	0	数据总线→端口 A
0	1	1	0	0	数据总线→端口 B
1	0	1	0	0	数据总线→端口 C
1	1	1	0	0	数据总线→控制字寄存器
					功能失效
×	×	×	×	1	数据总线→三态(禁止使用)
1	1	0	1	0	非法状态
×	×	1	1	0	数据总线→三态

A 组控制部件用来控制端口 A 和端口 C 高位部分(PC_7～PC_4)；

B 组控制部件用来控制端口 B 和端口 C 低位部分(PC_3～PC_0)；

控制字寄存器只能写入，而不允许读出操作。

4．端口 A、B 和 C

8255 A 有三个 8 位端口(A、B 和 C)。所有端口都可由程序设定为各种不同的工作方式。

(1) 端口 A 有一个 8 位数据输出锁存/缓冲器和一个 8 位数据输入锁存器。

(2) 端口 B 有一个 8 位数据输入/输出、锁存/缓冲器和一个 8 位数据输入缓冲器。

(3) 端口 C 有一个 8 位数据输出锁存/缓冲器和一个 8 位数据输入缓冲器 (输入没有锁存)。

二、8255 A 的方式选择

1．方式选择控制字

在使用 8255 A PPI 时，首先要由 CPU 输给 8255 A 的控制寄存器一个控制命令字。这个控制字是由用户用程序输入的，它决定了 8255 A 三个端口的功能和工作方式。然后，再用数据传送指令来执行 CPU 与 I/O 端口之间的数据传送。8255 A 的三种基本工作方式如图 13-3 所示。

8255 A 可由程序来选择的三种工作方式：

方式 0 (Mode 0)——基本输入/输出方式

方式 1 (Mode 1)——选通输入/输出方式

方式 2 (Mode 2)——双向输入/输出方式

当 RESET 输入为高电平时，所有端口都被置成输入方式，亦即 24 条引线全部为高阻状态。RESET 撤除后，8255 A 仍可保持这种输入方式，而不需要附加初始条件。端口 A 和端口 B 的工作方式可分别定义；而端口 C 可视需要，并根据端口 A 和端口 B 的定义而分成两部分。

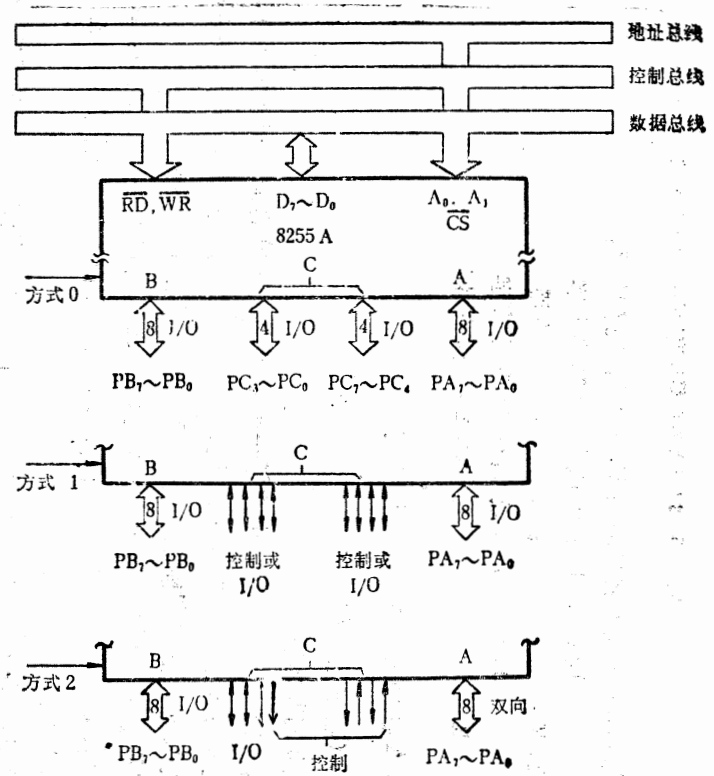


图 13-3　8255 A 的基本工作方法

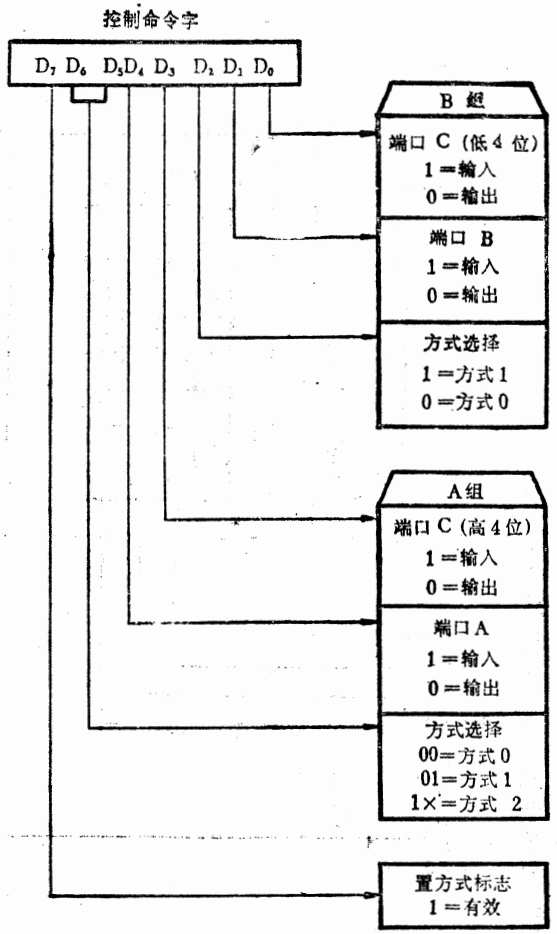


图 13-4　8255 A 方式控制字格式

几种工作方式可以组合起来使用。8255 A 工作方式的定义格式见图 13-4。

2．C 端口位置位/复位功能

8255 A 端口 C 的每一位都可以用 OUT 指令来置位或复位。在微型计算机控制应用中，这一特点使软件设计大为简化。例如，当 8255 A 工作在方式 1 或2时，可以用端口 C 的位置位/复位的功能使 INTE 触发器置“1”或置“0”，从而允许或禁止从端口 C 产生的中断请求信号。C 端口的位置位/复位控制字格式如图 13-5 所示。

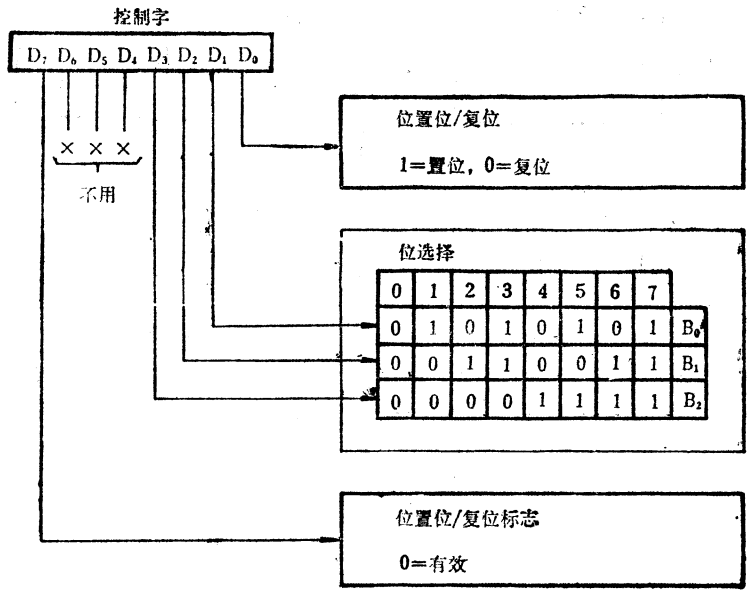


图 13-5　8255 A 端口 C 位置位/复位控制字格式

【例】 下面的程序段可将 C 端口 C_2 置“0”，C_3 置“1”：

```
MVI   A,      00000100B   ; 将 C2 位置“0”的控制字
OUT   CNTLR               ; 送 8255 A 控制寄存器
MVI   A,      00000111B   ; 将 C3 位置“1”的控制字
OUT   CNTLR               ; 送 8255 A 控制寄存器
```

三、8255 A 各种工作方式的功能说明

1．方式 0（基本输入/输出）

(1) 方式 0 的基本功能

这种逻辑结构对于三个端口中的每一个都可提供简单的输入和输出操作，而不需要“信息交换”，数据既可写入指定的端口，亦可从其中读取。在这种方式下，8255 A 可产生 16 种不同的 I/O 组合方式，此时输出可被锁存，但输入只有缓冲作用而不能锁存。

【例】 把控制字 10011000 (98 H) 送入 8255 A 的控制寄存器。这表明：A 端口用作输入，B 端口用作输出，C 端口低 4 位作为输出，C 端口高 4 位作为输入。这时，A 组和 B 组都处于方式 0 的工作方式。

若 8255 A 用在 8080 A/8085 A 微型计算机系统中，则首先要确定 8255 A 控制寄存器的地址码。例如，若把控制寄存器也当作 CPU 的一个 I/O 端口，并设定其地址为 CNTLR，则可

用下列两条指令来设置控制字。

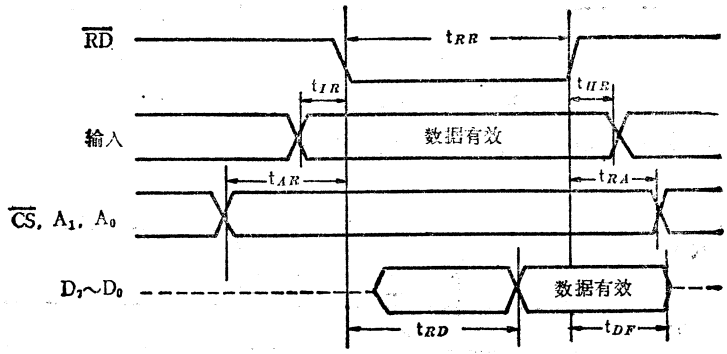
```
MVI  A, 98 H
OUT  CNTLR
```

这样，就可把控制字 98 H 送入控制寄存器，从而确定 8255 A 的工作方式。然后，再按照 A、B、C 的端口地址，进行 CPU 与 I/O 设备之间的数据传送。

(2) 方式 0 的时序

① 方式 0 的基本输入时序

在方式 0 时，基本的输入时序如图 13-6 所示。



符　号	参　数	8255 A		单　位
		min	max	
t_{RR}	读脉冲宽度	300		ns
t_{IR}	输入超前于 $\overline{RD}$ 的时间	0		ns
t_{HR}	输入滞后于 $\overline{RD}$ 的时间	0		ns
t_{AR}	地址稳定超前读信号的时间	0		ns
t_{RA}	读信号无效后地址保持时间	0		ns
t_{RD}	从读信号有效到数据稳定		250	ns
t_{DF}	读信号去除后到数据浮空	10	150	ns
t_{RV}	在两次读(或写)之间的时间间隔	850		ns

图 13-6　8255 A 方式 0 的输入时序波形图

若外设的数据已经准备就绪，CPU 用输入指令从 8255 A 读入此数据，则 $\overline{RD}$ 的宽度至少应为 300 ns，且地址信号必须在 $\overline{RD}$ 有效前 t_{AR} 时间有效。这样在 $\overline{RD}$ 有效后，经过 t_{RD} 时间，数据即可在数据总线上稳定。

Intel 8255 A 虽然是与 Intel 8080 A/8085 A CPU 配套的 I/O 接口电路，但也适用于 Z 80 CPU。图 13-7 示出 Z 80 CPU 的输入时序及有关参数。

若以 $\overline{IORQ}$ 与地址信号作为片选，则略领先于 $\overline{RD}$ 信号而可满足 t_{AR} 的要求。$\overline{RD}$ 信号的宽度约为两个半 T 时态，故远大于 $t_{RR}=300$ ns。另外，从 $\overline{RD}$ 有效到数据应该稳定的时间间隔为

$$T_1 = 2.5T - t_{DL\bar{\phi}}(RD) - t_{S\bar{\phi}}(D) = 1000 - 100 - 60 = 840 \text{ ns}$$

也远大于 $t_{RD}=250$ ns。因而，Z 80 的输入时序与 8255 A 在时间上是能够匹配的。

② 方式 0 的基本输出时序

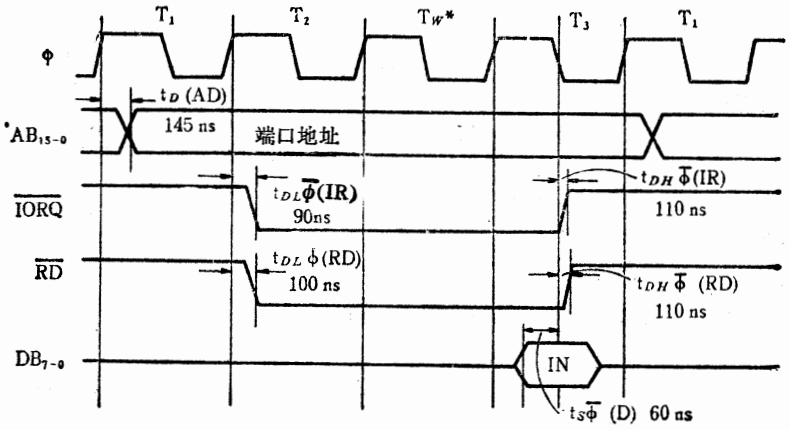
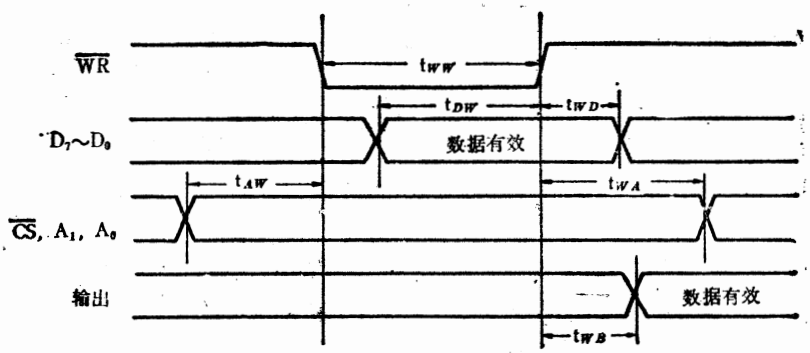


图 13-7　Z 80 CPU 的输入时序及有关参数

在方式 0 时，基本的输出时序如图 13-8 所示。



符　号	参　数	8255 A		单　位
		min	max	
t_{AW}	地址稳定超前写信号的时间	0		ns
t_{WA}	写信号后地址保持时间	20		ns
t_{WW}	写脉冲宽度	400		ns
t_{DW}	从写信号到数据有效	100		ns
t_{WD}	数据保持时间	30		ns
t_{WB}	从写信号结束到输出		350	ns

图 13-8　8255 A 方式 0 的输出时序波形图

CPU 用输出指令把数据输出给外设。对 8255 A 来说，要求写脉冲宽度至少为 400ns。且地址信号必须在写信号前 t_{AW} 时间有效，并在写信号结束后再保持 t_{WA} 时间。另外，输出的数据必须在写信号结束前 t_{DW} 时间有效(出现在数据总线上)，且在写信号结束后保持 t_{WD} 时间。这样，在写信号后最多 t_{WB} 时间写出的数据出现在输出端口上。

图 13-9 示出 Z 80 CPU 的输出时序及有关参数。

写脉冲的宽度大于两个 T 时态，它远大于 t_{WW} = 400 ns；输出的数据在 T_1 时态结束时已出现在数据总线上，故它与 $\overline{WR}$ 信号失效之间的时间也约为 2.5 T 时态，远大于 t_{DW} = 100 ns。因而，8255 A 的输出时序也能与 Z 80 的输出时序匹配。

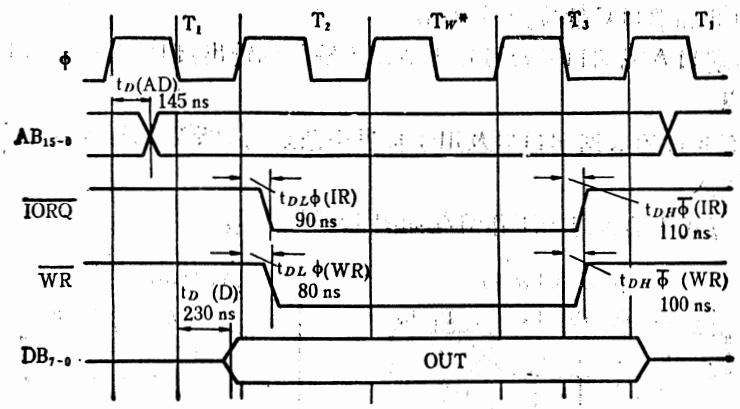


图 13-9　Z 80 CPU 的输出时序波形图

③ 方式 0 应用举例

通常，方式0可用于无条件传送方式的接口电路中，这时不需要状态端口，三个端口都可作为数据端口；也可用于查询式的 I/O 接口电路中，此时端口 A 和 B 可分别作为两个数据端口，而取端口 C 的某些位作为这两个数据端口的控制和状态信息。

图 13-10 示出了一个采用方式 0 工作的 8255 A 与 A/D、D/A 转换器连接的电路图。

图中 A/D、D/A 转换器都是 8 位转换器，其控制信号含义如下：

STB　DATA——数据选通信号，这是一个表示 CPU 把数据输出给 D/A 转换器的选通信号；

SAMPLE　EN——允许采样信号，这是一个允许 A/D 转换器开始采样的控制信号；

OUTPUT　EN——允许输出信号，这是一个表示 A/D 转换器转换结束后，CPU 可读入数据的控制信号。

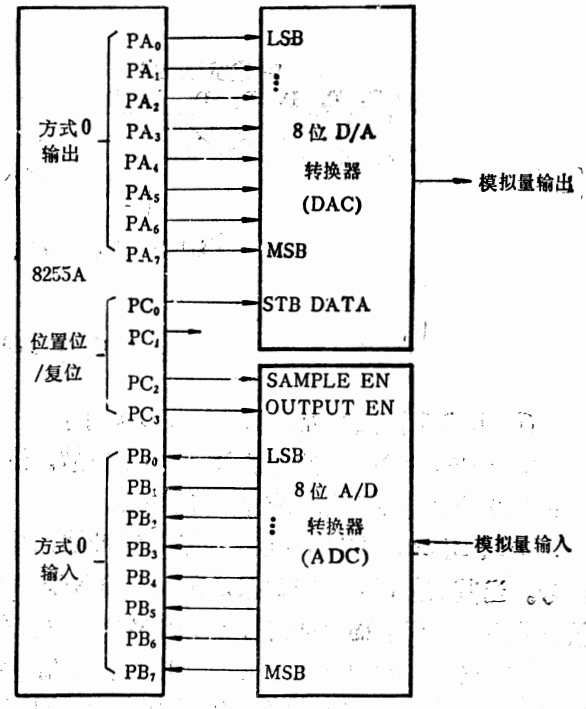


图 13-10　8255 A 方式 0 应用举例

在确定了各个端口的地址后，便可用初始化程序设定 8255 A 的控制命令字为 10000010 B(即 82 H)，并用输出指令把其送至 8255 A 的控制寄存器。这样，就确定了 8255 A 的工作方式为方式 0，端口 A 为输出口，端口 B 为输入口。根据外围设备的需要，把端口 A 作为 8 位 D/A 转换器的接口；端口 B 作为 8 位 A/D 转换器的接口；端口 C 若干位作为 D/A、A/D 转换器的控制信号线，它可以用端口 C 的位置位/复位控制字来设置。

2. 方式 1 (选通 I/O)

(1) 方式 1 的基本功能

方式 1 是一种应答式的 I/O 操作方式。它将三个端口分成 A、B 两组，即端口 A 和端口 C 高位为一组，端口 B 和端口 C 低位为另一组。

在方式 1 时，端口 A 或端口 B 都可作为数据的输入或输出，但同时规定端口 C 的某些位作为控制或状态信息。

每一组包含有 8 位的数据端口以及用于提供中断逻辑的指定的三条控制或状态线。

(2) 方式 1 输入

端口 A、B 在方式 1 输入时的工作情况如图 13-11 所示。

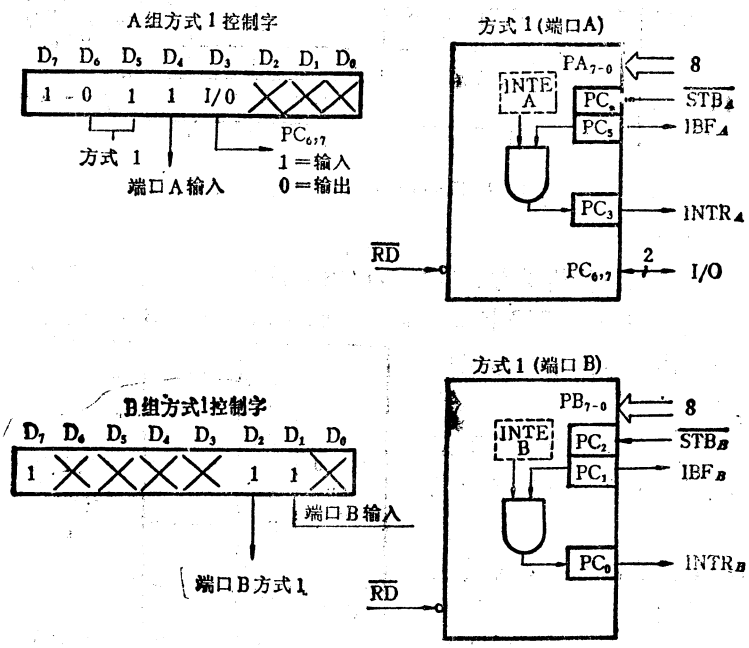


图 13-11 方式 1 输入

① 方式 1 输入时控制信号的定义如下：

$\overline{STB}$ (Strobe)——选通输入，低电平有效。这是由外设提供的输入信号，当其有效时，将输入设备来的数据送入端口的输入锁存器。

IBF(Input Buffer Full)——输入缓冲器满信号，高电平有效。这是由 8255 A 输出的状态信号。当其有效时，表明数据已输入至输入锁存器。当 $\overline{STB}$ 信号有效（低电平）时，IBF 就被置位（高电平），而 $\overline{RD}$ 信号的上升沿使其复位（低电平）。

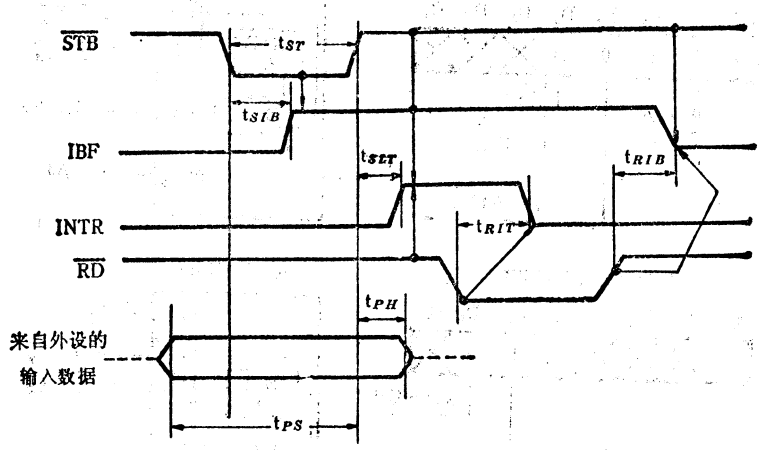
INTR(nterrupt Request)——中断请求信号，高电平有效。当有一输入设备请求服务时，8255 A 就用 INTR 输出端（高电平）作为向 CPU 提出中断请求信号，以请求 CPU 为其服务。当 $\overline{STB}$、IBF 和 INTE（中断允许）都为高电平时，INTR 输出端才被置成高电平。它由 $\overline{RD}$ 信号的下降沿清除。

INTE A (nterrupt Enable A)——端口 A 中断允许信号，由 PC_4 的置位/复位来控制，$PC_4 = 1$ 时允许端口 A 中断。

INTE B——端口 B 中断允许信号，由 PC_2 的置位/复位来控制，$PC_2 = 1$ 时，允许端口 B 中断。

② 方式 1 的输入时序

方式 1 的输入时序及有关参数如图 13-12 所示。



符　号	参　数	8255 A		单　位
		min	max	
t_{ST}	STB 脉冲宽度	500		ns
t_{SIB}	STB=0 到 IBF=1		300	ns
t_{SIT}	STB=1 到 INTR=1		300	ns
t_{RIB}	RD=1 到 IBF=0		300	ns
t_{RIT}	RD=0 到 INTR=0		400	ns
t_{PS}	数据提前 STB 无效的时间	0		ns
t_{PH}	数据保持时间	180		ns

图 13-12　方式 1 的输入时序波形图

当外设的数据已经准备就绪时，就用选通信号把数据锁入 8255 A 的输入锁存器，其脉宽至少为 500 ns。选通信号经过时间 t_{SIB} 后，IBF 有效，可供 CPU 查询；在选通信号结束后，经过 t_{SIT} 后向 CPU 发出 INTR 信号(如果中断是允许的话)，CPU 响应中断，进入中断服务程序，发出 $\overline{RD}$ 信号，把数据读入 CPU。在 $\overline{RD}$ 信号有效后，经过 t_{RIT} 时间就清除中断请求。当 $\overline{RD}$ 信号结束后，数据已读至 CPU，使 IBF 变为低电平。

(3) 方式 1 输出

端口 A、B 在方式 1 输出时的工作情况如图 13-13 所示。

① 输出时控制信号的定义如下：

$\overline{OBF}$(Output Buffer Full)——输出缓冲器满信号，低电平有效。这是 8255 A 输出给外设的一个控制信号。当其有效时，表示 CPU 已经把数据输出给指定的端口。它由输出命令 $\overline{WR}$ 的上升沿置成有效(低电平)，而由 $\overline{ACK}$ 信号的有效低电平使其恢复为高电平。

$\overline{ACK}$(Acknowledge)——应答输入信号，低电平有效。它是一个来自外设的应答信号，表明 CPU 输出给 8255 A 的数据已经被外设接受。

INTR——中断请求信号，高电平有效。当输出设备已经接收了 CPU 输出的数据后，8255 A 就用 INTR 输出端向 CPU 提出中断请求信号，要求 CPU 继续输出数据。INTR 是当 $\overline{ACK}$、$\overline{OBF}$ 和 INTE 都为高电平时，才被置成高电平的，而由 $\overline{WR}$ 信号的下降沿使其复位(低电平)。

INTE　A 由 PC_6 的置位/复位来控制。

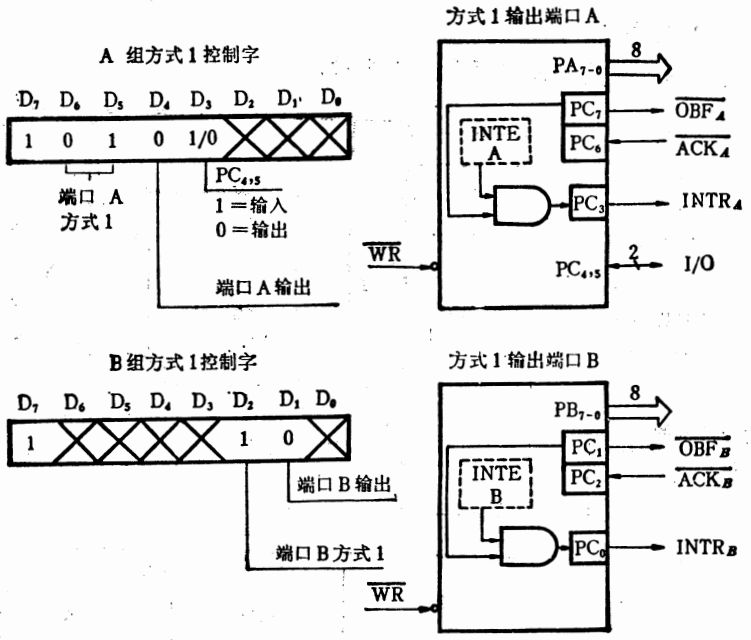
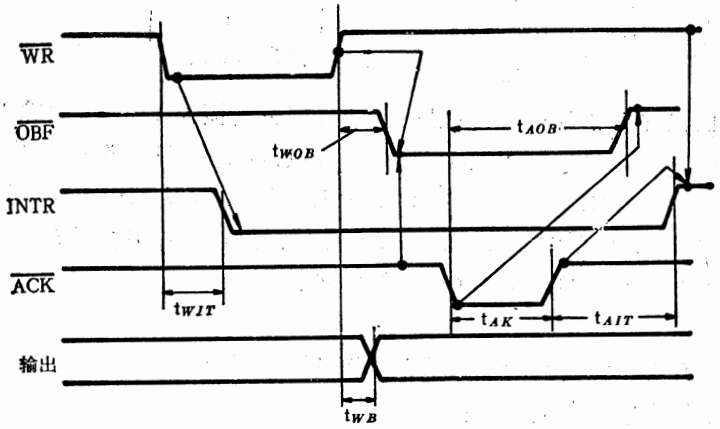


图 13-13 方式 1 输出

INTE B 由 PC_2 的置位/复位来控制。

② 方式 1 的输出时序

方式 1 输出时序及有关参数如图 13-14 所示。



符号	参数	8255 A		单位
		min	max	
t_{WOB}	WR=1 到 OBF=0		650	ns
t_{WIT}	WR=0 到 INTR=0		850	ns
t_{AOB}	ACK=0 到 OBF=1		350	ns
t_{AK}	ACK 脉冲宽度	300		ns
t_{AIT}	ACK=1 到 INTR=1		350	ns
t_{WB}	WR=1 到输出		350	ns

图 13-14 方式 1 的输出时序波形图

8255 A 工作在方式 1 时的输出过程如下：假设从 CPU 响应中断开始，在中断服务程序中，CPU 输出数据并发出 $\overline{WR}$ 信号，$\overline{WR}$ 信号经过时间 t_{WIT} 后清除 INTR，且在其上升沿后使 $\overline{OBF}$ 有效（低电平），通知外设接受数据，实际上 $\overline{OBF}$ 可作为外设接受数据的选通信号。在 $\overline{WR}$ 上升沿后，经过时间 t_{WB} 数据开始输出。当外设接受数据后，发出 $\overline{ACK}$ 信号，一方面使 $\overline{OBF}$ 变为无效(经过 t_{AOB})，另一方面在 $\overline{ACK}$ 的上升沿使 INTR 有效（经过 t_{AIT})，发出新的中断请求。

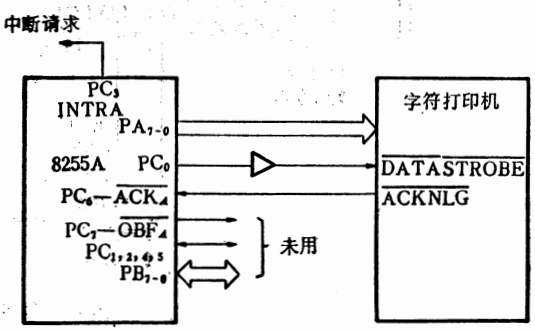


图 13-15　方式 1 输出举例

(4) 方式 1 应用举例

图 13-15 示出了工作在方式 1 的 8255 A，它用作按中断方式工作的打印机与 CPU 的接口电路。其中用端口 A 作为数据输出端口，用端口 C 的若干位作为“联络”信号。当外设不需要脉冲输入时，$\overline{OBF_A}$ 可以作为数据选通信号；但当要求一个脉冲的选通信号时，可由 CPU 通过 8255 A 的 PC_0 端来控制。$\overline{ACK_A}$ 接至打印机的 $\overline{ACKNLG}$ 端。

① 确定方式控制字

在初始化程序中，首先要确定端口地址，然后确定方式控制字，本例中的控制字如图 13-16 所示。

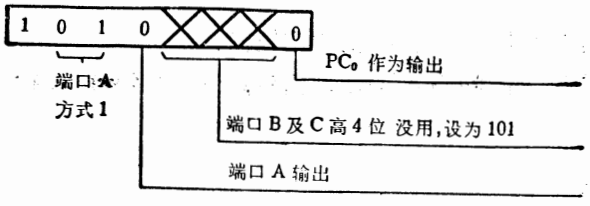


图 13-16　确定方式控制字

方式控制字为 ICW = 10101010B 即 0AAH。

然后，通过以下指令送入 8255A 的控制寄存器 CWR，即

```
MVI   A, ICW
OUT   CWR    ；设置 8255A 的工作方式
```

② CPU 打印输出过程

通常，打印输出总是一个数据块，输出过程是由用户程序中调用打印子程序 PSTART 开始的。在中断服务程序中处理字符输出，当字符已输送至输出缓冲器时，CPU 用将 PC_0 置位/复位的命令输出至打印机的 $\overline{DATA\ STROBE}$ 端，命令打印机接收数据。当打印机把 CPU 输出的字符打印完毕后，就发出 $\overline{ACK}$ 信号，使缓冲器满 $\overline{OBF}$ 恢复为高电平，并使 8255 A 产生中断请求。如果中断是开放的，则 CPU 响应中断，转去执行中断服务程序。在中断服务程序中，先保护现场，接着查询中断源（当有多个中断源时）。当查到是打印请求时，就输出下一个字符。在整个数据块打印完毕后，就返回主程序。

3．方式 2 (选通双向总线 I/O)

方式 2 的逻辑结构可以通过一组 8 位的数据总线与 I/O 设备进行双向通信，即该总线既能发送，也能接收数据。工作时可用程序查询方式，也可用中断方式。

(1) 方式 2 的基本功能

方式 2 只限于 A 组使用，它用一个 8 位双向总线端口 A 和一个控制端口 C 中的 5 位进行操作。输入、输出都能锁存。端口 C 中有 5 位用作端口 A 的控制和状态信息。端口 B 可用于方式 0 或方式 1。

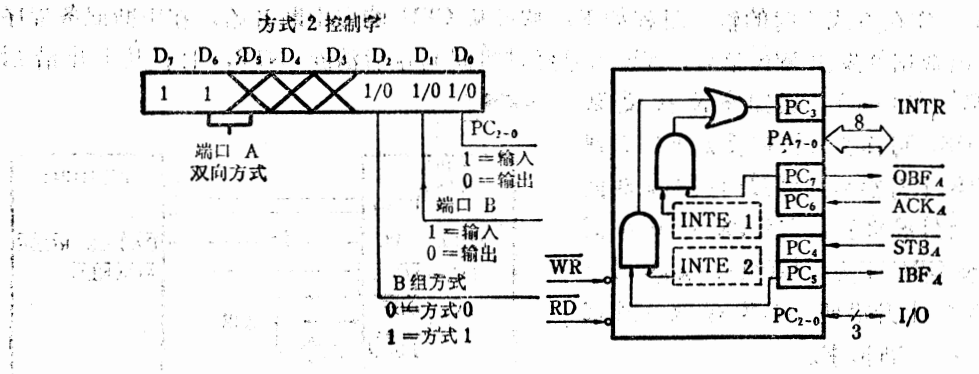


图 13-17 8255 A 方式 2

方式 2 的逻辑功能结构如图 13-17 所示。

(2) 选通双向操作时各控制信号的定义：

INTR——中断请求信号，高电平有效。在输入和输出时，都可用作为向 CPU 请求中断的信号。

$\overline{OBF}$——输出缓冲器满，低电平有效。它可用作对外设的选通信号，表示 CPU 已把数据输至端口 A。

$\overline{ACK}$——应答信号，低电平有效。当其有效时，表示开放端口 A 的三态输出缓冲器以输出数据，否则输出缓冲器处于高阻状态。

INTE1——与 $\overline{OBF}$ 有关的 INTE 触发器，它由 PC_6 的置位/复位来控制。

$\overline{STB}$——选通输入，低电平有效。它是外设供给 8255 A 的选通信号，当其有效时将输入数据选通至输入锁存器。

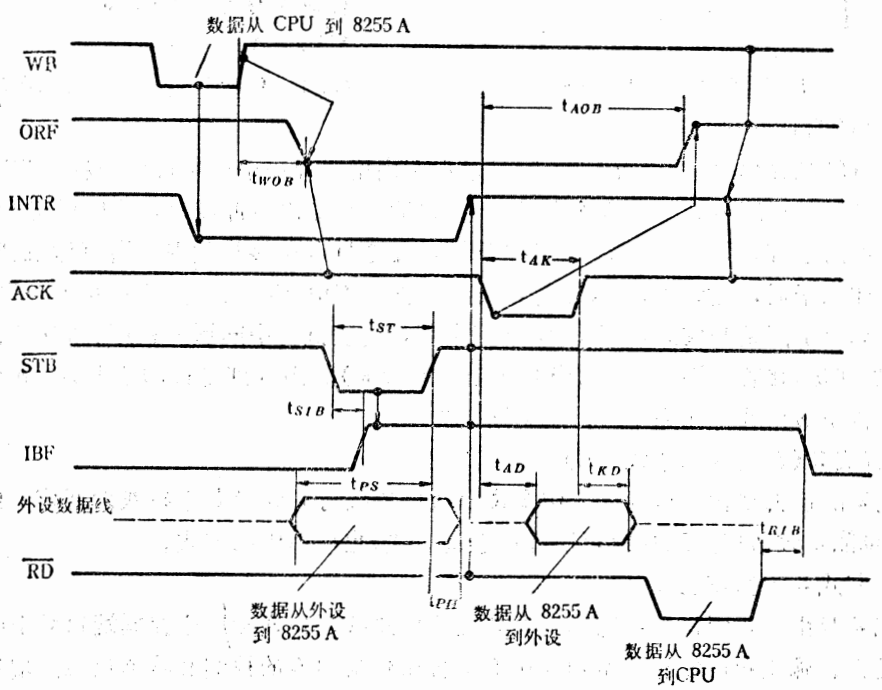


图 13-18 方式 2 的时序波形图

IBF——输入缓冲器满，高电平有效。它是一个状态信息，当其有效时表示数据已进入输入锁存器。

INTE-2——与 IBF 有联系的 INTE 触发器，它由 PC_4 的置位/复位控制。

(3) 方式 2 的时序

方式 2 的时序如图 13-18 所示。

它实质上是方式 1 的输入与输出方式的组合，故各个时间参数的意义也相同，不再重复。输出是由 CPU 执行输出指令，给出 $\overline{I/OW}$ 信号开始的，输入是由选通信号开始的。

上图中的输入、输出顺序是任意的，只要保证在 $\overline{ACK}$ 前出现 $\overline{WR}$ 和在 $\overline{RD}$ 前出现 $\overline{STB}$ 就能正常操作。

在输入和输出的情况下，都可以用中断方式。故

$$INTR = BF \cdot \overline{MASK} \cdot \overline{STB} \cdot \overline{RD} + \overline{OBF} \cdot \overline{MASK} \cdot \overline{ACK} \cdot \overline{WR}$$

其中 $\overline{MASK} = INTE$

4. 8255 A 中断控制功能

在 8255 A 按方式 1 或方式 2 进行程序设定时，能提供一个控制信号，用来作为向 CPU 的中断请求输入信号。如果要允许方式 2 中断，则可用端口 C 的置位/复位功能，对相应的中断触发器 INTE 置"1"(表示允许中断)或置"0"(表示禁止中断)来实现。

INTE 触发器定义如下：

INTE = 1　允许中断；

INTE = 0　禁止中断。

8255A 规定在方式 1 输入操作时，INTE A 由 PC_4 的置位/复位来控制；输出操作时，INTE A 由 PC_6 的置位/复位来控制。在输入和输出时 INTE B 均由 PC_2 的置位/复位来控制。

当 8255 A 工作在方式 2 时，规定当输出操作时，INTE 1 (与 $\overline{OBF}$ 有关的中断触发器) 由 PC_6 的置位/复位来控制；当输入操作时，INTE 2 (与 IBF 有关的中断触发器) 由 PC_4 的置位/复位 来控制。

【例】 下面一段程序可控制 8255 A 在方式 2 输出操作时允许中断和禁止中断：

```
MVI   A,  00001101B ; 将 PC6 位置"1"的控制字(允许输出中断)
OUT  CNTLR          ; 送 8255 A 控制寄存器
 ⋮
MVI   A,  00001100B ; 将 PC6 位置"0"的控制字(禁止输出中断)
OUT  CNTLR          ; 送 8255 A 控制寄存器
```

5. 读出端口 C 的状态

在方式 0 中，端口 C 可把数据送到外围设备，或者接受外围设备的数据。

当 8255 A 由程序设定在方式 1 或方式 2 时，端口 C 就根据不同的情况，产生或接受"联络"信号。读取端口 C 的内容可使程序员能够测试或检验每一外围设备的状态字，并相应地改变程序流程。可以由执行端口 C 的正常读操作来实现这一功能。

(1) 方式 1 输入状态字格式

其格式如图 13-19 所示。

(2) 方式 1 输出状态字格式

其格式如图 13-20 所示。

(3) 方式 2 状态字格式

其格式如图 13-21 所示。

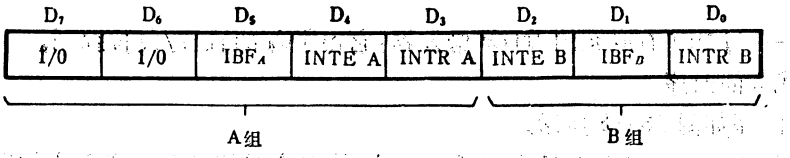


图 13-19 方式 1 输入状态字格式

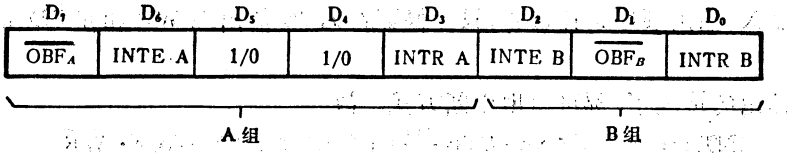


图 13-20 方式 1 输出状态字格式

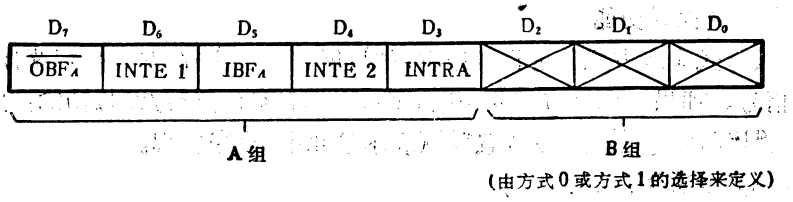


图 13-21 方式 2 状态字格式

四、8255 A 工作方式小结

8255 A 在各种工作方式中，各端口的功能小结如表 13-4 所示。概括地说，8255 A 的基本操作过程是：当 8255 A 被复位时，A、B、C 端口都被置于输入方式，这时 24 条引线都呈现高阻状态。在进行操作前必须进行工作方式的设定，这时需要把控制字方式标志位(D_7)置"1"，其它位可根据需要进行设定。然后，用 OUT 指令把控制字送到 8255 A 的控制寄存器。一旦工作方式设定后，只要不再进行重新设定或复位，它就一直保持其所设定的工作方式。这样，就可以根据设定的工作方式进行数据传送了。

表 13-4 8255 A 在各种方式中各端口的功能表

端口	方式 0		方式 1		方式 2	
	输入	输出	输入	输出	仅 A 组	
$PA_7 \sim PA_0$	IN	OUT	IN	OUT	I/O	
$PB_7 \sim PB_0$	IN	OUT	IN	OUT	无	仅有方式 0 或方式 1
PC_0	IN	OUT	$INTR_B$	$INTR_B$	I/O	
PC_1	IN	OUT	IBF_B	$\overline{OBF}_B$	I/O	
PC_2	IN	OUT	$\overline{STB}_B$	$\overline{ACK}_B$	I/O	
PC_3	IN	OUT	$INTR_A$	$INTR_A$	$INTR_A$	
PC_4	IN	OUT	$\overline{STB}_A$	I/O	$\overline{STB}_A$	
PC_5	IN	OUT	IBF_A	I/O	IBF_A	
PC_6	IN	OUT	I/O	$\overline{ACK}_A$	$\overline{ACK}_A$	
PC_7	IN	OUT	I/O	$\overline{OBF}_A$	$\overline{OBF}_A$	

当控制字的 $D_7 = 0$ 时，说明设置的是置位/复位标志，此时 D_0 的置位/复位功能，是表示对端口 C 的置位或复位。$D_0 = 1$ 表示置位；$D_0 = 0$ 表示复位。$D_3 \sim D_1$ 为位选择。

8255 A 各种工作方式的详细说明，请参阅 MCS-80/85 用户手册的有关部分。

§13-2 可编程序的计数器/定时器电路

(CTC——Counter/Timer Circuit)

微型计算机应用于控制系统时，常常要求有一些外部实时时钟，以实现定时或延时控制，也常常需要有外部计数器。Z 80 CTC 和 Intel 8253 都是能实现这样要求的电路。本节着重讨论 Z 80 CTC 的原理与应用，也简略地介绍 Intel 8253 芯片。

一、Z 80 CTC 的组成和功能

Z 80 CTC 计数器/定时器是一种具有 4 个独立通道(即具有 4 个独立的计数器/定时器)的可编程序的器件。每个通道都可为采用 Z 80 作为 CPU 的微型计算机系统提供计数和定时的功能。它采用 N-MOS 工艺，28 条引线的双列直插式封装。它仅需单一的 +5 V 电源和一个单相时钟。它的输入和输出都和 TTL 兼容。

CPU 可使 CTC 通道与多种外设接口并按它们所需要的方式和状态工作。

1．CTC 的功能说明

(1) CTC 共有 4 个独立的计数器/定时器通道(通道 0、1、2、3)，每一个通道都可由程序来设定工作在定时方式还是计数方式。

(2) 当 CTC 的某一通道作为定时器使用时，它能利用 Z 80 的时钟进行计时，并按程序的设置，定时地发出脉冲，同时还能发出一个中断请求信号。

(3) 当 CTC 的某一通道作为计数器使用时，它能对外界事件进行计数。当达到程序规定的计数次数时，就向 CPU 发出一个中断请求。

(4) 三个通道(通道 0、1、2)有“计数回 0”/“时间到”(ZC/TO)输出信号，能够驱动达林顿晶体管，可作为控制信号。

(5) CTC 具有链形优先权中断逻辑，不需要外加的硬件线路就可以形成中断排队结构。

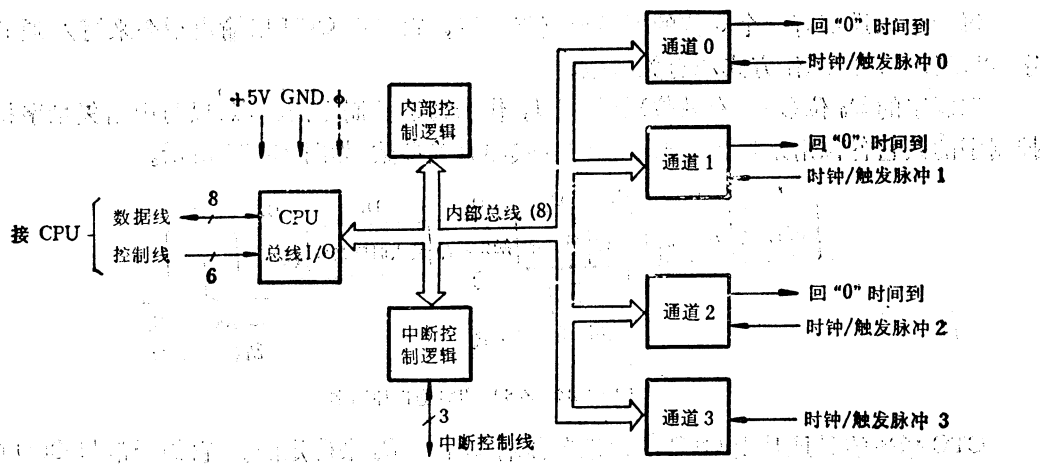


图 13-22 Z 80 CTC 方框图

4个通道的中断优先次序是：通道0最高，通道3最低。

2．CTC的结构

(1) CTC内部结构框图

CTC内部结构框图如图13-22所示。

CTC由CPU总线接口逻辑、内部控制逻辑、四个计数器/定时器通道逻辑和中断控制逻辑等4个部分组成。CTC的每个通道都能产生中断请求，并能给出一个唯一的中断矢量(按中断方式2)，自动地指向一个中断服务程序入口地址表。这四个通道可按通道号顺序连接在标准Z 80优先级链中，其中0号通道具有最高优先权。CPU总线接口逻辑允许CTC在不需要其它外接逻辑线路情况下直接与CPU接口。但是，对于较大的系统还得需要增加端口地址译码器和缓冲器。

(2) 通道的逻辑结构

四个计数器/定时器通道中任何一个的逻辑结构如图13-23所示。

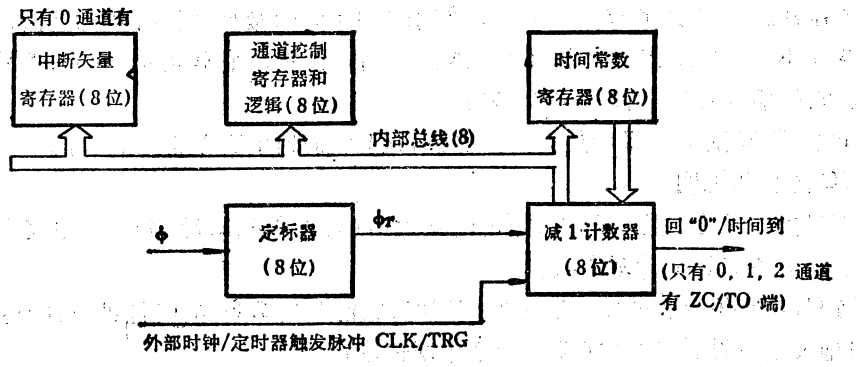


图13-23 Z 80 CTC通道逻辑结构图

它由一个8位的通道控制寄存器及控制逻辑、一个8位的时间常数寄存器、一个8位的减1计数器(CPU可读的)和一个8位的定标器(仅用于定时器方式)所组成。此外，0通道还包含一个中断矢量寄存器。现分述如下：

① 通道控制寄存器(8位)和控制逻辑

每一个通道都有一个8位的通道控制寄存器，它可由CPU用输出指令来写入通道控制字，以选择通道的工作方式和有关参数。

控制字的D_0位(最低有效位)始终是1，作为通道控制字的标志，以与中断矢量字相区别。控制字的其它各位用来选择通道的操作方式和有关参数，如图13-24所示。

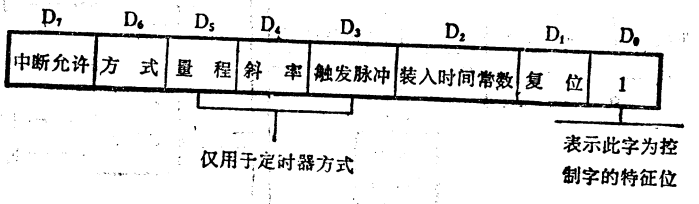


图13-24 Z 80 CTC通道控制字

CTC通道的寻址是由CTC的通道选择端CS_1、CS_0来确定的。它们通常与CPU的地址总线的A_1和A_0相连，用于选择四个不同的通道。其真值表如表13-5所示。

表13-5 CTC 通道选择表

	CS_1	CS_0
Ch0	0	0
Ch1	0	1
Ch2	1	0
Ch3	1	1

② 定标器(8 位)

定标器仅用于定时器工作方式，它可由通道控制器的第 5 位来设定对系统时钟进行 16 ($D_5 = 0$)或 256($D_5 = 1$)分频，然后作为减 1 计数器的输入脉冲。实际上，这相当于再一次对系统时钟分频，以扩大定时的范围。

③ 时间常数寄存器(8 位)

这个寄存器在计数器和定时器方式工作时都能起作用。在程序初始化时，CPU 输出通道控制字后接着必须将一个 1～256 的整数(即时间常数)装入该寄存器，否则 CTC 不会开始工作。当 CTC 初始化时，该寄存器将把由程序装入的计数初始值送入减 1 计数器中。每当减 1 计数器减到"0"后，同一计数值将自动地再次送入减 1 计数器中。当一个通道正在计数或定时操作时，如果有一个新的时间常数输入到该寄存器中，则要等减 1 计数器完成现行计数操作而回"0"后，才能将新的时间常数送入减 1 计数器，作为新的初始值。

④ 减 1 计数器(8 位)

它是一个 8 位的减 1 计数器，即每输入一个脉冲，计数器的值减 1，不论在计数器或定时器工作时都要用到它。该计数器由时间常数寄存器预置初始值，以后每当回"0"时，由时间常数寄存器重新装入。当它以计数器方式工作时，每输入一个外部时钟脉冲，使该计数器减 1；当它以定时器方式工作时，则由系统时钟脉冲经定标器分频后的输出脉冲使该计数器减 1。

CPU 可以将这个减 1 计数器作为一个输入端口，在任何时候都可以用一个简单的输入指令来读取这个计数器的值。CTC 中的每一个通道可由通道控制字的第 7 位 ($D_7 = 1$) 允许该通道中断，则每当减 1 计数器回"0"时，产生一个中断请求信号。

在通道 0、1、2 中，每当计数器回"0"时，在相应的"回 0"/"时间到"(ZC/TO)输出端输出一个脉冲信号，可用作外部控制。由于受封装引线的限制，通道 3 没有这条引线，因而通道 3 仅可用于不需要这一输出脉冲的场合。

⑤ 中断矢量寄存器(8 位)

CTC 四个通道共用一个中断矢量寄存器（在通道 0 中)，所以，在初始化程序中，只需要送一个中断矢量到通道 0。当 CPU 响应中断(按方式 2)时，CTC 的中断控制逻辑能自动提供相应于各通道的不同的中断矢量。

3. 引线说明

Z 80 CTC 引线布置图如图 13-25 所示。

每条引线的功能说明如下：

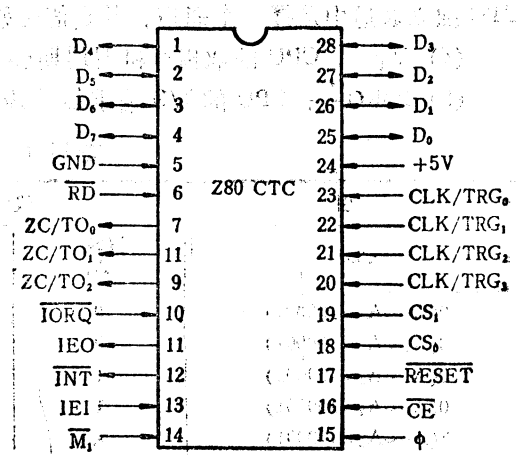


图 13-25(a) Z 80 CTC 引线图

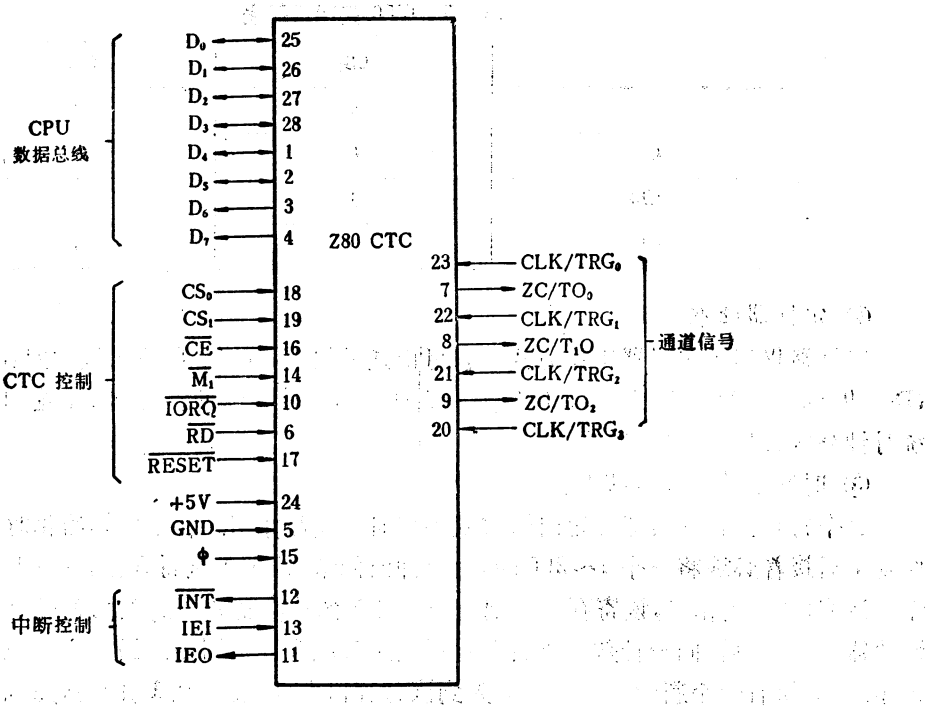


图 13-25(b)　Z 80 CTC 逻辑符号图

(1) $D_7 \sim D_0$——数据总线(双向、三态)。

此数据总线用于在 Z 80 CPU 和 Z 80 CTC 间传送数据和控制字。总线包含 8 位,其中 D_0 是最低位。

(2) $\overline{CE}$ (Chip Enable)——片选信号,输入、低电平有效。

只有当这条引线为低电平时,才允许 CTC 在 I/O 写周期内接受来自 Z 80 数据总线的控制字、中断矢量或时间常数数据字;在 I/O 读周期内读取减 1 计数器的内容到 CPU。通常,这个信号是根据地址总线低八位中的 $A_7 \sim A_2$ 及 $\overline{IORQ}$ 译码得出的。

(3) CS_1、CS_0——通道选择,输入。

这两条引线通常接至地址总线的 A_1、A_0 位,从而形成一个两位二进制地址码,以便在四个 CTC 独立通道中选择一个通道。其真值表如表 13-6 所示。

(4) $\overline{M_1}$——CPU 的取指令机器周期(M_1)信号,输入,低电平有效。

(5) $\overline{IORQ}$——CPU 的 I/O 请求信号,输入,低电平有效。

表 13-6　CTC 端口选择表

引线			选中单元	单元地址
$\overline{CE}$*	CS_1 A_1	CS_0 A_0		
0($A_7 \sim A_2$=100001)	0	0	通道 0	84H
0($A_7 \sim A_2$=100001)	0	1	通道 1	85H
0($A_7 \sim A_2$=100001)	1	0	通道 2	86H
0($A_7 \sim A_2$=100001)	1	1	通道 3	87H
1($A_7 \sim A_2 \neq$100001)	×	×	芯片未被选中	

* Z 80 STARTER KIT 是按此地址连接的

$\overline{M_1}$ 与 $\overline{IORQ}$ 同时有效时，作为 CPU 的中断响应信号。

(6) $\overline{RD}$——CPU 的读命令，输入，低电平有效。

$\overline{RD}$ 与 $\overline{IORQ}$ 相结合以控制 CPU 和 CTC 之间传送数据和控制字。

CTC 没有专门的写命令引线，而是通过一个 $\overline{RD}$ 信号的“非”来表示写信号。

上述三种控制信号的作用如表 13-7 所示。

表 13-7 CTC 控制信号的功用

引线			功能说明
$\overline{M_1}$	$\overline{IORQ}$	$\overline{RD}$	
0	0	1	中断响应
0	1	0	检查中断服务是否结束
1	0	0	CPU 从 CTC 的减 1 计数器读取内容
1	0	1	将控制字、时间常数、中断矢量等从 CPU 写入 CTC
其它值			没有作用

(7) IEI——中断允许输入信号，输入，高电平有效。

当有多个中断源时，本信号被用来构成优先权中断链。若 IEI 为高电平，则表示 CPU 目前没有为比本芯片的优先权更高的其它器件服务。此时允许芯片向 CPU 请求中断。

(8) IEO——中断允许输出信号，输出，高电平有效。

只有当 IEI 为高电平，且 CPU 没有为本芯片的中断服务时，本信号才为高电平。若 CPU 为本芯片或中断优先权更高的芯片执行中断服务时，本信号均为低电平，从而封锁优先权较低的器件发出中断请求。

(9) $\overline{INT}$——中断请求信号，输出，漏极开路，低电平有效。

当任一已编程序允许中断的 CTC 通道的减 1 计数器回“0”时，这个信号变为有效。

(10) $\overline{RESET}$——复位信号，输入，低电平有效。

当 $\overline{RESET}$ 为低电平时，CTC 复位，这一信号中止所有通道的工作，禁止 CTC 产生中断请求，并使 ZC/TO 输出变为无效状态。IEO 重复 IEI 的状态，即 IEO 与 IEI 状态相同。同时，CTC 的数据总线输出驱动器变为高阻状态。

(11) CLK/TRG(3～0)（或称 C/T3～0)——4 个通道的“外部时钟/定时器触发”脉冲信号，输入。可由用户规定其为上升沿或下降沿有效。

在计数器方式时，加到这个输入端的计数脉冲控制减 1 计数器减 1；在定时器方式时，加到这个输入端的控制脉冲启动定时操作。

至于是外加脉冲的上升沿还是下降沿起作用，由用户写入的控制字预先设定。

(12) ZC/TO(2～0)(或称为 ZC 2～0)——3 个通道(通道 3 除外)的“回零/时间到”脉冲，输出，高电平有效。每当通道 0～2 的减 1 计数器回“0”时，发出一个有效信号。

(13) 其它：ϕ 为时钟脉冲；+5 伏为电源；GND 为地。

二、Z 80 CTC 的工作方式

在 Z 80 CTC 通道开始工作之前，CPU 必须将一个通道控制字和一个时间常数数据字写

入该通道的通道控制寄存器和时间常数寄存器中。若四个通道中的任一个通道已由程序将其通道控制字第 7 位置"1"(允许中断),则还必须将其中断矢量写入 CTC 中的中断矢量寄存器。

CTC 的每个通道都有两种工作方式——定时器和计数器工作方式。

1. 定时器方式

CTC 的任一通道可先由通道控制字($D_6 = 0$)设定为定时器工作方式。接着把计数值(相隔多少时间发一次中断)预置入时间常数寄存器中。定时器何时开始计时有两种方式:

(1) 将计数值置入时间常数寄存器的时刻作为开始计时信号,称为自动启动;

(2) 由外部送给 CLK/TRG 端一个触发脉冲作为开始计时信号,称为外触发启动。

究竟选用哪一种方式由通道控制字的 D_3 位来确定。

当定时器开始工作时,时间常数寄存器中的计时值即输至减 1 计数器作为初始值,接着由定标器对系统时钟 ϕ 进行分频,即每隔 16 个或 256 个 Z 80 时钟脉冲(由程序选择),使减 1 计数器的计数值减 1。当减 1 计数器回 "0" 时,便从 ZC/TO 端输出一个正脉冲。若程序规定 CTC 允许中断的话,此时便会产生中断请求,使 $\overline{INT}$ 变为有效。

由此可知,在定时器工作方式时,CTC 使系统时钟 ϕ 通过两个串行的计数器——定标器和减 1 计数器传送,由此产生系统时钟整数倍的定时间隔。在通道 0～2 的 ZC/TO 输出端可输出一个周期精确的均匀的脉冲序列,其周期为

$$T = t_C \times P \times TC$$

其中,t_C 是系统时钟周期;P 是定标器系数,可由程序设定为 16 或 256;TC 是由程序预置的时间常数。

2. 计数器方式

CTC 的任一通道也可由通道控制字($D_6 = 1$)设定为计数器工作方式,接着把计数值送入时间常数寄存器。然后,由外部输至 CLK/TRG 端的输入脉冲来控制计数。每当 CLK/TRG 端输入一个触发脉冲,在脉冲的有效边沿后,与下一个 ϕ(系统时钟)的上升沿同步时,使减 1 计数器减 1。但减 1 计数器减 1 的脉冲有效边沿是上升沿触发还是下降沿触发则由通道控制字设定。

当减 1 计数器回"0"时,CTC 可向 CPU 发出中断请求,同时通道 0～2 的 ZC/TO 端还会输出一个正脉冲。

三、Z 80 CTC 的编程

接通电源后,CTC 处于不确定状态。可用 $\overline{RESET}$ 信号使 CTC 复位。但在任一通道开始工作前,CPU 必须将通道控制字和时间常数写入该通道相应的寄存器。如果该通道控制字中是允许中断的,则还必须将一个中断矢量字写入该通道中断控制逻辑。当 CPU 向 CTC 写入上述三个字之后,CTC 就会按编程所要求的方式工作。CTC 的每一通道只占用一个端口地址,因而需要规定一些特征来区别上述三种字。

通常,CTC 的初始化程序设定的顺序如下:

(1) 首先要确定中断服务程序入口地址表存放的位置,即 16 位中断矢量地址指针,以便设定 CPU 的中断矢量寄存器 I 的值(即 16 位中断矢量地址的高 8 位);

(2) 设定中断矢量低 8 位,以便 CPU 响应中断时和 I 寄存器组合成 16 位中断矢量地址指针找到中断服务程序入口地址表;

(3) 设置通道控制字，以确定 CTC 的工作方式；

(4) 设置时间常数值，以便设置定时方式的定时间隔或计数方式的计数值。

现分述如下：

1．设定 CPU 中的中断矢量寄存器 I——中断矢量地址的高 8 位。

Z 80 CTC 规定中断是按方式 2 工作的。因而需要确定 16 位中断矢量地址指针，假定其为 2400 H，则应将高 8 位 24 H 装入 I 寄存器。这只需用两条指令即可实现。

即 LD A，24 H

LD I，A

2．设定中断矢量地址的低 8 位

CTC 的中断矢量的格式如图 13-26 所示。

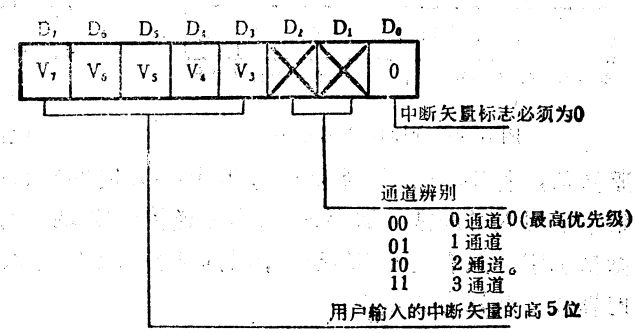


图 13-26 中断矢量字

当 Z 80 CTC 某通道在 CPU 程序控制下按中断方式 2 响应时，必须形成一个 16 位中断矢量地址指针，以便从存贮器中的中断服务程序入口地址表获得一个相应的中断服务程序入口地址。这个指针的高 8 位由 CPU 的 I 寄存器提供，而低 8 位必须由 CTC 来提供一个中断矢量字。

这个中断矢量的高 5 位 $V_7 \sim V_3$ 由用户设定的中断服务程序入口地址表的安排来确定，并在 CTC 程序设计时写入通道 0（中断矢量只需要也只能写入通道 0）；D_2、D_1 由 CTC 中断控制逻辑自动提供，如上述中断矢量格式所示。作为中断矢量字标志的 D_0 位必须为 0。

3．通道控制字

通道控制字的格式如图 13-24 所示。

应当指出，CTC 中的 4 个通道可以用不同的方式工作或完成不同的功能，因而必须对每个通道设置通道控制字而不能公用。

各位的含义如下：

(1) $D_7 = 1$ 表示允许通道中断，当计数器回“0”时，向 CPU 发出中断请求信号 $\overline{INT}$；

$D_7 = 0$ 表示禁止通道中断。

(2) $D_6 = 1$ 表示通道选择计数器工作方式；

$D_6 = 0$ 表示通道选择定时器工作方式。

(3) $D_5 = 1$ 表示定标器系数为 256，即每 256 个时钟脉冲(ϕ)使减 1 计数器减 1；

$D_5 = 0$ 表示定标器系数为 16，即每 16 个时钟脉冲(ϕ)使减 1 计数器减 1。

(4) $D_4 = 1$ 表示 CLK/TRG 的上升沿为有效，即在定时器方式时，由脉冲上升沿触发启动定时器工作；在计数器方式时，用脉冲上升沿使计数器减 1。

$D_4 = 0$ 表示 CLK/TRG 的下降沿为有效。

(5) D_3 仅用于定时器方式。$D_3 = 1$ 为外触发启动方式，即由 CLK/TRG 来启动定时器工作。当时间常数写入通道时，定时器并不立即启动，一直要等到该通道的 CLK/TRG 端从外部接受一个有效脉冲边沿后的第一个 ϕ 脉冲的上升沿，定时器才开始工作。

$D_3 = 0$ 为自启动方式，即由装入时间常数来启动定时器工作。当装入时间常数后，第一个系统时钟定时器即自动开始工作。

(6) $D_2=1$ 表示在通道控制字后写入的字是时间常数；

$D_2=0$ 表示下一个写入的字不是时间常数，此时，通道将不工作。

(7) $D_1=1$ 表示使通道复位。即通道停止计数或定时，ZC/TO 不起作用，禁止 CTC 产生中断请求。

$D_1=0$ 表示每当减 1 计数器到达“0”时，时间常数寄存器立即将其内容装入减 1 计数器，通道继续现行操作。

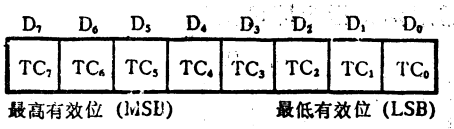


图 13-27 时间常数数据字

(8) $D_0=1$ 为通道控制字标志。

4．时间常数

时间常数数据字的格式如图 13-27 所示。

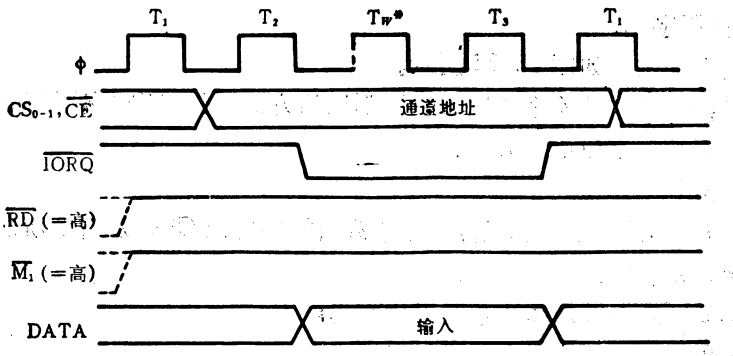
时间常数数据字可以是 1～256 中的任何一个整数值，若 TC_7～TC_0 8 位全为 0，则表示时间常数的最大值 256。时间常数是紧跟在通道控制字后写入时间常数寄存器的，当通道控制字 $D_2=1$，则表示通道控制字后要写入的是时间常数数据字。只有当 CPU 将一个时间常数数据字写入时间常数寄存器后，通道才能开始进行定时器或计数器方式工作。

四、Z 80 CTC 的时序

本节将说明几种操作——将一个字写入 CTC、从 CTC 读出一个字、计数和定时的时序关系。

1．CTC 写周期

CTC 写周期的时序如图 13-28 所示。



* 从 Z80 CPU 自动装入

图 13-28 CTC 写周期的时序波形图

CTC 写周期实际上是 CPU 输出指令的时序。只不过 CPU 的输出信号经过译码器、缓冲器等再送至 CTC，有相应的延时。在 T_1 时态，地址总线的低 8 位使 $\overline{CE}$ 和 CS_1、CS_0 有效，选中 CTC 芯片及规定的通道；此时，CTC 的 $\overline{RD}$ 为高电平，表示 CPU 处于写周期的准备状态。其后，在 T_2 时态，$\overline{IORQ}$ 变为有效，CTC 开始进入写周期。应当指出，此时 $\overline{M_1}$ 必须无效，以区别中断响应周期。在 T_2 时态后，CPU 自动插入 T_W^* 时态，以便外设在需要时提出 WAIT 请求。CPU 输出时，在 T_1 的 ϕ 的下降沿已将要输出的数据送到数据总线上，以使外设有足够的时间写入信息，但在 T_3 的 ϕ 的后沿必须将数据写入。

2．CTC 读周期

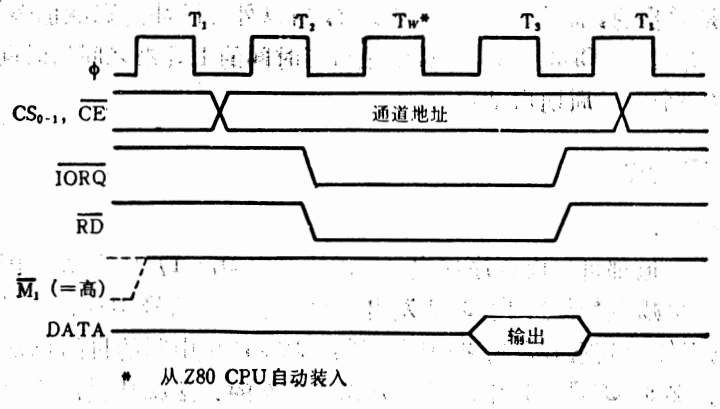


图 13-29 CTC 读周期的时序波形图

其时序如图 13-29 所示。

CTC 读周期实际上是 CPU 的输入指令的时序。在 T_1 时态，地址总线的低 8 位使 $\overline{CE}$ 和 CS_1，CS_0 有效；在 T_2 时态，CTC 的 $\overline{RD}$、$\overline{IORQ}$ 都变为有效。Z 80 CPU 开始进入读周期。同样，此时 $\overline{M_1}$ 必须为无效，以区别于中断响应周期。在 T_2 时态后，CPU 自动插入一个 T_W^* 时态。在 T_3 的 ϕ 的后沿，CPU 采样数据线，以获得由 CTC 来的数据。

3．CTC 计数和定时

图 13-30 示出 CTC 计数方式和定时方式的时序图。

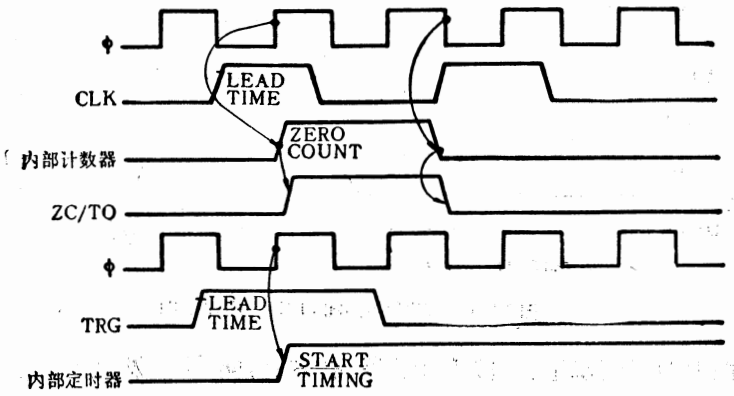


图 13-30 CTC 计数和定时方式的时序波形图

在计数器工作时，减 1 计数器是由外部硬件送至 CLK/TRG 端的脉冲信号的边沿来控制减数的。由于外部脉冲的出现与系统是异步的，所以计数器是在外部脉冲的有效边沿（脉冲上升沿）输入后的下一个时钟的上升沿减数，以达到同步工作。因此，这个 CLK/TRG 脉冲宽度必须大于一个最小值，且其周期必须不小于系统时钟周期的两倍。（参见 CTC 交流特性的规定。）此外，虽然在 CLK/TRG 的有效边沿和系统时钟的上升沿之间，没有严格的时间要求，但如果这个时间比规定的最小值小，则计数器将延迟一个 ϕ 时态减数。

当计数器减至“0”时，ZC/TO 端立即输出一个正脉冲。

在定时器方式工作时，如果由外部脉冲启动时，则加到 CLK/TRG 输入端的触发脉冲（可由用户选定是上升沿或是下降沿起作用）将在下一个时钟脉冲的上升沿时启动定时器。同样，

外部脉冲的出现是异步的，而启动定时是同步的，所以外部脉冲的宽度也必须大于规定值(见CTC交流特性)；并且，在CLK/TRG的有效边沿与时间的上升沿之间的时间也必须大于规定时间，否则将延迟一个时钟周期启动定时。

五、Z 80 CTC 的中断

1．CTC 中断服务

CTC 的每一个通道都可以由程序规定（通道控制字 $D_7=1$）是否允许中断。若允许中断，则在减 1 计数器每次减至“0”时，向 CPU 发出一次中断请求信号 $\overline{INT}$。

正如 CPU 响应任一其它外设请求中断一样，CTC 产生中断的目的也是迫使 CPU 执行一个中断服务程序。Z 80 CTC 规定中断是按方式 2 工作的。这就是说，当一个 CTC 通道请求中断并被响应时，必须形成一个 16 位中断矢量地址指针，以便从存贮器的中断服务程序入口地址表中获得一个指定的中断服务程序的入口地址。为此，必须由申请中断的外设提供一个 8 位的中断矢量(作为低 8 位)，以便与 CPU 中的 I 寄存器的内容(作为高 8 位)相结合，形成 16 位中断矢量地址指针。因此，CTC 在初始化时必须用程序向 CTC 写入一个中断矢量。

2．中断响应周期

图 13-31 示出中断响应周期的时序图。

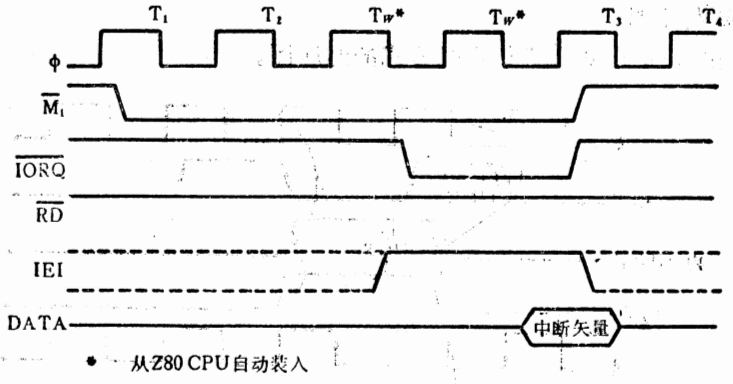


图 13-31　CTC 中断响应周期时序波形图

在 CTC 中断开放的情况下，当减 1 计数器回“0”时，向 CPU 发出中断请求。如果此时 CPU 的中断也是开放的，则 CPU 在现行指令周期的最后一个 T 时态的上升沿采样 $\overline{INT}$ 信号，从而在下一机器周期转入中断响应周期。其过程如下：在 T_1 的上升沿后 $\overline{M_1}$ 有效，因为是中断响应，CPU 自动插入两个 T_W^* 时态，使链形优先权电路有时间(必要时再插入 T_W 周期) 确定申请中断的最高优先权中断源；在第一个 T_W^* 时态的下降沿 $\overline{IORQ}$ 有效，它与 $\overline{M_1}$ 信号一起形成 CPU 的中断响应信号，使选中的 CTC 通道将中断矢量送入系统数据总线，CPU 则在 T_3 时态的上升沿采样数据总线，以读入中断矢量。

中断服务程序的末尾安排一条 RETI 指令，以便为正确控制嵌套优先级中断处理的链路“允许”线恢复初始状态。CTC 内部会译出两字节的 RETI 码，并且确定该通道的中断服务是否结束。

3．链形中断优先权电路

CTC 与其它为 Z 80 配套的 I/O 接口电路一样，有 IEI 和 IEO 两条信号线，用来形成链形

中断优先权电路。其中，最靠近 CPU 的外设具有最高优先权。在 CTC 中，当几个通道同时请求中断时，其内部中断控制逻辑能自动确定优先权级别，其优先级由通道编号预先编定：0 号通道具有最高优先权，依次降低优先权级别，3 号通道的优先权级别最低。

CTC 的中断控制逻辑能保证按照 Z 80 CPU 的规定实现中断嵌套和从中断返回。

在 CTC 中可能出现的一个典型嵌套中断的时序如图 13-32 所示。

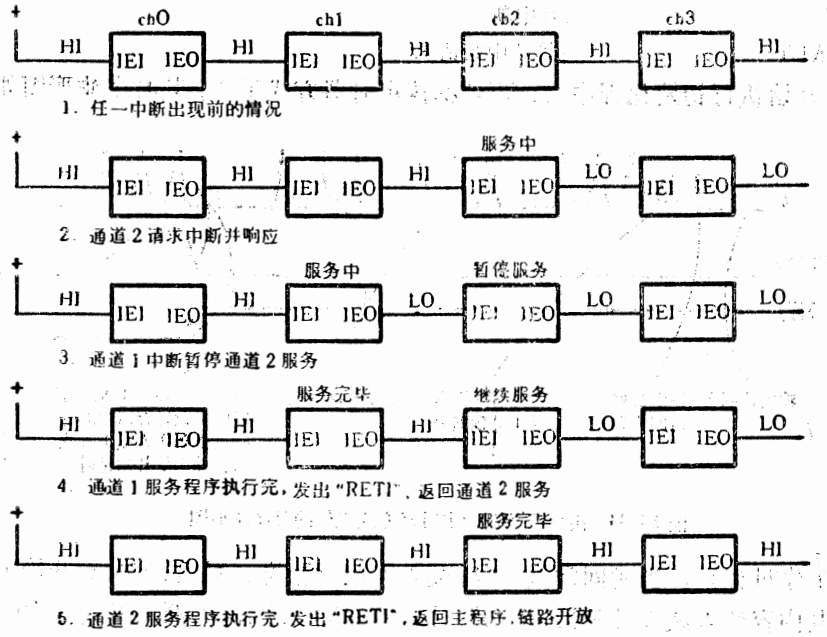


图 13-32　CTC 中断嵌套时序

图中，通道 0 的 IEI 端(即 CTC 的 IEI 输入端)接正电源。当没有通道申请中断时，通道 0 的 IEO = 1，使通道 1、2、3 的 IEO = 1，即 CTC 的输出 IEO = 1。

当通道 2 请求中断时，它的 IEO = 0，屏蔽了通道 3。但通道 1 的 IEI 仍为"1"。因此，如果在通道 2 的中断服务过程中，通道 1 请求中断，则可向 CPU 发出中断请求信号。若此时中断是开放的，则 CPU 就暂停对通道 2 的中断服务，而去响应通道 1 的中断请求。当通道 1 的中断服务程序执行完毕，就返回通道 2 的中断服务程序。当通道 2 中断服务程序执行完毕，就返回主程序，使链形电路恢复初始状态。

六、Z 80 CTC 程序设计举例

1．定时器工作方式

【例 1】 设定通道 0 为定时器方式，自动启动定时器工作(即装入时间常数启动定时)，定标系数选定 16，时间常数设定为 03 H，开放 CTC 中断，并设中断服务程序入口地址表指针为 2450 H。

初始化程序如下：

```
START: LD  A,  24 H ;设定中断矢量高 8 位
       LD  I,  A
       LD  A,  50 H ;设定中断矢量低 8 位
```

```
OUT   (CTC0),   A
LD    A,        85 H  ;设定通道控制字
OUT   (CTC0),   A
LD    A,        03 H  ;装入时间常数,定时器开始工作
OUT   (CTC0),   A
IM    2                ;设定中断方式 2
EI                     ;开中断
HALT                   ;等待中断请求
```

从 START 开始执行初始化程序后，CTC 就按定时器方式工作，其时序波形图如图 13-33 所示。

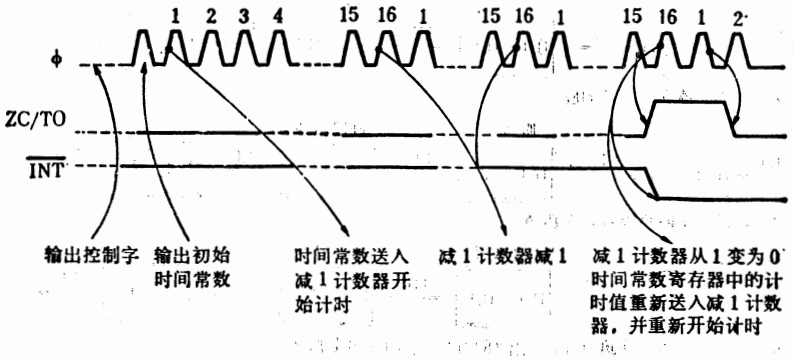


图 13-33　由程序控制的定时器方式工作的时序波形图

定时器的工作过程是：当时间常数 03 H 装入时间常数寄存器后，定时器开始工作。时间常数寄存器将其内容装入减 1 计数器，定标器对时钟 ϕ 进行计数，本例中每输入 16 个 ϕ（因为控制字 $D_5=0$），定标器输出一个 ϕ_T（见图 13-33），使减 1 计数器减 1，定标器输出第三个 ϕ_T 时，减 1 计数器由 1 减为 0，ZC/TO 端发出一正脉冲，$\overline{INT}$ 端变为低电平，向 CPU 请求中断。本例中因为控制字 $D_1=0$，所以时间常数寄存器将其内容再一次装入减 1 计数器，并重复上述操作。可见 ZC/TO 上每隔 48 个 ϕ 出现一个正脉冲，直到写入一个停止其操作的控制字（$D_1=1$）为止。

本例中 ZC/TO 上发出正脉冲的时间间隔可以由程序设定。发脉冲的起始和终止也由程序设定。

【例 2】 用外界触发脉冲加入 CLK/TRG 端来启动定时器的工作（以通道 1 为例）。设外部触发脉冲是上升沿有效，定标系数选为 256，中断服务程序的入口地址表指针为 2452H。

初始化程序如下：

```
START: LD    A,        24 H   ;设定中断矢量高 8 位
       LD    I,        A
       LD    A,        50 H   ;外设提供中断矢量低 8 位
       OUT   (CTC0),   A
       LD    A,        0BDH   ;设定工作方式
       OUT   (CTC1),   A      ;输入通道控制字
       LD    A,        03 H   ;装入时间常数
       OUT   (CTC1),   A
       IM    2                ;设定中断方式 2
```

```
EI                      ;开中断
HALT                    ;等待中断请求
```

从 START 开始执行初始化程序后，CTC 就按定时器工作。其时序波形图如图 13-34 所示。

应当注意：中断矢量低 8 位一定要写入通道 0，所以仍为 50 H。

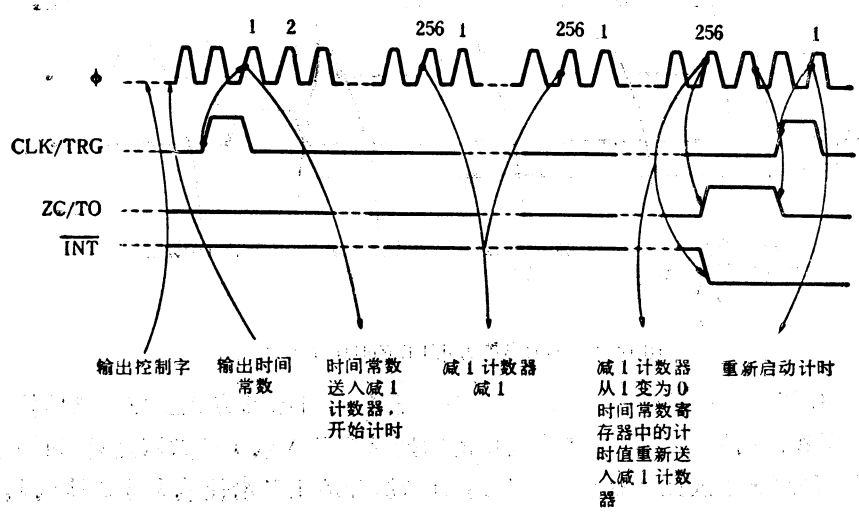


图 13-34　CLK/TRG 由外界控制的定时器方式工作的时序波形图

定时器的工作过程是：当时间常数 03 H 装入时间常数寄存器后，定时器等待 CLK/TRG 端信号的到来。一旦出现 CLK/TRG 信号(本例中控制字 $D_4=1$，故上升沿有效)，在 CLK/TRG 有效后第一个 ϕ 的上升沿，定时器开始工作，其后的过程与例 1 基本相同，所不同的是：①每当出现第 256 个 ϕ 后(不是 16 个，因为控制字中 $D_5=1$) 减 1 计数器减 1；② 当减 1 计数器减到零后，ZC/TO 端出现正脉冲，$\overline{\text{INT}}$ 端变低电平，定时器停止工作。只有当再一次出现 CLK/TRG 信号时，才能再次启动定时器的工作。

2．计数器工作方式

这种工作方式主要用于对外部(异步)脉冲信号进行计数。当到达规定数值后，ZC/TO 发出正脉冲，并使 $\overline{\text{INT}}$ 端变为低电平。

【例 3】 设定通道 2 为计数器方式工作，计数脉冲的有效边沿是上升沿，计满 3 个脉冲后发出一个中断请求，设中断服务程序入口地址表指针为 2454 H。

初始化程序如下：

```
START: LD    A,      24 H    ;设定中断矢量高 8 位
       LD    I,      A
       LD    A,      50 H    ;外设提供中断矢量低 8 位
       OUT   (CTC0), A
       LD    A,      0 D 5H  ;设定工作方式
       OUT   (CTC2), A       ;输入通道控制字
       LD    A,      03 H    ;装入时间常数
       OUT   (CTC2), A
       IM    2               ;设定中断方式 2
```

```
EI                  ；开中断
HALT                ；等待中断请求
```

从START开始执行初始化程序后，CTC通道2按计数器方式工作，其时序波形图如13-35所示。

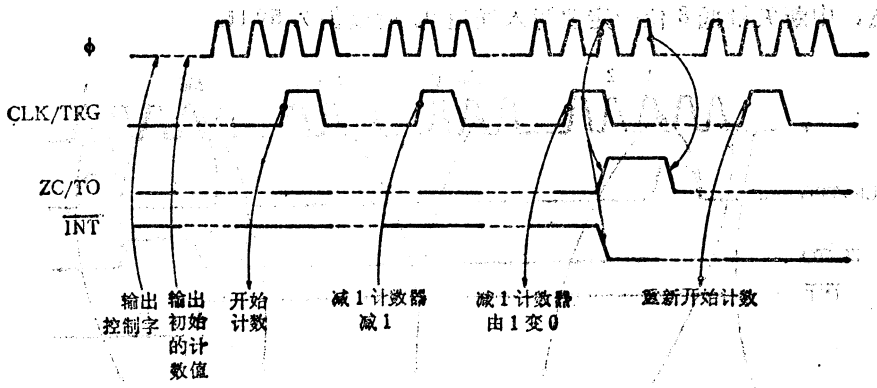


图13-35　按计数器方式工作的时序波形图

计数器的工作过程是：首先输入通道控制字，规定按计数器方式工作。然后输入时间常数，计数器开始工作。时间常数寄存器将其中的时间常数装入减1计数器之后，每当输入一个CLK/TRG脉冲信号(表示发生一次外界事件)，计数脉冲的上升沿使减1计数器减1。直到减1计数器到达"0"时，ZC/TO发出一个正脉冲，同时$\overline{INT}$端变为低电平，时间常数寄存器再次将初始常数装入减1计数器，重复上述计数操作。

3．CTC应用举例

若有一程序，要求每隔20 ms使累加器A中的内容左移一位，则可设定CTC某一通道为定时器方式，并开放该通道的中断。每当计时20 ms，CTC就发出中断请求，使CTC转去执行中断服务程序，使A左移一位。

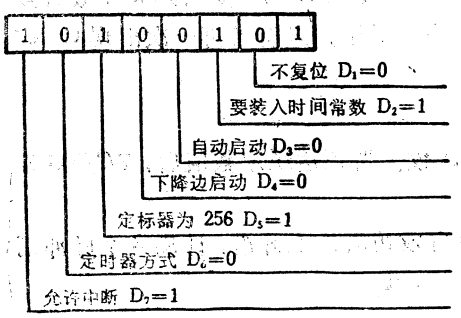


图13-36　例中的通道控制字

程序设定如下：

(1) CTC初始化

假设用CTC通道0，该通道端口地址为CTC 0。

① 确定通道控制字

设CTC工作在定时器自动启动方式，定标系数为256，并允许中断，反复工作，故其通道控制字如图13-36所示。即控制字为10100101B = 0A5 H。

② 确定时间常数

本例要求延时时间间隔为20 ms。如前所述，

$$T = t_C \times P \times TC$$

其中，t_C为系统时钟周期，$t_C = \frac{1}{f_C}$。若用TP-801单板机，其f_C = 1.9968 MHz；P为定标系数，现为256；TC为要写入的时间常数。

因为要求　T = 20 ms，

故

$$TC = \frac{Tf_C}{P} = \frac{20 \times 1000 \times 1.9968}{256} = 156 = 9CH$$

③ 确定中断矢量

D_0 位按规定为 0。因是通道 0，故 $D_2D_1=00$。若取 $V_7\sim V_3=00000$，则中断矢量为 00 H。

(2) 主程序

在主程序中要输出通道控制字，首先要确定 CTC 的端口地址。现设定通道 0 端口地址为 CTC0，其主程序如下：

```
        ORG   2000 H
        LD    SP,    27C0H  ;设堆栈指针
        LD    A,     21H    ;设置中断矢量高 8 位
        LD    I,     A
        LD    A,     00H    ;设置中断矢量低 8 位
        OUT   (CTC0), A
        LD    A,     0A5H   ;设置通道控制字
        OUT   (CTC0), A
        LD    A,     9CH    ;设置时间常数
        OUT   (CTC0), A
        LD    A,     01H
        IM    2              ;设定中断方式 2
        EI                   ;开中断
LOOP:   HALT
        JR    LOOP
        END
```

(3) 中断服务程序

首先要把中断服务程序的入口地址，存入由I寄存器和中断矢量地址所指向的表格中。

```
; 中断服务程序入口地址表
  2100H    00
  2101H    22
; 中断服务程序
  ORG      2200H
  RLCA
  EI
  RETI
```

在本例中断服务程序中，保护现场和恢复现场均省略。

七、Z 80 CTC 与 CPU 的连接

图 13-37 示出了 CTC 与 CPU 的连接线路图，此线路连接简单，此处不再赘述。

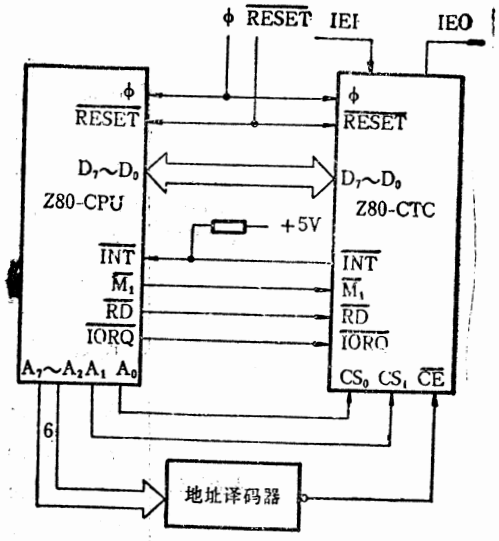


图 13-37 CTC 与 CPU 的连接线路图

§13-3 可编程序计数器/计时器电路 Intel 8253

Intel 8253 是可编程序的计数器/计时器电路芯片。它由三个独立的 16 位计数器组成，其中每个计数频率可达 2 MHz。全部操作方式可通过程序设定。

一、8253 的方框图和引线

8253 的方框图和引线布置图如图 13-38 和 13-39 所示。表 13-8 示出引线的名称。

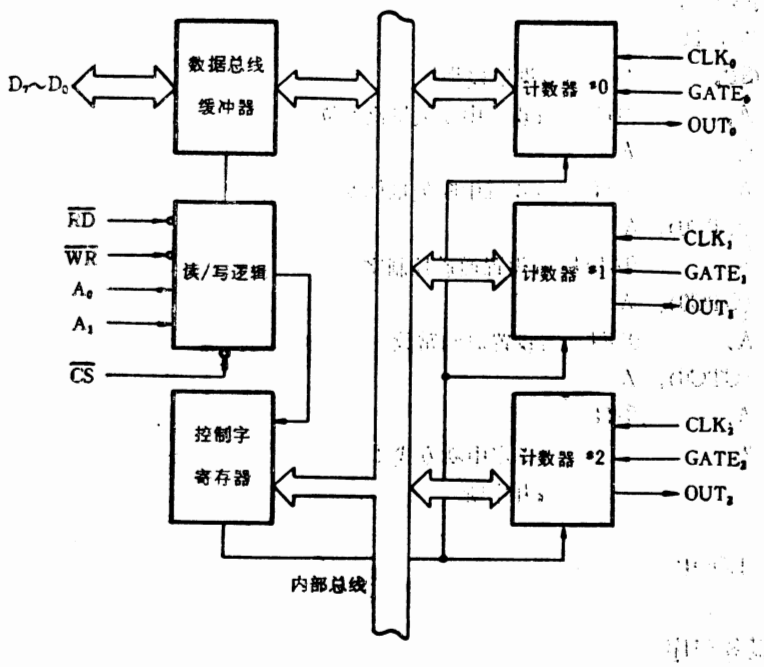


图 13-38 8253 的方框图

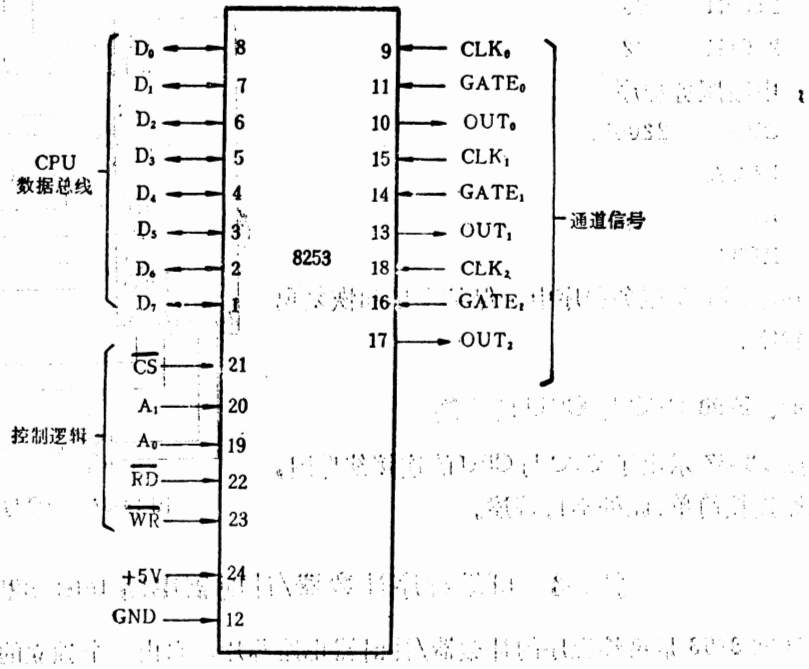


图 13-39 8253 的引线图

表 13-8 引线名表

$D_7 \sim D_0$	数据总线(8位)	$\overline{WR}$	写命令字或写数据
CLKN	计数器时钟输入	$\overline{CS}$	片选
GATEN	计数器选通输入	A_0, A_1	计数器选择
OUTN	计数器输出	V_{CC}	+5V
$\overline{RD}$	读计数器	GND	地

二、8253 的操作说明

8253 的工作方式可由系统软件来设定。操作前，CPU 必须向 8253 送出一组控制字，并设定计数寄存器的字节数(1～2 个)，对每一个计数器进行初始化。这些控制字规定了 8253 的工作方式、存数次序及二进制或二-十进制的计数方式。

8253 控制逻辑有 5 个控制信号 $\overline{RD}$、$\overline{WR}$、$\overline{CS}$、A_1 和 A_0，其中，A_1 和 A_0 接地址总线的低 2 位。对应于 5 个控制信号的组合和执行如表 13-9 所示的操作。

表 13 9 8253 控制信号的组合和执行表

$\overline{CS}$	$\overline{RD}$	$\overline{WR}$	A_1	A_0	执行操作
0	1	0	0	0	装初值到 0 号计数器
0	1	0	0	1	装初值到 1 号计数器
0	1	0	1	0	装初值到 2 号计数器
0	1	0	1	1	写控制字到控制寄存器
0	0	1	0	0	读 0 号计数器值
0	0	1	0	1	读 1 号计数器值
0	0	1	1	0	读 2 号计数器值
0	0	1	1	1	非法，三态
1	×	×	×	×	非选中，三态
0	1	1	×	×	非法，三态

8253 控制字格式定义如图 13-40 所示。

在这里我们仅介绍方式 0——计数完中断。其工作过程是：当程序送一控制字将所选的计数器置于所设定的方式后，该计数器的输出为“低”，当计数器初值装入被选的计数器后，在外部输入的门控电位(高电平)控制下，通过各自的计时脉冲进行递减计数。此时，其输出仍然为“低”。当计数器从初始值减到全“0”时，便产生一高电平输出，利用此输出信号可向 CPU 发出定时中断，通知微型计算机“时间已到”，此中断请求一直保持到程序再次向计数器送入初值为止。若在计数过程中再装初值，则会导致装第一字节停止现行计数，装第二字节后就从新的初值开始计数。

送方式控制字的顺序和计数器号的顺序无关。因为每个计数器的方式控制寄存器具有各自的地址(SC_0、SC_1)。但是向计数寄存器存入实际值时，必须严格地按方式控制字中程序所设定的执行顺序(RL0、RL1)进行存入操作。置初值不一定要紧跟在相应的置方式字之后，可以在置方式字后的任何时刻，只要依次存入正确的字节数即可。

所有计数器都是递减计数器，若把全“0”存入计数器中去，将会得到最大计数值(对二进制，计数为 2^{16}，对二-十进制为 10^4)。

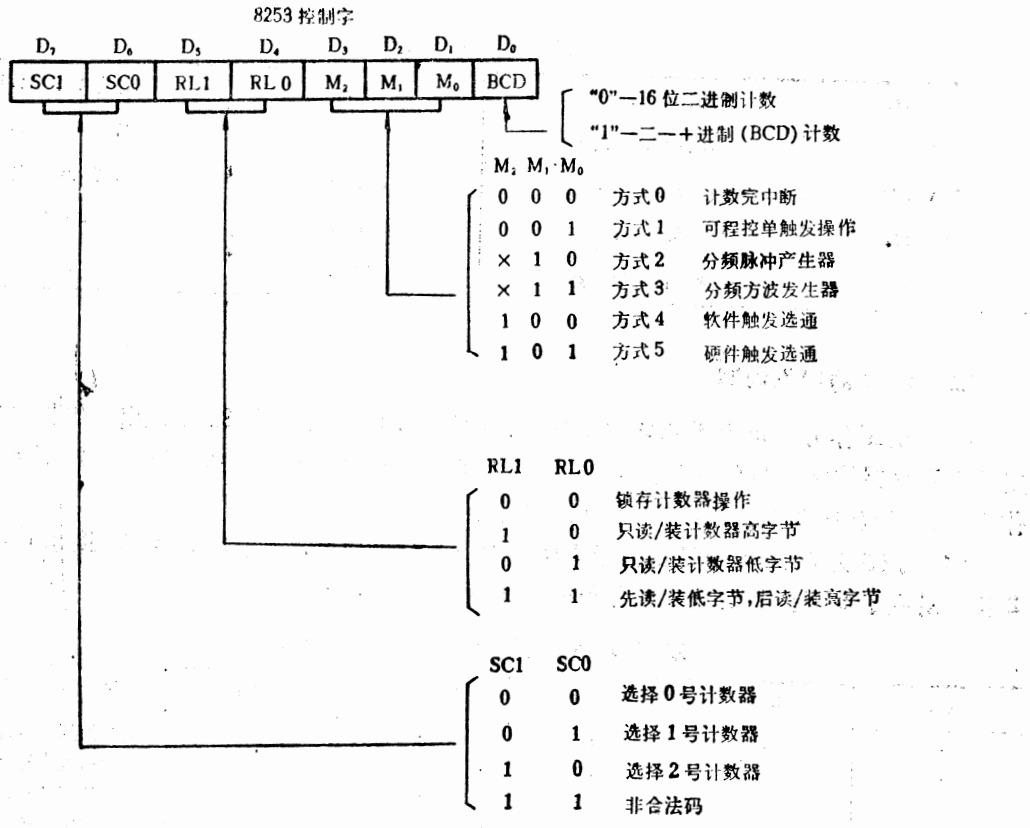


图13-40 8253的控制字

关于 Intel 8253 详细的操作说明请参阅 MCS-80/85 技术手册。

§13-4 可编程序并行 I/O 接口电路 Z 80 PIO(Parallel Input Output)

Z 80 PIO 是一种可编程序的 8 位并行的 I/O 接口芯片。它能在 CPU 和外设之间实行并行数据的传送。

Z 80 PIO 的工作方式可由系统软件来设定，使它能与大多数常用的 I/O 设备连接而不需要外加逻辑电路，使用十分方便，灵活。

一、Z 80 PIO 的组成和功能

1．主要功能

(1) PIO 芯片有两个独立的 8 位并行双向 I/O 端口，每个端口有两条"联络"(handshake)信号线，用以控制数据的传送。

(2) 每个端口可由程序设定为几种不同的工作方式：

① 带联络线的输出——方式 0；

② 带联络线的输入——方式 1；

③ 带联络线的双向——方式 2 (只有端口 A 可选用)；

④ 位控——方式 3。

上述工作方式都配有实现中断所必须的各种信息。当端口 A 选用双向方式时，必须借用端口 B 的“联络”信号线。因此，此时端口 B 只能选用位控方式。

(3) 有实现链形优先权中断的逻辑，有中断允许输入 IEI 线和中断允许输出 IEO 线，可以实现中断优先排队，并能自动提供中断矢量，而不需要外加逻辑电路。

(4) 当使用 Z 80 PIO 时，I/O 设备与 CPU 之间的数据传送是在中断控制(中断方式 2)下实现的。并且可通过改变程序，使得在外设中出现指定状态时，就向 CPU 发出中断请求，这就大大减少了 CPU 查询外设状态的时间。

2．PIO 的结构

Z 80 PIO 的方框图如图 13-41 所示。

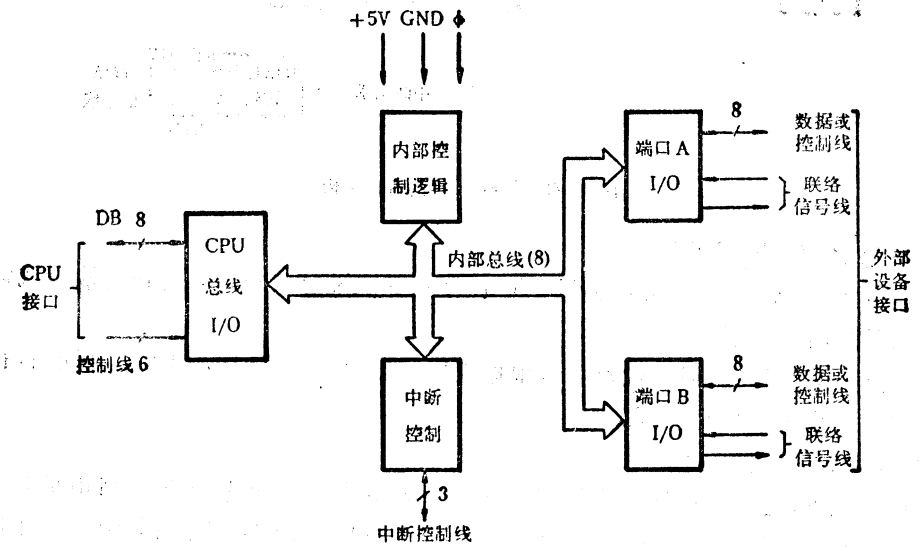


图 13-41　Z 80 PIO 方框图

Z 80 PIO 的内部结构是由 Z 80 CPU 总线接口、内部控制逻辑、两个独立的 I/O 端口 A 和 B 以及中断控制逻辑所组成。端口 A 和 B 由 PIO 内部控制逻辑根据 CPU 的程序来决定端口的工作方式。所有的数据和控制信号都由总线接口与 CPU 直接相连而不需要外加逻辑。但是，在组成较大的系统时，可能还需要地址译码器或缓冲器。

3．I/O 端口结构

PIO 的每个端口由 6 个寄存器和中断控制逻辑组成，如图 13-42 所示。

(1) 6 个寄存器如下：

① 方式控制寄存器(2 位)

它由 CPU 装入控制字以选择端口的工作方式(输出、输入、双向或位控方式)。

② 数据输入和数据输出寄存器(8 位)

这两个都是 8 位寄存器。它们用于 I/O 设备与 CPU 之间的数据传送，这种传送由 CPU 执行 I/O 指令来实现。它们合用一个端口地址，每个端口还配有 2 条联络信号线，用以控制 PIO 和外设之间的数据传送。

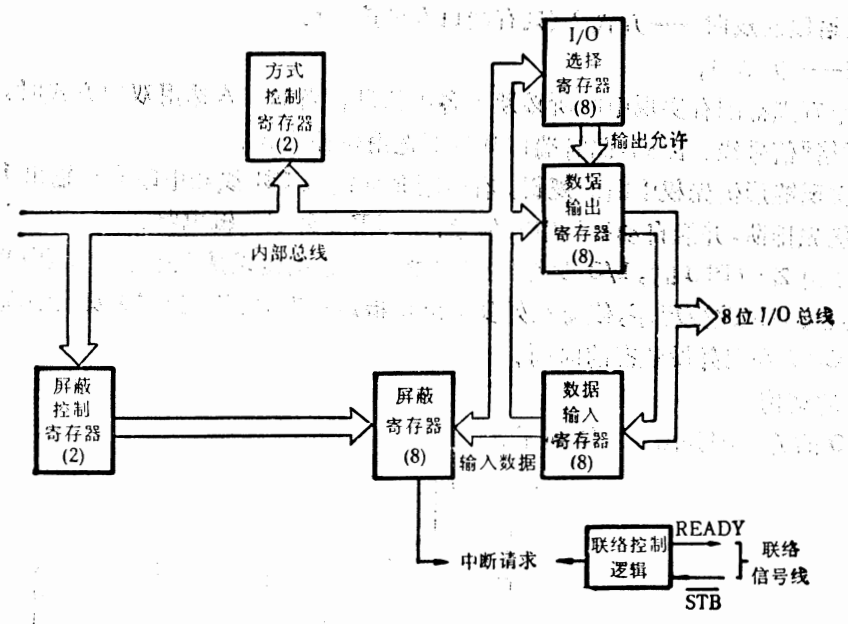


图 13-42　Z 80 PIO 的端口结构

③ I/O 选择寄存器(8 位)

在位控方式下，CPU 可用程序通过 I/O 选择寄存器设定端口的各位作为输出或输入。

④ 屏蔽寄存器(8 位)

在位控方式下，端口的每一位都可以根据程序的设定向 CPU 请求中断。屏蔽寄存器可用来对端口的每一位分别实行屏蔽。

⑤ 屏蔽控制寄存器(2 位)

在位控方式下，这个寄存器用来指定引起中断的条件，即可由程序规定高电平有效还是低电平有效。并且指定是当所有未屏蔽引线有效时(即相“与”的情况下)引起中断，还是在任一个未屏蔽引线有效时(即相“或”的情况下)引起中断。

上述第③、④、⑤寄存器仅用于位控方式。

6 个寄存器中，数据输入和输出寄存器合用一个端口地址(数据端口)；其余 4 个寄存器统称为控制寄存器，它们共用一个端口地址(控制端口)。在程序初始化时，由 CPU 来设定这 4 个寄存器的初值，以赋于 PIO 相应的功能。

(2) 中断控制逻辑

中断控制逻辑用来实现嵌套优先权中断。每一个 PIO 都有 IEI 和 IEO 两条信号线，由它们在系统链形结构中的位置来决定 PIO 芯片的优先权。在同一 PIO 内，规定端口 A 的优先权高于端口 B。在输入、输出或双向方式中，每当外设请求传送一个新字节时，就引起一次中断。在位控方式中，则当外设状态与程序规定相符时引起中断。

PIO 规定按中断方式 2 工作时，端口 A 或 B 各自有一个独立的中断矢量。当 CPU 响应中断时，申请中断的端口必须向 CPU 提供一个 8 位的中断矢量。PIO 可从 CPU 数据总线上直接译码 RETI 指令。这样，系统中每个 PIO 即使未与 CPU 有任何通信，也随时都知道 CPU 是否在执行中断服务程序。

4. 引线说明

Z 80 PIO 采用耗尽型负载 N-MOS 工艺，并装在 40 条引线的双列直插式封装中。PIO 的引线布置如图 13-43 所示。这些引线的作用如下：

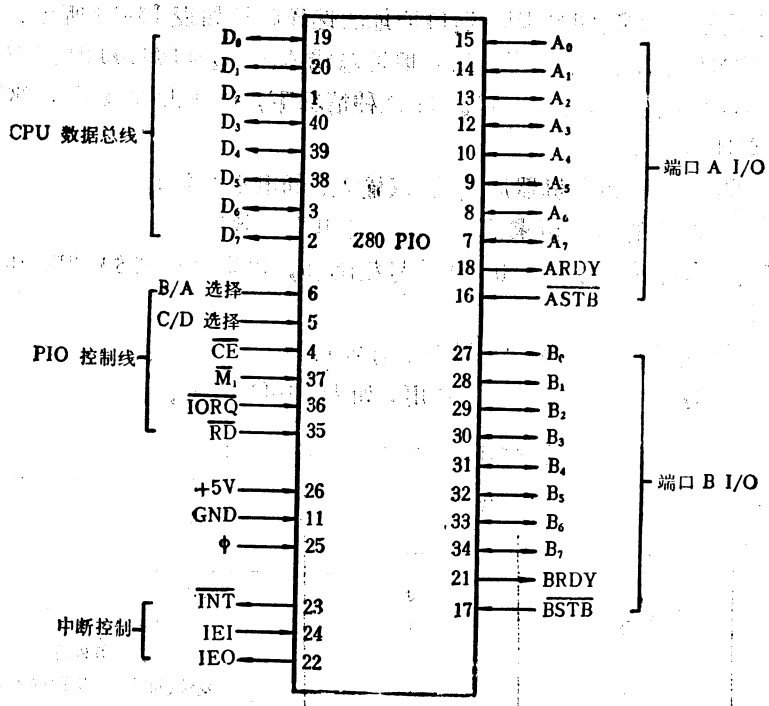


图 13-43 Z 80 PIO 逻辑符号图

(1) $D_7 \sim D_0$——数据总线(双向、三态)。

这个总线用于在 CPU 和 PIO 之间传送数据和命令。D_0 是总线的最低位。

(2) $\overline{CE}$——芯片允许(输入、低电平有效)。

只有当此信号为低电平时，才允许 CPU 和该 PIO 芯片进行数据通信。通常 $\overline{CE}$ 接至译码器输出端。

(3) B/A——选择端口 A 或端口 B (输入)。

当该信号为低电平时，选中端口 A；当为高电平时，选中端口 B。通常将此引线接至 CPU 地址总线的 A_0 位。

(4) C/D——选择数据寄存器或控制寄存器(输入)。

表 13 10 PIO 端口地址的选择

引线			选中单元	单元地址
CE	C/D(A_1)	B/A(A_0)		
0($A_7 \sim A_2$=100000)	0	0	端口 A 的数据寄存器	80H
0($A_7 \sim A_2$=100000)	1	0	端口 A 的控制寄存器	82H
0($A_7 \sim A_2$=100000)	0	1	端口 B 的数据寄存器	81H
0($A_7 \sim A_2$=100000)	1	1	端口 B 的控制寄存器	83H
1($A_7 \sim A_2 \neq$100000)	×	×	芯片未被选中	

注：Z 80 STARTER KIT 是按此地址连接的。不同的微型计算机系统或不同的 PIO 芯片，$A_7 \sim A_2$ 可为不同值。

当该信号为低电平时，访问数据寄存器，即数据总线被用来传送数据；当该信号为高电平时，访问控制寄存器，即CPU通过数据总线向PIO写入的信息是一个控制命令字。通常将此引线接至CPU地址总线的A_1位。

从以上三个信号的含义，可将PIO端口地址的选择归纳如表13-10所示。

表13-10中"×"表示可为任意值。如果地址总线$A_7 \sim A_2 = 100000$时，$\overline{CE}$被译码成逻辑"0"。如果$A_7 \sim A_2$为其它值时，$\overline{CE} = 1$。在这种情况下，PIO占有4个外部设备的端口地址80H、81H、82H、83H。

(5) $\overline{M_1}$——CPU的取指令机器周期信号(输入、低电平有效)。

(6) $\overline{IORQ}$——CPU的I/O请求信号(输入、低电平有效)。

$\overline{IORQ}$信号与B/A、C/D、$\overline{CE}$和$\overline{RD}$信号相配合，以实现在Z 80 CPU和Z 80 PIO之间传送命令和数据。

(7) $\overline{RD}$——CPU的读命令(输入、低电平有效)。

P O中$\overline{M_1}$、$\overline{IORQ}$和$\overline{RD}$控制信号的功用，如表13-11所示。

表13-11 PIO控制信号的功用

引线			功能说明
$\overline{M_1}$	$\overline{IORQ}$	$\overline{RD}$	
0	0	0	没有作用
0	0	1	中断响应
0	1	0	检查中断服务程序是否结束
0	1	1	复位(即PIO逻辑进入复位状态)
1	0	0	CPU从PIO读出
1	0	1	从CPU写入PIO
1	1	0	没有作用
1	1	1	没有作用

(8) IEI——中断允许输入信号(输入，高电平有效)。

(9) IEO——中断允许输出信号(输出，高电平有效)。

IEI和IEO是为了实现芯片间的中断链形优先权排队的信号，已在Z 80 CTC中介绍过。

(10) $\overline{INT}$——中断请求(输出，漏极开路，低电平有效)。当PIO正在向CPU发中断请求时，$\overline{INT}$变为有效。

(11) $PA_7 \sim PA_0$——端口A的总线(双向，三态)。

它们用来在端口A和外部设备之间传送数据或控制信息。

(12) $\overline{A\ STB}$——外设送至端口A的选通脉冲(输入，低电平有效)。

此信号的作用与端口A的工作方式有关。

① 输出方式：外设送来这一选通脉冲的前沿(下降沿)，作为收到PIO输出数据的回答。

② 输入方式：外设送来这一选通脉冲(前沿)，是为了将外设来的数据装入端口A的输入寄存器中。当此信号有效时，数据就被装入PIO。

③ 双向方式：当此信号有效时，从端口A输出寄存器来的数据被开放而进入端口A的双向数据总线上。选通脉冲的前沿作为收到数据的回答。

④ 位控方式：选通信号在内部被封锁。

(13) A RDY——端口A准备就绪信号(输出,高电平有效)。

此信号的作用与端口A的工作方式有关。

① 输出方式:本信号有效时,表示端口A输出寄存器中已装有数据,且外设数据总线也稳定了,因而已为向外设传送数据作好了准备。

② 输入方式:当端口A输入寄存器已空,且已准备好接受外设来的数据时,本信号有效。

③ 双向方式:当端口A输出寄存器已有数据可向外设传送时,本信号有效。在本方式中,除非 $\overline{\text{A STB}}$ 有效,否则数据不会放到端口A的数据总线上去。

④ 位控方式:本信号被封锁,并强制变为低电平。

(14) $PB_7 \sim PB_0$——端口B的总线(双向,三态)。

它们用来在端口B与外部设备之间传送数据和控制信息。端口B总线能在1.5 V下提供1.5 mA电流,以便驱动复合晶体管。

(15) $\overline{\text{B STB}}$——端口B选通脉冲(来自外部设备,输入、低电平有效)。

此信号的作用与端口B和端口A的工作方式有关。本信号的含义类似于 $\overline{\text{A STB}}$,但有以下例外:在端口A双向方式时,本信号将外设来的数据选通送入端口A的输入寄存器中。

(16) B RDY——端口B准备就绪信号(输出,高电平有效)。

此信号的作用与端口B和端口A的工作方式有关。本信号的含义也类似于A RDY,但也有以下例外:在端口A双向方式中,当端口A输入寄存器已空,并且准备好接收外设来的数据时,本信号有效。

(17) 其它:ϕ 为时钟;+5V为电源;GND为地。

二、Z 80 PIO的编程

1. 复位

Z 80 PIO在下列两种情况下自动进入复位状态:

(1) 接通电源时;

(2) 当 $\overline{\text{RD}}$、$\overline{\text{IORQ}}$ 信号无效(高电平),而 $\overline{M_1}$ 信号有效(低电平)时。

后一种复位功能可以使用户方便地控制 $\overline{M_1}$ 信号,以使PIO复位,而不必非经过掉电后再通电来复位。

复位状态执行以下功能:

(1) 两个端口的屏蔽寄存器都被复位(置为高电平),使端口的全部数据位被屏蔽。

(2) 端口数据总线被置成高阻状态,且RDY为无效(低电平),自动选用方式1(输入方式)。

(3) 矢量地址寄存器未被复位。

(4) 两个端口的中断允许触发器被复位。

(5) 两个端口的输出寄存器被复位。

一旦PIO进入复位状态,它就一直保持着,直到它接收到CPU来的一个控制字为止。

2. PIO寻址

每片PIO有两个端口:端口A和端口B。每个端口有两个可寻址的单元:一个是数据寄存器;另一个是控制寄存器。数据输入寄存器和数据输出寄存器共用一个数据端口。

写入控制寄存器的控制字共有6种,其中有3种仅用于位控方式,这将在位控方式中介

绍。为了能区分不同的控制字写入同一个控制端口地址，必须在送入控制寄存器的控制字中设置若干位作为寻址用，称之为特征位，还必须根据规定的先后顺序写入控制字。

3．PIO 的初始化

在 Z 80 PIO 开始执行数据传送前，必须首先进行程序初始化。如上所述，由 CPU 写入 PIO 相应端口控制寄存器的控制字有 6 种，其中 3 种仅用于位控方式，下面先介绍另外更常用的 3 种控制字的格式及含义。

(1) 装入中断矢量

PIO 是按中断方式 2 工作的，故在初始化程序时，应输入一个中断矢量，以便当中断响应时，能向 CPU 输出此中断矢量，其格式如图 13-44 所示。

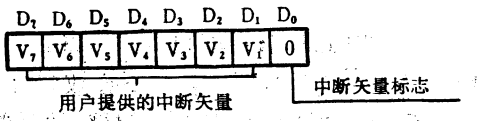


图 13-44 PIO 中断矢量字

其中规定 $D_0=0$ 作为中断矢量的标志。$V_7\sim V_1$ 由用户根据中断服务程序的入口地址在整个入口地址表中的安排来确定。假设端口 A 和端口 B 的中断矢量地址指针和中断服务程序入口地址如表 13-12 所示。

表 13-12 中断服务程序入口地址表

2340 H	0	0	端口 A 中断服务程序的入口地址
	2	2	
2342 H	1	B	端口 B 中断服务程序的入口地址
	2	2	
2344 H			

其中，设 CPU 中断矢量寄存器 I = 23 H，又设端口 A 的 $V_7\sim V_1=0100000$，即 PIO 的中断矢量 = 40 H，两者合成一个完整的中断矢量地址指针 2340 H。于是可由 2340 H 和 2341 H 单元中找到中断服务程序的入口地址 2200H。因为中断服务程序的入口地址占用两个单元，所以中断矢量可以安排成都是偶数，如 2340 H、2342 H、2344 H 等，即它的最低位必然为 0。由此可以与其它控制字相区别。

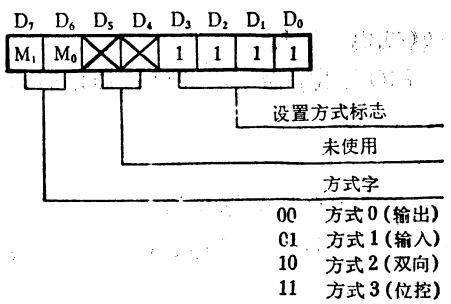


图 13-45 PIO 方式控制字

(2) 选择工作方式

PIO 的端口 A 具有 4 种工作方式。端口 B 具有除双向方式外的其它 3 种工作方式，如表 13-13 所示。每个端口的工作方式是在 PIO 初始化时，将方式选择控制字写入方式控制寄存器来设置的。选择工作方式控制字的格式如图 13-45 所示。

(3) 中断允许控制字

当 PIO 选用方式 0、1、2 时，可用中断允许控制字开放端口的中断允许触发器，其格式如图 13-46 所示。

规定：当 $D_7=1$ 时，PIO 端口的中断触发器被置“1”，此时端口允许中断请求；当 $D_7=0$ 时，中断允许触发器置“0”，端口禁止中断请求。$D_3\sim D_0=0011$ 是中断允许控制字的标志。

表 13-13 PIO 的操作方式

M_1 D_7	M_0 D_6	操作方式	说明
0	0	方式 0	带联络的输出
0	1	方式 1	带联络的输入
1	0	方式 2	带联络的双向 I/O
1	1	方式 3	位控操作方式

注：只有端口A能工作在方式2，此时端口B的联络线被端口A所占用，因而端口B只能工作在方式3。详细说明将在后面介绍。

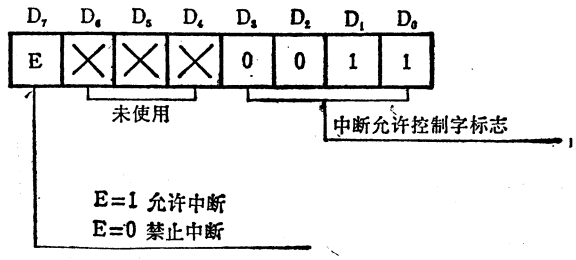


图 13-46 允许中断控制字

只要利用以上 3 种控制字，便可以设置 PIO 工作于方式 0、方式 1 或方式 2。

当 PIO 工作于位控方式时，除必须设置另外一种中断控制字外，还要另外附加两种控制字，总共 5 种控制字才能设置 PIO 工作于方式 3。

现分述如下：

(1) 方式 0——输出

执行下列一组指令后，可使 PIO 的端口 A 或端口 B 进入方式 0：

```
LD    A,    00001111B    ; 输出方式
OUT   (PIOCR A/B), A     ; 设置端口A或B为输出方式
```

此后，CPU 可通过一条 OUT 指令向端口 A 或 B 的输出寄存器写入数据并锁存起来。这样输出数据就出现在端口的数据线上。同时，输出寄存器的内容，任何时刻都可由 CPU 用一条 IN 指令读入 CPU。

在方式 0 工作时，CPU 向 PIO 写入一个数据，就使端口的 RDY 线变为高电平，通知外设可以取走这一数据。RDY 一直保持为高电平，直到从外设收到一个选通信号为止。选通信号的上升沿引起一个中断请求(若中断已被开放)，并使 RDY 线变为无效(低电平)。

(2)方式 1——输入

执行下列一组指令后，可使 PIO 的端口 A 或 B 进入方式 1：

```
LD    A,    01001111B    ; 输入方式
OUT   (PIOCR  A/B), A    ; 设置端口A或B为输入方式
```

此时，CPU 只要对端口 A 或 B 执行一次读操作，就会使 PIO 的 RDY 线有效，它指示外设应把数据装入已空的输入寄存器。于是，外设就用选通信号把数据送入端口输入寄存器。同时，在选通信号的上升沿将产生中断请求（若中断已被开放），并使 RDY 变为无效（低电平）。

(3) 方式 2——双向

执行下列一组指令后，可使 PIO 的端口 A 进入方式 2：

```
LD    A,          10001111B  ; 双向工作
OUT   (PIOCRA),   A          ; 设置端口 A 为双向方式
```

方式 2 是一种双向数据传送方式，它需要用四条“联络线”，故只有端口 A 可以用于方式 2，此时端口 B 只能工作在位控方式。

在这种方式下，输出控制使用端口 A 的 A RDY 和 $\overline{\text{A STB}}$ 信号线，而输入控制使用端口 B 的 B RDY 和 $\overline{\text{B STB}}$ 信号线。因此，A RDY 和 B RDY 可以同时起作用。

方式 2 是方式 0 和方式 1 的组合。方式 2 输出部分与方式 0 的唯一区别是：从端口 A 输出寄存器来的数据只有在 $\overline{\text{A STB}}$ 有效时，才能进入端口数据总线，以便实现双向功能。

(4) 方式 3

方式 3 用于提供控制和状态信息，它不需要联络信号。在这种方式中，可由程序设定端口的每一位为输入或输出，并当外设出现程序所规定的状态时，向 CPU 发出中断请求。

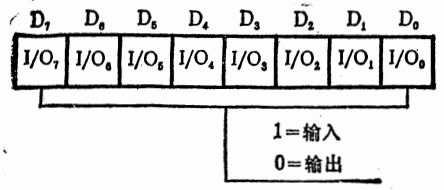


图 13-47　输入输出控制字

① 设置 I/O 选择字

当选用方式 3 时，在送入 PIO 的方式选择控制字后规定：必须紧跟着送入一个 I/O 选择控制字，锁存在 I/O 选择寄存器中，以确定端口数据总线中哪几条作为输入，哪几条作为输出。这个控制字的格式如图 13-47 所示。

该格式中，若 I/O = 1，则表示相应的数据位用作输入；若 I/O = 0，则表示相应的数据位用作输出。

若设定端口为位控方式工作，且数据线全部为输入，则可执行下列一组指令：

```
LD    A,          11001111B  ; 方式 3
OUT   (PIO CRA/B), A         ; 设置端口 A 或 B 为位控方式
LD    A,          0FF        ; 全部数据线为输入
OUT   (PIO CRA/B), A
```

若设定端口为位控方式工作，且数据线全部是输出，则可执行下列指令：

```
LD    A,          11001111B  ; 方式 3
OUT   (PIO CRA/B), A         ; 设置端口 A 或 B 为位控方式
XOR   A                      ; 全部数据线为输出
OUT   (PIO CRA/B), A
```

若设定端口为位控方式，令其 1、5、6 位为输入；令其 0、2、3、4、7 为输出，则可执行下列指令：

```
LD    A,          11001111B  ; 方式 3
OUT   (PIO CRA/B), A         ; 设置端口 A 或 B 为位控方式
LD    A,          01100010B  ; 位 1、5、6 为输入，位 0、2、3、4、7 为输出
OUT   (PIO CRA/B), A
```

在方式 3 工作时，对选通信号不予理会，而 RDY 线保持为“低”。在此期间，Z 80 CPU 可在任何时刻向一端口写入数据，或从一端口读出数据。在读一端口时，读回 CPU 的数据将是

由两类数据组成：一类是指定作为输入的端口数据总线上来的数据（即真正的输入信号），另一类是指定作为输出的那些线上来的输出寄存器的数据。

② 设置中断控制字，

PIO 中的每一端口的中断控制字格式如图 13-48 所示。

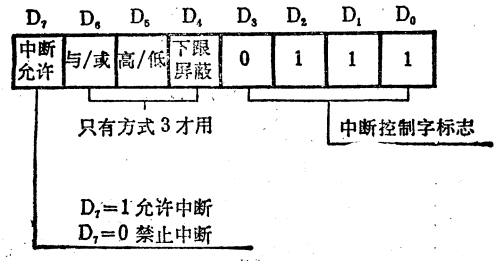


图 13-48 中断控制字

其中 D_6、D_5 和 D_4 只有在位控方式时才起作用。在其它方式时，这三位不起作用，可为任意值。当由 CPU 向 PIO 写入该字时，D_6、D_5 被锁存在屏蔽控制寄存器中。

在任何工作方式期间，若使中断控制字的 D_4 位置“1”，则将使挂起的中断（即有中断请求而未被响应）复位。

中断控制字各位的含义如下：

$D_7 = 1$ 表示开中断；$D_7 = 0$ 表示关中断。

$D_6 = 1$ 表示对被监视的信号（即未被屏蔽位）进行“与”操作，即这些位必须都进入规定状态才产生中断请求；

$D_6 = 0$ 表示对被监视的信号进行“或”操作，即只要任一指定位进入起作用状态就产生中断请求。

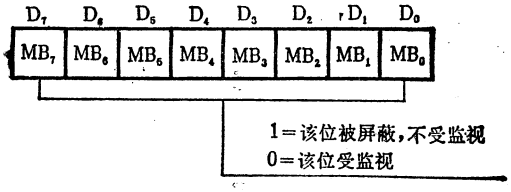


图 13-49 中断屏蔽字

$D_5 = 1$ 表示被监视的信号为高电平时有效；$D_5 = 0$ 表示被监视的信号为低电平时有效。

$D_4 = 1$ 表示跟在中断控制字后送入的控制字应为中断屏蔽字；$D_4 = 0$ 表示其它情况。

③ 屏蔽字

此字紧跟在中断控制字（且 $D_4 = 1$）之后写入，它被锁存在屏蔽寄存器中。其格式如图 13-49 所示。

MB = 0 表示该位受监视，允许引起中断；

MB = 1 表示该位的监视被屏蔽。

三、Z 80 PIO 的时序

下面我们将结合初始化程序来分析 PIO 的工作过程和时序关系。

1. 输出方式——方式 0

其初始化程序为：

```
LD    A,          23H   ; 设置中断矢量的高8位
LD    I,          A
LD    A,          40H   ; 设置中断矢量的低8位
OUT   (PIO CRA/B), A
LD    A,          0FH   ; 设定工作方式0
OUT   (PIO CRA/B), A
LD    A,          83H   ; 允许PIO请求中断
OUT   (PIO CRA/B), A
```

```
IM    2                    ; 设置中断方式 IM 2
EI                         ; 开中断
LD    A,          (HL)
OUT   (PIO DA/B), A        ; 向端口输出一个数据
```

注：PIO CRA/B 和 PIO DA/B 分别表示端口A或B的控制寄存器、数据寄存器的地址。

其工作过程如下：

(1) 设定中断矢量为 2340 H。

(2) 设定端口 A 为方式 0 工作。

(3) 对 PIO 端口 A 开中断，也就是使其中断允许触发器置位。

(4) 设定 CPU 的中断方式为 IM2，并使 CPU 开放中断。

(5) CPU 执行一条输出指令 OUT (PIO DA/B)，端口 A 或 B 即开始方式 0 的输出周期，如图 13-50 所示。在 CPU 执行输出指令期间，PIO 产生一个 $\overline{WR}$* 脉冲，它将 CPU 数据总线来的数据，锁存在所访问的端口 A 或端口 B 的输出寄存器中。然后，在 $\overline{WR}$* 脉冲上升沿后的一个时钟 ϕ 的下降沿，使 RDY 变为高电平，指示外设端口中已有数据可供使用。

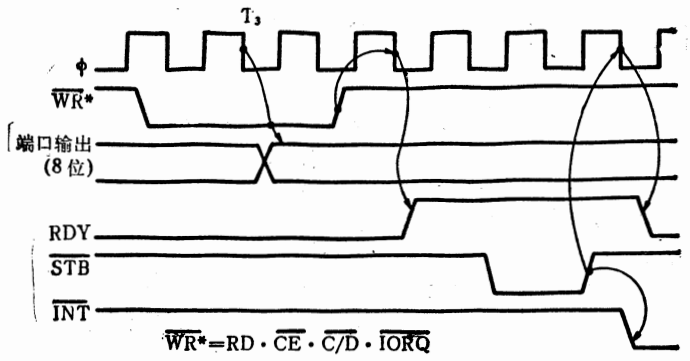


图 13-50 PIO 的输出时序波形图

在大多数系统中，RDY 的上升沿可用作外设的选通信号(若外设需要时)。此后，RDY 将保持有效，直到从外设收到一个选通信号 $\overline{STB}$ 为止。当外部设备在 $\overline{STB}$ 线上发出一个负脉冲，并在 $\overline{STB}$ 的上升沿将端口中数据取走后，如果中断是开放的，且该端口是申请中断源中优先权最高的，就可自动产生中断请求，即 $\overline{INT}$ 变为低电平，并使待命线 RDY 失效。此后，如果 CPU 已开放中断，则 CPU 响应此中断请求，端口将中断矢量 40 H 与 CPU 中的 I 寄存器中的 23 H 合成一个完整的中断矢量 2340 H，在 2340 H 和 2341 H 单元中取出端口中断服务程序的入口地址，例如，如表 13-11 所示的 2200 H。接着执行入口地址为 2200 H 的中断服务程序，CPU 执行相应的操作，并向端口输出一个数据，此后的过程与上述相同。

由此可知，外部设备与 CPU 之间的全部数据传送是在中断控制下实现的。

2．输入方式——方式 1

图 13-51 示出了一个输入周期的时序图。

这一周期是在 CPU 执行了一次读操作以后，RDY 变为有效(高电平)，通知外设端口输入寄存器已空，可输入新的数据。当外设把新数据准备好后，利用选通信号 $\overline{STB}$ 来启动 PIO。当 $\overline{STB}$ 变为有效(低电平)时，就从外设将数据装入 PIO 端口内的数据输入寄存器中，开始了一个新的输入过程。$\overline{STB}$ 的上升沿将使 $\overline{INT}$ 电平变为有效(低电平)。如果这时中断允许触发器

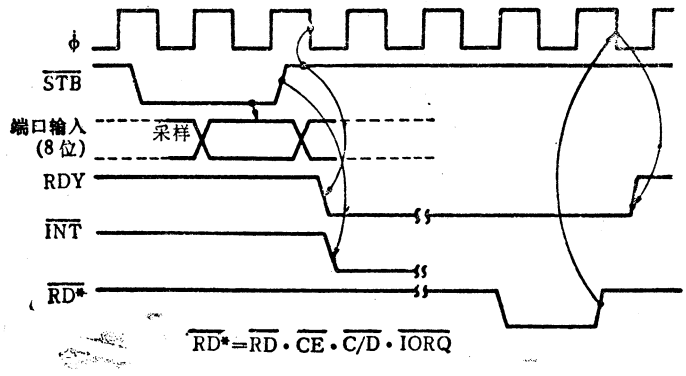


图 13-51 I/O 的输入时序波形图

已被置位，且这一器件的中断请求是优先权最高的话，则向 CPU 发出中断请求。同时，在 $\overline{STB}$ 上升沿后的时钟下降沿使 RDY 变为低电平，表示数据输入寄存器已满，禁止外设输入新的数据。此后，CPU 在执行中断服务程序过程中，执行一次 I/O 读操作，从端口输入寄存器中读取数据（当在 $\overline{RD^*}$ 上出现一个负脉冲时）。同时，在 $\overline{RD^*}$ 信号的上升沿的时钟 ϕ 的下降沿时，使 A RDY 又变为高电平，指示输入寄存器已空，允许外设将新的数据装入端口。

3．双向方式——方式 2

这一方式是方式 0 和方式 1 的组合，它需要四根联络线。即除了需要端口 A 的两条联络线 $\overline{A\ STB}$ 和 A RDY 作为输出的联络控制用外，还需要占用端口 B 的两条联络线 $\overline{B\ STB}$ 和 B RDY作为端口 A 输入的联络控制用。此时，端口 B 只能工作在方式 3（位控方式）。

图 13-52 示出了方式 2 的时序图。

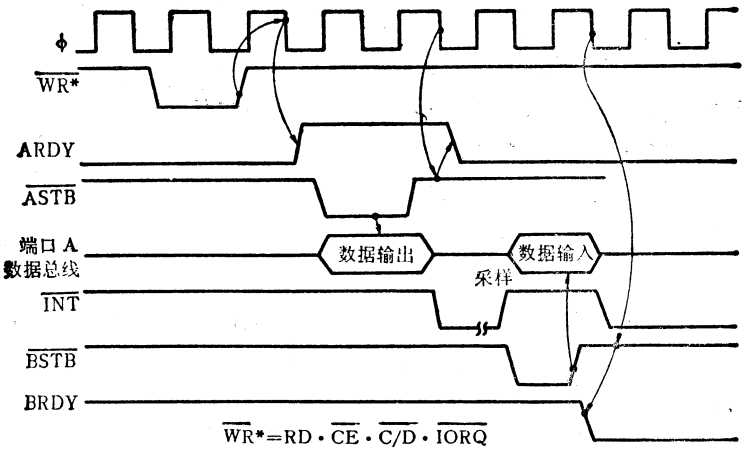


图 13-52 PIO 双向方式时序波形图

它与以上方式 0 和方式 1 所介绍的情况几乎是相同的，其差别仅在于：在方式 2 中，只有当 $\overline{A\ STB}$ 为低电平时，才允许数据送到端口 A 的总线上，这一选通的上升沿可用来将数据锁入外设。值得注意的是，在双向工作时必须将端口 A 和端口 B 的中断允许触发器同时置位（即开放中断），才能实现中断驱动的双向传送。此外，还应注意两点：① 当 $\overline{A\ STB}$ 有效时，外设决不能将数据送到端口总线上，以免发生总线冲突。② 对端口 B 方式 3 时的中断和对端口 A 方式 2 时的输入中断，两者所送回的中断矢量是相同的。因此，此时端口 B 应工作在查询方式

或端口B屏蔽寄存器为全1（封锁所有各位）状态，以免发生混乱。

4．位控方式——方式3（以端口B为例）

这一控制方式不使用联络信号，并可在任何时刻执行常规的端口写入或读出操作。在读PIO时，送回CPU的数据由两部分组成：一部分是数据输出寄存器中相应于被指定为输出位的内容；另一部分是数据输入寄存器中相应于被指定为输入位的内容。

方式3操作时序如下：在写入时，数据将被锁入端口的输出寄存器，其时序波形和方式0相同。当端口A工作在方式3时，A RDY线将被强制变为低电平。当端口B工作在方式3时，B RDY将保持低电平，除非端口A处于方式2。

在方式3读PIO时，输入寄存器中保存的数据是刚好在$\overline{RD}$下降沿之前出现在数据总线上的数据。如图13-53所示。

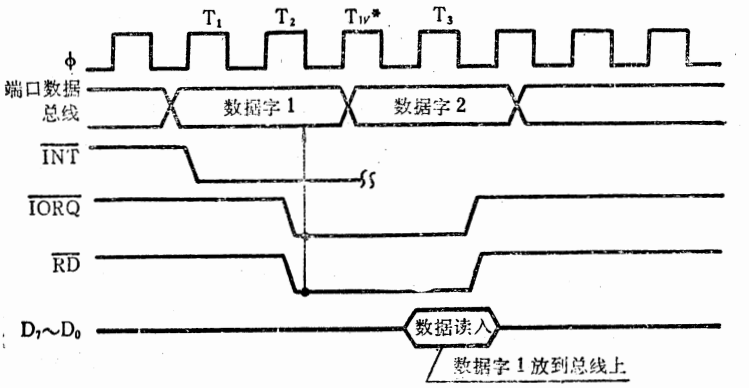


图13-53 PIO方式3读入时序波形图

如果端口的中断是开放的，并且端口数据线上的数据满足8位屏蔽寄存器和2位屏蔽控制寄存器所规定的逻辑式，则产生中断。在逻辑式的状态发生变化之前，是不会发生另一个中断的。也就是说，方式3的中断只有在下列情况下产生，即方式3逻辑操作的结果从"0"变成为"1"。如果假定方式3逻辑式为"或"操作，当一个未被屏蔽的端口的一条数据线有效并发出中断请求时，若有第二条未屏蔽端口数据线与第一条同时有效，将不产生新的中断请求，因方式3逻辑操作的结果并未发生变化。

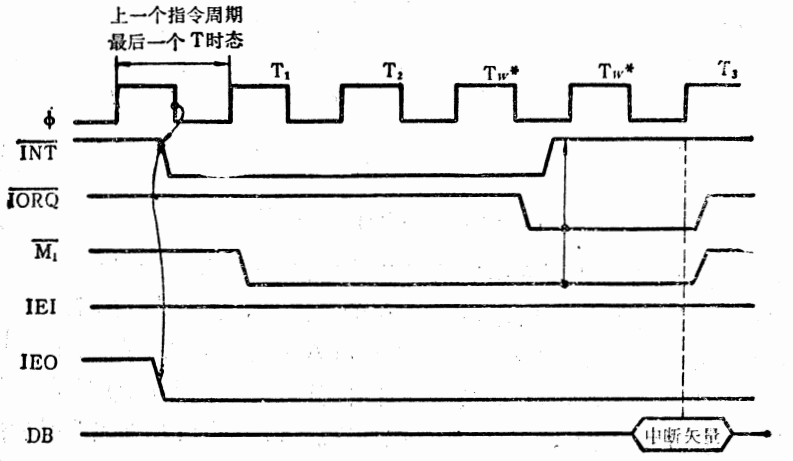


图13-54 PIO中断响应时序波形图

5．中断响应时序

在PIO发出中断请求之后，如果CPU的中断是开放的，则将送出一个中断响应（$\overline{M_1}$和$\overline{IORQ}$）信号。这时，PIO中断逻辑将判定哪一个是优先权最高的中断请求端口。在中断响应期间，优先权最高的端口将把其中断矢量放到Z 80数据总线上。其时序如图13-54所示。

当PIO的一个端口有中断请求之后，就使它的IEO变为低电平，封锁优先权低的端口申请中断，IEO一直保持低电平，直到它的请求被响应处理完毕，接受到中断返回指令（RETI）为止。若这一中断请求未被响应，则在PIO译码到操作码"ED"后，经过一个$\overline{M_1}$周期，IEO将被强制变为高电平，保证两字节的RETI指令可被适当的PIO端口所译码（详见中断一章）。

PIO同样可以实现嵌套中断，其过程与Z 80 CTC类似，故不再重复。

四、Z 80 PIO应用举例

1．Z 80 PIO与μ80打印机的接口设计

μ80打印机（MICROLINE 80 MATRIX PRINTER）是一种含有微处理器的新颖的点阵式打印机。

（1）μ80打印机主要的接口信号

① $D_{7\sim0}$——8位数据线。

$D_{7\sim0}$用来接收从CPU发来的信息。由于μ80打印机仅能识别ASCII码，因此欲打印的字符要预先由程序变换成ASCII码。

② $\overline{DATA\ STROBE}$——数据选通，低电平有效。

当处于选通工作的接口器件（如PIO）收到来自CPU的数据后，将通过数据选通信号线通知打印机接收数据。

③ $\overline{ACK}$——响应信号，低电平有效。

打印机从接口器件接收数据并存入缓冲存贮器后，将由$\overline{ACK}$信号线向接口器件发出应答信号，表示数据已被接收。

④ BUSY——"忙"信号，高电平有效。

当打印机的缓冲存贮器已装满信息或正在执行某种功能而不能接收数据时，将通过BUSY线向接口器件发出"忙"信号，使CPU暂停送数。

（2）μ80打印机的工作时序

μ80打印机的工作时序如图13-55所示。

其工作过程如下：

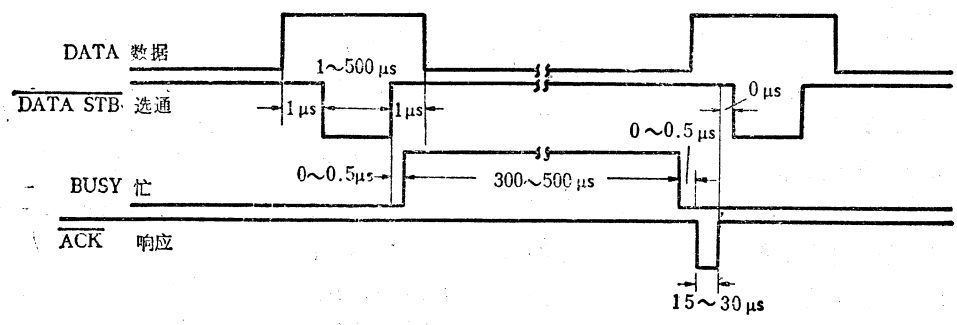


图13-55 μ80打印机工作时序图

① 当来自接口器件的并行数据送到打印机的数据线上时，打印机中的微处理器并不立即接收数据而是等待选通信号的到来。为确保数据的有效，$\overline{\text{DATA STB}}$ 必须在数据稳定后 1μs 才能变为有效，当其撤消后，有效数据还应维持 1 μs。

② 当 $\overline{\text{DATA STB}}$ 到来后，打印机接收数据，并使 BUSY 变为高电平，表示打印机处于“忙”的状态，禁止接收数据。BUSY 从 $\overline{\text{DATA STB}}$ 的上升沿后的 0～0.5μs 开始变为高电平，维持 300～500μs 后自动变为低电平，然后再经过 0～0.5 μs 使 $\overline{\text{ACK}}$ 变为有效(低电平)，并维持 5～30 μs 后即可再次接收 $\overline{\text{DATA STB}}$ 的有效信号。打印机发出 $\overline{\text{ACK}}$ 有效信号，通知 CPU 输出的数据已被接收完毕，可以送来新的数据。

(3) Z 80 PIO 输出时序与 μ 80 打印机工作时序的配合。

Z 80 PIO 与 μ 80 打印机的接口电路如图 13-56 所示。

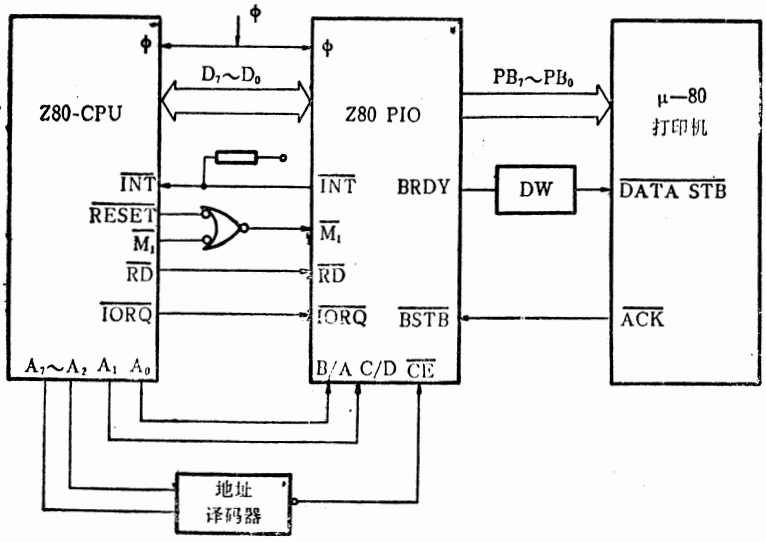


图 13-56 Z 80 PIO 与 μ80 打印机的接口电路

为了使 PIO 与 μ 80 打印机之间的时序能正确地配合。在 PIO 端口 B 的 B RDY 端接入一个单稳电路(比如 74 LS 123)。其目的是利用 B RDY 信号的上升沿，经过单稳电路后形成一个负脉冲，以作为 μ 80 打印机的选通信号 $\overline{\text{DATA STB}}$。

μ 80 打印机的输出信号 ACK 可直接作为 PIO 端口 B 的输入信号 $\overline{\text{B STB}}$，用来通知 CPU 可以再次输出新的数据。

(4) 打印过程

设打印机和 CPU 采用程序中断方式传送数据。首先，CPU 通过输出指令将需要打印的数据送入 P O 端口 B，当端口 B 数据准备好时，B RDY 信号变为有效(高电平)并经单稳电路形成一个符合 μ 80 打印机要求的选通数据信号(负脉冲)，通知打印机接收数据。如果打印机不“忙” (BUSY = “0”)，将接收数据并存入缓冲存贮器。此间打印机使 BUSY 变为有效(高电平)，表示不能接收新的数据。当打印机接收数据完毕，BUSY 自动变为低电平，然后发出 $\overline{\text{ACK}}$ 有效信号，表示数据已被接收。PIO 接受打印机发来的 $\overline{\text{ACK}}$ 信号后，向 CPU 发出中断请求。若 CPU 开放中断，则响应中断后，转入执行中断服务程序。在中断服务程序中，再向打印机输出新的数据。打印机在接到 CPU 发出的信息后将进行判别，若是“打印字符”，则存入缓冲存

贮器，待装满一行再进行打印。若是“动作信号”，则进行相应的动作。

(5) μ80 打印机控制字格式

由于μ80 打印机中含有 CPU，因此它有可编程的功能。其符号和控制字及功能说明见表 13-14。

表 13-14 μ80 打印机控制字

符　　号	控　制　字　(H)	功　能　说　明
CR	0D	设置打印机初始状态，并回车至行首
LF	0A	换行
GS	1D	打印格式为 16.5 字符/英寸
RS	1E	打印格式为 10 字符/英寸
US	1F	打印格式为 5 字符/英寸
ESC-6	1B-36	使行间距为 6 行/英寸
ESC-8	1B-38	使行间距为 8 行/英寸
ESC-A	1B-41	使每行字符数为 80 个
ESC-B	1B-42	使每行字符数为 64 个

注：① 当打印机上电后，它立即设置为 10 字符/英寸、6 行/英寸和 80 个字符/行。

② ESC-i 为两字节控制字，应连续送两个代码。

③ 一旦每英寸字符数、行数及每行字符数设置好后，它们将保持不变，直到输入新的控制字代码为止。

(6) 程序编制

设有地址为 20B0H 的数据缓冲区。用作存放待打印的 ASCII 码，然后通过程序将它们逐个送至打印机，只有当送满一行的字符数或送出 0DH (CR 回车)或 0AH(LF 换行)码时，打印机才开始打印。输出的字节数为N个。

再设中断矢量地址指针为 2060 H，中断服务入口地址为 2080 H。并利用 Z 80 STARTER KIT 单板机上的 PIO B 端口作为输出端口。

其打印程序如下：

```
PRINT: LD   A,   20H    ; 中断矢量高 8 位
       LD   I,   A
       LD   A,   60H    ; 中断矢量低 8 位
       OUT  (83H), A
       LD   A,   0FH    ; 设 PIO B 端口为输出方式
       OUT  (83H), A
       LD   A,   83H    ; 设 PIO 允许中断
       OUT  (83H), A
       IM   2           ; 设中断方式 2
       LD   C,   81H    ; PIO 端口地址送 C 寄存器
       LD   HL,  20B0H  ; HL 指向打印缓冲区首地址
       LD   B,   N      ; 打印字节数送 B 寄存器
LOOP:  OUTI             ; 将打印缓冲区数据逐个送打印机
       EI               ; 开放 CPU 中断
       HALT             ; 暂停
       JR   NZ,  LOOP   ; 未送完返回 LOOP 继续再送
```

```
        HALT                    ; 打印完毕,停机
    ; 中断服务程序入口地址表
2060H   80
2061H   20
    ; 中断服务程序
2080H   RETI
```

2．控制接口——位控方式的应用

图 13-57 示出了一个典型的控制方式的应用实例。

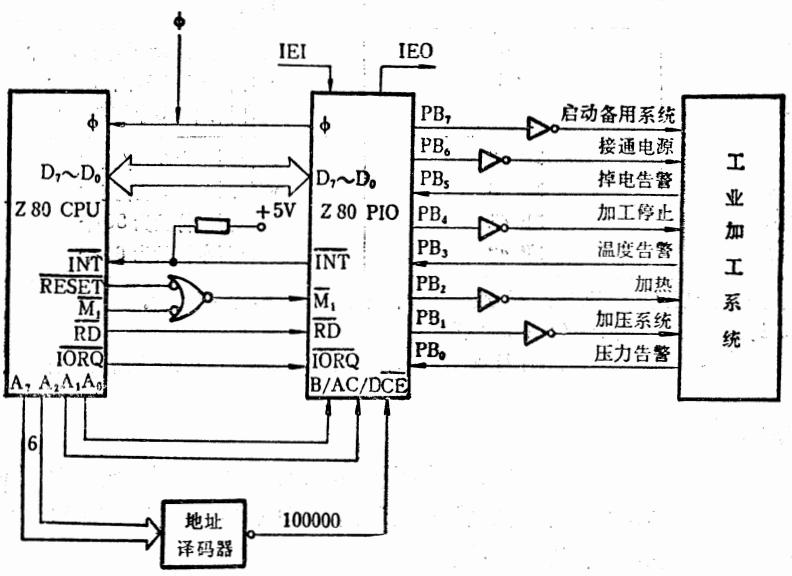


图 13-57　Z 80 PIO 控制接口图

假设有以 CPU 为中心组成的控制系统，要监视一个工业加工过程。当任何不正常工作情况发生时，都将报告 CPU。这个过程的控制和状态字具有下列格式：

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
启动备用系统	接通电源	掉电告警	加工停止	温度告警	加热	加压系统	压力告警

若由端口 B 工作于方式 3 来实现上述要求，则需要写入如下的控制字：

(1) 方式控制字

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	1	×	×	1	1	1	1

置为 00

当选用方式 3 时，送到端口的下一个控制字必须是一个 I/O 选择字。

(2) I/O 选择字

本例中要求 PB_0、PB_3、PB_5 作为输入，其余为输出，故需写入如下 I/O 选择字：

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	1	0	1	0	0	1

(3) 中断矢量

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	1	0	0	0	0	1	0

(4) 中断控制字

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	1	1	0	1	1	1
中断允许	或逻辑	高电平有效	下跟屏蔽字		中断控制		

此中断控制字表示对受监视的输入线进行“或”操作，并设定高电平有效。因为工作在方式3，故在中断控制字后，必须写入一个屏蔽字。

(5) 屏蔽字

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	1	0	1	0	1	1	0

这个屏蔽字表示 PB_0、PB_3 和 PB_5 线受监视，其余的都被屏蔽。在这种情况下，当 PB_0、PB_3、PB_5 任何一个出现高电平时，都会引起中断，要求 CPU 进行告警处理。

若设端口B地址为：数据端口 = 81 H

控制端口 = 83 H

中断矢量地址指针为 2342 H，中断服务程序入口地址为 2200 H，则其初始化程序如下：

```
INITIA: LD    A,      23H   ; 设定中断矢量高8位
        LD    I,      A
        LD    A,      42H   ; 设定中断矢量低8位
        OUT   (83H),  A
        LD    A,      0CFH  ; 设置方式控制字
        OUT   (83H),  A
        LD    A,      29H   ; 设置 I/O 选择字
        OUT   (83H),  A
        LD    A,      0B7H  ; 设置中断控制字
        OUT   (83H),  A
        LD    A,      0D6H  ; 设置屏蔽字
        OUT   (83H),  A
        IM    2
        EI
        LD    HL,     2200H ; 把中断服务程序首地址 2200H 送入 2342H、2343H 单元
        LD    (2342H), HL
```

随后，就进入执行主程序。

在编制中断服务程序时，如需要用到 CPU 中的各寄存器，则应该把各寄存器中的内容推入堆栈(保护现场)，待中断服务程序执行完毕时，再把存入堆栈的各寄存器的原有内容按存入时相反的次序弹回各寄存器中(恢复现场)。然后再安排 EI 和 RETI 两条指令。如果此中断服务程序允许被另一优先权更高的中断服务程序所中断，则在保护现场完成后，立即加上 EI 指令，在恢复现场之前加上禁止中断指令 DI。这样在执行 PUSH 或 POP 指令时禁止中断，待各寄存器的内容推入堆栈或弹出堆栈后才允许中断，故中断服务程序的首尾应为：

```
SUT: PUSH   AF
     PUSH   BC
     PUSH   DE
     PUSH   HL
     EI             ; 以下允许中断
     ⋮              ;
     ⋮              ;  }中断服务程序的内容
     ⋮              ;
     DI             ; 以下禁止中断
     POP    HL
     POP    DE
     POP    BC
     POP    AF
     EI             ; 返回后允许中断
     RETI
```

第十四章　可编程序串行 I/O 接口电路

本章讨论串行通信及其 I/O 接口电路。

§14-1　串 行 通 信

微型计算机与一些常用的外围设备(如电传打字机 TTY、CRT 终端、调制解调器等)之间的数据交换,往往需要采用串行通信方式。尤其是在远程计算机通信中,串行通信更是一种不可缺少的通信方式。

串行通信是指数据一位一位顺序传送。它可由两种方式来实现:一种是将 8 位通道中的一位直接用于串行数据传送,它是依靠软件来实现的,这样会降低 CPU 的利用率;另一种是利用专用的通信接口来实现并行码和串行码之间的转换。常用的有下列几种:

通用异步接收-发送器 UART(Universal Asynchronous Receiver/Transmitter);

通用同步/异步接收-发送器 USART (Universal Synchronous Asynchronous Receiver/Transmitter);

异步通信接口适配器 ACIA(Asynchronous Communication Interface Adapter);

远程数据通信接口 TDI(Telecommunication Data Interface)。

由于串行通信只需用一条传送线,它比并行通信更加方便,而且节省传送线,因此,在远程数据通信中,一般都采用串行通信方式。当然它的传送速度不如并行通信快。

一、串行通信的基本方式

在串行通信中,有两种基本的通信方式:

1. 异步通信 ASYNC(Asynchronous Data Communication)
2. 同步通信 SYNC(Synchronous Data Communication)

现分述如下:

1. 异步通信

在异步通信中,CPU 与外围设备之间有两项约定:

(1) 字符格式。即字符的编码形式及规定,如 ASCII 码规定:每个串行字符由以下 4 个部分组成:

① 1 个起始位;

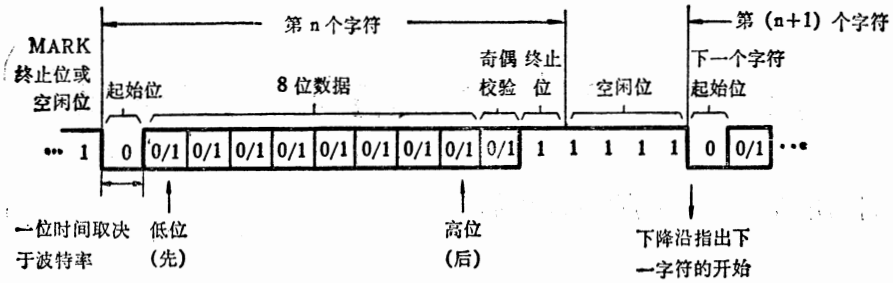
② 5～8 个数据位;

③ 1 个奇偶校验位(作为检错用);

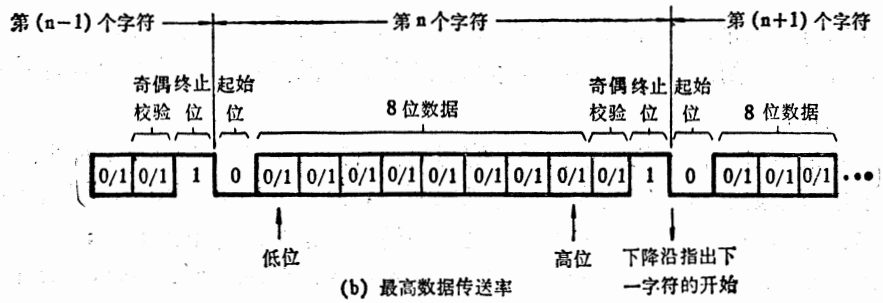
④ 1～2 个终止位(停止位);

图 14-1 示出这种串行码字符传送的两种格式。

起始位后面紧跟的是要传送字符的最低位,每个字符的结束是一个高电平的终止位,起始位至终止位构成一帧。相邻两个字符之间的间隔可以是任意长度的,以便使它有能力处理实时的串行数据。两个相邻字符之间叫空闲位。然而,下一个字符的开始,必然以高电平变成低



(a) 小于最高数据传送率



(b) 最高数据传送率

图 14-1 异步串行码字符格式

电平起始位的下降沿作为标志。图 14-1(a) 示出小于最高数据传送率的格式，其最高数据传送率的格式即为在相邻字符之间去除空闲位后的格式。

(2) 波特率(Baud rate)

波特率是指每秒传输代码(字符)的位数。

假如数据传送速率是 120 字符/秒，而每一个字符格式规定包含十个数据位(起始位、终止位、八个数据位)，则这时传送的波特率为：

$$10 \times 120 = 1200 \text{ 位/秒} = 1200 \text{ 波特}$$

而每个数据位的传送时间 T_d 即为波特率的倒数：

$$T_d = \frac{1}{1200} = 0.000833 \text{ s} = 0.833 \text{ ms}$$

波特率也是衡量传输通道频宽的指标。应当指出，它是指传送代码的速率，这与传送数据位的速率有所区别。因为每个数据只占 8 位，所以数据的传送速率为 $8 \times 120 = 960$ 波特。

异步通信的传送速度一般在 50 到 9600 波特之间，常用于计算机到 CRT 终端和字符打印机之间的通信等。

2. 同步传送

在异步传送中，每一个字符要用起始位和终止位作为字符开始和结束的标志，占用了一些时间，因而在数据块传送时，为了提高速度，就要设法去掉这些标志，而采用同步传送。此时，在数据块开始处要用同步字符来指明，如图 14-2 所示。

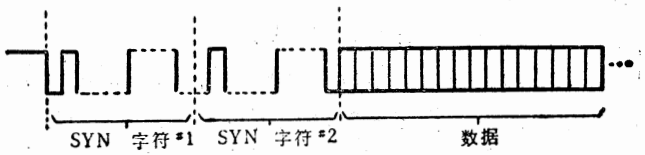


图 14-2 同步字符

同步传送速度高于异步传送速度，可达 56 千波特(KILOBAUD)。但它要求有时钟来实现发送端及接收端之间的同步，故硬件电路比较复杂。通常用于计算机之间的通信或计算机到 CRT 等外设之间的通信等。

二、串行通信中的基本技术

1. 数据传送方向

在串行通信中，数据通常是在两个站(如 A 和 B)之间进行双向传送的。根据需要又分为两种：

(1) 半双工(Half Duplex)

图 14-3 表示这种传送方式。

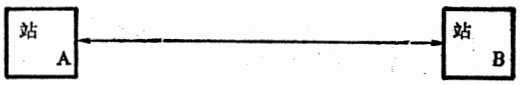


图 14-3 半双工示意图

这种方式是指通信时双方都能收和发，但不能同时收和发的通信方式。在这种通信方式中，通信双方只能轮流地进行发送和接收，即时而 A 站发送，B 站接收，时而 B 站发送，A 站接收。

(2) 全双工(Full Duplex)

图 14-4 表示这种传送方式。

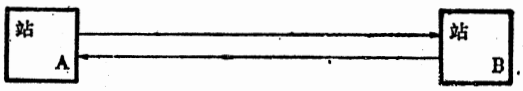


图 14-4 全双工示意图

这种方式是指可以同时在两个站之间进行发送和接收的通信方式。可见，全双工需要两路传输线。

2. 信号的调制和解调

Modem 一词是英文 Modulator-Demodulator 即调制-解调器的缩写。它是计算机的远程通信中必须采用的一种辅助的外设。

计算机通信是一种数字信号的通信。如图 14-5 所示。

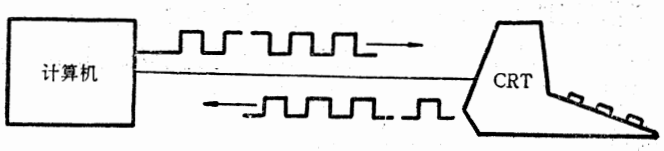


图 14-5 通信信号示意图

它要求传送的频带很宽，而计算机在远程通信时，通常是通过载波电话(Carrier telephone)线传送的，它不可能有这样宽的频带。如果用数字信号直接通信，那么经过传送线，信号就会产生畸变，如图 14-6 所示。

因此，在发送端必须采用调制器把数字信号转换为模拟信号，即对普通载波电话线上的高

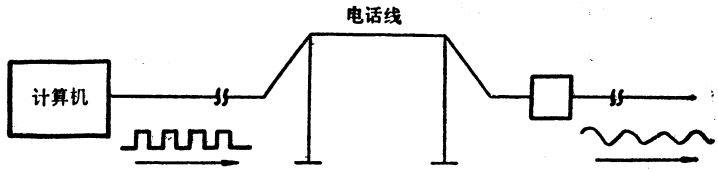


图 14-6　数字信号通过电话线传送产生的畸变

频载波进行调制；而在接收端又必须用解调器检测发送端来的模拟信号，再把它转换成数字信号，如图 14-7 所示

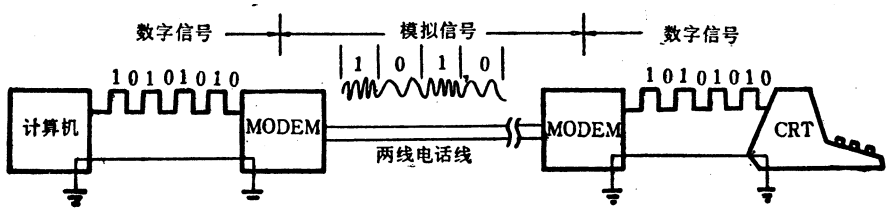


图 14-7　调制与解调示意图

从图可知，Modem 在发送端相当于 D/A 转换器，而在接收端则相当于 A/D 转换器。

按调制方式，Modem 可分为三类：即调幅、调频和调相。其中，调频方式是常用的一种调制方式。调频时，数字信号“1”与“0”被调制成易于鉴别的两个不同频率的模拟信号。这种形式的调制称为频移键控 FSK(Frequency Shift Keying)，其原理如图 14-8 所示。

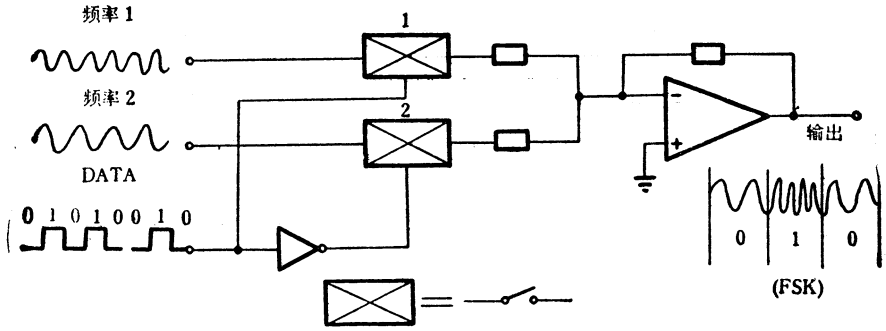


图 14-8　FSK 调制法原理图

两个不同频率的模拟信号，分别由电子开关控制，在运算放大器的输入端相加，而电子开关由需要传输的数字信号(即数据)来控制。当信号为“1”时，控制 1 号电子开关导通，送出一串频率较高的模拟信号；当信号为“0”时，控制 2 号电子开关导通，送出一串频率较低的模拟信号，于是在运算放大器的输出端，就得到了调制后的信号。

3. Modem 的接口——EIA RS-232 C 接口

Modem 与数字装置连接时需要通过接口，国外现有数据通信系统中所用 Modem 的接口，通常采用美国电子工业协会(EIA)的 RS-232 C 标准接口。

EIA(Electronic Industry Association) RS (Recommended Standard)-232 C 是目前最常用的串行通信接口之一，它用作数据终端设备(DTE)与数据通信设备(DCE)之间的串行接口。实际上，它是一个 25 只脚的标准串行连接器，它的每一个连脚的信号及电平规定都已标准化，

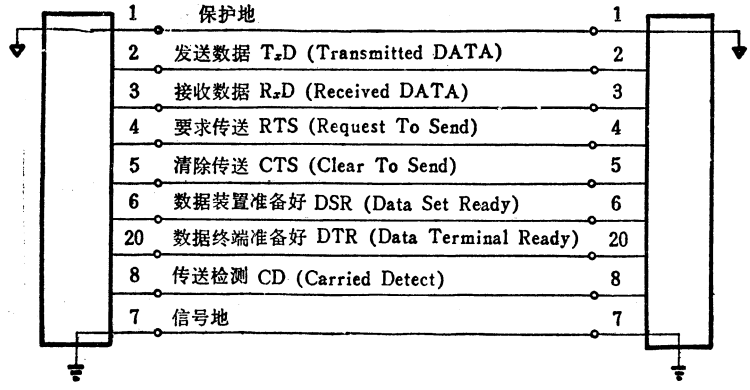


图 14-9 RS-232 C 的连脚图

以便作为通用接口。其基本的信号规定如图 14-9 所示。

对于 RS-232 C 而言，逻辑"1"电平为 −5V～−15V，而逻辑"0"电平为 +5V～+15V。因此，RS-232 的电平不一定能够直接地与 TTL 电平相兼容。为了使 RS-232 的信号能与 TTL 电路相连接，必须进行电平转换，即把 RS-232 的电平转换为 TTL 电平和把 TTL 电平转换为 RS-232 的电平。这就需要使用专门的传输线驱动器和传输线接收器。常用的有 1488 型 RS-232 C/TTL 电压电平转换器和 1489 型 TTL/RS-232 C 电压电平转换器，以及 8 T 15 和 8T 16EIA 传输线驱动器和传输线接收器，这些器件都能进行电压电平的转换，并提供所需的驱动电流及信号调节性能。1488 型传输线驱动器和 1489 型传输线接收器，如图 14-10 所示。

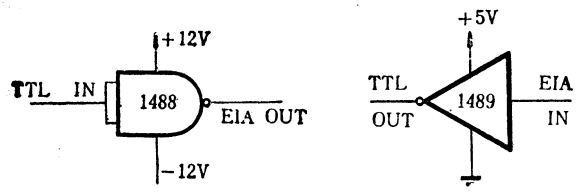


图 14-10 1488 型和 1489 型电路示意图

RS-232 C 接口广泛用于传送速率不高或中等传送速率的场合。

RS-232 C 有以下几种标准的传送速率(波特)：

19200	9600	4800	2400
1200	600	300	150
110	75	50	

计算机和远程以及当地终端(用查询方式交换信号)的连接示意图如图 14-11 所示。

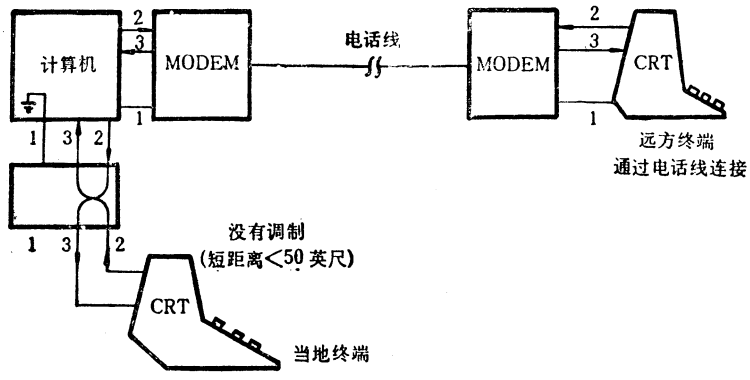


图 14-11 计算机与远程终端和当地终端连接示意图

当地终端可直接通过 RS-232 C 接口连接；而远程终端需要经过调制后通过电话线传送。在数据终端与调制器之间用 RS-232 C 接口。

三、通用异步收发器 UART

如上所述，CPU 与外界的串行通信可以用软件也可用硬件实现。UART 是用硬件实现串行通信的通信接口电路之一。

1．组成

UART 通常由三个部分：接收器、发送器和控制器组成的，如图 14-12 所示。

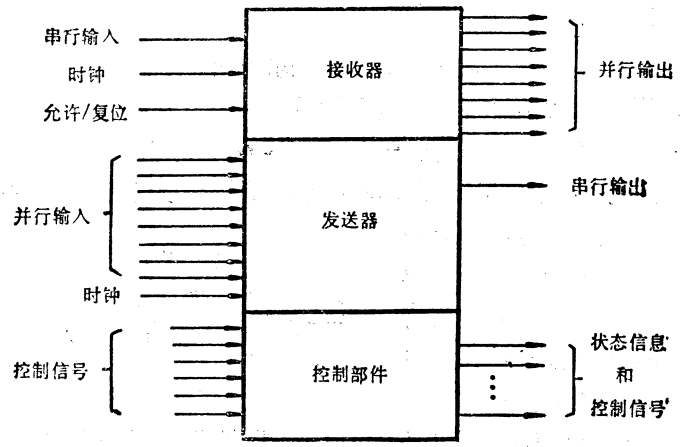


图 14-12　UART 的组成框图

接收器用来把串行码转换为并行码；发送器用来把并行码转换为串行码；而控制器则用来接收 CPU 的控制信号，执行 CPU 所要求的操作，并输出状态信息和控制信号。

2．功能

UART 的功能是既能接收异步串行输入码并将其转换为 CPU 所需要的并行码，也能将 CPU 内部的并行码转换为串行码输出。串行码输入转换成并行码的原理如图 14-13 所示。

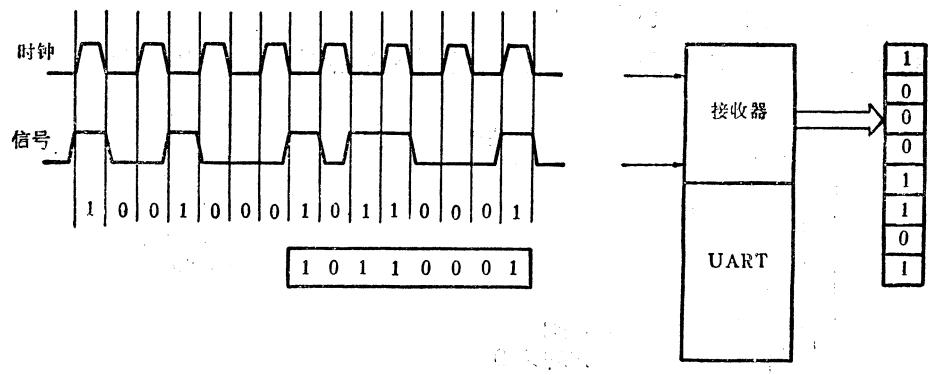


图 14-13　UART 的串行码转换成并行码

在 UART 工作时，其接收器始终监视着串行输入端，当发现一个起始位时，就开始一个新的字符的接收过程。并在接收过程中，自动检查每个字符的最后的终止位（逻辑“1”），以便得

到同步。如果发现终止位为逻辑“0”，则将发出一个出错状态位。CPU 将检测此状态位，并作出相应的处理动作。

UART 是用外部时钟来和接收的数据进行同步的。外部时钟的周期 T_c 和每个数据位的周期 T_d 有以下关系：

$$T_c = \frac{T_d}{K}$$

其中 K = 16 或 64

外部时钟和接收数据的同步如图 14-14 所示。

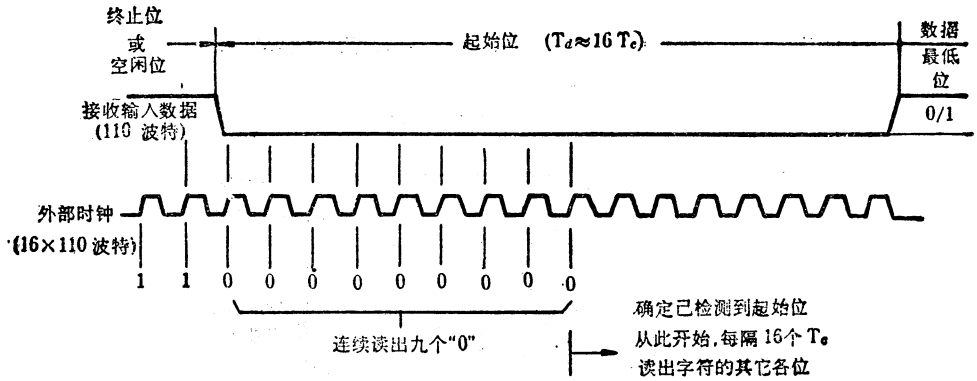


图 14-14　外部时钟和接收数据起始位同步

设每个数据位周期 T_d 为外部时钟周期的 16 倍，则 UART 在终止位和若干个空闲位以后，要找出起始位，使它与外部时钟同步。在每个外部时钟的上升沿采样接收数据线，在找出连续九个“0”以后，开始读出接收数据的每个数位值，然后，每隔 16 个外部时钟脉冲采样一次数据线，如图 14-15 所示。由图可知，采样的时间正好在数据位的中间。

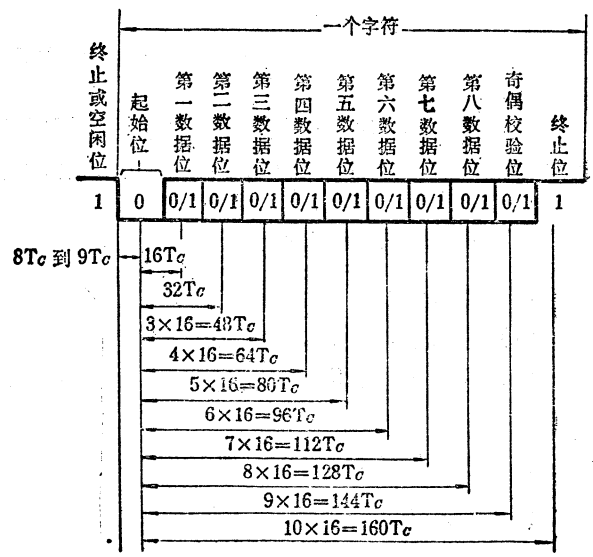


图 14-15　UART接收数据定时方式

为了检测远距离传送数据的正确性，需要加一位奇偶校验位。在数据传送时，UART 检查每个要传送字符中“1”的个数。若采用偶校验，则在发送时 UART 会自动在奇偶校验位上添“1”或“0”，使得“1”的总和(包括奇偶校验位)为偶数。CPU 将检查这个奇偶校验位，并做出相应操作。UART 处理奇偶校验位的方法如图 14-16(a) 和 14-16(b)所示。

随着大规模集成电路技术的发展，通用的可编程序的同步和异步通信接口电路芯片种类繁多。常用的有 Intel 8251、Zilog Z80-SIO 和 Motorola 的 MC 6850 (ACIA) 等。

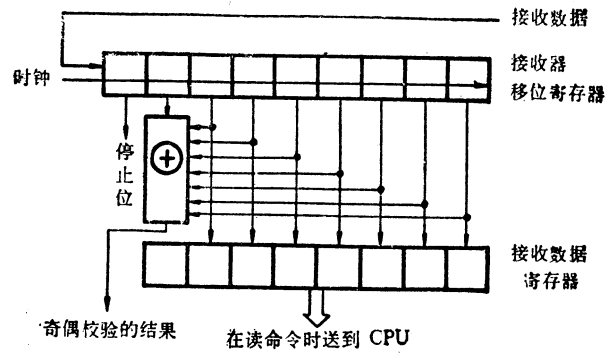


图 14-16(a)　奇偶校验电路在接收时的工作情况

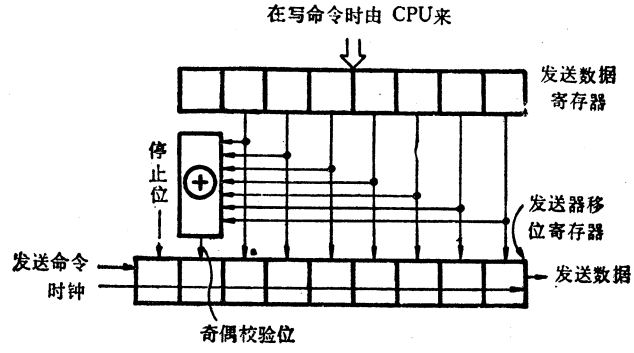


图 14-16(b)　奇偶校验电路在发送时的工作情况

§14-2　Intel 8251A USART 可编程通信接口电路

Intel 8251 A USART 是可编程序的串行通信接口电路。它既可以进行异步的传送，又可以进行同步的传送，故称为通用同步与异步接收和发送器，简称 USART。

一、8251 A 的组成和功能

1. 主要性能

(1) 可用于同步和异步传送；

(2) 同步传送：5～8 位字符；内同步或外同步；自动插入同步符号 SYNC（单同步或双同步）；

(3) 异步传送：5～8 位字符，时钟频率为通信波特率的 1 倍、16 倍或 64 倍；

(4) 可产生断点字符(Break Character)；可产生 1 位、1 $\frac{1}{2}$ 位或 2 位的终止位；

(5) 波特率：DC——19.2 K(异步)，　DC——64 K(同步)；

(6) 全双工双缓冲器的发送/接收器；

(7) 误差检测——具有奇偶、溢出和帧错误等检测电路；

(8) 与 8080 A/8085 ACPU 完全兼容；

(9) 所有输入和输出电路都与 TTL 兼容。

2. 8251 A 的方框图

8251 A 的方框图如图 14-17 所示。

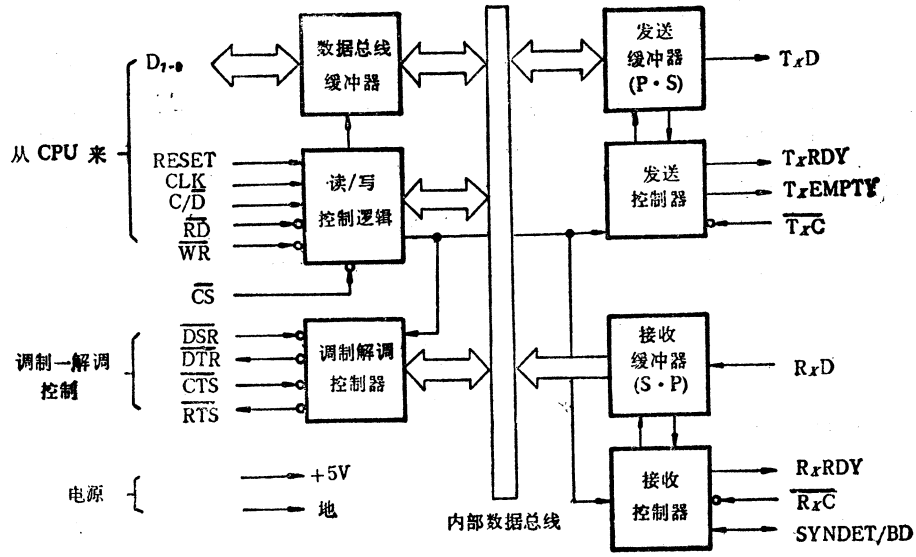


图 14-17 8251 A 结构方框图

由图可见 8251 A 是由发送器、接收器和控制电路组成的。它是专门为微型计算机系统进行数据通信而设计的。可以通过 CPU 对 8251 A 编程序而使其适用于现行任何一种串行数据的通信。

(1) 接收器

它能接受从 R_xD 端输入的串行码，并按规定把串行码转换成并行码后，存放在接收数据缓冲器中。

当 8251 A 工作于异步方式且允许接收和准备好接收数据时，它监视 R_xD 线。当无字符传送时，R_xD 线上为高电平(即所谓 Mark)；当发现 R_xD 线上出现低电平时，即认为它是起始位(即所谓 Space)，于是启动内部计数器，当计数到一个数据位宽度的一半（若时钟脉冲频率为波特率的 16 倍时，则为计数到第 8 个脉冲)时，又重新采样 R_xD 线，若其仍为低电平，则确认为起始位，而不是噪声信号。此后，每隔 16 个脉冲采样一次 R_xD 线作为输入信号，送到移位寄存器，经过移位，奇偶校验和去除终止位后就得到了转换为并行的数据，经过 8251 A 的内部数据总线传送至接收数据缓冲器，同时发出 R_xRDY 信号，通知 CPU 字符已经可用。

在同步方式中，USART 监视 R_xD 线，将一个数据位送到接收缓冲寄存器，然后用比较法误别字符，在 R_xD 线上一次一位地移动数据，每接收一位就比较一位，直至使接收寄存器的内容与含有同步字符(由程序给定)的寄存器相等为止(若规定为两个同步字符，则出现在 R_xD 线上的两个相邻字符 x 须与规定的字符相同)，则置 SYNDET 信号，表示已找到同步字符。

在找到同步字符后，利用时钟采样和移位 R_xD 线上的数据位，按规定的位数，把它送至接收数据缓冲器，同时发出 R_xRDY 信号。

(2) 发送器

发送器接收 CPU 送来的并行数据，加上起始位、奇偶校验位和终止位后，由 T_xD 输出线发送到外设。

在异步方式时，发送器加上起始位，检查并根据程序规定的检验要求(奇校验还是偶校验)加上适当的校验位，最后根据程序规定，加上 1 位、$1\frac{1}{2}$ 位或 2 位终止位。

在同步方式中，发送器在数据发送前插入一个或两个同步字符(这些都在初始化时，由程序设定)，而在数据中除了奇偶校验位外，不再插入别的位。在 USART 工作于同步发送方式，而 CPU 来不及把新的字符送给它时，USART 会自动地在 T_xD 线上插入同步字符(因为在同步方式时，字符间是不允许存在间隙的)。

不论在同步还是异步工作方式，只有当程序设置了 T_xEN(Transmitter Enable——允许发送)和 $\overline{CTS}$(Clear to Send——清除发送，这是对调制器发出的请求发送的响应信号)有效时，才能发送。

此外，发送器还能发送断点字符。断点字符是由通信线上的连续的 Space 字符组成的。它用来在全双工通信中中止发送终端。只要 8251 A 的命令寄存器的位 3 (SBRK) 为 "1"，则 USART 就始终发送断点字符。

(3) 读/写控制逻辑

读/写控制逻辑对 CPU 输出的控制信号进行译码以实现表 14-1 所示的读/写功能。

表 14-1　8251A 读写操作真值表

$\overline{CS}$	C/D	$\overline{RD}$	$\overline{WR}$	说　　明
0	0	0	1	8251 A 数据 ⟶ 数据总线
0	0	1	0	数据总线的数据 ⟶ 8251A
0	1	0	1	状态信息 ⟶ 数据总线
0	1	1	0	数据总线的命令字 ⟶ 8251 A
0	×	1	1	数据总线为三态
1	×	×	×	数据总线为三态

$\overline{WR}$ 和 $\overline{RD}$ 分别表示写入(CPU 输出给 8251 A)和读出(CPU 从 8251 A 输入)的命令。C/$\overline{D}$ 表示控制/数据，用以给 8251 A 指出在数据总线 D_7～D_0 上的信息是数据还是控制字(或状态信息)；若 C/$\overline{D}$ = 1 则表示是控制字；若 C/$\overline{D}$ = 0，则表示是数据。此端通常连到 CPU 地址总线的 A_0。$\overline{CS}$ 表示片选信号，它由 CPU 的 $\overline{IORQ}$ 及地址信号经译码后供给。

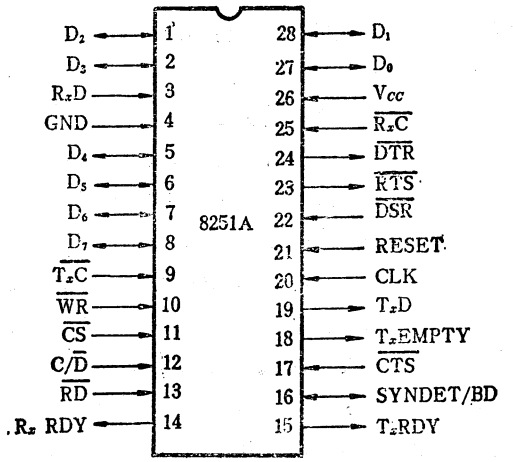


图 14-18　8251 A 的引线图

3．引线说明

8251 A 有 28 条引线，如图 14-18 所示。

它的接口信号可分为两组：一组为与 CPU 接口的信号；另一组为与外设(调制器)接口的信号。

(1) 与 CPU 的接口信号

① D_7～D_0——数据总线(8 位、三态、双向)

CPU 与 8251 A 之间的命令、数据以及状态信息都是通过这组数据总线传送的。

② CLK——时钟脉冲(TTL)

CLK 用来产生 8251 A 的内部时序。CLK 的频率在同步方式工作时，必须大于接收器和发送器输入时钟频率的 30 倍；在异步方式工作时，必须大于输入时钟频率的 4.5 倍。

在 8080A CPU 系统中，CLK 通常可接至 8224 时钟发生器的 ϕ_2 (TTL) 输出端；在 8085 CPU 系统中，CLK 通常接至 8085 的 CLK OUT 端。

③ RESET——复位

当此线为高电平时，使 8251 A 进入“空闲”状态。

④ T_xRDY(Transmitter Ready)——发送准备好信号

只有当 USART 允许发送 ($\overline{CTS}$ 为低电平，T_xEN 为高电平)，且发送命令/数据缓冲器为空时，此信号有效。它用以通知 CPU：8251 A 已准备好接收一个数据或命令。当 CPU 与 8251 A 之间用查询方式交换信息时，此信号可作为一个“联络”信号；在用中断方式交换信息时，此信号可作为 8251 A 的一个中断请求信号。当 USART 从 CPU 接收了一个字符时，T_xRDY 复位。

⑤ T_xE(Transmitter Empty)——发送器空信号

当 T_xE 有效(高电平)时，表示发送器中并行到串行转换器已空。在同步方式工作时，若 CPU 来不及输出一个新的字符，则它变为高电平，同时发送器在输出线上插入同步字符，以填补传送空隙。

⑥ R_xRDY(Receiver Ready)——接收器准备好信号

如果命令寄存器的 R_xE(Receive Enable)置位时，当 8251 A 已经从它的串行输入端接收了一个字符，并可以传送到 CPU 时，此信号有效。在查询方式时，此信号可作为一个“联络”信号；在中断方式时，可作为一个中断请求信号。当 CPU 读取一个字符后，R_xRDY 复位。

⑦ SYNDET(Synchonous Detect)——同步检测信号

此信号仅用于同步方式。它用于输出还是输入取决于初始化程序对 8251 A 的设定为内同步还是外同步。在 RESET 有效时，此信号复位。

当工作于内同步方式(输出)时，8251 A 检测到所要求的同步字符，SYNDET 为高电平，输出以指示 USART 已经达到同步。若 8251 A 由程序规定为双同步字符时，此信号在第二个同步字符的最后一位的中间变为高电平。当 CPU 执行一次读状态操作时，SYNDET 复位。

在外同步方式(输入)工作时，从这个输入端输入的一个上升沿，使 8251 A 在下一个 $\overline{R_xC}$ 的下降沿开始收集字符。SYNDET 输入高电平至少应维持一个 $\overline{R_xC}$ 周期，直到 $\overline{R_xC}$ 出现下一个下降沿。

$\overline{WR}$、$\overline{RD}$、$\overline{CS}$、C/$\overline{D}$ 等信号的含义如前所述，不再重复。

(2) 与装置的接口信号

① $\overline{DTR}$ (Data Terminal Ready)——数据终端准备好(输出，低电平有效)

它是一个通用的输出信号，可由命令字的位 1 置“1”，而变为有效，用以表示 CPU 准备就绪。

② $\overline{DSR}$ (Data Set Ready)——数据装置准备好(输入，低电平有效)

它是一个通用的输入信号，用以表示调制解调器或外设的数据已准备好。CPU 可通过读入状态操作，在状态寄存器的位 7 检测此信号。

③ $\overline{RTS}$(Request To Send)——请求发送(输出，低电平有效)

此信号等效于 $\overline{DTR}$，它用于通知调制器：CPU 已准备好发送。它可由命令字的位 5 置“1”而变为有效(低电平有效)

④ $\overline{CTS}$(Clear To Send)——清除发送信号(输入，低电平有效)

它是调制解调器或其它外设送到调制解调器控制器的信号。当其有效时，表示允许 USART 发送数据。

⑤ $\overline{R_xC}$——(Receiver Clock)接收器时钟

这个时钟控制 USART 接收字符的速度。

在同步方式时，$\overline{R_xC}$ 等于波特率，由调制解调器供给。

在异步方式时，$\overline{R_xC}$ 是波特率的 1 倍、16 倍或 64 倍。由方式控制字预先选择。USART 在 $\overline{R_xC}$ 的上升沿采样数据。

⑥ R_xD(Receiver Data)——接收数据

USART 在这条线上串行地接收字符，并转换为并行的字符。高电平表示 Mark 即“1”。

⑦ $\overline{T_xC}$(Transmitter Clock)——发送器时钟

这个时钟控制 USART 发送字符的速度。在时钟速度和波特率之间的关系同 $\overline{R_xC}$。数据在 $\overline{T_xC}$ 的下降沿由 USART 移位输出。

⑧ T_xD(Transmitter Data)——发送数据

由 CPU 送来的并行的字符，在这条线上被串行地发送。高电平表示 Mark 即“1”。

二、8251 A 的编程

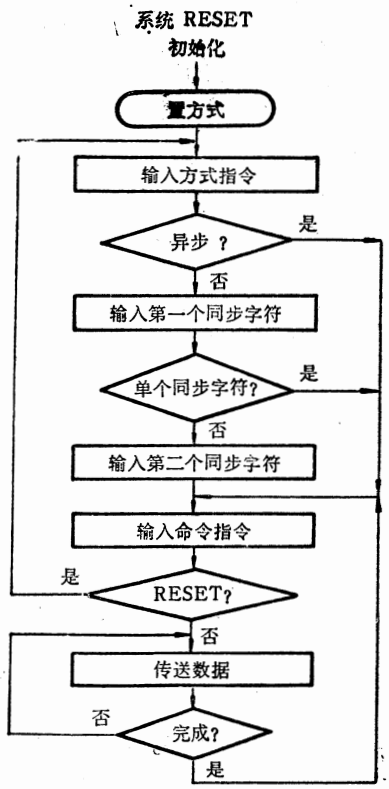


图 14-19　8251 A 初始化编程的流程图

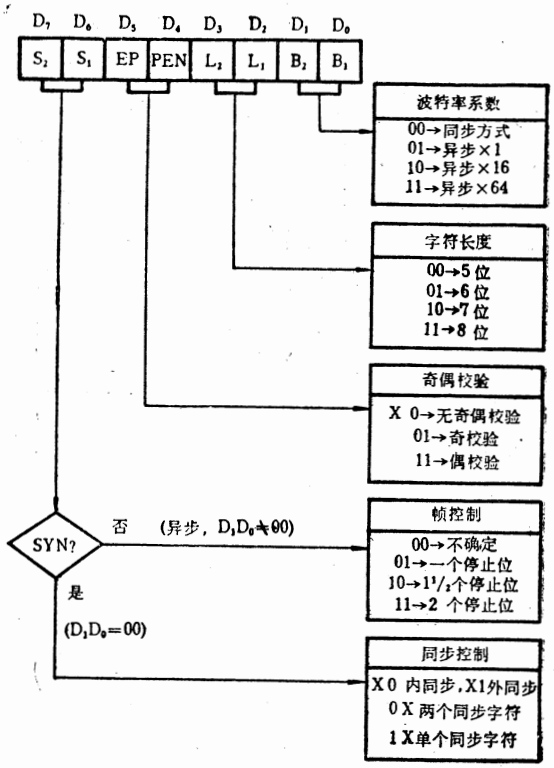


图 14-20　传送方式选择字格式

8251A 是一个可编程序的通信接口。它在系统 RESET 之后开始发送或接收数据之前，必须先由 CPU 对它进行初始化编程，以确定它的工作方式(同步或异步)、传送的波特率、字符格式及终止位位数等。

8251 A 初始化编程的流程图如图 14-19 所示。

1．方式选择控制字

其格式如图 14-20 所示。

此方式选择控制字可分为 4 组，每组 2 位。其中：

(1) $D_1D_0 = 00$ 同步方式；

$D_1D_0 \neq 00$ 异步方式，且 D_1D_0 的三种组合用于选择输入时钟频率与波特率之间的系数。

(2) D_3D_2 用作确定字符的位数。

(3) D_5D_4 用作确定奇偶校验的性质。

(4) $D_7D_6 = 00$ 用于同步控制时确定内同步还是外同步，以及同步字符的个数。

$D_7D_6 \neq 00$ 用于异步控制时规定终止位的位数。

在同步方式时，紧跟在方式选择控制字后面的是由程序输入的同步字符。它是用与方式选择控制字类似的方法由 CPU 输给 USART 的。

2．命令控制字

在输入同步字符后，或在异步方式时，在方式选择控制字后面应由 CPU 输给命令控制字，其格式如图 14-21 所示。

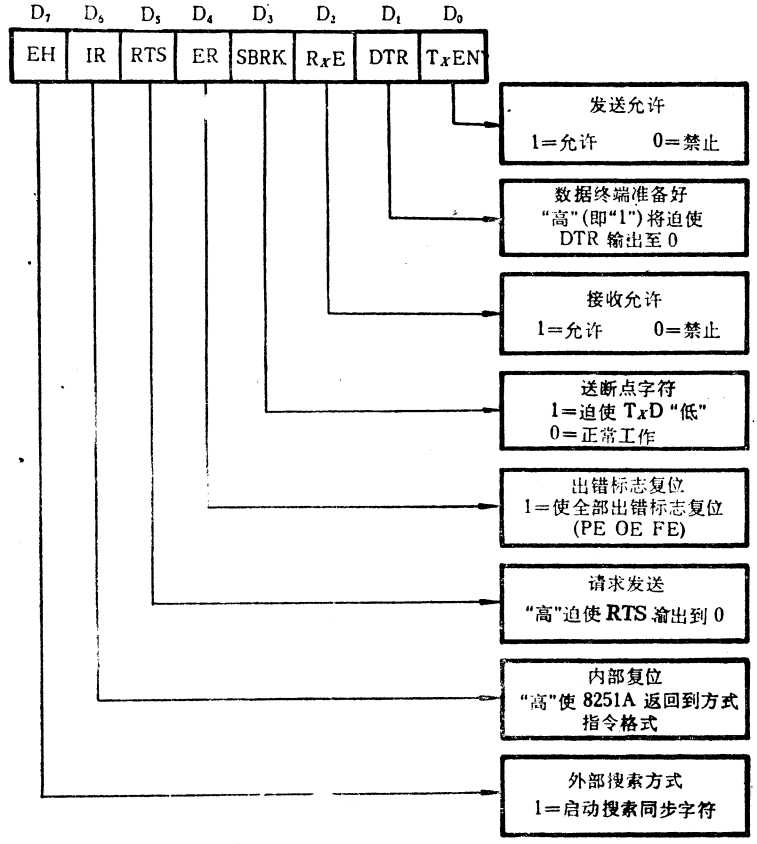


图 14-21 8251 A 的命令指令格式

命令控制字直接使 8251 A 处于规定的工作状态，以准备发送或接收数据。

8251 A 中还有状态寄存器，CPU 可通过 I/O 读操作，将 8251 A 的状态字读入 CPU，用以控制 CPU 与 8251 A 之间的数据交换。

读状态字时，C/$\overline{D}$ 端为“1”。状态字的格式如图 14-22 所示。

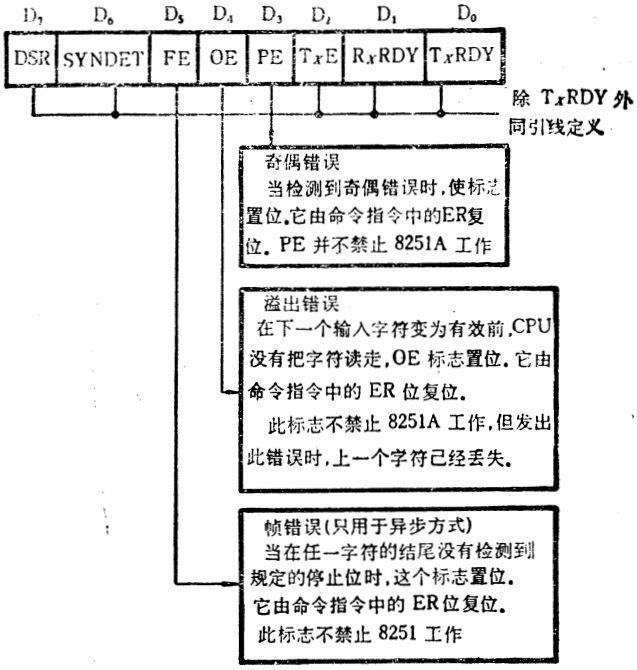


图 14-22　8251 A 状态寄存器格式

应当指出，状态寄存器的 T_xRDY 与输出端 T_xRDY 是有区别的。前者只要数据缓冲器一空就置位，而后者只能当条件：

数据缓冲器空，$\overline{CTS}\cdot T_xEN$ 成立时，才置位。

三、8251 A 应用举例

8251 A 可用来作为 CPU 与外设或调制器之间的接口，如图 14-23 所示。

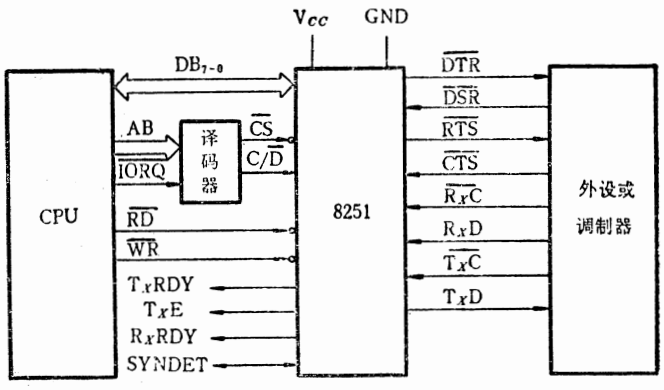


图 14-23　8251 A 作为 CPU 与外设的接口

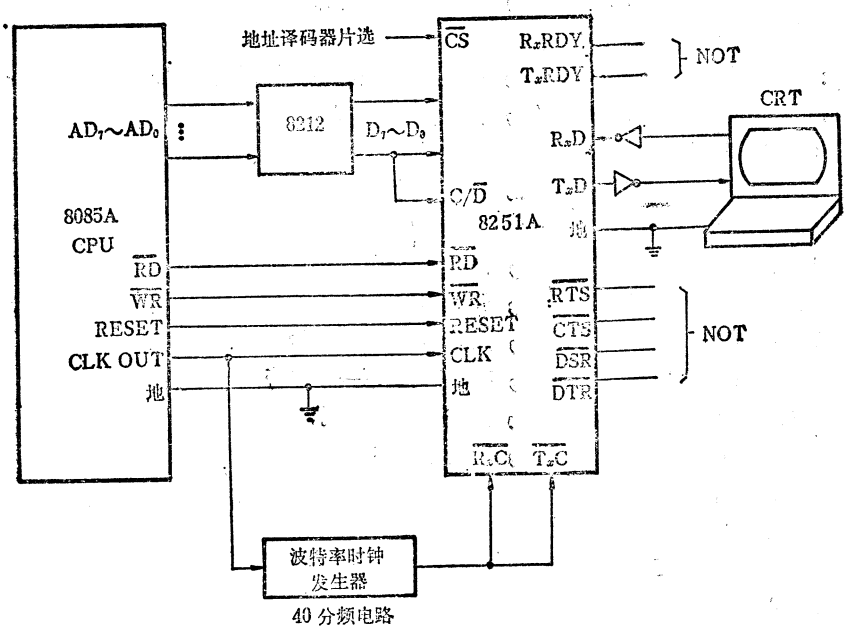


图 14-24　8251 A 与 CRT 的接口电路

在 MCS-85 微型计算机系统中，8251 A 用作 CPU 与 CRT 终端的接口，如图 14-24 所示。

由图可知，将系统工作的基准时钟 40 分频后，与 8251 A 的接收器时钟($\overline{R_xC}$)和发送器时钟($\overline{T_xC}$)相连接，而 8251 A 的数据发送端(T_xD)和数据接收端(R_xD)分别经隔离及电平匹配电路与 CRT 的相应信号端相连接。由于 8085 A 的基准时钟频率为 3.072 MHz，因此经 40 分频后的信号频率为 76.8 KC。

设初始化条件为：

数据传输方式：异步

停 止 位 数：1 位

奇 偶 校 验：偶校验

字 符 长 度：8 位

波特速率因子：64

根据 8251 A 异步工作方式选择控制字和命令控制字的格式，它们应为：

(1) 方式选择控制字为 01111111 B(7 FH)；

(2) 命令控制字为 00100111 B(27 H)。

按上述定义时，CRT 数据传输波特速率为 1200。若将波特速率因子改为 16 时，CRT 数据传输波特速率将是 4800，此时 8251 A 的方式选择控制字应是 7 EH。

若 8251 A 数据端口地址为 30 H，命令/状态端口地址为 31 H，则在 CRT 屏幕上不断逐行地显示 HELLO 字样的显示程序如下：

```
标号    指令(助记符)          注释
        ORG  2000H
        LXI  SP,   20C2H
        MVI  A,    7FH     ；设置 8251A 的方式选择控制字
        OUT  31H
```

```
        MVI   A,      27H      ; 设置 8251A 的命令控制字
        OUT   31H
 LOOP:LXI   H,      STAR     ; 设置显示字符存放单元的首地址
NEXT:MOV   A,      M        ; 将字符送 CRT
        OUT   30H
        MVI   D,      10H      ; 设置延时
        CALL  DELAY
        INX   H                ; 内存地址增量
        MVI   B,      05H      ; 设置字符数
        DCR   B                ; 显示字符送完了吗?
        JNZ   NEXT             ; 否,继续送下一个字符
        JMP   LOOP             ; 是,返回重复显示
DELAY:DCX   D                  ; 延时子程序
        MOV   A,      D
        ORA   E
        JNZ   DELAY
        RET
 STAR:DB    48H, 45H, 4CH, 4CH, 4FH
```

§14-3 Z 80 串行 I/O 接口电路——Z 80 SIO

Z 80 SIO(Serial Input/Output) 电路是可编程的双通道的通信接口电路。它是专门为微型计算机系统进行串行数据通信而设计的，能工作于异步的、同步的或 SDLC(Synchronous Data Link Control——同步数据链路控制)、或 HDLC(High-level Data Link Control——高级数据链路控制)方式中。

一、Z80 SIO 的组成

1. 组成

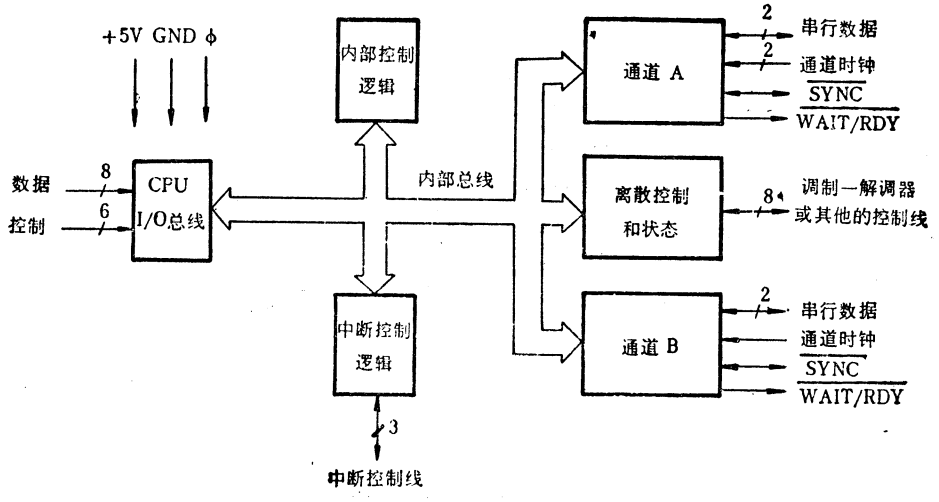


图 14-25 SIO 方框图

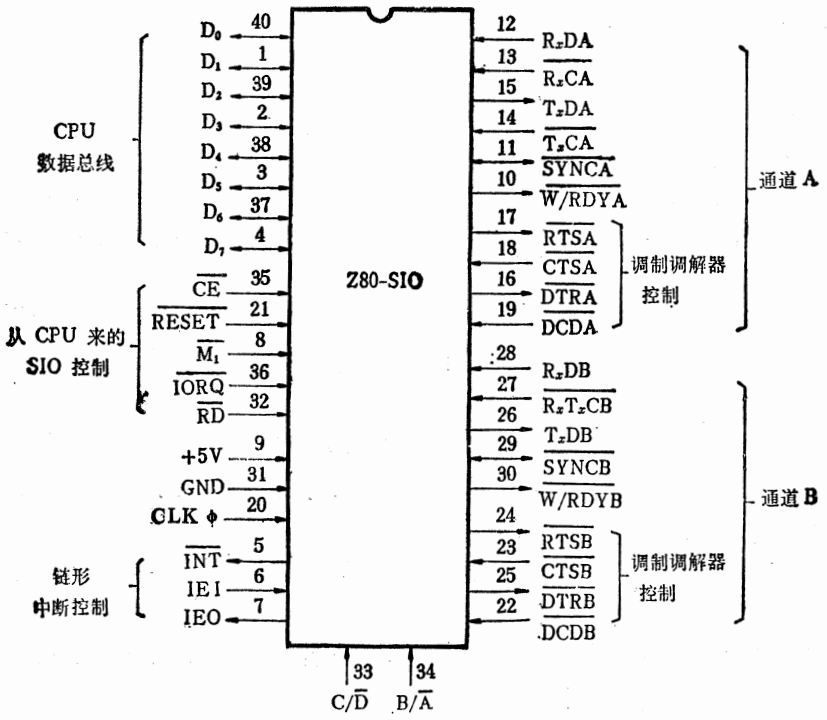


图 14-26 SIO 引线图

Z80 SIO 的内部结构框图及引线图如图 14-25 及图 14-26 所示。

Z80 SIO 包括：Z80 CPU 总线接口、内部控制逻辑、中断控制逻辑和两个全双工通道。每个通道包含读和写寄存器组、发送逻辑和接收逻辑、离散的控制/状态逻辑。SIO 通道的控制部分有 8 个 8 位写寄存器，供 CPU 向 SIO 写入命令字，以选择 SIO 各通道的工作方式。此外，还有 3 个读寄存器，用来向 CPU 提供各通道的工作状态，以便 CPU 检验。SIO 中断矢量字写入 B 通道 RR_2 内，此中断矢量是 A、B 通道共用的。

2．Z80 SIO 引线功能

顺便指出，Z80 SIO有Z80 SIO/0、SIO/1、SIO/2 等型号，它们的个别引线有所区别，使用时应加以注意。以下介绍的是 Z80 SIO/0，其 $\overline{T_xCB}$ 和 $\overline{R_xCB}$ 共用一条引线 $\overline{R_xT_xCB}$，这主要是由于封装的限制而采取的措施。如果对 $\overline{R_xCB}$ 和 $\overline{T_xCB}$ 要求不同的时钟频率时，则应选用 Z80 SIO/1 型。

(1) $D_7 \sim D_0$——系统数据总线(双向，三态)，用于写入数据、命令，读出数据、状态。

(2) $B/\overline{A}$——通道 B 或 A 选择(输入，高电平选择通道 B，低电平选择通道 A)。

(3) $C/\overline{D}$——控制或数据选择(输入，高电平是控制，低电平是数据)。

(4) $\overline{CE}$——片选(输入，低电平有效)。

(5) $\overline{M_1}$——从 Z80 CPU 来的机器周期 M_1 的信号(输入，低电平有效)。

(6) $\overline{IORQ}$——从 Z80 CPU 来的输入/输出请求(输入，低电平有效)。

(7) $\overline{RD}$——从 Z80 CPU 来的读命令(输入，低电平有效)。

(8) ϕ——系统时钟(输入)。

(9) $\overline{RESET}$——复位(输入，低电平有效)，清除所有寄存器内容，封锁通道中的发送和接

收逻辑以及中断逻辑。强行使 T_xDA 和 T_xDB 为“1”；调制-解调器控制信号 $\overline{RTS}$、$\overline{DTR}$ 为高电平所有寄存器全部清除，封锁全部中断；数据总线置为高阻抗状态。

(10) $\overline{INT}$——中断请求(输出，漏极开路，低电平有效)。

(11) IEI——中断开放输入(输入，高电平有效)。

(12) IEO——中断开放输出(输出，高电平有效)。IEI 和 IEO 用来构成优先中断控制链。

(13) $\overline{W/RDYA}$ 和 $\overline{W/RDYB}$ (通道 A、B 各有一引线)——在编程中，可选择连到 CPU $\overline{WAIT}$ 或连到 DMA $\overline{READY}$，其输出低电平使 CPU 或 DMA 等待，然后，再进行对 SIO 的读或写，以达到同步的目的。

(14) $\overline{CTSA}$、$\overline{CTSB}$——清送电路(通道 A、B 各有一引线，输入，低电平有效)。

它们为外部设备提供控制发送器的“接通/断开”开关，其输出低电平将使发送逻辑开放，但通过编程也可对其关闭，而作为其它的控制线。

(15) $\overline{DCDA}$、$\overline{DCDB}$——数据传送检测(通道 A、B 各有一条引线，输入，低电平有效)。

它们为外设提供控制接收器的“接通/断开”开关。其输出低电平将使接收逻辑开放，但通过编程也可对其关闭，而用作其它控制线。

(16) R_xDA、R_xDB——接收串行数据(通道 A、B 各有一条引线，输入，高电平有效)。数据是在 R_xC (本通道接收时钟)前沿(上升沿)接收的。

(17) T_xDA、T_xDB——发送串行数据(通道 A、B 各有一条引线，输出，高电平有效)。数据在 T_xC (本通道发送时钟)后沿(下降沿)发送出去。

(18) $*\overline{R_xCA}$、$\overline{R_xCB}$——接收器时钟(输入，低电平有效)。在异步工作方式中，时钟可为 ×1、×16、×32 或 ×64 的数据率。

(19) $*\overline{T_xCA}$、$\overline{T_xCB}$——发送器时钟(输入，高电平有效)，可以是×1、×16、×32 或×64 的波特率，但是倍乘数要与接收器相同。T_xC 和 R_xC 输入由施米特触发器缓冲，以满足较慢上升和下降时间的要求。

(20) $\overline{RTSA}$、$\overline{RTSB}$——请求发送(通道 A、B 各有一条引线，输出，低电平有效)，当 $\overline{RTS}$ 位置位时，$\overline{RTS}$ 引线为低电平。在异步工作方式下，当 $\overline{RTS}$ 位复位时，只有在发送器空了以后，$\overline{RTS}$ 引线变为高电平。在同步工作方式时，$\overline{RTS}$ 严格按 $\overline{RTS}$ 位的状态输出。

(21) $\overline{DTRA}$、$\overline{DTRB}$ 数据终端准备(通道 A、B 各有一引线，输出，低电平有效)。引线输出跟随由 DTR 位编制的状态。

(22) $\overline{SYNCA}$、$\overline{SYNCB}$ 外部字符同步(通道 A、B 各有一引线，输入/输出，低电平有效)。如果选择外同步工作方式，字符的组合将从下一个 $\overline{R_xC}$ 的上升沿开始。如果选择内同步，则引线作为输出，但只是在时钟周期某部分，即同步字符被识别的区间内才有效。同步条件不闩锁，每次识别出同步方式时，此引线将为有效，而不管字符的边界。在异步工作方式时，这些引线只输入给状态寄存器 0 中的搜索/同步(Hunt/Sync)位，还可作为所希望的输入操作。

二、Z 80 SIO 的操作说明

Z 80 SIO 的功能特性是由系统程序设定的。系统软件发布一系列命令字以预置所需要的基本操作方式，并发布其它一些命令用来规定所选工作方式的各种状态，也就是终止位、位/字符、同步字符等。Z80 SIO 命令结构设计成能运用强有力的 Z80 成组传送 I/O 指令，以便简化程序编制。

Z80 SIO 两个通道的每一个都有命令寄存器，它必须在操作之前通过系统软件进行程序设置。通道选择输入端(B/$\overline{A}$)和控制/数据输入端(C/$\overline{D}$)通常由 Z80 CPU 的地址总线来控制(见表14-2)。

表 14-2　Z80 SIO 的通道选择

C/$\overline{D}$		
0	0	通道A数据
0	1	通道B数据
1	0	通道A命令/状态
1	1	通道B命令/状态

Z80 S O 的每个通道包含有 8 个寄存器，可通过编制程序，选择各通道的工作方式。

关于 Z80 S O 控制字格式和状态字格式的详细使用说明，可参阅 Zilog 公司的有关技术手册。

第十五章　微型计算机的输入/输出设备及其接口技术

微型计算机是通过输入输出设备与外界进行通信的。由于微型计算机本身成本低、体积小，而在许多场合下，输入输出设备的价格往往超过了微型计算机本身的价格，因此，在组成一个微型计算机系统时，选择合适的输入输出设备是十分重要的。

本章仅介绍若干常用的输入输出设备的性能、工作原理及其与 CPU 的接口技术。

§15-1　简单开关及其与CPU 的接口

在微型计算机应用系统中，常用控制面板上的各种开关给微型机置数。这里需要解决的接口技术主要是去抖动问题。众所周知，任何电气-机械开关的断开与闭合，都要经过几毫秒的抖动之后才能真正接触。从开关最初被合上到接触稳定，通常需经过 10～20ms。开关断开时亦发生同样的问题。抖动的前沿和后沿如图 15-1 所示。解决这个问题的方法之一是设置去抖动线路。

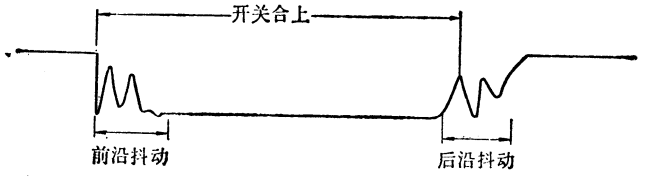


图 15-1　抖动波形图

如果用的是单刀双掷开关，这种开关有一个公共点，一个常开触点(NO)和一个常闭触点(NC)。它们不是处于常闭状态就是处于常开状态。这种机械式的单刀双掷开关可利用单稳电路，软件延时或由两个与非门组成的触发器来消除抖动。

现设置一个去抖动线路，再和 I/O 端口电路 8212 相连接，则其线路如图 15-2 所示。

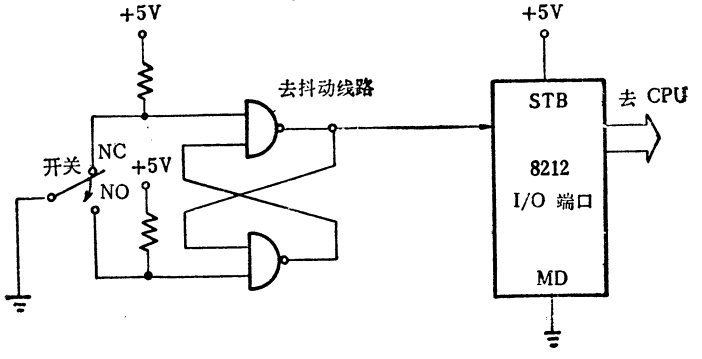


图 15-2　开关与 CPU 的接口

图中，去抖动线路是由两个“与非”门组成的 RS 触发器。其输出接到 8212 I/O 端口电路中的一位。8212 用作一个编有地址的缓冲器，它的输出端接到 CPU 的系统总线上。去抖动电路保证开关的每次动作只产生一次跳变，即当开关第一次接触时，电路就稳定成一定的状态，使开关因抖动而产生的重复脉冲不至于影响该电路的状态。

如图 15-2 所示开关可以给微型机输入有用的信息，这里有两种情况：

(1) 监控开关的输出，当常开触点接通时，把某存贮单元清零。

(2) 监控开关的输出，当开关状态变化时，把某存贮单元清零。

下面分别讨论它们工作的程序：

1．等待开关接通

这里，把 8212 看成是 CPU 输入端口，其通道的编码为 PORT，将去抖动线路的输出接到 8212 的某一位，比如接到第 4 位，即位 3。我们选择存贮单元 2040H 作为标志单元。如图 15-2 所示，NO 触点断开，此时，当输给 8212 位 3 是“1”时，存贮单元 2040H 中存放的是“1”。CPU 不断检查开关位置。如果常开触点 NO 接通，则 8212 位 3 变成“0”。这时，作为控制标志的存贮单元就清零。

这一段程序的流程见图 15-3。

检查开关状态的源程序如下：

```
        Z80                          8080A
        LD   HL, 2040H               LXI   H, 2040H ;标志单元置“1”
        LD   (HL), 01                MVI   M, 01
WAITC:  IN   A, (PORT)       WAITC:  IN    PORT     ;读入开关状态
        AND  00001000B               ANI   00001000B ;检查开关状态是否接通（指
                                                     ;NO是否接地）
        JR   NZ, WAITC               JNZ   WAITC    ;否，循环等待
        DEC  (HL)                    DCR   M        ;是，标志单元置“0”
        HALT                         HLT            ;暂停
```

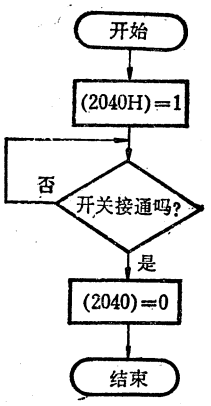


图 15-3　检查开关状态的流程图

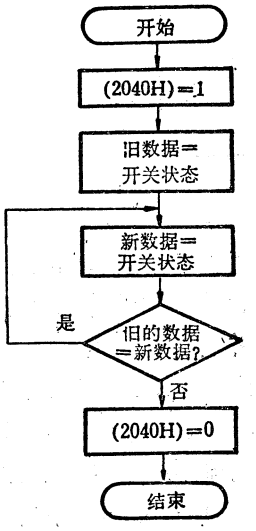


图 15-4　检查开关状态变化的流程图

应当指出，可将存贮单元置“0”的方法很多，这里是用DCR(DEC)指令来清零的。

2．等待开关状态变化

这时，CPU将不断检查开关状态。如果状态没有变化，则存贮单元2040H一直保持为“1”；如果状态发生变化，则存贮单元2040H清零。图15-4示出其程序流程图。

检查开关状态变化的源程序如下：

```
        Z80                              8080A
        LD    HL,     2040H      LXI   H,     2040H
        LD    (HL),   01         MVI   M,     01      ;标志单元置“1”
        IN    A,      (PORT)     IN    PORT            ;取旧开关状态
        AND   00001000B          ANI   00001000B
        LD    B,      A          MOV   B,     A
SRCH:   IN    A,      (PORT)  SRCH:IN    PORT          ;取新开关状态
        AND   00001000B          ANI   00001000B
        CP    B                  CMP   B               ;新状态等于旧状态吗?
        JR    Z,      SRCH       JZ    SRCH            ;是,等待
        DEC   (HL)               DCR   M               ;否,标志单元清零
        HALT                     HLT                   ;暂停
```

实际上，用SUB B或XRA B指令都可以取代程序中的CMP B (CP B)指令。然而，这两条指令都将改变累加器的状态。

§15-2 七段发光二极管显示器及其与CPU的接口

七段发光二极管显示器(Light Emitting Diode 简称LED)应用非常普遍，从袖珍计算器到复杂的仪器都要用到它。它由七段发光二极管组成。如图15-5所示。

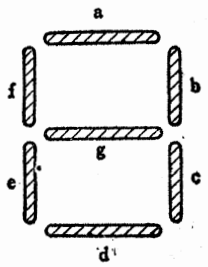


图15-5 七段发光二极管显示器

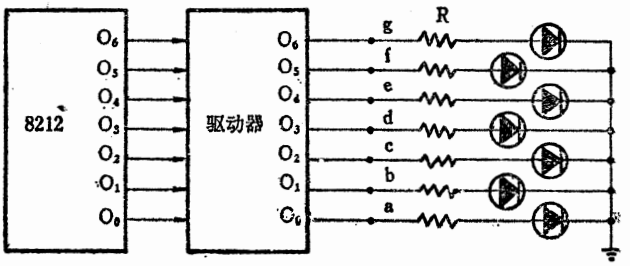


图15-6 七段发光二极管显示器的接口电路

它既可以用作显示十进制数字，也可用作显示部分英文字母。

发光二极管是一种当外加电压(阳极电压比阴极电压为正)超过额定电压时发生击穿，从而流过电流，并发出可见光的器件。一般采用多个发光二极管来组成7段或8段发光数码管显示器。当段组合发亮时，便显示某一数码或字符。

图15-6表示七段发光二极管显示器的接口电路。

8212的7个输出O_0～O_6分别接到从a到g的7个发光二极管。限流电阻R应使流过LED的最大电流小于50mA，平均电流小于10mA。大多数I/O端口不能直接驱动LED，故必须用驱动器或晶体管来驱动。

表 15-1　七段 LED 表示的十进制数字

十进制数字	七位数字的状态 D_6 g	D_5 f	D_4 e	D_3 d	D_2 c	D_1 b	D_0 a	共阴极原码表示 (H)	共阳极反码表示 (H)	存贮单元地址
0	0	1	1	1	1	1	1	3F	40	SSEG
1	0	0	0	0	1	1	0	06	79	SSEG+1
2	1	0	1	1	0	1	1	5B	24	SSEG+2
3	1	0	0	1	1	1	1	4F	30	SSEG+3
4	1	1	0	0	1	1	0	66	19	SSEG+4
5	1	1	0	1	1	0	1	6D	12	SSEG+5
6	1	1	1	1	1	0	1	7D	02	SSEG+6
7	0	0	0	0	1	1	1	07	78	SSEG+7
8	1	1	1	1	1	1	1	7F	00	SSEG+8
9	1	1	0	0	1	1	1	67	18	SSEG+9

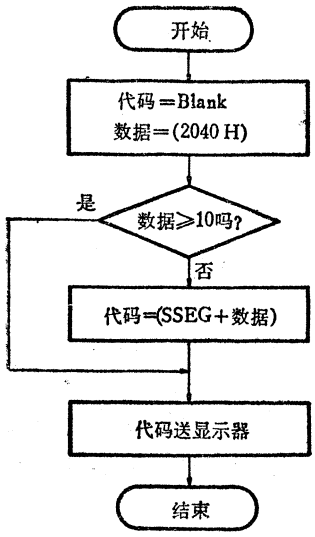


图 15-7　数据在七段显示器上显示的流程图

由于采用了发光二极管共阴极的接法，故驱动器输出为“1”时，发光二极管点亮。当然，也可以把发光二极管阳极共同连接到 +5 伏电源。这样，当驱动器输出为“0”时，发光二极管点亮。

共阴极接法的七段 LED，可用一个 8 位二进制码来表示一个十进制数字。见表 15-1。

我们先把这十个码编成“表格”，放在存贮器的十个单元中，以备程序查找。

表中，SSEG 是存放在存贮器中 7 段发光二极管显示 “表格”的起始地址。当 CPU 查“表”时，要显示的数字的存贮单元地址正好等于(SSEG + 数字值)。如要显示数字 5，则存贮器表格地址为 SSEG + 5。在编写程序时，假定 7 段显示的二进制数据存放在存贮单元 2040H 中。如果取出后，这个数据小于 10，则被显示。如果取出的数据大于 10(如 128)，则显示出一个“空白”(blank)标记。这段程序的流程图如图 15-7 所示。

七段发光二极管显示数据的程序如下：

```
        Z80                          8080A
        LD    B,    blank      MVI  B,  blank  ;取 blank 码
        LD    A,    (2040H)    LDA  2040H     ;取要显示的数据
        CP    10               CPI  10        ;数据≥10?
        JR    NC,   DSPLY      JNC  DSPLY     ;是,显示 blank 标志
        LD    DE,   SSEG       LXI  D,  SSEG  ;否,七段显示“表格”的起始地址
        LD    H,    00         MVI  H,  00
        LD    L,    A          MOV  L,  A
        ADD   HL,   DE         DAD  D         ;准备查“表格”
        LD    B,    (HL)       MOV  B,  M     ;从表格中取出七段显示码
DSPLY:  LD    A,    B    DSPLY: MOV  A,  B
        OUT   (PORT1), A       OUT  PORT1     ;送字模端口显示
        HALT                   HLT            ;暂停
```

程序中，PORT1 是指 8212 端口的设备码。对共阴极显示器，空白标志 blank 一般用“00”码表示；对共阳极显示器，它是“FF”。

LED 也可用来显示某些英文字母（如正体小写 b、d、h 等等）。读者可按照同样原理，写出 LED 显示英文字母的程序。

§15-3 键盘及其与 CPU 的接口

在微型计算机系统中，键盘是最常用的一种输入设备。它有两种类型：全编码键盘和非编码键盘。

全编码键盘能自动提供对应于被按的键的 ASCII 码，并能同时产生一选通脉冲通知微处理器。此外，它一般还具有去抖动和多键串键（Roll-Over）保护电路。这种键盘的优点是使用方便，但需用较多的硬件，故价格昂贵，一般用于复杂的键盘，即包含有64键或更多键的键盘。

非编码键盘恰如一组开关，仅简单地提供行和列的矩阵。其全部工作包括按键的识别、按键代码的产生、防串键和去抖动等问题，都靠程序去实现。因此，它所需要的硬件较少，价格也便宜。

键盘输入信息的过程如下：

1．CPU 检查是否有键按下（通常键都以矩阵形式排列）；

2．查出按下的是哪一个键；

3．把此键所代表的信息翻译成计算机所能识别的代码，如 ASCII 码或其它预先约定的编码。

在这过程中，由于按键与开关相似，故也存在着抖动问题。因此，应采取措施在键按下稳定后再查键的信息。

在前面讨论开关接口时，曾介绍过可用RS 触发器，也可以用 RC 滤波器去抖动。这些都是在键的数量较少的情况下采用硬件去抖动的技术。在键数较多（如 16 个以上）时，常用软件延时程序去抖动。一般经过 15～20 ms 延迟，待键状态稳定后再去读取键的信息。

在按键时，用户可能因偶尔同时按下一个以上的键而造成编码出错，这称之为串键。此时需要通过硬件线路或软件进行处理。例如，采用一种硬件线路，使得有一个键按下时产生一个脉冲信号，此信号经放大送入电磁线圈产生磁力去阻止其它键的按下，直到第一键被释放或它的信息被读入为止。又如，可用程序查询的方法扫描键盘，如果只有一个键按下，则进行编码；如果同时有几个键按下，则不进行编码，或者只对其中一个键的信息进行编码（通常取最后一个释放的键），而其它按下的键均无效。

第二步、第三步可以用硬件也可以用软件完成。如主要用硬件完成，则称为编码键盘；如主要用软件完成，则称为非编码键盘。

下面分别加以说明。

一、非编码键盘

非编码键盘通过程序查询来识别按键并产生相应的键识别码。

键的识别是指 CPU 不仅能识别出键是否已经按下，而且还要识别出哪一只键已经按下。

如果按键为数不多，且每个按键分别接到输入端口的一位，则此时键盘接口与一组开关接

口是一样的。这种结构仅适用于小键盘。

按键超过 8 个的键盘，可以把按键以矩阵的形式连接起来，以减少所要求的输入线的条数。如图 15-8 所示，每一按键表示一列和一行间的一条可能的连接线。例如键 4 能使行线 1 和列线 1 接通。

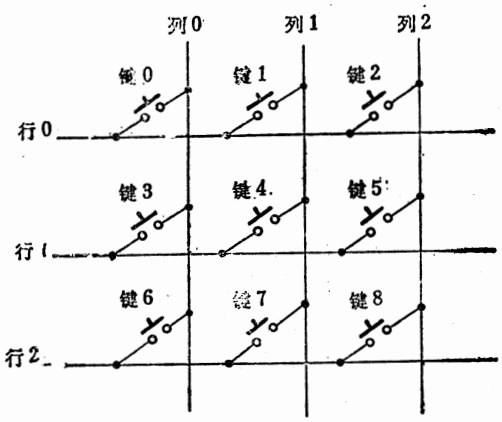


图 15-8　键盘矩阵结构

键盘矩阵要求 n＋m 条外接线，n 是行数，m 是列数。相比之下，如每个按键是单独的，则需要 n×m 条外接线。

我们以 3×3＝9 个键盘为例(见图 15-9)，来说明如何用软件来解决键盘接口技术问题。这个键盘矩阵分成三行、三列。如果第 4 号键按下，则行 1 线和列 1 线接通。若输给行 1 线为“0”电平，则列 1 线上将输出“0”电平。

本例采用键盘扫描方法来识别键的号码。扫描时，先把行 0 接“0”电平，然后检查列线。如果有一条线为“0”电平，则表示该行 0 上有一个键已经按下，因而使行和列接通。然后，确定哪一条列线接地，即输入端口的哪一位是“0”电平，便可确定是哪一个按键按下了。如果没有一条列线接地，那末再把行 1 接“0”电平，并检查列线，如此继续进行行扫描，直到最后一行为止。

如果要很快地检查是否有任何一个键已经按下(不管其键号是多少)，可以把所有各行同时全部接“0”电平，然后，检查其列线上是否有“0”电平。

键盘扫描法需要将行线接到一个输出端口和将列线接到一个输入端口，如图 15-9 所示

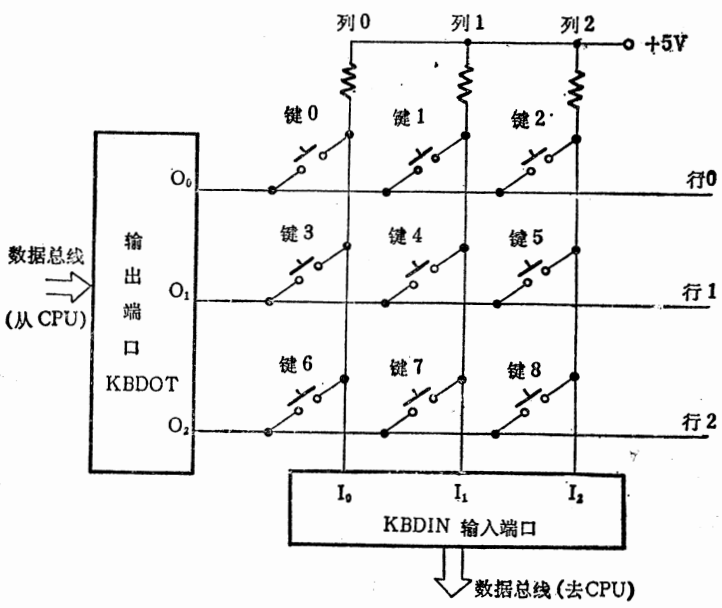


图 15-9　键盘扫描接口电路

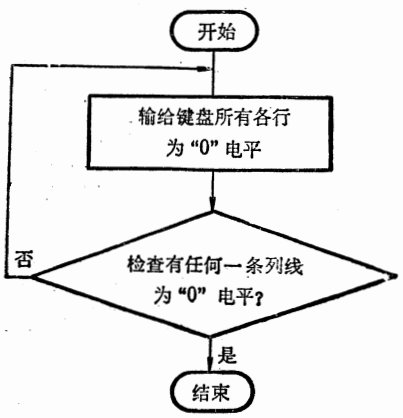


图 15-10　等待按键闭合程序的流程图

我们仍以 3×3 矩阵为例。CPU 使输出端口的某一位为 0，而其余各位为 1，就可以使某一行线接“0”电平。CPU通过检查输入端口的数据(列的状态)来确定哪号键已经按下。具体步骤如下：

第一，识别是否有键按下(即等待是否有按键按下)

首先，用输出指令使所有各行同时为“0”电平，即使输出端口的 O_0、O_1 和 O_2 三位都输出“0”电平。然后，从输入端口读入数据。只要有任何一条列线为“0”电平，便可知道已有一个键按下。如果没有一条列线是“0”电平(即列线输出为全 1)，则表示无按键按下，程序就在循环中等待。这个程序的流程如图 15-10 所示。

这段程序如下：

```
        Z80                          8080A
WAITK:  LD    A,  11111000B  WAITK:  MVI  A,11111000B
        OUT   (KBDOT),  A            OUT  KBDOT      ;使所有各行为“0”电平
        IN    A,  (KBDIN)            IN   KBDIN      ;读入各列线数据
        AND   00000111B              ANI  00000111B  ;屏蔽列线中无用位
        CP    00000111B              CPI  00000111B  ;是否有任一列线接地
        JR    Z,  WAITK              JZ   WAITK      ;否，等待按键
        HALT                         HLT             ;是，完成
```

程序中 KBDOT 和 KBDIN 分别是输出端口和输入端口的设备号。

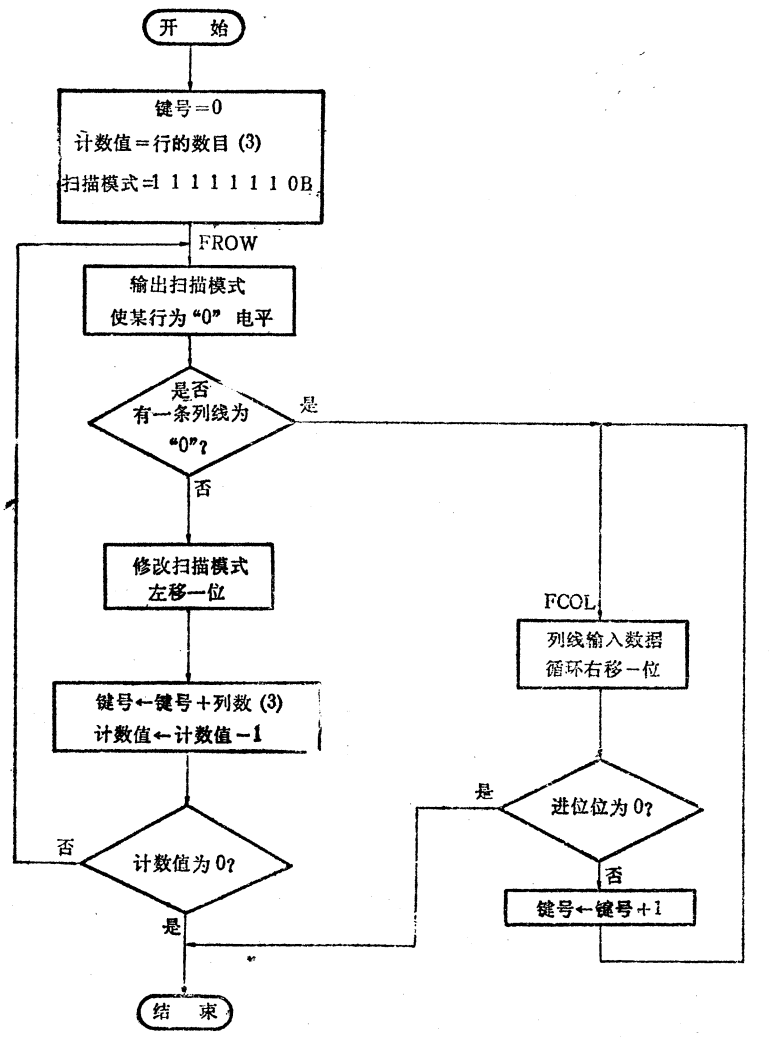


图 15-11 判定按键号码程序的流程图

第二，判定按键的位置(即判定已按下的按键的键号)

这段程序的流程见图 15-11 所示。

先令键号数寄存器置零，置计数值为行的数目(本例为 3)，并置扫描的模式。接着使行 0 线为"0"电平，其它各条线为"1"电平。然后，检查是否有任何一条列线为"0"电平。若无，则改变扫描模式，即把它左移一位，使行 1 线接"0"电平，同时使键号数增 3，即从第 4 个键开始检查；计数值减 1，一直检查到计数值为零；若有一条列线为"0"电平，则把列线数据输入循环右移一位(由于设最低位相当于列线 0 的状态)。如果这一位(进位位)为"0"电平，则表示为 0 号键按下，否则将键号增 1，重复循环右移。由于已经知道该行上已有一个键按下，故在各条列线上一定可以检查出某一列线为"0"电平。

根据图 15-11 所示的流程图可写出如下程序：

```
        Z80                          8080A
        LD    B, 00                  MVI   B, 00        ;键号=0
        LD    C, 11111110B           MVI   C, 11111110B ;置初始扫描模式，使行0为
                                                        ;"0"电平
        LD    D, 03                  MVI   D, 03        ;计数值=行数
FROW:   LD    A, C           FROW:   MOV   A, C
        OUT   (KBDOT),A              OUT   KBDOT        ;扫描一行
        RLCA                         RLC                ;修改扫描模式，用于下一行
        LD    C, A                   MOV   C, A
        IN    A, (KBDIN)             IN    KBDIN        ;列线数据输入
        AND   00000111B              ANI   00000111B    ;屏蔽无用位
        CP    00000111B              CPI   00000111B    ;有任一条列线接地?
        JR    NZ, FCOL               JNZ   FCOL         ;有，转去找该列线
        LD    A, B                   MOV   A,B          ;否，键号←键号+列数使之
                                                        ;适合下一行
        ADD   A, 03                  ADI   03
        LD    B, A                   MOV   B, A
        DEC   D                      DCR   D            ;所有各行都已扫描过了吗?
        JR    NZ, FROW               JNZ   FROW         ;否，扫描下一行
        JP    DONE                   JMP   DONE         ;是，程序结束，没有按键动
                                                        ;作
FCOL:   RRA                  FCOL:   RAR                ;是否这一条列线接地?
        JR    NC, RIGHT              JNC   RIGHT        ;是，程序结束
        INC   B                      INR   B            ;否，键号←键号+1
        JP    FCOL                   JMP   FCOL         ;检查下一列
DONE:   JP    DONE           DONE:   JMP   DONE
RIGHT:  JP    RIGHT          RIGHT:  JMP   RIGHT
```

这里是以 3 行 ×3 列的键盘为例的。如果把行数、列数、屏蔽模式等用 EQU 伪指令写出一个通用变量名，则上述程序可以更为通用化。

以上采用的键盘扫描法是传统的"行扫描"(row-scanning)法，也可以采用较新的"线路反转"(Line-reversal)法，这时必须用可编程序 I/O 接口电路来识别被按的键。

假设用 Z80 PIO 作为键盘的接口，这时必须用程序将其设置为位控方式，即每条线或每组

线可以独立程控作为输入或输出，这是问题的关键。

采用线路反转法进行键识别可分为三步进行：

(1) 将全部列线接地，并保存行线输入；

(2) 将全部行线接地，并保存列线输入；

(3) 利用行线输入和列线输入，组成键识别码与预先编制的一个表格进行比较，从而确定键号。

线路反转法比行扫描法更好，但其所需的可编程序 I/O 接口电路的成本比行扫描法所常用的锁存器和缓冲器更贵些。

二、编码键盘

编码键盘采用硬件线路来实现键盘编码。下面举例说明编码键盘硬件线路的工作原理。

在一个专用微型计算机的面板上，有 12 个数字键和 7 个功能键，如图 15-12 所示。它们经过硬件线路编码后，产生一个 8 位状态字送入 CPU，其编码方式如图 15-13 所示。

实现上述编码功能的硬件线路如图 15-14 所示。

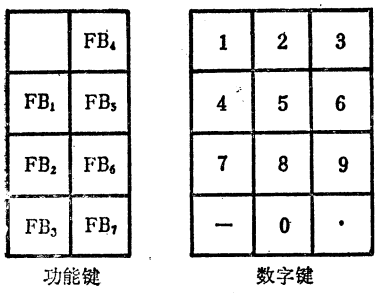


图 15-12 键盘面板

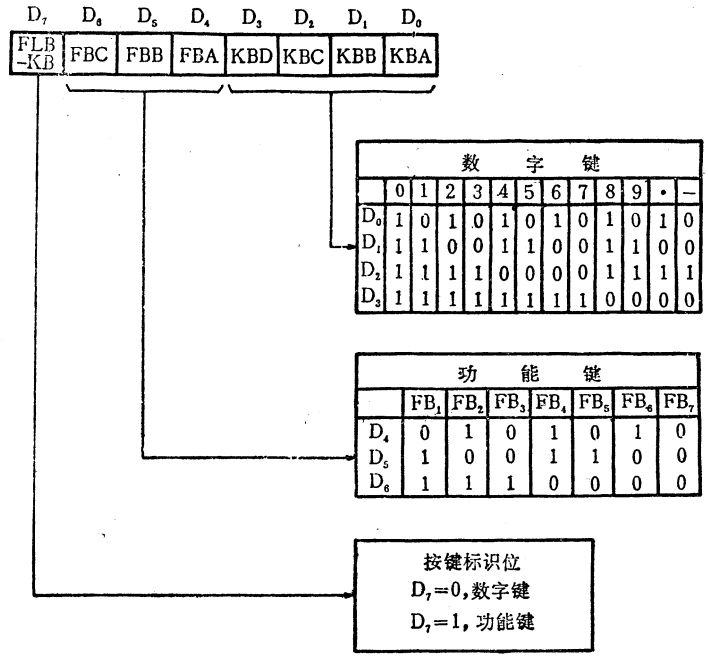


图 15-13 键盘编码格式

它由三个编码器和若干“与门”组成。编码器的输入为十进制数(低电平有效)，输出为此十进制数的二进制编码(反码形式)。按键控制编码器的输入。例如，将 FB_4 键按下，则编码器 1 的 4 端输入为低电平，因此，其输出编码为：$A_2=0$，$A_1=A_0=1$ (二进制数 100 的反码)，即 FBA = FBB = 1，FBC = 0。由于编码器 2、3 无输入，因此它们的 $\overline{GS}$ 端为高电平，这就使与门 4 的输出为高电平，即 FLB-KB 为高电平，表示按下的键是功能键。按下 FB_4 键时，在数据总线上 $D_7=1$，$D_6=0$，$D_5=D_4=1$。

这种硬件线路的编码方式易于实现。它的缺点是对一个键必须有一条输入线，所以仅适用于小键盘。如果键的个数较多，为了减少输入线的数量，可采用硬件线路扫描键盘的工作方式。

编码键盘与微型机的接口可以由一个输入缓冲器和一个中断请求触发器组成(见图 15-15)。按任何一个键都能使中断请求触发器置“1”，使 $\overline{INT}=0$，向 CPU 发中断请求信号。

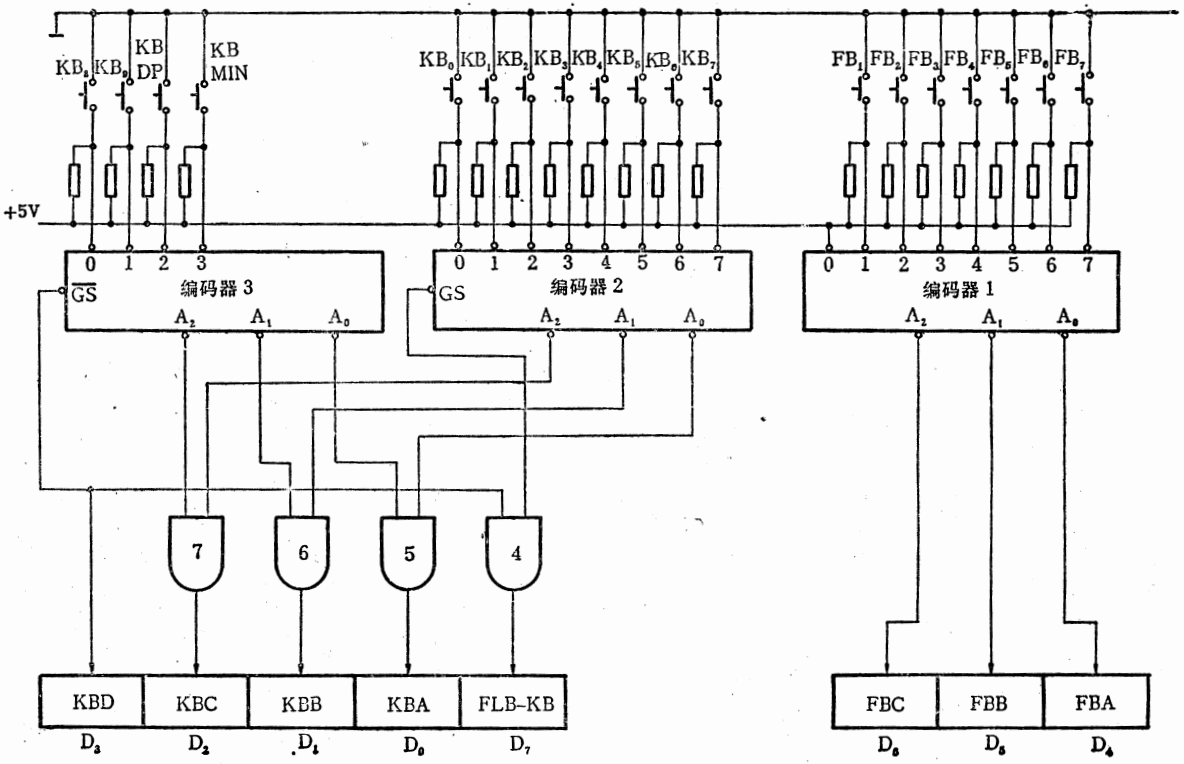


图 15-14　编码硬件线路

CPU 响应中断，按输入缓冲器的地址执行输入指令，于是由编码器产生的 8 位编码状态字经输入缓冲器读入 CPU，然后 CPU 可以通过识别 D_7 位来区分功能键与数字键。若按下的是功能键，则根据它的特征编码的不同而转入为其服务的子程序。若按下的是数字键，则按数字键的编码转换成二进制数字（输入的数字 0～9 是二进制的反码，而小数点与负号则作为专用字符处理）。

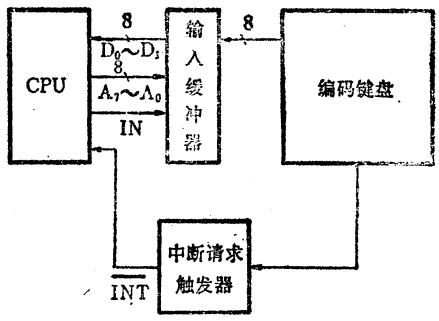


图 15-15　编码键盘与微型计算机的接口

以上的键盘接口技术可以扩展至较大的键盘，如 64 个键的键盘矩阵。这时，必须另加译码器。如果还要求“移位”和“控制”等更多的功能键，则需要有更多的硬件和软件支持。这时，采用专用的键盘/显示器控制器，如 Intel 8279 更为合适。

§15-4　模拟通道接口

一、模拟通道的作用和组成

微型计算机只能处理数字形式的信息，但是在实际工程中大量遇到的却是连续变化的物理量。所谓连续，包括两方面的含义：一方面从时间上来说，它是随时间连续变化的；另一方面从数值上来说，它的数值也是连续变化的。这种连续变化的物理量通常称为模拟量。例如温度、压力、流量、位移、转速以及连续变化的电压电流等等。模拟通道接口的作用就是实现实际的模拟量和数字量之间的转换。模/数（A/D）转换就是把输入的模拟量变为数字量，供微型计算机处理。数/模（D/A）转换就是将微型计算机处理后的数字量转换为模拟量形式的控制信号。

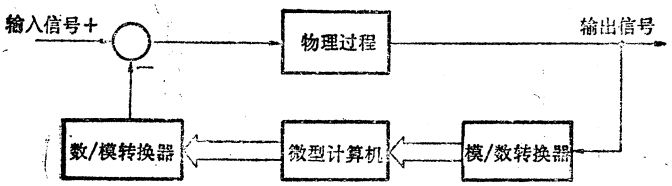


图 15-16　微型计算机对物理过程的控制

在微型计算机控制系统中，当控制对象有连续变化的模拟信号发送和接收时，必须设置 A/D 和 D/A 转换电路。如图 15-16 所示。

微型计算机控制系统对所要监视和控制的生产过程的各种参数，如温度、压力等，必须先由传感器（Transducer）进行检测，并转换为电信号，然后经过数据放大器放大，这时输出一般为 0～5 伏电平的模拟信号。接着，通过 A/D 转换器将标准的模拟信号转换为等价的数字信号，再传送给微型机。微型机对各种信号进行处理后输出数字信号，再由 D/A 转换器将数字信号转换为模拟信号，作为控制装置的输出去控制生产过程的各种参数。其过程如图 15-17 所示。

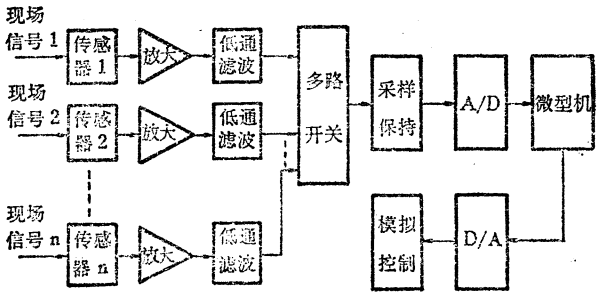


图 15-17　微型机与控制系统的接口

图中：

(1) 传感器——检测生产过程各种现场参数，并且转换为电信号。

(2) 数据放大器——将传感器的信号(通常为毫伏～微伏级)放大到 A/D 转换器所需要的量程范围。

(3) 低通滤波器(Low-Pass Filter)——抑制干扰。

(4) 多路转换开关 (Multiplexer)——通常需要监视和控制的生产过程的现场参数，虽然往往有若干个，但由于它们的变化是比较缓慢的，所以可以用一个 A/D 转换器来处理多路输入信号。这样必须采用多路切换装置对若干个输入信号进行顺序切换或选译切换，实现这一功能的部件称多路转换开关。

(5) 采样-保持电路(Sample/Hold)——它是为了提高转换精度和消除转换时间的不确定性而设置的。

下面简要介绍 D/A 和 A/D 转换器的原理，然后着重讨论 D/A、A/D 转换器与 CPU 的接口以及它们的应用。

1. D/A(Digit to Analog)转换器原理

D/A 转换器基本上由四个部分构成，即权电阻网络、模拟开关、基准电源和运算放大器。其原理线路如图 15-18(a)、(b)所示。对并行 D/A 转换器而言，它接收并行输入的二进制信息。在转换器中具有同二进制位数相等的模拟开关，每一位二进制码输入线控制一个模拟开关。电阻网络通过模拟开关接在基准电源上，电阻网络根据输入数字信息的控制作用，通过模拟开关的通断转换为相应的电压输出，运算放大器在 D/A 转换器中常用来对各输出分量求和。

图 15-18(a) 是 10 位二进制“权电阻解码网络”，每一位对应一个电阻及由该位代码所控制的双向模拟开关，图中下方标出的是二进制各位的“权”。从图中可看出，每一位电阻的阻值是与这一位“权”相对应的，权值愈大，其阻值愈小。在这里，电阻阻值与每一位的“权”相对应，故称为“权电阻解码网络”。

由于数字量的每一位代码都代表一定的“权”，因此 D/A 转换器必须将每一位二进制代码按其“权”的数值转换为相应的模拟量。又由于权电阻网络是一个线性网络，因此可以用迭加原理，将代表各位的模拟分量迭加，即得到总的模拟量输出，而这个总的模拟输出量恰好与输入的数字量成正比，从而实现了数字-模拟转换。

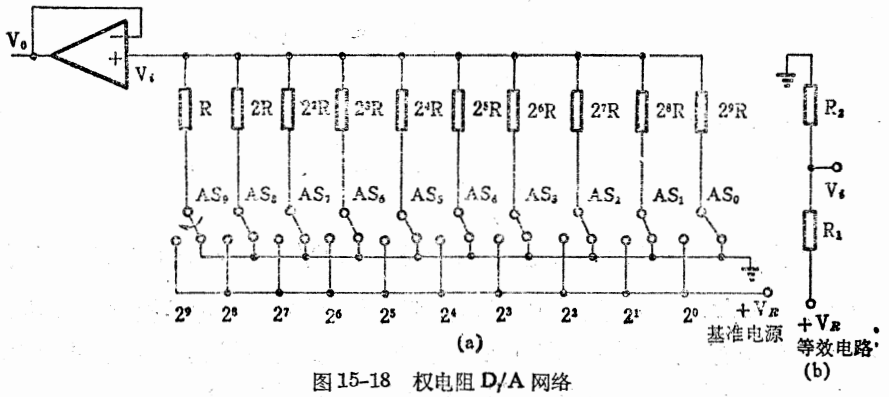


图 15-18 权电阻 D/A 网络

上图表示二进制十位的情况。二进制数码从高位至低位分别控制模拟开关 AS_9～AS_0。当相应位的数码为“1”时，模拟开关接到基准电源 $+V_R$；当相应位的数码为“0”时，模拟开关与地(零电平)相接。

假设数码为1000000000，则 AS_9 接 $+V_R$，其它模拟开关都接地。这时等效电路如图15-18(b)所示。

设电压跟随器的输入阻抗为 R_i，则输出电压分量 V_{i9} 为

$$V_{i9}=\frac{R_2}{R_1+R_2}\cdot V_R=\frac{\dfrac{R_2}{R_1R_2}}{\dfrac{R_1+R_2}{R_1R_2}}\cdot V_R=\frac{\dfrac{1}{R_1}}{\dfrac{1}{R_1}+\dfrac{1}{R_2}}\cdot V_R$$

式中 $R_1=R$

$R_2=2R\,/\!/\,2^2R\,/\!/\,2^3R\,/\!/\cdots/\!/\,2^9R\,/\!/\,R_i$（即为其它各接地电阻的并联值）

因此

$$V_{i9}=\frac{1}{1+\dfrac{1}{2}+\dfrac{1}{2^2}+\cdots+\dfrac{1}{2^9}+\dfrac{R}{R_i}}\cdot V_R$$

一般情况下，因为网络的输出端有跟随器，而跟随器的输入阻抗 R_i 比网络中的电阻 R 大得多，故可略去 R_i 的影响。此时有

$$V_{i9}=\frac{1}{1+\dfrac{1}{2}+\dfrac{1}{2^2}+\cdots+\dfrac{1}{2^9}}\cdot V_R\approx\frac{1}{2}V_R\left(=\frac{1}{2^1}V_R\right)$$

同理，当数码为0100000000时，即模拟开关 AS_8 接基准电源，其它模拟开关接地，此时有

$$V_{i8}=\frac{R_2}{R_1+R_2}\cdot V_R=\frac{1/R_1}{1/R_1+1/R_2}\cdot V_R$$

式中 $R_1=2R$

$R_2=R\,/\!/\,2^2R\,/\!/\,2^3R\,/\!/\cdots/\!/\,2^9R$

因此

$$V_{i8}=\frac{\dfrac{1}{2R}}{\dfrac{1}{2R}+\dfrac{1}{R}+\dfrac{1}{2^2R}+\cdots+\dfrac{1}{2^9R}}\cdot V_R\approx\frac{1}{4}V_R\left(=\frac{1}{2^2}V_R\right)$$

同理可以求出对应于其它数码时的输出电压如表15-2所示。

假设二进制数码 $D=D_9D_8\cdots D_2D_1D_0$，其中 $D_i(i=0,1,2,\cdots9)$ 或者为“0”或者为“1”，则根据迭加原理，数码D对应的输出电压为

$$V_{i\Sigma}=\frac{V_R}{2^{10}\left(1-\dfrac{1}{2^{10}}\right)}\sum_{i=0}^{9}D_i2^i\approx\frac{V_R}{2^{10}}\cdot\sum_{i=0}^{9}D_i2^i$$

对于n位D/A转换器可推广得

$$V_{i\Sigma}=\frac{V_R}{2^n\left(1-\dfrac{1}{2^n}\right)}\sum_{i=0}^{n-1}D_i2^i\approx\frac{V_R}{2^n}\cdot\sum_{i=0}^{n-1}D_i2^i$$

由此可知，D/A转换器总的输出电压 V_0 与输入到D/A转换器的二进制数码值 $\sum_{i=0}^{n-1}D_i2^i$ 成正比，从而实现了D/A转换。从原理上说，只要位数足够多，这种转换器可以达到很高的精度，实际上电阻值总是有一定的误差，而且受温度变化的影响，开关也不可能是理想的，这些都将是影响转换器精度的主要因素，而原理误差往往只占很小的成分。

上述电路的缺点是构成网络的电阻数值种类太多，相差也大，尤其当位数增多时，阻值分散性将很大，而为了保证转换精度，阻值又要求很精确，这就给生产上带来一定的困难。为此常采用T形解码网络的D/A转换器。网络中只用R和2R两种电阻，每节有两个电阻，一

表 15 2

数码	电压	
	准确值	近似值
1000000000	$\frac{1}{2^1}\left(\frac{2^{10}}{2^{10}-1}\right)V_R$	$\frac{1}{2^1}V_R$
0100000000	$\frac{1}{2^2}\left(\frac{2^{10}}{2^{10}-1}\right)V_R$	$\frac{1}{2^2}V_R$
0010000000	$\frac{1}{2^3}\left(\frac{2^{10}}{2^{10}-1}\right)V_R$	$\frac{1}{2^3}V_R$
0001000000	$\frac{1}{2^4}\left(\frac{2^{10}}{2^{10}-1}\right)V_R$	$\frac{1}{2^4}V_R$
0000100000	$\frac{1}{2^5}\left(\frac{2^{10}}{2^{10}-1}\right)V_R$	$\frac{1}{2^5}V_R$
0000010000	$\frac{1}{2^6}\left(\frac{2^{10}}{2^{10}-1}\right)V_R$	$\frac{1}{2^5}V_R$
0000001000	$\frac{1}{2^7}\left(\frac{2^{10}}{2^{10}-1}\right)V_R$	$\frac{1}{2^7}V_R$
0000000100	$\frac{1}{2^8}\left(\frac{2^{10}}{2^{10}-1}\right)V_R$	$\frac{1}{2^8}V_R$
0000000010	$\frac{1}{2^9}\left(\frac{2^{10}}{2^{10}-1}\right)V_R$	$\frac{1}{2^9}V_R$
0000000001	$\frac{1}{2^{10}}\left(\frac{2^{10}}{2^{10}-1}\right)V_R$	$\frac{1}{2^{10}}V_R$

个开关，相当于二进制数的一位，开关由该位的代码所控制。由于电阻接成 T 型，故称 T 型解码网络。

把权电阻网络和 T 型电阻网络的电路形式的特点综合起来可以得到新的网络形式，称为变形权电阻网络。目前，这种权电阻网络已被广泛采用。其原理分析与上述相似，故不另作叙述。

2．A/D(Analog to Digit)转换器原理

如上所述，A/D 转换器的作用就是将模拟信号转换为二进制的数字信号供微型机或数字控制系统进行处理。

A/D 转换器种类繁多，常用的 A/D 转换器有三种类型：一是利用计数比较器技术；二是利用双斜率积分法；三是利用逐次逼近法。由于采用逐次逼近法进行 A/D 转换，无论在转换速度上或在精度上都能得到满意的结果，因此它是目前最广泛使用的一种 A/D 转换器。

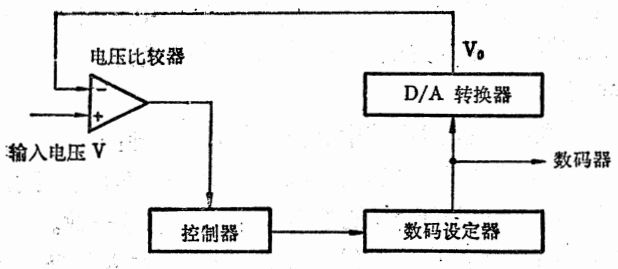


图 15-19　比较式电压数字转换原理框图

采用逐次逼近法的 A/D 转换器是由一个模拟量比较器和一个 D/A 转换器组成的，如图 15-19 所示。

在数码设定器中先设定一个数码，经 D/A 转换器转换为电压 V_0，称之为反馈电压。反馈电压 V_0 与被转换电压 V 进行比较，若 $V>V_0$，则控制器使数码设定器的数码增加；反之，若 $V<V_0$，则控制器使数码设定器的数码减小；数码设定器中的数码改变后，V 与 V_0 再进行比较。这样从高位到低位逐位比较，直至比较到最低位为止。最后反馈电压 V_0 与被转换电压 V

相接近，其误差小于一个量化单位。此时，数据设定器中的数码即为转换结果。

改变数码设定器中的数码有多种方法，最常用的是逐位比较法(或称逐次逼近法)。下面主要介绍逐位比较式电压数字转换器。

逐位比较式电压数字转换器的工作过程是用一系列基准电压与被转换电压V相比较，由高位至低位逐位地确定各位数码是“1”还是“0”。

现举例说明逐次比较法转换的步骤：

第一步，由控制器置数码设定器数码为100…00，若 $V>V_0$，则第一个“1”保留；若 $V<V_0$，将第一个“1”变为“0”。

第二步，由控制器置数码设定器第二位数码为“1”，此时若 $V>V_0$，则第二个“1”保留；若 $V<V_0$，则第二个“1”变为“0”。

依次类推，直到最末一位为止。表 15-3 给出了十位逐位比较式电压数字转换器转换过程实例。设转换电压 V 为 675(量化单位)。D/A 转换器每位的基准电压分别为 512、256、128、64、32、16、8、4、2、1(量化单位)。表 15-3 中的“留”是指这一步编码的那位“1”应保留；“去”是指这一步编码的那位“1”应变为 0。

表 15-3

步骤	数码设定器内容										十进制读数	比较器判断
	512	256	128	64	32	16	8	4	2	1		
1	1	0	0	0	0	0	0	0	0	0	512	+ 留
2	1	1	0	0	0	0	0	0	0	0	768	− 去
3	1	0	1	0	0	0	0	0	0	0	640	+ 留
4	1	0	1	1	0	0	0	0	0	0	704	− 去
5	1	0	1	0	1	0	0	0	0	0	672	+ 留
6	1	0	1	0	1	1	0	0	0	0	688	− 去
7	1	0	1	0	1	0	1	0	0	0	680	− 去
8	1	0	1	0	1	0	0	1	0	0	676	− 去
9	1	0	1	0	1	0	0	0	1	0	674	+ 留
10	1	0	1	0	1	0	0	0	1	1	675	+ 留
结果	1	0	1	0	1	0	0	0	1	1	675	

图 15-20(a)、(b)分别示出：(a) 各位基准电压的量化单位；(b) 工作过程中反馈电压 V_0 的波形图。由此可见，反馈电压 V_0 逐次逼近转换电压 V，故称为逐步逼近式转换器。

3．采样-保持电路(Sample/hold circuit)

采样-保持电路广泛应用于数据采集系统和实时控制系统中，它的功能有两种：

(1) 采样跟踪状态：在此期间应尽可能快地接受输入信号，使输出和输入信号相一致。

(2) 保持状态：把采样结束前瞬间的输入信号保持下来，使输出和保持的信号一致。

由于模/数转换需要一定时间，在转换期间，要求模拟信号保持稳定，因此当输入信号变化速率较快时，都应采用采样-保持电路，如果输入信号变化缓慢，则可不用保持电路。

采样-保持电路的工作原理如图 15-21 所示。

此电路由模拟开关 K，存储元件 C，运算放大器 A 所组成。模拟开关由数字指令控制，当数字指令为“1”时，模拟开关 K 接通，存贮在 C 上的信号跟踪输入信号，经缓冲放大器输出，这就是采样过程。这段时间称为采样时间。当数字指令为“0”时，模拟开关 K 断开，存贮电容 C

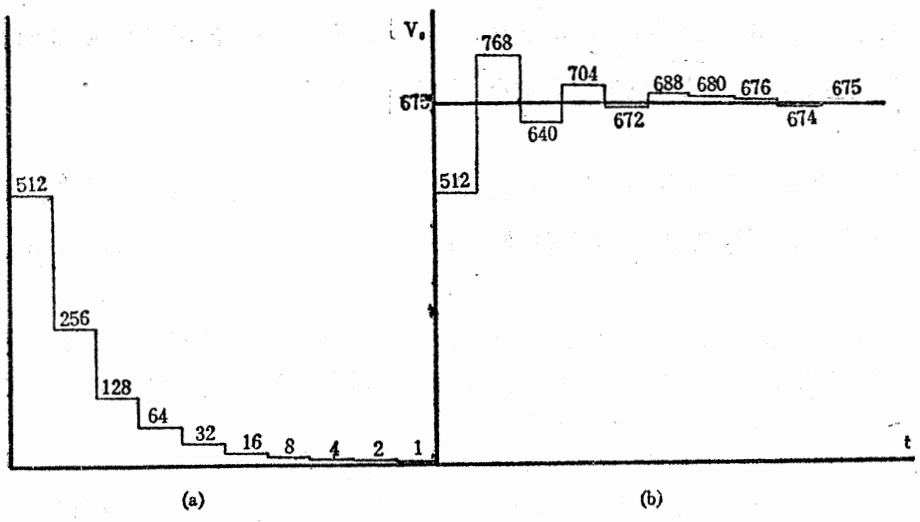


图 15-20 (a) 各种基准电压的量化单位(b)反馈电压 V_0 的波形图

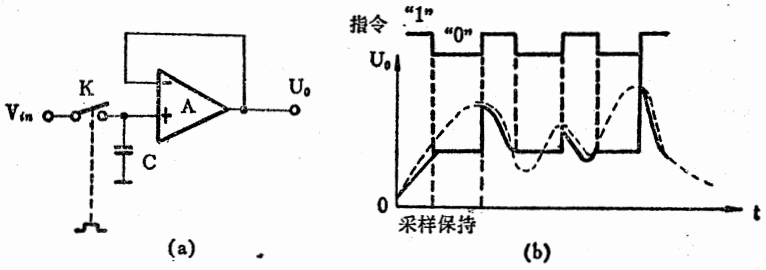


图 15-21 采样-保持电路和工作原理图

将 K 断开瞬间的输入信号保持下来，并通过放大器输出，即输出信号保持断开瞬间的输入信号，这个过程称为保持过程。这段时间称为保持时间。

在实际的数据采集系统或控制系统中，往往有多个模拟输入或模拟输出通道。目前，仍以用一个公用的转换器，而对多路的模拟输入或模拟输出信号实行分时控制为主要方式。多路转换开关可实现多个模拟输入通道分时采样，以输入到一个公用的 A/D 转换器；多路分配开关可实现把一个公用的 D/A 转换器输出信号分时地送到多个模拟输出通道去。

微型计算机控制系统的特点是必须用采样开关将连续变化的模拟信号，转变为离散的脉冲序列。究竟采用多大的采样频率对模拟信号进行采样，才能复现原来的模拟信号而不丢失信息？香农(Shannon)采样定理指出：当采样器的采样频率高于或等于连续信号的最高频率的两倍时，原信号才能通过采样器而无失真地复现出来。但在实际应用中，还需解决许多技术问题。

二、数据转换器的主要指标

1. A/D 转换器的主要指标

(1) 分辨率(resolution)

分辨率是指转换器所能分辨的被测量的最小值。

通常用位数来表示 A/D、D/A 转换器的分辨率，如 8 位、10 位、12 位等。对于 n 位转换

器，其分辨率为整个模拟量的 $\frac{1}{2^n}$，n 为数字的位数。例如，一个 10 位转换器的分辨率为 $\frac{1}{1024}$。显然，位数愈多，最低位的数值愈小，分辨率就愈高。

(2) 精度

有绝对精度和相对精度两种表示方法。常用数字量的位数作为度量绝对精度的单位，如精度为最低位的 ±1/2，即 $\pm\frac{1}{2}$ LSB（最低有效位）；用百分比表示满量程时的相对误差，如 ±0.05%。

应当指出，精度与分辨率是两个不同的概念。精度指的是转换后所得结果相对于实际值的准确度；而分辨率指的是能对转换结果发生影响的最小输入量。分辨率很高，但由于温度漂移、线性不良等原因，精度并不一定很高。

(3) 量程(满刻度范围)(FSR——Full Scale Range)

量程即指转换器的满刻度范围，亦即最大和最小模拟值之差。例如，转换器具有 0～10V 的单极性范围或 −5～+5V 的双极性范围，在这两种情况下 FSR = 10 V。

应当指出，A/D、D/A 转换器的输出永远达不到满刻度值。例如，设 D/A 输出范围为 0～10V，那末 10V 就是名义上满刻度值。如 8 位分辨率的转换器的最大输出为 $255/256\times10=9.961$V；而 12 位分辨率的转换器的最大输出为 $4095/4096\times10=9.9976$(V)。在上述两种情况下，最大输出都比名义满刻度电压小一位。这是因为模拟量的零值是 2^n 个转换器状态中的一个，故对 A/D、D/A 转换器而言，在 0 值之上仅有 2^n-1 个阶梯。为简便起见，数据转换器的模拟量范围，总是用名义满刻度来确定。

(4) 总转换误差(total conversion error)

总转换误差包括整个转换器的设备误差和量化误差，用 $\pm\frac{1}{2}$ LSB，±1LSB 来表示。

(5) 量化误差(quantizing error)

将模拟量变为数字量，总有一个最小的数字分度单位，这就构成量化误差。它决定于数字位数最低位所对应的电压值，其最大值等于 $\pm\frac{1}{2}$ LSB。

(6) 线性度误差(Linearity error)

这是指转换器传递函数曲线与理想值(直线)不同而出现的误差，用 % 或 LSB 表示。

(7) 总转换时间(total conversion time)

自发布对输入信号进行采样的指令开始，直到获取整个数字信号时止，完成一次转换所需的时间叫做总转换时间。其中包括：输入信号多路采样切换的时间、缓冲放大器的稳定时间、采样时间、具体转换时间以及装入缓冲寄存器完成串行或并行转换的时间。总转换时间与转换位数有关。一般转换精度要求愈高，总转换时间也愈长。

除上述指标外，还有输出逻辑电平(多为与 TTL 兼容)，对电源的要求及环境条件(温度、湿度、外界电磁场、振动、冲击等)等指标。

2. D/A 转换器的主要指标

(1) 分辨率，它与 A/D 转换器分辨率的含义相同。

(2) 稳定时间(Settling time)

当放大器的输入电压快速改变时，其输出电压稳定下来，达到一定正确数值所需的时间叫

做稳定时间。它比 A/D 转换时间短得多，一般为 μs 或 ns 数量级。

三、CPU 与 D/A 转换器的接口

CPU 与 D/A 转换器的接口如图 15-22 所示。

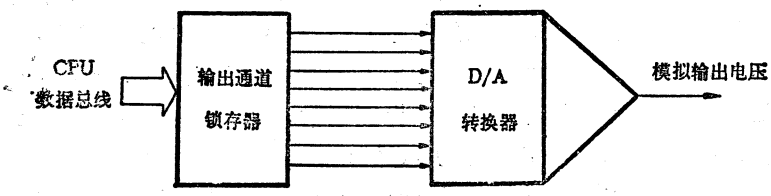


图 15-22　D/A 转换器与输出端口的连接

由图可知，一个 8 位的 D/A 转换器，可以直接与作为输出通道的 8 位锁存器(如 8212、74LS273 等)的输出端连接，实现数/模转换。如果 D/A 转换器本身具有数据输入寄存器或锁存器，而端口选择信号又可以直接加到 D/A 转换器的选通输入端，那么数据输出端口便可省略。如果要求更高精度的数/模转换(如 10、12、16 位的 D/A 转换)，则需要有两个端口来保存数据，此时微处理器先输出 8 位数据到一个锁存器，接着再输出剩下位数的数据到另一个锁存器，然后微处理器发出一个控制信号选通保存数据的两个锁存器，将分两次送至锁存器的数据同时送到 D/A 转换器。

目前，常用的 LSI 的 D/A 转换器有：

8 位 D/A 转换器：AD1408, DAC0800, DAC0832, DACIC8B。

10 位 D/A 转换器：AD7520, AD7533, 5G7520 (国产)。

12 位 D/A 转换器：AD7521, DAC-68IC 等。

1. CPU 与 8 位 D/A 转换器的接口

AD1408/AD1508 是单片 8 位 D/A 转换器，它由匹配好的双极型开关、一个精密电阻网络和一个控制放大器所组成，采用 16 引线的双列直插式封装。使用时的连接方法如图 15-23 所示。

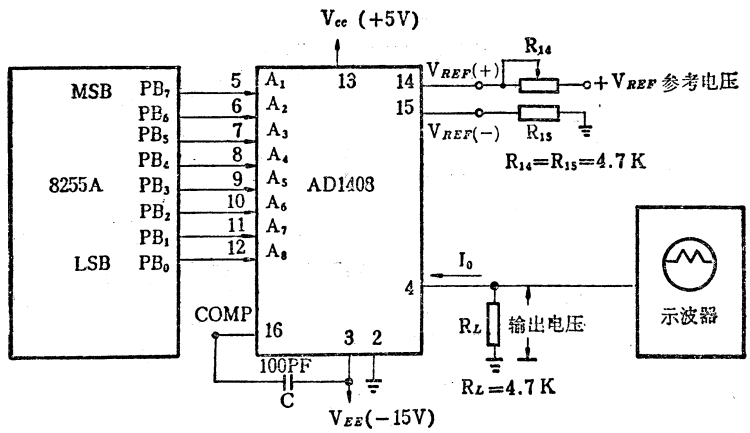


图 15-23　AD1408 (使用正参考电压) 的连接方法

注：引线 1 为空脚(NC)，使用时悬空。

其主要性能如下：分辨率 8 位，线性度 $<\pm\frac{1}{4}$ LSB，稳定时间 250 毫微秒，最大输出电流 2 毫安。

A_1～A_8 是转换电路的数据输入端，其逻辑电平与 TTL 兼容。每一位的输入控制 AD1408 内部的一个模拟电流开关，该开关有效地馈送一个正比于该位"权"的电流，整个输出电流等于所有内部开关的电流之和，它正比于输入的二进制数码(A_1～A_8)的值（注意：标号 A_1 是数字的最高位，而 A_8 是最低位）。整个可变换的电流受参考电压 V_{REF} 的控制，V_{REF} 通过电阻 R_{14} 被转换为电流，这个参考电流确定输出电流的标度系数，同时也受 R_{14} 的影响，它的最大输出电流 $=V_{REF}/R_{14}\leqslant 2mA$（满标度输出电流为 1.992mA）。$R_{15}$ 选用接近于 R_{14} 的阻值，它为参考控制放大器提供了基本的电流补偿，以便使温度漂移减至最小值。COMP 端接入补偿电容器 C，用于对电路内部的参考放大器的相位补偿，以提高对噪声的抑制能力，保证内部线路的稳定性。在负载电阻 R_L 上可取单向(负电压)的电压信号，但必须与高阻抗输入电路相接(如示波器等)，其输出电压与输入数字信号 A_1～A_8 有如下关系：

$$V_{OUT}=-\left(\frac{V_{REF}}{R_{14}}\times R_L\right)\times\left(\frac{A_1}{2}+\frac{A_2}{4}\cdots\frac{A_8}{256}\right)=-5V\left(\frac{A_1}{2}+\frac{A_2}{4}\cdots\frac{A_8}{256}\right)$$

$$(V_{REF}=5V,\ R_L=R_{14})$$

当输入值为 00H 时，D/A 转换器输出为 0V。

当输入值为 FFH 时，D/A 转换器输出为 −4.98V。

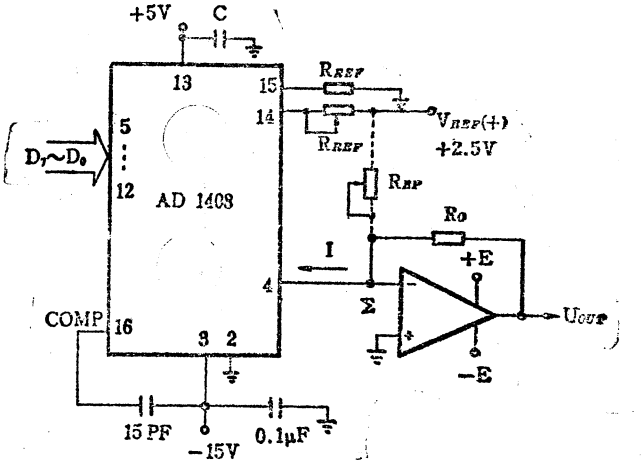
如图 15-23 所示，把 AD1408 接到 8255 并行 I/O 接口芯片的 PB 端口。必须预先对 8255A 初始化，即送一控制字到 8255A 的控制寄存器，把 PB 端口设置为输出方式。

若把 AD1408 接到 Z80 PIO 并行 I/O 接口芯片的 PB 端口，同样必须预先对 PIO 初始化，即送一方式控制字到 PIO 的 PB 端口控制寄存器，把 PB 端口设置为输出方式。

如果要提高 AD1408 输出的带载能力，该电路必须与运算放大器配合使用。

图 15-24 示出 AD1408 在增加带负载能力情况下的电路连接图。

这时，AD1408 的输出端(引线 4)可直接与任何型号的运算放大器的求和端 Σ 相接，其输出电压与输入数字信号 A_1～A_8 有如下关系式：



注：R_{REF}=1.25K R_0=5.00K R_{BP}=2.5K

图 15-24 D/A 转换的典型电路

$$V_{OUT}=V_{REF}\times\frac{R_0}{R_{REF}}\left(\frac{A_1}{2}+\frac{A_2}{4}+\frac{A_3}{8}+\frac{A_4}{16}+\frac{A_5}{32}+\frac{A_6}{64}+\frac{A_7}{128}+\frac{A_8}{256}\right)$$

调整 V_{REF}、R_{RET}、R_0 三个参数可改变 V_{OUT} 的输出范围。例如，若取 $V_{REF}=2.5V$，$R_0=5K\Omega$，$R_{REF}=1.25K\Omega$，则当输入数为 00H 时，$V_{OUT}=0V$；输入数为 FFH 时，$V_{OUT}=9.961V$，此时输出为单极性。

如果希望得到双极性的电压输出范围，则必须在运算放大器的求点和，通过 R_{BP}（如虚线所示）加入一个正信号 V_{REF}，则输出特性将向负方向平移。当取 $R_0=2R_{BP}$ 时，V_{OUT} 下移 5V，此时，数值 00H 对应的输出电压为 −5V；80H 对应的输出电压为 0V；FFH 对应的输出电压为 4.961V。

获得上述双极性电压输出范围时所取参数为 $V_{REF}=2.5V$，$R_{REF}=1.25K\Omega$，$R_0=5.00K\Omega$，$R_{BP}=2.50K\Omega$，$C=15PF$

【例 1】 编写一个三角波程序经 AD 1408 送示波器不断反复地显示。

① 在 Intel 8080A/8085A CPU 系统中，用 8255A 的端口 B 作接口时，其程序流程图如图 15-25 所示，其延时子程序见图 15-26 所示。

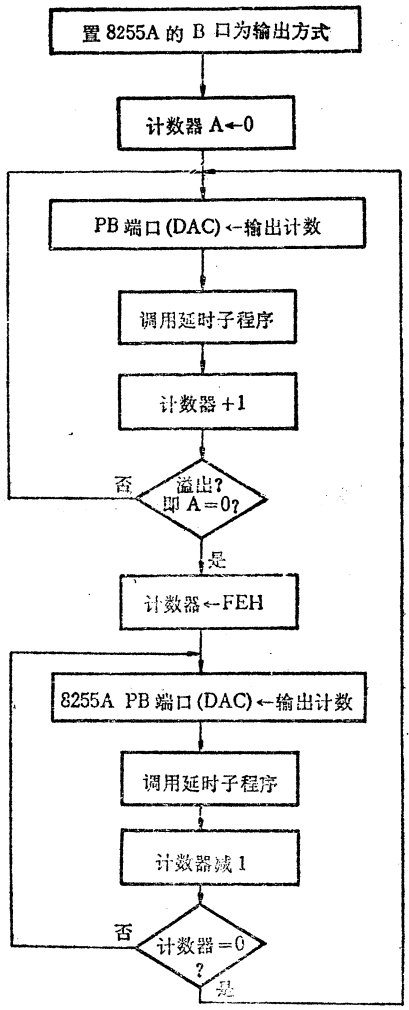


图 15-25 三角波程序流程图

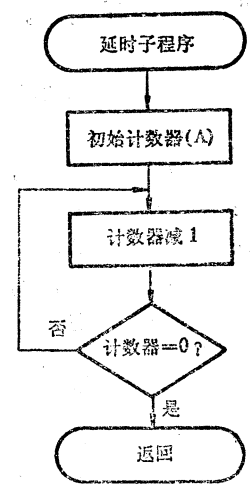


图 15-26 延时子程序

三角波主程序：

```
标号    指令    (助记符)        注释
        ORG     2000H
        LXI     SP,     20C2H
        MVI     A,      INITW   ；送控制字到 8255A
        OUT     CTLPRT
        XRA     A               ；计数器清零
```

```
UP:     OUT   DAC
        CALL  DELAY        ; 调用延时子程序
        INR   A            ; 计数器增量
        JNZ   UP           ; 若 A<FFH，则继续计数
        MVI   A,    0FEH   ; 若 A=0，则置下降值
DOWN:   OUT   DAC
        CALL  DELAY        ; 调用延时
        DCR   A            ; 计数器减量
        JNZ   DOWN         ; 继续减一直到 0 为止
        JMP   UP           ; 循环执行
```

延时子程序

```
DELAY:  PUSH  PSW          ; 保存 A、F
        MVI   A,    05H    ; 设延迟值
DEL2:   DCR   A
        JNZ   DEL2
        POP   PSW
        RET
DAC:    EQU   35H
CTLPRT: EQU   37H
INITW:  EQU   91H
```

② 在 Z80 CPU 系统中(假设为 TP-801 单板机)，用 Z80-PIO 作接口时，可以画出类似的程序流程图如图 15-27 所示。

三角波主程序：

```
        ORG   2000H
        LD    A,    0FH    ;
        OUT   (83H), A      ;   } 送控制字到 Z80 PIO 的端口 B
TLOOP:  LD    B,    01H
TLOOQ:  LD    A,    B
        OUT   (81H), A
        INC   B
        JR    NZ,   TLOOQ-$
        LD    B,    0FEH
TLOOR:  LD    A,    B
        OUT   (81H), A
        DJNZ  TLOOR-$
        JR    TLOOP-$
```

2. CPU 与 12 位 D/A 转换器的接口

在许多实际应用中，要求 D/A 转换器有更高的精度，这时可用 10 位、12 位或 14 位 D/A 转换器。

下面以 Intersil 公司的单片 12 位 D/A 转换器 AD7521 为例说明其与 CPU 的接口原理。AD7521 芯片内部由 12 只 CMOS 电流模拟开关及其驱动逻辑和 R-2R 梯形电阻网络所组成，数据输入可与 TTL/DTL/CMOS 逻辑电平兼容。其主要性能指标如下：分辨率 12 位，线性度

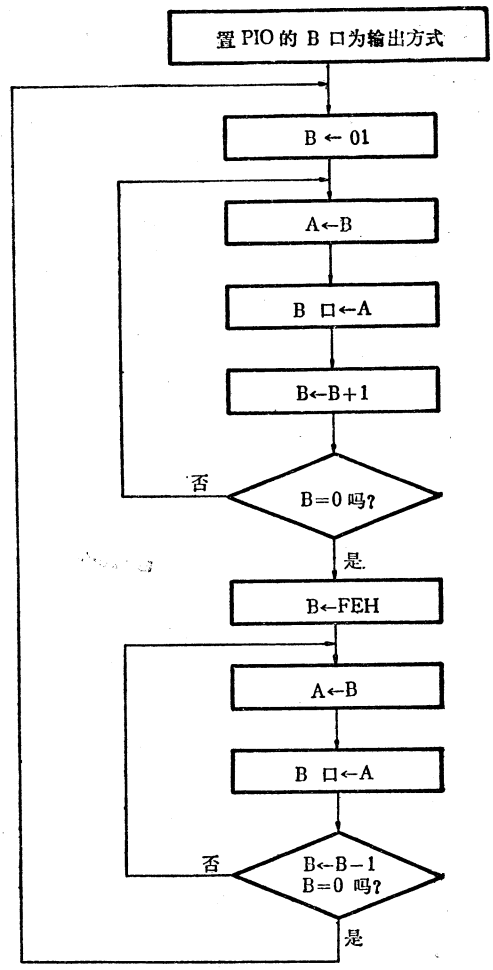


图 15-27　三角波程序流程图

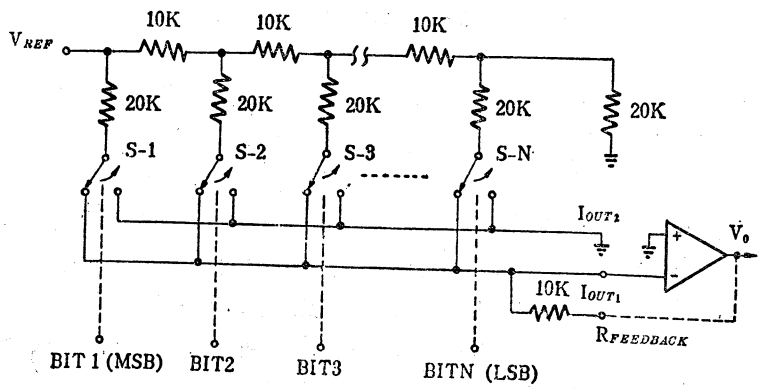


图 15-28　AD7521 原理电路图

8、9、10 位，稳定时间 500 ns，供电电压为 +5～+15 V。

AD7521 工作原理电路图如图 15-28 所示。引线图如图 15-29 所示。

电流模拟开关分别由12位的数字量来控制。若设该位为"1"时，开关合向 I_{OUT1}；若设该位为"0"时，开关就合向 I_{OUT2}。当把 I_{OUT2} 端接地时，则合向 I_{OUT1} 端的开关越多，I_{OUT1} 输出的电流值就越大，故 I_{OUT1} 的值能反映输入的数字量的大小。详细的电路原理分析见本节第一部分。同理可得

$$V_0 = -V_{REF}\left(\frac{A_1}{2^1}+\frac{A_2}{2^2}+\cdots+\frac{A_{12}}{2^{12}}\right)$$

式中，A_i 为"1"或"0"依输入数据而定，每一位代码都代表一定的"权"，并按"权"的数值转换为相应的模拟分量。经过迭加之后的总的模拟量输出与输入的数字量成正比。

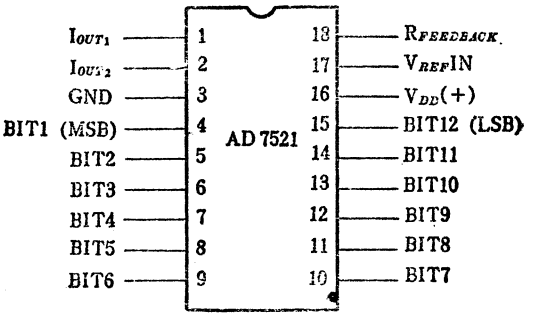


图15-29 AD7521的引线图

AD7521与CPU的接口电路如图15-30所示。

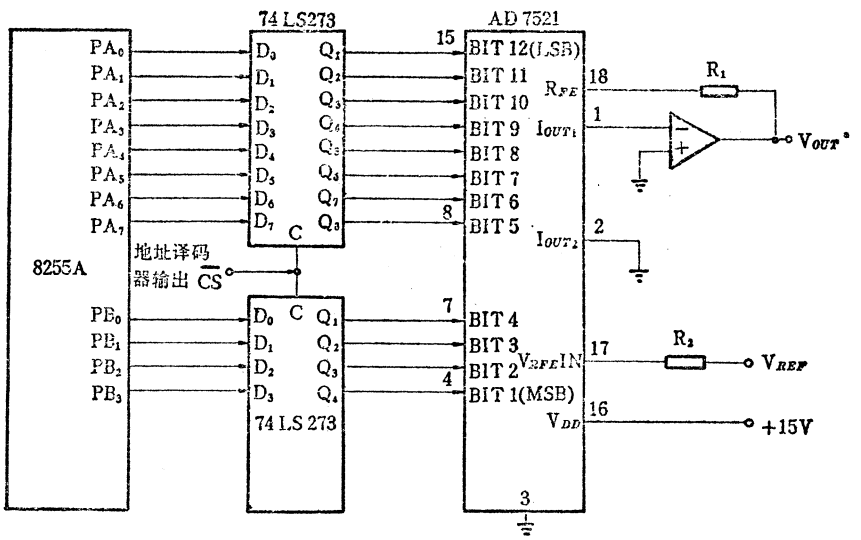


图15-30 CPU与AD7521的接口电路

在这里，输出的模拟量是单极性的。图中的运算放大器起着将输出电流 I_{OUT1} 转换为电压输出 V_{OUT} 的作用。运算放大器所需的反馈电阻已集成在AD7521芯片内，在外接时可串入一个0～500Ω的可调电阻 R_1 作为增益的微调电阻。当 R_1 增加时，可使输出的 V_{OUT} 相应的增加。若需减小增益，可在 V_{REF} 到17引线之间串入一个0～500Ω的可调电阻 R_2。当 R_2 增加时，增益略有下降。当输入的数字量变化时，输出的模拟量变化见表15-4。

如取 $V_{REF}=4.096$ V，则最低位对应的模拟量电压 $V_{OUT}=-V_{REF}(2^{-12})=-1$ mV。

从图15-30看出：在8255A与AD7521之间连接了两片8位锁存器(74LS273)，把12位分成两段，第一次CPU先输出低8位到一片锁存器，第二次再把高4位送到另一片锁存器上，待12位数据都已输出到锁存器上时，再由CPU发出输出选通信号，将两片锁存器中的12位数据同时输送到AD7521进行D/A转换，然后经运算放大器输出模拟电压，这样可以避免产生毛刺。

表 15-4 单极性接法的 D/A 转换编码表

数字量输入	模拟量输出
111111111111	$-V_{REF}(1-2^{12})$
100000000001	$-V_{REF}(1/2+2^{-12})$
100000000000	$-\frac{V_{REF}}{2}$
011111111111	$-V_{REF}(1/2-2^{-12})$
000000000001	$-V_{REF}(2^{-12})$
000000000000	0

注：$1LSB=2^{-12}V_{REF}$

AD7521 也可以接成双极性输出方式，其实用线路及编码表如图 15-31 及表 15-5 所示。

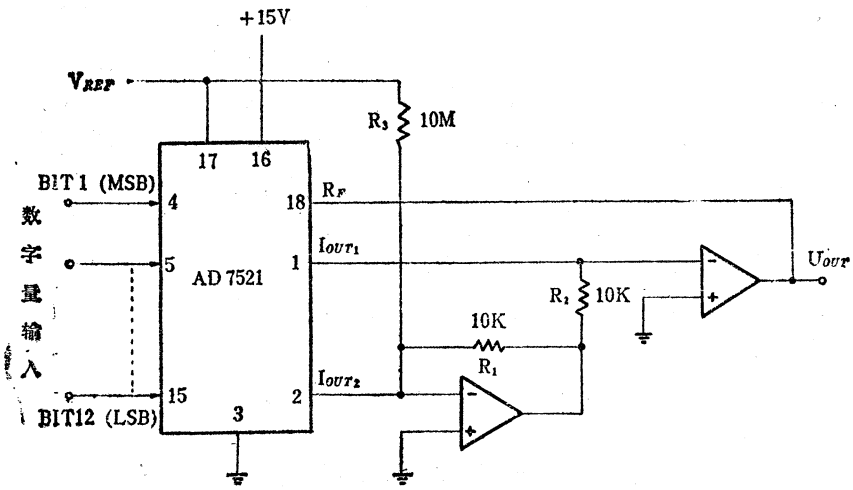


图 15-31 双极性接法的D/A 线路

表 15-5 双极性接法的 D/A 转换编码表

数字量输入	模拟量输出
111111111111	$-V_{REF}(1-2^{-11})$
100000000001	$-V_{REF}(2^{-11})$
100000000000	0
011111111111	$V_{REF}(2^{-11})$
000000000001	$V_{REF}(1-2^{-11})$
000000000000	V_{REF}

注：$1LSB=2^{-11}V_{REF}$

此时，I_{OUT2} 端不接地，输出的电压 V_{OUT} 由 I_{OUT1} 和 I_{OUT2} 按一定关系组合来确定。根据输入的数字量的变化，输出的模拟电压可有正负两种极性。

12 位 D/A 转换程序与 8 位 D/A 转换程序大致相同，故不另作说明。

四、CPU 与 A/D 转换器的接口

前已述及，现在常用的模/数转换方法是逐次逼近法，其中又分为：软件逐次逼近法和硬件逐次逼近法。当模拟输入信号变化不大，对转换速度要求不高时，为了降低成本，采用软件

逐次逼近法较为适宜。当需要高速 A/D 转换时，采用硬件逐次逼近法更为适合。

1．软件逐次逼近转换

图 15-32 给出了软件逐次逼近法的原理图。

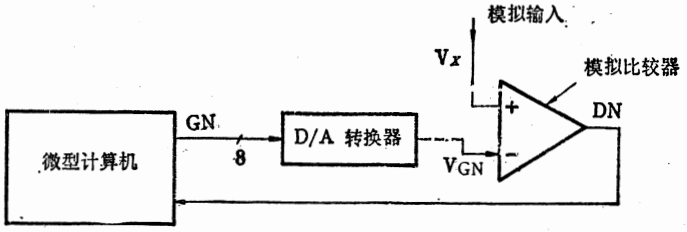


图 15-32　软件逐次逼近法原理图

在这种逐次逼近法方案中，微型计算机执行一段程序，便能求出模拟输入电压的数字表示。模拟比较器的输出给微型计算机一个指示信号，表明 CPU 送出的猜想电流值 GN 是太高(这时 DN = 0)，还是太低(这时 DN = 1)。假定所有的模拟输入电压是相近的，对于逐次逼近型的转换器而言，最有效的搜索算法即所谓的对分搜索法。对分搜索法的基本思路是：微型计算机的每一次猜测将产生有关模拟输入电压值的最多的信息。设第一次猜想 (G1) 是 10000000，因为模拟电压高于这个值或低于这个值是相近的。若第一次猜想值太低(D1 = 1)，则下次猜想值为 11000000。若第一次猜想值太高(D1 = 0)，则下次猜想值为 01000000。用此方法，仅需 8 次这样的猜想就能得到答案。而如果从 00000000 开始，然后每次简单地将猜想值加 1，则平均需要猜想 128 次。

我们可以用数学方式把这样对分搜索的方法作如下描述：令 DN 是第 N 次猜想后立即输出的模拟比较器的值，并且设 V_X 是模拟输入电压。

当 $V_{GN} > V_X$ 时，则 DN = 0；而当 $V_{GN} < V_X$ 时，DN = 1，则猜想值如下

G1 = 10000000

G2 = $D_1$1000000

G3 = $D_1D_2$100000

G4 = $D_1D_2D_3$10000

G5 = $D_1D_2D_3D_4$1000

G6 = $D_1D_2D_3D_4D_5$100

G7 = $D_1D_2D_3D_4D_5D_6$10

G8 = $D_1D_2D_3D_4D_5D_6D_7$1

G9 = $D_1D_2D_3D_4D_5D_6D_7D_8$

用软件实现逐次逼近转换的接口电路如图 15-33 所示。

其中，电压比较器 LM311 是构成 A/D 转换器的主要环节之一，它的分辨率、稳定度、动作速度等指标直接影响着 A/D 转换器的性能。当比较器正端电压大于负端电压时，LM311 的输出为“1”电平，反之，当比较器正端电压小于负端电压时，LM311 的输出为“0”电平。可见电压比较器的输入为模拟量，输出为数字量。这样，比较器就成为模拟量转换为数字量的媒介。

该电路的工作原理是：先由微型计算机输出一个 8 位二进制数经 8255A 的端口 PB 给 D/A 转换器 AD1408。AD1408 的输出与待测量的模拟输入电压相比较，如果比较器 LM311的输出为“0”，说明 D/A 转换器的输出电压比模拟输入电压高，即输入到 AD1408 的二进制数比

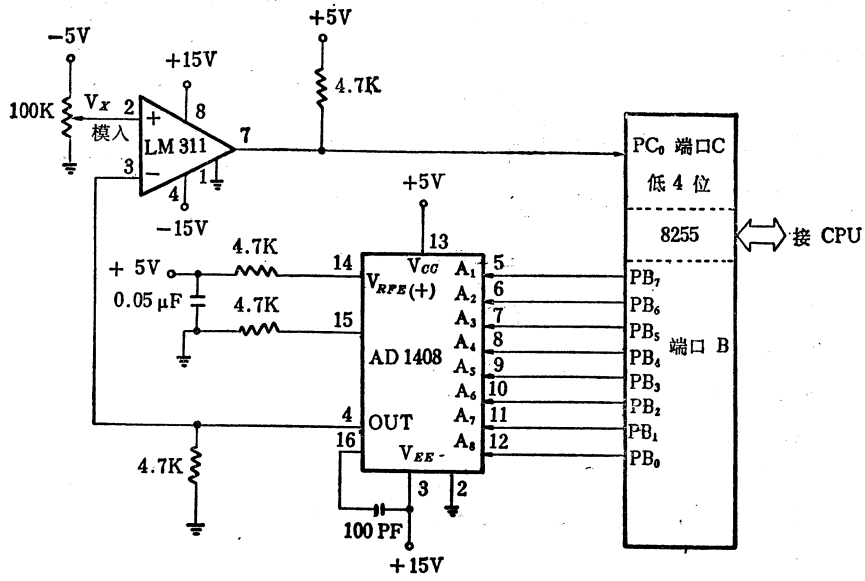


图 15-33　用软件实现的逐次逼近转换的接口电路

模拟输入电压所对应的二进制数值低，所以必须增加 D/A 转换器输入端的二进制值；若比较器输出为“1”，说明 D/A 转换器输出电压比模拟输入电压低，所以必须减少 D/A 转换器输入端的二进制值。

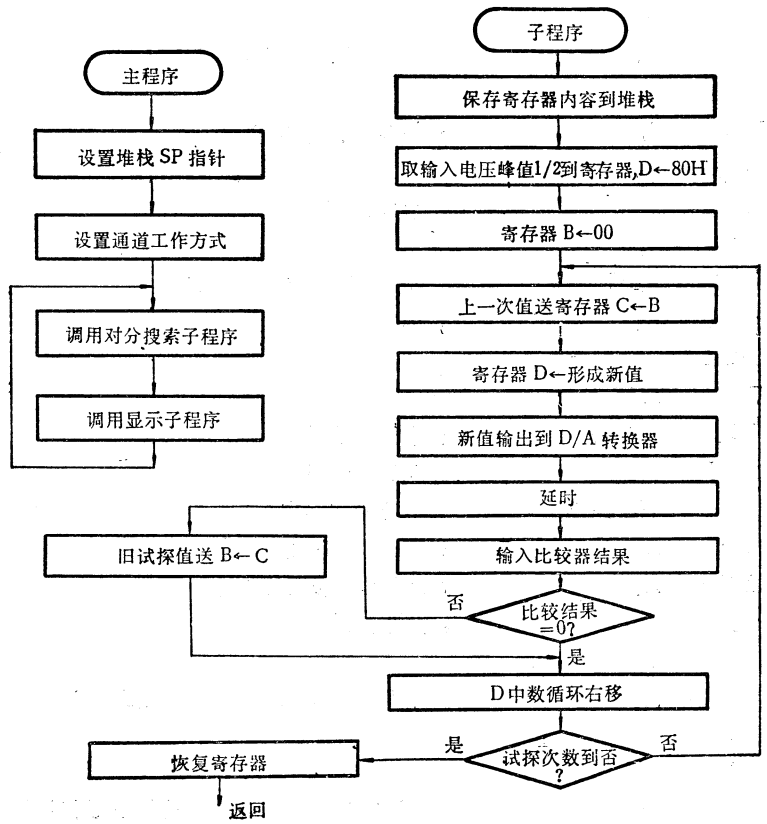


图 15-34　由软件实现A/D转换的流程图

应当指出，上述所指的电压的高或低，都是相对于输入模拟电压本身是负，DAC 的输出电压也是负而言的。

当采用对分搜索法实现软件逐次逼近时．先由输入电压最大值的二分之一与输入电压比较，比较结果可缩小搜索范围，在这 $\frac{1}{2}$ 中再做对分搜索又可缩小 $\frac{1}{2}$ 的搜索范围。依次类推，逐次比较直至相等为止。对 AD1408 而言，其输入与输出的传递特性为

$$V_{OUT} = -5 \times \left(\frac{IN}{256}\right) \quad (伏)$$

图 15-34 是用软件实现 A/D 转换的流程图。

用 Intel 8080A 汇编语言写出源程序如下：

对分搜索法主程序

```
        ORG   2000H
        LXI   SP,   20C2H   ; 设置 SP 指针
        MVI   A,    INITM   ; 8255A 初始化
        OUT   CTLPRT
MAIN;   CALL  SAAD          ; 调用对分搜索子程序
        CALL  UPDDT【注】    ; 调用显示子程序
        JMP   MAIN          ; 循环
```

对分搜索法子程序

```
SAAD:   PUSH  B             ; 保存 CPU 状态
        PUSH  D
        MVI   D,    80H     ; 从最高位开始，D←80H
        MVI   B,    00H
NXTBIT; MOV   C,    B       ; 保存旧的猜测值
        MOV   A,    B
        ORA   D             ; 形成新值
        MOV   B,    A
        OUT   DAC           ; 把新值送 D/A 转换器
        MVI   A,    02H     ; 延时
TIM;    DCR   A
        JNZ   TIM
        IN    COMP          ; 取比较结果
        ANI   01H           ; 判比较结果
        JZ    HGUESS        ; 若比较器输出为"0"，则猜测值太高(绝对值太低)
        MOV   B,    C       ; 若比较器输出为"1"，则猜测值太低(绝对值太高)
HGUESS: MOV   A,    D       ; 修改数位位置的指针
        RRC                 ; 循环右移
        MOV   D,    A
        JNC   NXTBIT        ; 无进位，8次未到转 NXTBIT
        MOV   A,    B
```

【注】显示子程序 UPDDT 可以调用系统中现成的程序，如 SDK-85 单板机中的显示子程序 UPDDT 入口地址为 036 EH，也可以另行编制。

```
        POP   D
        POP   B
        RET
DAC:    EQU   35H
COMP:   EQU   36H
CTLPRT: EQU   37H
INTTM:  EQU   91H
```

2．硬件逐次逼近转换

当需要高速A/D转换时，对分搜索算法可以由微处理器外部的硬件来实现。用硬件来执行这个转换的方法之一是使用2502逐次逼近寄存器(SAR——Successive Approximation Register)如图15-35所示。当使用外部时钟、一个D/A转换器和一个模拟比较器时，2502能执行所有逐次逼近转换所需的逻辑功能。用这个硬件转换方法，其转换速度大约比上述软件转换的速度快100倍。

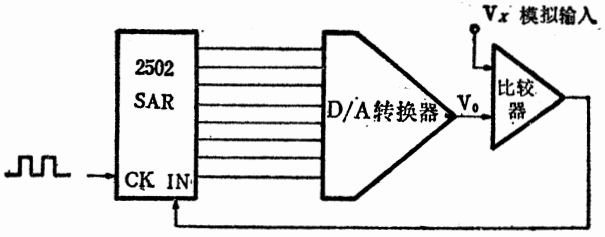


图15-35　硬件逐次逼近式转换示意图

这里，转换原理与上述软件转换相同，只不过用SAR代替软件逐次逼近算法程序而已。最后由SAR输出的即为A/D转换后的数字量输出。这种A/D转换的精度主要取决于SAR的位数。位数愈多，精度愈高，但转换时间愈长。

3．CPU与8位A/D转换器的接口

这里以常用的ADC 0809为例说明其与CPU接口的原理。

(1) ADC 0809的组成及引线

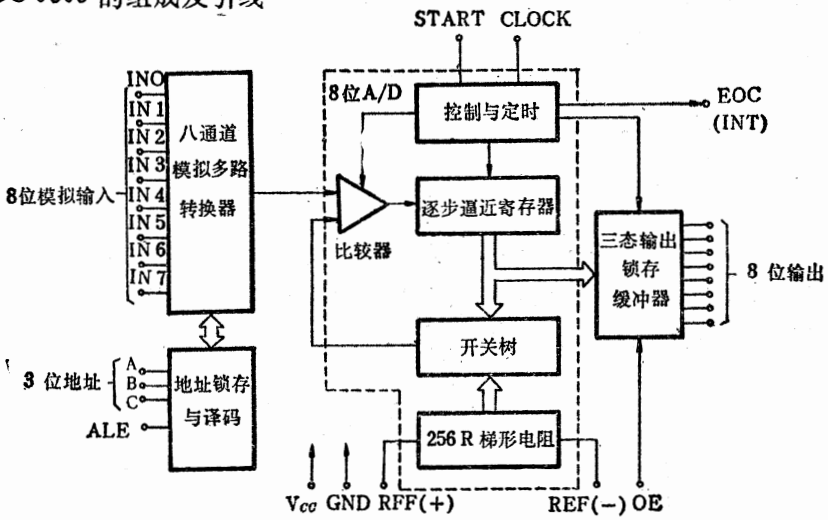


图15-36　ADC 0809结构框图

ADC 0809 是用 CMOS 工艺制成的 8 位单片 A/D 转换器，它采用双列直插式 28 引线封装。片内包括一个 8 路模拟开关，一个模拟开关的地址锁存与译码电路，一个比较器，一个 256R 电阻解码网络(梯形电阻网络)，一个树状电子开关电路，一个逐次逼近寄存器，一个控制与时序电路以及一个三态输出锁存缓冲器。如图 15-36 所示。

ADC 0809 的引线如图 15-37 所示。

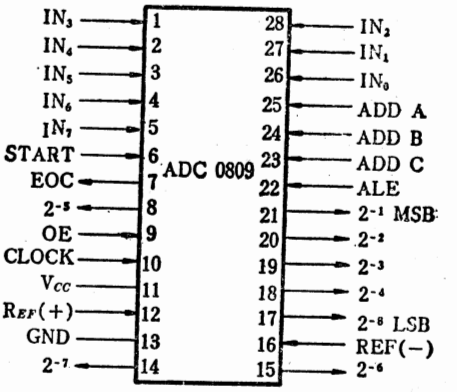


图 15-37 ADC 0809 的引线图

各引线说明如下：

① $IN_7 \sim IN_0$——模拟量输入端，可直接与 8 路模拟量连接。

② 2^{-1}MSB～2^{-8}LSB——A/D 转换器的数字输出端，分别从高位 D_7 到低位 D_0。

③ START——转换启动控制端，当其有效时，转换器即开始转换。

④ EOC(END OF CONVERSION)——转换结束脉冲输出端，当其有效时，表示转换结束，CPU 可读入数据。

⑤ OE(OUTPUT ENABLE)——输出允许控制端，当其有效时，将输出缓冲器中的数据送至数据总线上。

⑥ CLOCK——时钟输入端。

⑦ V_{CC}——电源输入端 + 5V。

⑧ $R_{EF}(+)$ $R_{EF}(-)$——参考电源输入端。

⑨ GND——地线。

⑩ ADD A, B, C——8 路模拟开关的 3 位地址线。

⑪ ALE——地址锁存控制端，当其有效时，地址锁存与译码器即选择指定的通道。

(2) 主要性能指标

① 分辨率——8 位；

② 总的不可调误差——±1/2LSB 和 ±1LSB；

③ 转换时间——100 μs；

④ 具有锁存控制功能的 8 路模拟开关，可对 8 路模拟电压分时进行转换；

⑤ 可用单 5V 电源供电，此时模拟输入范围为 0～5V，不需要外部的调零和满量程调整；

⑥ 输出与 TTL 兼容。

(3) 工作原理

① 多路开关——芯片中有一只 8 路单端模拟信号输入的多路开关，通过 ADD A, B, C (芯片 25、24、23 引线)三端地址译码选通 8 路中的一路进行 A/D 转换。

地址译码与输入选通的关系如表 15-6 所示。

表 15-6

被选通模拟通路		IN_0	IN_1	IN_2	IN_3	IN_4	IN_5	IN_6	IN_7
地	A	L	H	L	H	L	H	L	H
址	B	L	L	H	H	L	L	H	H
线	C	L	L	L	L	H	H	H	H

② 转换器——由三部分组成，们是 256R 电阻解码网络，逐次逼近寄存器和比较器。

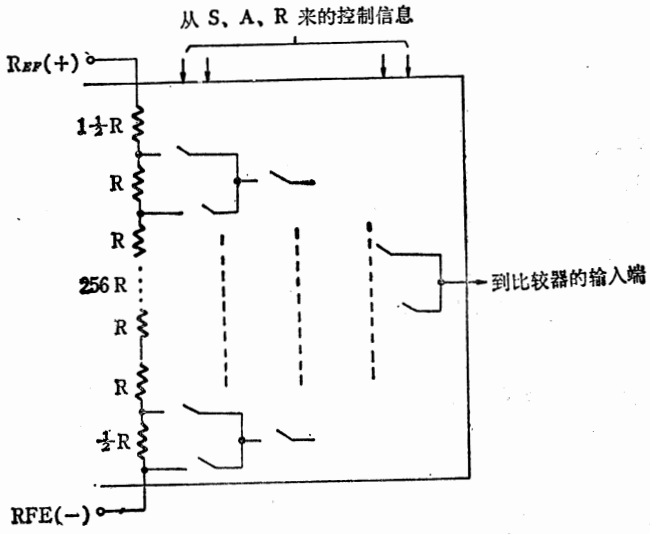


图 15-38　电阻网络及开关

电阻解码网络——该芯片的电阻解码网络实际上是一个 256 R 的电阻分压电路（见图 15-38）。

它与树状电子开关组成 D/A 转换器。

逐次逼近寄存器 S．A．R——提供 8 位迭加的数字量去逐次逼近模拟输入电压，以完成 8 位 D/A 转换。

比较器——它是转换器的关键部件，对最后的转换精度起决定性的影响。

多路开关的输出接到内部比较器的一个输入端，开关树的输出接到比较器的另一个输入端。被转换的模拟电压同来自开关树的电压在比较器中进行比较。这里，开关树的输出模拟电压由 S．A．R 的输出来控制。这种比较共进行 8 次，每次确定 8 位数字中的某一位是“1”或是“0”。8 位数字的判别顺序是由高位到低位，即第一次比较并确定 D_7，第二次比较并确定 D_6 位，…第 8 次比较并确定 D_0 位。

每次比较需要 8 个时钟周期，8 次比较总共需要 64 个时钟周期。当 ADC 0809 时钟为 640kHz 时，时钟周期为 1.5625μs，转换时间为 $1.5625\times 64=100(\mu s)$。转换结果送到“三态输出锁存器”。在下一个模拟量转换之前，“三态输出锁存器”中的数据随时可被 CPU 读出。“三态输

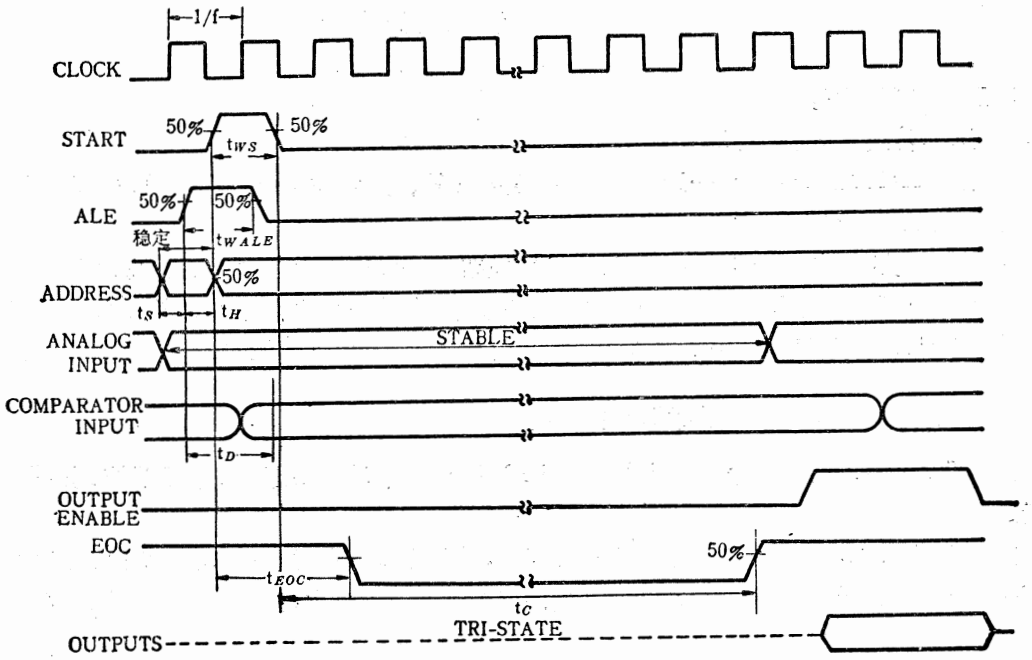


图 15-39　ADC 0809 的时序图波形图

出锁存器”可以直接连接到有关的系统总线上。

(4) 转换时序

ADC 0809 的转换时序如图 15-39 所示。

图中：f_C——时钟频率 640kHz；

t_{WS}——最小的启动脉宽 100～200 ns；

t_{WALE}——最小的 ALE 脉宽 100～200 ns；

t_S——最小的地址建立时间 25～50 ns；

t_H——最小的地址保持时间 25～50 ns；

t_D——从 ALE 开始的模拟开关延时时间为 1～2.5 μs；

t_C——转换时间 100 μs；

t_{EOC}——转换结束延迟时间 (8 个时钟周期 + 2μs)。

ADC 0809 典型的接口方法如图 15-40 所示。

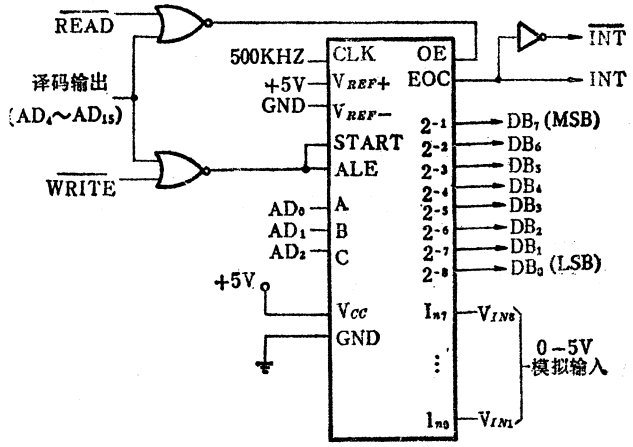


图 15-40 ADC 0809 典型的接口方法

其工作过程如下：

① 选择转换通道(见表 15-6)。

② 给 START 和 ALE 加一正脉冲，转换立即开始。此脉冲可由程序启动，也可以由 Z80-CTC 的输出脉冲(ZC/TO)启动。

③ 在转换过程中，EOC 为低电平，一旦转换结束，EOC 即由低电平变为高电平。

④ CPU 给OE端发出一个正脉冲，打开“三态输出锁存器”，将转换结果输至数据总线上。

⑤ COMPARATOR INPUT 由 ADC 0809 内部产生，无须外部触发。

五、用 A/D 转换器组成的数据采集系统

应用 TP 801-Z80 单板微型机上的 Z80 PIO 和 ADC 0809 组成数据采集系统，如图 15-41 所示。

现要求从 ADC 0809 的 0 通道输入一个 0～5V 的模拟电压，连续采样 256 次，每次转换得到一组 8 位二进制数，存入从 2200H 开始的 RAM 区。参考电压 $V_{REF}(+)=V_{CC}$；$V_{REF}(-)$ 接地。

图 15-41 中，除 TP801-Z80 单板机和 ADC 0809 外，还需要单稳态电路 DW (如 74LS123)、

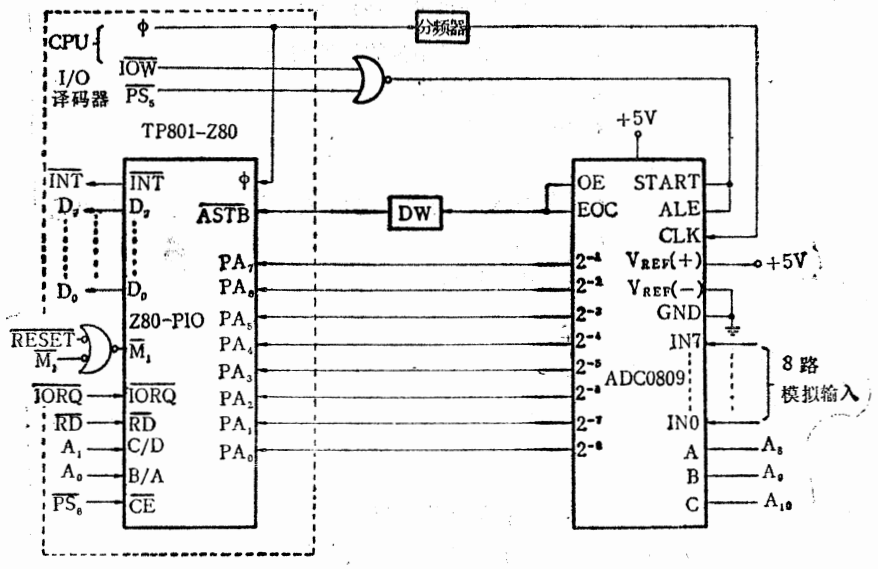


图 15-41 用 A/D 转换器组成的数据采集系统

分频器(如 74LS74)、或非门(74LS02)及与门(74LS08)。单稳态电路(74LS123)用于将 ADC 0809 的每次转换结束时发出的 EOC 的状态信号变换为合适的负脉冲信号，送 Z80 PIO A 端口的 $\overline{ASTB}$。分频器由双 D 触发器 74LS74 组成，用于将 TP801-Z80 的主频 φ(2MHz)进行 4 分频后作为 ADC 0809 的时钟信号。74LS02 或非门用于形成 ADC 0809 的 ALE 和 START 的工作脉冲信号。

在数据采集系统的工作过程中，将选择通道地址装入 B 寄存器，以便在执行 OUT (C), A 指令时，将选择通道地址放至与 ADC 0809 的 A、B、C 相连的 A_8、A_9、A_{10} 上。用程序启动 START 和 ALE 后，ADC 0809 即开始转换，转换结束后，EOC 由"低"变"高"，由于 OE 与 EOC 相连，因此打开三态输出锁存缓冲器。EOC 经单稳电路 DW 变换为负脉冲送入 PIO 的 $\overline{A\ STB}$ 端，通过 $\overline{INT}$ 端向 CPU 发出中断请求；如果 CPU 开放中断，则响应中断后，转入执行中断服务程序，将转换结果读入 CPU。

当 $V_{IN}=0$ 伏时，A/D 转换器输出为 00H；

当 $V_{IN}=V_{REF}$ 时，A/D 转换器输出为 FFH。

顺便指出，ADC 0809 不必调零，也不需要满量程调节。由于其接地点仅有一个，故应将它的 GND 与系统的数字地相接。

数字采集程序如下：

```
ORG   2000H
LD    A,     23H    ; 设置中断矢量高 8 位
LD    I,     A
LD    A,     40H    ; 设置中断矢量低 8 位
OUT   (9AH), A
LD    A,     4FH    ; 设置 PIO A 端口为输入工作方式
OUT   (9AH), A
LD    A,     83H    ; 设置 PIO A 端口允许中断
OUT   (9AH), A
```

```
        LD      D,    0FFH    ; 设置每个通道采集数据的次数
        LD      HL,   2200H   ; 设置存放转换结果的地址指针
        LD      B,    00H     ; 通道0开始转换
        IM      2             ; 中断方式2
        LD      C,    94H
UP:     OUT     (C),  A       ; PS5 有效,B 寄存器内容放至 A15~A8 上,转换开始
        EI                    ; 开中断
        HALT                  ; 等待中断请求
        JP      UP
```

中断服务程序入口地址表

```
2340H  00
2341H  24
```

中断服务程序

```
2400H  IN      A,      (98H)      ; 读转换结果
       LD      (HL),   A          ; 转换结果存入 RAM 区
       INC     HL                 ; 地址指针加1
       DEC     D                  ; 转换次数减1
       JR      Z,      DONE-$     ; 256 次转换结束?
       RETI
DONE:  HALT                       ; 数据采集完毕。
```

§15-5 盒式磁带机 (Casstte tape unit) 及其与 CPU 的接口

盒式磁带机是一种利用录音磁带串行记录二进制的数字信息，并能把已记录在磁带上的信息读出的装置。磁带是用塑料(聚脂树脂)制成的长带，在这长带表面涂着一层磁性材料作为记录磁层，它具有能记录二进制信息的性能。其主要特点是存贮容量较大、具有非易失性、体积小、价格低等，但其出错率较高。

为了降低微型机成本，通常选用家用的磁带录音机作为盒式磁带机。本节先简要介绍磁带录音机的工作原理，然后以 Z80 STARTER KIT 单板机与磁带机的接口为例来加以说明。

一、录音机的工作原理

录音机是应用声和电的转换，以及电和磁的转换原理工作的。图 15-42 是其结构示意图。它由机械传动装置、音频放大器和磁头三部分组成。机械传动部分包括电动机和机械连动机构；音频放大电路包括录音、放音放大器和超音频振荡器等电路；磁头有抹音磁头和录放两用磁头。

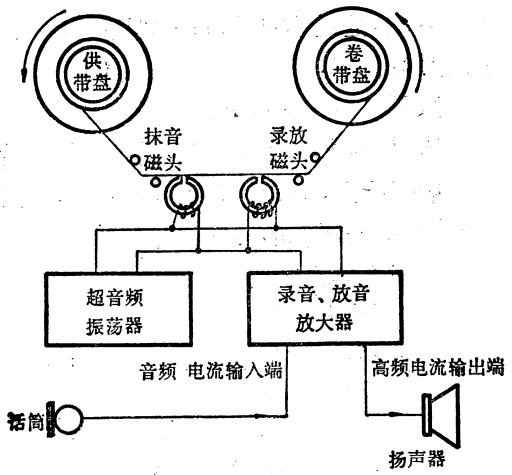


图 15-42 录音机结构示意图

录音的工作过程如图 15-43 所示。

声音经过话筒变为音频信号电流，然后把它送入录音放大器进行放大。放大了的音频信

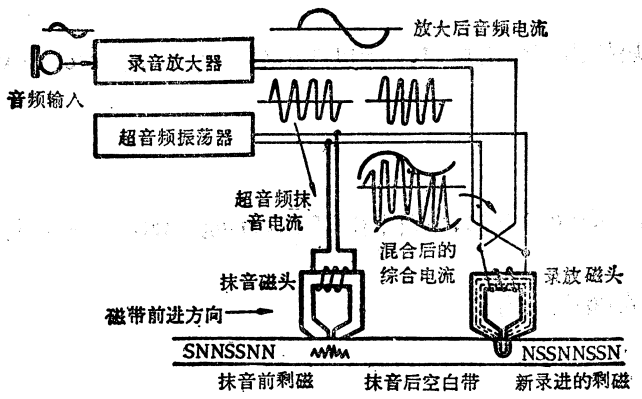


图 15-43　录音原理示意图

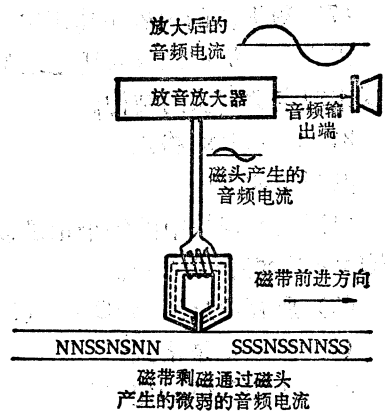


图 15-44　放音原理示意图

号电流通过录音磁头线圈，便产生了随音频电流变化的磁场。由于磁带与录音磁头的缝隙紧贴在一起，而又不断地向前移动，变化的磁场就通过磁头缝隙前的磁带而形成回路，这样就把这一段磁带磁化而把声音录了下来。在录音时还要给录音磁头加上一个超音频偏磁电流，它的作用是使录下来的声音失真小、杂音低。图 15-43 中磁带上的 NS 符号表示录下的信息。

放音的工作过程如图 15-44 所示。

已录好音的磁带，按录音时同样的速度紧贴着放音磁头缝隙前进，由于磁头铁芯对磁场的阻力比磁头缝隙空气的阻力要小得多，因此磁带上所录下的音频剩磁磁力线容易通过铁芯而形成回路，这个随剩磁大小变化的磁力线切割放音磁头的线圈，就产生原来录音时的音频电流。这个电流经过放大器放大，再送到扬声器，于是就放出原来的声音。

二、磁带记录的标准

用于盒式磁带上记录数据的格式必须遵守两个标准：

(1) 用 KC(Kansas City)标准来规定记录的信息是“1”还是“0”。

(2) 用 Intel Hex 格式来表示数据块记录的格式。

下面分别加以说明：

1. KC 标准

KC 标准是 1975 年 11 月在美国堪萨斯城由 BYTE 杂志主持的专题讨论会上制订的。这次会议的目的是在制造商间制订一个音频盒式磁带记录技术的标准。

其要点如下：

(1) 用频率为 2400Hz 的 8 个周期表示逻辑 1(Mark)。

(2) 用频率为 1200Hz 的 4 个周期表示逻辑 0(Space)。

(3) 一个字符的构成格式为：一个逻辑 0 的起始位、7 或 8 个数据位和 1～2 个终止位(Z80 STARTER KIT 用 7 位 ASCII 数据字符和 2 个终止位)。

(4) 7 个 ASCII 数据位的次序是：最低有效位在先，最高有效位在后。

(5) 在整个数据块前至少有 30 秒的空闲位作为导引段，数据块结束后至少有 5 秒的空闲位作为结尾段。

(6) 数据速率为 300 Baud (即每位宽 3.33 ms)。

(7) 数据块的内容不作规定。

因为 KC 标准没有对所记录的数据块的内容作出规定，所以选用另一个标准 Intel Hex 格式来规定数据块的结构。

2. Intel Hex 格式

其规定如下：

(1) 在数据块中的每个记录以冒号：(Colon)开始，以回车 CR(Carriage Return)和换行 LF(Line Feed)作为结束。

(2) 所存信息用 ASCII 码(7 位，无奇偶位)记录。

(3) 数据记录格式：

字节 1 Colon(:)定义符

字节 2～3 规定该记录中的 ASCII 字节数，最多为 32 个 ASCII 字节。

字节 4～5 该记录起始地址的高位字节。

字节 6～7 该记录起始地址的低位字节。

字节 8～9 ASCII 码的零

字节 10～ 自第 10 个字节开始为用 ASCII 码表示的数据字节，在一个记录内最多为 32 个 ASCII 字节。

最后两个字节存放的是除定义符、回车符和换行符以外的所有字节的校验和(Checksum)。这个校验和是记录中的所有二进制数的和的负数。

每一个记录以回车和换行符作为结束标志。

(4) 文件结束记录的格式：

字节 1	冒号(:)定义符
字节 2～3	ASCII 码零
字节 4～5	ASCII 码零
字节 6～7	ASCII 码零
字节 8～9	记录类型 01(ASCII 0, ASCII 1)
字节 10～11	校验和

三、磁带机与 CPU 的接口

Z80 STARTER KIT 用软件控制的方法通过 Z 80 CPU 和 Z 80 CTC 形成一个软件 UART，以完成串行变并行和并行变串行，以及发送和接收数据等功能。故其接口电路比较简单，如图 15-45 所示。

现分两部分说明：

1. 盒式磁带转贮(Dump)

该接口由 Z80 CTC 和录音机接口电路组成，如图 15-45 所示。

所谓磁带转贮，即指信息由 CPU 写入磁带的过程。磁带上记录的“1”或“0”是以不同频率的信号来表示的，而不同频率的信号是利用 Z80 CTC 通道 1 来产生的。

写入磁带的过程如下：

(1) 对CTC 的通道 1 进行初始化编程，确定写入磁带的信息块的起始和终止地址等以及设立 40 秒的导引段。

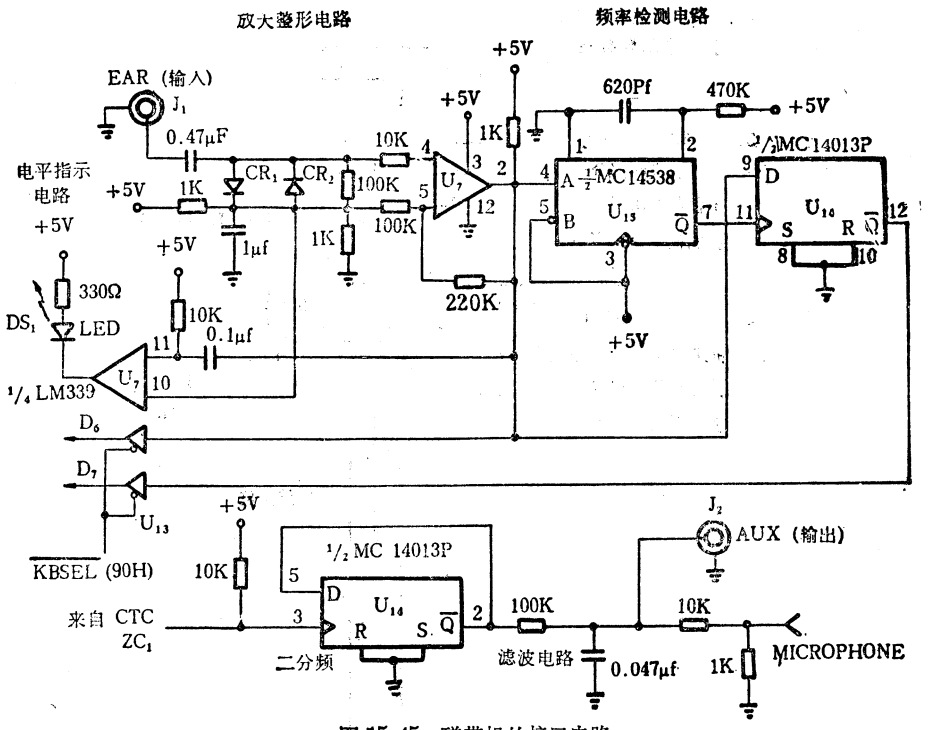


图 15-45　磁带机的接口电路

(2) "写磁带"程序将需要输出的二进制数转换成 ASCII 码。即将每个十六进制数折成两部分：高 4 位表示一位数，低 4 位表示另一位数，然后把每位数变为相应的 ASCII 码。

(3) 把每个 ASCII 码另加 1 位起始位、2 位终止位，共计 10 位，形成写磁带的一个数据字节，然后把每个字节转换成具有标准记录格式的串行数据形式。

(4) 按 Intel HEX 格式的规定组织数据块，先要输出冒号，然后比较数据块的长度是否比一个记录(最大为 32 个 ASCII 字节)长，若长，则按 Intel HEX 格式组织数据块并输出；若短，则检查是否是文件的结尾段，若是则按文件结束标志输出。

(5) 在写磁带时，Z80 CTC 向 CPU 请求中断(按方式 2)，当 CPU 响应后就转入中断服务程序。该中断服务程序先把要输出的周期数("1"——16 个周期；"0"——8 个周期)减 1，然后检查是否为 0，若不为"0"则开中断，返回。即表示正在输出"1"或"0"的过程中，直至完成应输出的周期数为止。若已为"0"，则表示一个数位输出已经完成，就继续输入下一个数位，根据它是"1"还是"0"，重新设置 CTC_1 的方式控制字和时间常数(决定输出频率)，开始一个新的数位。若输出为"1"，则通过 CTC_1 变为频率为 4800Hz 的 16 个周期波(在 U_{14} 中二分频为 8 个周期的频率为 2400 赫的信号)；若输出为"0"，则通过 CTC_1 变为频率为 2400Hz 的 8 个周期波(在 U_{14} 中二分频为 4 个周期的频率为 1200 赫的信号)。

(6) 得到的表示 1 和 0 的对应频率的方波，经滤波电路滤掉方波的高频率部分后，经由 J_2 送到录音机。

下面举例说明将内存中的十六进制数写入磁带的过程，如图 15-46 所示。

2．盒式磁带装入(Load)

其接口电路如图 15-45 所示。

这种操作方式用于将盒式磁带中的信息输入到微型计算机的 RAM 中，此时磁带机的

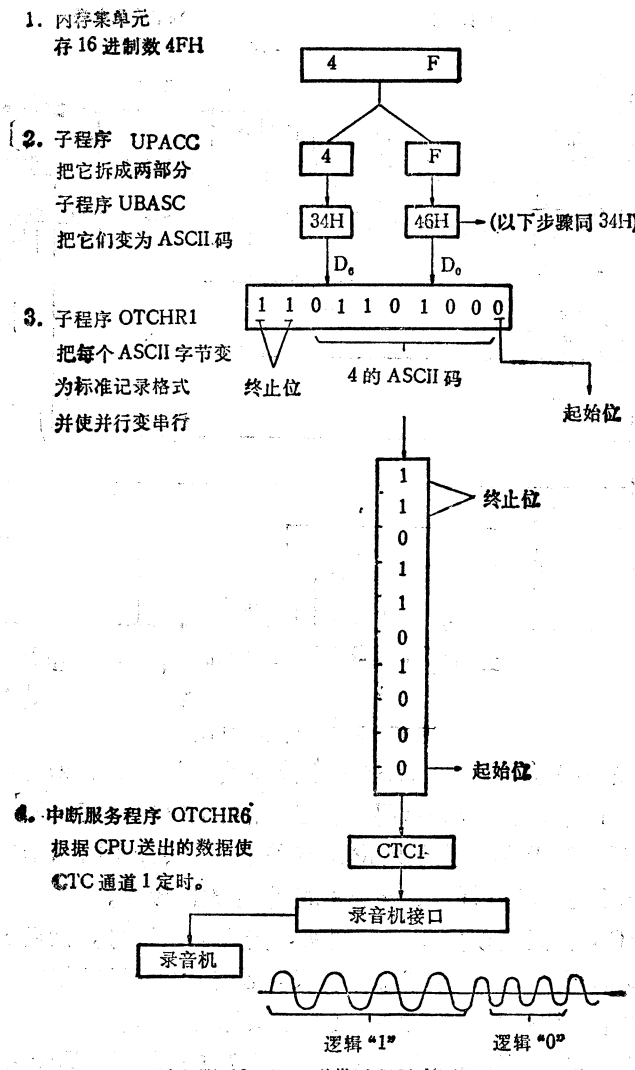


图 15-46 写入磁带过程示意图

EAR (Earphone) 插座用一根音频软线连接至单板机的 J_1 处。U_7 是一个限幅和整形电路的组合，它用于将输入信号进行限幅和放大处理后，输出一个不畸变的方波给 U_{15}。从磁带机来的波形的峰值必须在 2V 左右。U_7 的另一部分是一个发光二极管 LED 的驱动器，当 U_{15} 接收到正确的信号幅度时，它就发光显示。U_{15} 的一半和 U_{14} 形成一个频率检测器，用于分辨 1200 Hz ("0") 和 2400Hz ("1") 两种频率。当 U_7 输出 1200 Hz 脉冲 ("0") 时，U_{14} 的 $\overline{Q}$ 输出为"0"；当 U_7 输出 2400Hz 脉冲 ("1") 时，U_{14} 的 $\overline{Q}$ 输出为"1"。这样，U_{14} 的 $\overline{Q}$ 输出解调了数据序列，由 CPU 通过三态缓冲器 U_{13} 选通读入 Z80-CPU 的数据总线 D_7 上，并通过软件拼装为 ASCII 码和转换成二进制数，去掉记录格式中其余非数据字节，然后送到内存保存起来。对该单板机而言，原数据从哪个首地址写入录音机，则从录音机送出时，也是从记录中的该首地址处开始放入内存。

装入磁带的过程可归纳如下：

(1) 当 CPU 从磁带上读取信息时，磁带上记录的"1"或"0"由硬件加以鉴别后，将结果送至数据总线的 D_7 上。

(2) 在装入磁带主程序中先设立15秒的导引段，然后调用从磁带上输入字符的子程序INCHR。

(3) 子程序 INCHR 输入数据，检查是否起始位，若是，利用 CTC_3 作为延时，经延时半位后再检查是否起始位，当确信为起始位后，每隔一位的时间输入数据，经过移位把串行的变为并行的，并去掉起始位和终止位，保留在A中。

(4) 在输入一个记录时，先检查是否为冒号，若不是则重复上述过程，直到查到冒号后，才是记录开始，然后接 Intel HEX 格式输入数据。

(5) 通过调用输入两个相邻字符的子程序，将 ASCII 码转换为二进制数，并拼装成一个字节，然后送内存保存起来。

(6) 为了检查所记录的数据是否正确，程序对所记录的数据字节经累加后形成"校验和"。若录、放过程中的"校验和"不相符，则由程序控制显示器向用户报告出错。

四、盒式磁带的转贮和装入流程框图

Z80 STARTER KIT 系统采用硬件、软件相结合的方法，巧妙地实现了盒式磁带的转贮和装入，其示意图如图 15-47 所示。

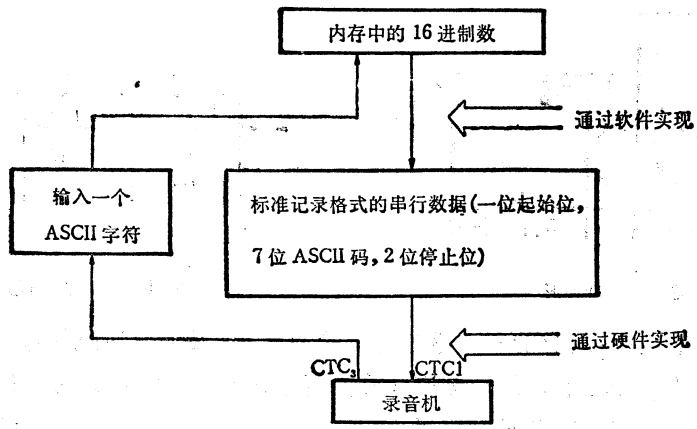


图 15-47 磁带转贮和写入示意图

Z80 STARTER KIT 系统盒式磁带转贮和装入的流程图和程序清单，可参阅 Z80 STARTER KIT 技术手册。

§15-6 软磁盘 (Floppy disc) 及其与 CPU 的接口

磁盘是一种表面涂上磁性材料的园盘片，与磁带相似，它靠磁性材料的磁化方向来存贮信息。磁盘在存贮容量、存取速度及可靠性等方面都比磁带优越，价格也不算高，是微型计算机系统中常用的一种外存贮器。它可用来存放软件，特别是操作系统，也可用于存放用户产生的文件。

磁盘有两大类：软磁盘和硬磁盘。目前常用的是软磁盘。

一、软磁盘

微型计算机使用的软磁盘有两种规格：标准软盘(8″盘)和小软盘(5″盘)。目前常见的一

表 15-7　软盘参数和性能表

软盘		每面磁道数	每道区段数	存贮字节	写保护口
5″盘双面单密度		35	18	181K	写保护时缺口贴片
8″盘	单面单密度	77	26	256K	与5″盘相反
	单面双密度	77	52	512K	

些软盘参数和性能如表 15-7 所示。

两种规格软磁盘的外型如图 15-48 所示。它们均套装在一个塑料(或纸质)的保护套内。塑料套有三个孔：一个长形孔是供读写磁头寻找磁道用的；一个中心大孔是供驱动电机旋转磁盘用的；第三个小孔是检索孔，作为检查起始位置标记用的。塑料套的边缘还有一个写保护缺口，若用纸带贴死此缺口，则 5″ 磁盘被保护起来，不能再写入信息(对 8″ 磁盘则相反)。另外，在塑料套上通常都有标明磁盘规格型号的标牌。

磁头在磁盘上读写信息的原理与磁带机相似。每一个数据位都以一个启动脉冲开始，如数字是 1，则在此数据单元的中央加入一个脉冲；如数字是 0，则不加入脉冲，其波形见图 15-49 数字信息波形。

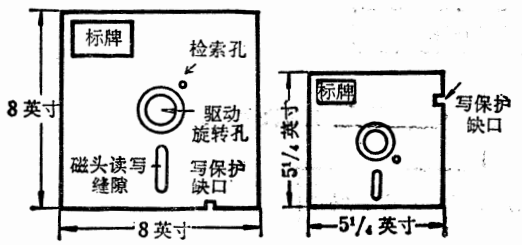


图 15-48　软磁盘外形

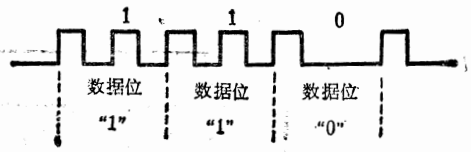


图 15-49　数字信息波形

由于磁盘是圆形结构，因此数据在磁盘上存贮的方式与磁带有所不同，而与唱片相似，通常是把一个磁盘分成若干磁道。每一磁道又分为若干区段 (Sector)。每一区段记录一定字节数的数据信息。例如，TRS-80 微型计算机系统的 5″ 软磁盘，每一个磁盘有 35 个磁道，每一磁道又分成 10 个区段，每一区段可记录 256 字节的数据。图 15-50 标出了此种磁盘记录数据的格式。

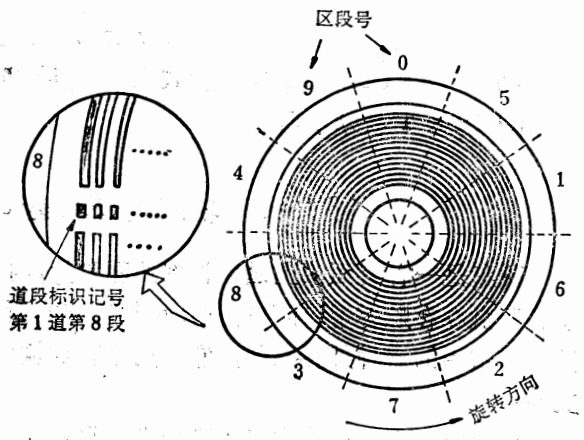


图 15-50　磁盘记录数据的格式

二、软磁盘驱动器的结构和工作原理

CPU 与软磁盘之间必须通过磁盘控制器与磁盘驱动器进行接口。当要进行磁盘操作时，CPU 并不直接控制磁盘操作，而是以命令的形式发送给软盘控制器，由软盘控制器产生若干控制信号送给软盘驱动器。然后，由软盘驱动器将软盘控制器送来的控制信号转换成驱动软盘的各种电气和机械的动作，驱动软盘完成 CPU 命令所要求的操作，同时，软盘驱动器还将软盘的现行状态传送给软

盘控制器，作为 CPU 或软盘控制器正确控制软盘操作条件。

软盘驱动器由读写磁头、步进电机(控制磁头移动到某一磁道上读取信息)、驱动电机(旋转磁盘)以及控制读写和电机旋转速度的逻辑电路等部分组成。图 15-51 是小型软磁盘驱动器的外形图。工作时，软磁盘盘片从面板上的狭缝插入驱动器。图 15-52 是驱动器内部结构的示意图。下面分别加以说明。

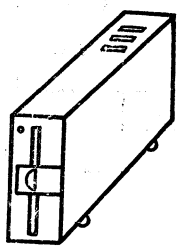


图 15-51　软盘驱动器的外形

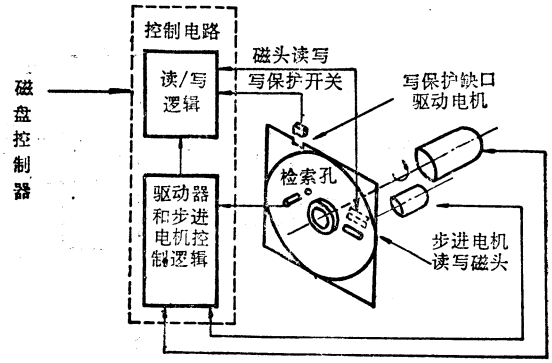


图 15-52　软盘驱动器结构示意图

1．驱动电机和步进电机

驱动电机用来旋转磁盘，它从启动到磁盘旋转稳定约需 1 到 2 秒，工作时要求转速稳定。另一个是步进电机，它通过齿轮、齿条机构带动磁头沿着磁盘径向移动，以找到需要读写的磁道。这两个电机的工作都受磁盘控制电路的控制。

2．磁盘控制电路

该电路包括读/写逻辑、驱动电机和步进电机的控制逻辑。读/写逻辑控制磁头产生磁头读写所需要的电流。当磁盘上的写保护缺口贴死时，写保护开关把这一状态信息送入读/写逻辑，读/写逻辑就禁止写电流通过磁头，使信息不能写入磁盘。驱动电机和步进电机控制逻辑通过检索小孔检测磁道(或区段)的起始位置。再经过接口线路送入计算机，计算机通过软件加以识别，并通过接口线路发出控制命令，此控制命令经控制逻辑电路控制驱动电机和步进电机工作。

现在把软磁盘的工作概括如下：

首先接通电源，驱动电机带动磁盘旋转，约 1 到 2 秒达到稳定转速(例如 300r/min)后，计算机就可以对磁盘存取信息了。计算机发控制命令，通过磁盘控制电路使步进电机转动，把磁头移到所需读写的磁道。同时控制电路又通过检索小孔检测磁道(或区段)的起始位置。一旦找到需要的磁道和区段号后，计算机就可以通过读写逻辑控制磁头从磁盘上存取数据。

三、软磁盘控制器 FDC(Floppy Disc Controller)

软磁盘是由磁盘驱动器驱动和控制的，而软磁盘驱动器必须经过软磁盘控制器与 CPU 接口。

软磁盘接口线路应具有下列功能：

1．提供磁盘读写逻辑、驱动电机和步进电机所需要的控制信息，并能存取数据。例如，它能把软磁盘中的串行信息转换成并行信息送入 CPU 或把来自 CPU 的并行信息转换成串行信息写入软磁盘；寄存读出或写入的磁道地址、区段地址；进行磁头定位、标志磁盘运行状态(检索磁道与区段的位置，标志出错信息等)；产生循环冗余校验码 (CRC——Cyclic Redundancy

check)，进行出错校验运算等。

2．把软磁盘控制器和 CPU 系统总线相连，进行接口地址译码，传送数据以及传送 CPU 对软磁盘控制器的控制命令等。图 15-53 是软磁盘与微型机的接口方框图。

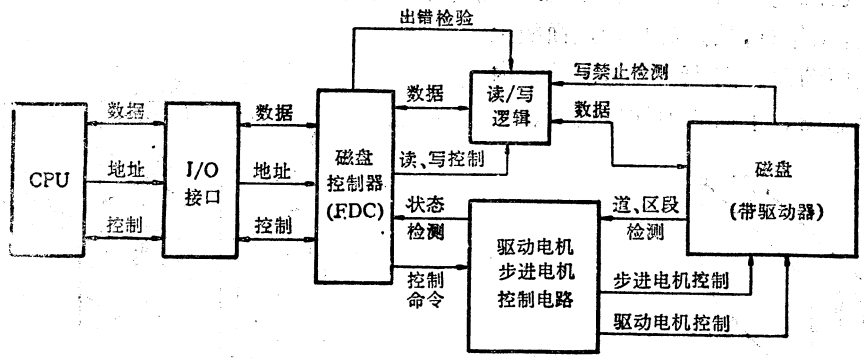


图 15-53　软磁盘与微型机的接口方框图

在该接口方框图中，其读写操作步骤如下：

(1) 读操作

① 把磁道号送入磁道寄存器；

② 发出找磁道的命令；

③ 等待找到所需要的磁道；

④ 软磁盘控制器向 CPU 发中断请求，CPU 响应中断，读入磁盘信息；

⑤ 进行 CRC 校验。

(2) 写操作

①、②、③步骤同读操作步骤；

④ 发一个写命令；

⑤ CPU 在收到发送数据请求应答信号后，送出第一个数据；

⑥ 送其余数据；

⑦ 送循环冗余校验码 CRC。

上述读、写操作是对硬区段磁盘而言的，对软区段磁盘应按其规定的记录格式进行读写操作。

近年来，已陆续研制成单片软盘控制器，例如 Intel 8271、Motorola 6843 等。它们除了提供上述控制功能外，还能通过编程改变步进电机的步进速率，改变数据传输速率和数据传输方式(采用 DMA 传送或程序传送)等。

§15-7　CRT 显示器及其与 CPU 的接口

CRT 显示器能将计算机的数据转换为各种直观的图象和字符，并在萤光屏幕上显示出来。它具有速度快、无噪音、无机械磨损、使用简便、可靠性高等优点，因而在微型计算机系统中得到广泛应用。特别是带键盘的显示器，操作人员可以借助系统的硬件和软件的功能，在 CRT显示器上随时增删、修改显示的内容，因此，CRT显示器是一种实现人机联系的重要工具。

本节着重讨论 CRT 显示器的工作原理及其与 CPU 的接口。

一、CRT 显示器的工作原理

1．光栅扫描

CRT 萤光屏上的图象是由电子束的扫描产生的。如图 15-54 所示，受视频信号控制其强度的电子束，在水平偏转和垂直偏转的信号作用下，自左至右，从上而下地逐步扫描。水平方向的扫描叫行扫描，在每行扫描的正行程(图中用实线表示)，电子束的强弱受图象信号影响。当到达右端时，电子束被消隐并飞回到起始端的下一行，然后，进行新一行的横向扫描。如此往复，从上到下扫完整个屏幕。这就使屏幕上出现了有明暗层次的一帧画面。扫到右下角的电子束消隐后返回左上角的起始端，再开始新一帧的扫描显示。由于选取了合适的帧频，利用人眼的视觉暂留，因此，人们看到的是一幅稳定的画面。

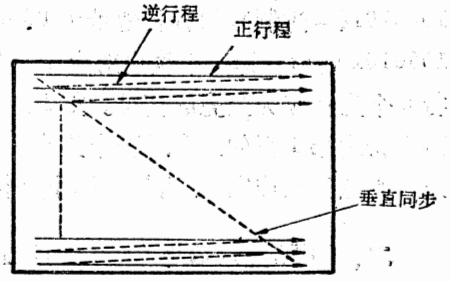


图 15-54　光栅扫描方式

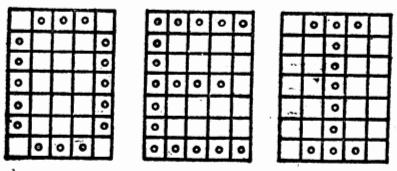


图 15-55　5×7 点阵字符

2．字符的产生

在 CRT 上显示的字符是由点阵组成的。最常用的点阵是横向 5 个点，竖向 7 个点组成的 5×7 点阵，所有的 ASCII 码字符均可由 5×7 点阵的不同光点表示。图 15-55 表示组成字母 O、E、I 的点阵图案。

在 CRT 屏幕上字符显示的格式如下：192 行扫描行组成 16 行字符，每行字符在竖向占 12 个扫描行，其中 7 个扫描行用来产生字符，5行空白扫描线作为字符之间的间距。另外，每个字符的横向由 5 个光点组成，两个字符之间空一个光点的位置，每 12 个扫描行，即一个字符行自左到右可以产生 64 个字符(见图 15-56)。

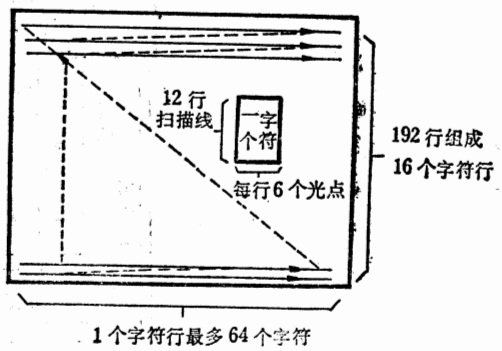


图 15-56　CRT 显示器的扫描格式

为了把 ASCII 码变成 5×7 点阵字符形式，需要使用字符发生器来进行变换。就是说，字符发生器将字符的 ASCII 码变换成 5×7 点阵信息，作为图象信息发送出去控制电子束的强弱以产生表示该字符的光点图案。

应当指出，CRT 的扫描工作方式不是扫描出一个字符再扫描下一个字符，而是一次扫描 64 个字符的同一光点扫描行信息(即 64 个字符的 1/7 信息)，连续七次扫描就把同一字符行的 64 个字符一起扫描出来。

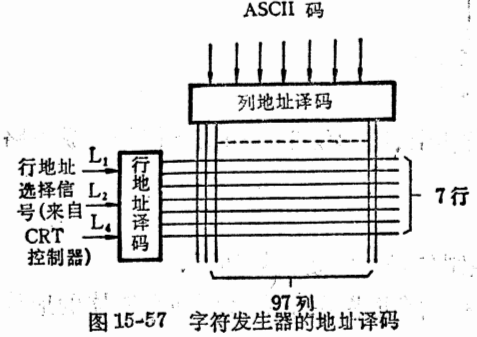


图 15-57　字符发生器的地址译码

5×7 点阵字符发生器通常是由字长 5 位的 ROM 组成的。ROM 中每个存贮单元存一个字符的一个光点扫描行(5 个光点)信息。常用的 7 位 ASCII 码的字符数最多为 97 个，而每个字符由 7 行(每行 5 个点)信息组成，因此 ROM 所需的最大容量为 97×7 = 679。这 679 个 ROM 存贮单元可按 7 行

×97 列矩阵排列，并由 3 位行地址（产生 7 个行）和 7 位列地址(可产生 97 列)组成的地址译码矩阵选择，如图 15-57 所示。

字符发生器的列地址译码输入来自锁存器中的 7 位 ASCII 码，而行地址译码输入受 CRT 控制器所发的行地址选择信号 L_1、L_2、L_4 的控制。例如，锁存器送入字母 E 的 ASCII 码 1000101(=进制)，扫描行地址为 000($L_1=L_2=L_4=0$)，字符发生器立即从行地址 000，列地址 1000101 的地址译码中读出对应于字母 E 第一行的 ROM 单元内容为 11111，即字母 E 的第 1 行 5 个光点应为全亮的信息。如 $L_4=L_2=L_1=0$，而字母为 I，则经图 15-57 的字符发生器译码后得到第 1 行的第 1、5 两个光点为暗，第 2、3 和 4 光点为亮。当把一个字符行 64 个字符的第一光点行扫描完毕，再扫描64个字符的第二光点行，共扫描 7 个光点行，就显示出一行字符。

二、CRT 显示器与 CPU 的接口

从前面讨论中可知，要在 CRT 显示器上显示 CPU 的数字信息，必须把 CPU 的数字信息转换成 CRT 显示器所能接收的图象信息。这一功能是通过 CRT 接口电路完成的。图 15-58 示出了 CRT 接口的方框图。

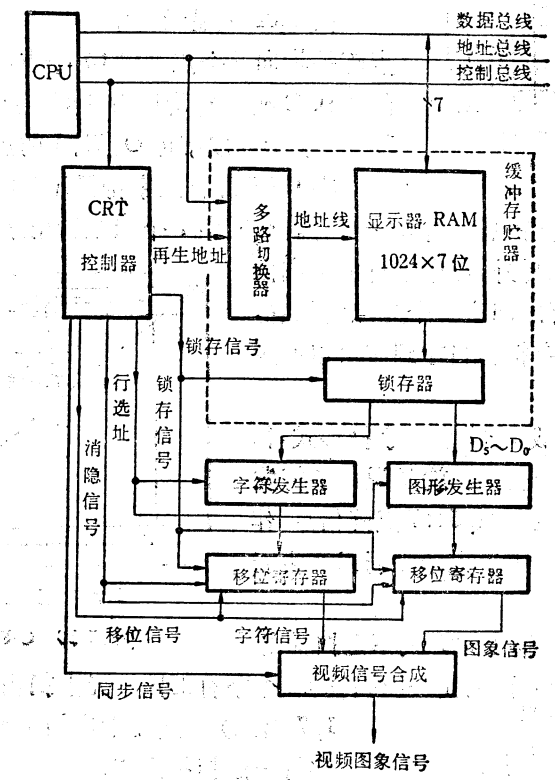


图 15-58　CRT 接口方框图

由图可知，它由 CRT 控制器、缓冲存贮器、字符发生器、图形发生器和视频信号合成电路等部分组成。下面分别加以说明。

1. CRT 控制器

CRT 控制器提供接口电路工作所需的各种时序控制信号。主要信号如下：

(1) 以一定的场频（垂直扫描频率，例如 60Hz）提供存贮图象信息的缓冲存贮器的地址

(称再生地址)。

(2) 提供字符发生器和图形发生器的扫描行地址，保证字符或图形以正确的格式在屏幕上显示。

(3) 提供串行移位信号，使图象信号、消隐信号和同步信号混合成串行信息流，然后发送出去。

(4) 提供消隐控制信号和水平、垂直同步信号。

为了保证电子束能正确无误地扫描出一帧稳定的图象，图象信号、消隐信号、同步信号这三部分信息是缺一不可的。通常这三部分信号总称为视频图象信号(简称视频信号)。

由于大规模集成电路的发展，现在已有多种单片 CRT 控制器可供选用。例如 Intel 8275 可编程序 CRT 控制器。

8275 的基本功能是缓冲从显示存贮器来的信息并产生 CRT 同步用的垂直与水平的定时信号，从而刷新 CRT 显示图象。8275 CRT 控制器是一种可编程序的接口器件，这使得它能够只需要附加很少的硬件，便可几乎与任何光栅扫描显示器相连。8275 的显示格式为每行 1～80 个字符、每帧 1～64 行、每行 1～16 条水平扫描线，可通过编程序选定。

2．缓冲存贮器

缓冲存贮器由显示 RAM、多路切换器和锁存器组成。显示 RAM 中存贮待显示的一帧图象信息，其存贮容量与一帧画面所能显示的最大字符数相等。在本例中，萤光屏所能显示的最大字符数为 $16\times64=1024$ 个字符，而每个字符用七位 ASCII 码表示，因此显示 RAM 的容量为 1024×7 位。每一个字符的 ASCII 码存于显示 RAM 的存贮单元中，某一字符的存贮地址是与该字符在萤光屏上的位置一一对应的。

每次显示的信息应先由 CPU 送入显示 RAM，此时多路切换器将显示 RAM 的地址线与 CPU 地址总线相连，CPU 通过存数指令将要显示的图象信号送入显示 RAM。然后多路切换器将显示 RAM 的地址线连到 CRT 控制器上，由 CRT 控制器以每行扫描 64 个字符的速率改变 RAM 的地址，并依次把 RAM 存贮单元中的数据送入锁存器，再通过字符发生器(或图形发生器)和视频信号混合电路发送给 CRT 显示器。

3．视频图象的合成

视频图象合成电路的功能是将图象信号、消隐信号、水平同步信号、垂直同步信号按一定的时序和电平的大小合成后串行地发送给 CRT 显示器。

字符发生器的功能已如上述，图形发生器的功能是把来自锁存器的图形信息转换成图形光点扫描信息，经移位寄存器送视频合成电路，形成视频图象信号发送给 CRT 显示器。

三、CRT 终端

所谓终端(terminal)是指信息可以进入或离开通信网络的地点，它可向网络发送信息，也可以从网络接受信息。终端的特点是可以通过终端设备，例如带键盘的 CRT 显示器远距离地使用计算机网络中的计算机。

现代 CRT 终端多是“智能终端”，即指自身带有一个微处理器的终端，它本身具有很强的处理功能，目前已得到广泛的应用。

第十六章　微型计算机系统的组成

本章讨论如何将微处理器、存贮器、I/O 接口电路和典型的外设等组件有机地连接起来，以组成一个完整的微型计算机系统，使读者对微型计算机系统的工作有一个全面的完整的了解。

§16-1　标准微型计算机系统的组成

一、标准微型计算机系统结构

下面以 8 位微型计算机系统为例来说明微型计算机系统的互连。标准微型计算机系统结构框图如图 1-2 所示。如果需要扩大系统的功能，只要结合系统的实际需要加以扩充即可。

标准微型计算机系统的基本特征是采用总线结构，其中包括 8 位双向数据总线、16 位单向地址总线和由控制信号组成的控制总线。CPU、存贮器、I/O 接口等标准组件都挂在系统总线上。

二、总线驱动器电路

对于一个规模较小的微型计算机系统，可以不用总线驱动器，因为 CPU 的地址总线和数据总线的输出本身具有驱动一个标准 TTL(1.6mA)和若干电容负载的能力。例如，Intel 8080A 的数据总线输出吸收电流为 1.9 mA，Intel 8085 A 的为 2.0 mA，Z80 的为 1.8 mA 等，这对于小系统来说是足够的了。但当 CPU 和大容量的标准 ROM、RAM 一起使用或扩展成一个多插件系统时，为了驱动系统总线，就必须增设总线驱动器。

总线驱动器可以根据数据传送方向的要求，采用单向或双向驱动器。例如对于地址总线，它仅由 CPU 发出地址信号，所以采用单向驱动器；而对于数据总线，CPU 要通过数据总线，将信息从存贮器或 I/O 端口读出或写入，故应采用双向驱动器。它们的作用是为系统总线提供足够的驱动能力，并能起到缓冲作用以便保护 CPU。

常用的总线驱动器有 4 位双向总线驱动器 8216/8226 和 8 位 I/O 端口 8212。

1. 8216/8226 4 位并行双向总线驱动器

(1) 8216/8226 框图及引线

8216/8226 是 4 位并行双向总线驱动器/接收器，为了驱动 MOS 电路，DO 输出端能提供 3.65 V 的高电平输出，DB 引线提供 50 mA 的电流输出驱动能力。它们的引线布置图和原理框图如图 16-1、图 16-2 和图 16-3 所示。

8216 和 8226 的区别仅在于 8216 输出端是不反相的，而 8226 输出端是反相的，这样便于适用各种各样的应用场合。

(2) 8216/8226 的功能原理

由图 16-2、图 16-3 可知，4 位驱动器的每条缓冲线由两个分开的三态缓冲器所组成，因此能直接与总线接口，并具有双向传送的能力。在驱动器的一边，一个缓冲器的输出端同另一

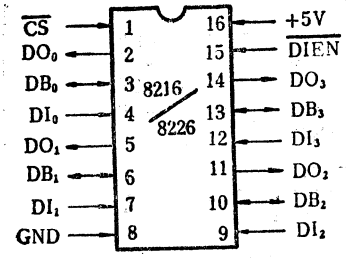


图 16-1　8216/8226 的引线图

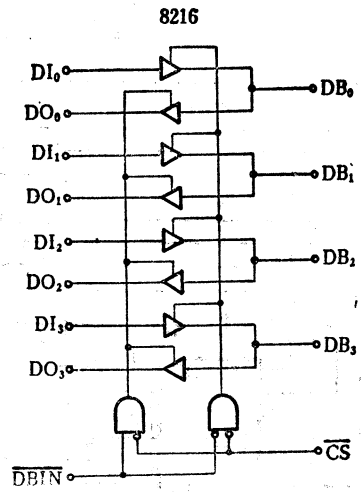


图 16-2　8216 原理框图

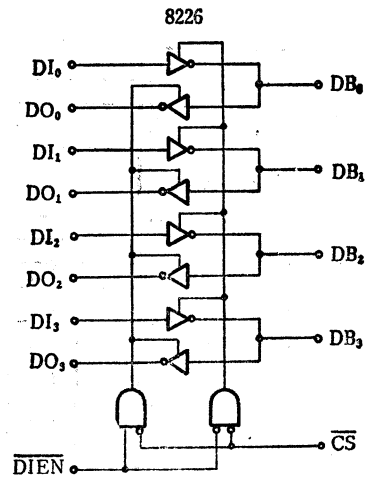


图 16-3　8226 原理框图

引线名称

$DB_3 \sim DB_0$	双向数据总线
$DO_3 \sim DO_0$	数据输出
$DI_3 \sim DI_0$	数据输入
$\overline{DIEN}$	数据方向控制
$\overline{CS}$	片选

个缓冲器的输入端连接在一起(称为 DB 端),由于它的接口能直接同 TTL 相容,并具有较高的驱动能力(50mA),所以常用来连接系统器件(如存贮器或 I/O 等)。在驱动器的另一边,为了获得最大的灵活性,各输入端和输出端是相互分开的。当然它们也可以连在一起,以使驱动器能用来缓冲双向数据总线。

8216 的片选端 $\overline{CS}$ 是器件选择输入端。当它为高电平时,驱动器输出被强置为高阻抗状态;当它为低电平时,器件被选中(允许工作)。而数据流向则由 $\overline{DIEN}$ 输入端来确定。

$\overline{DIEN}$ 输入端将按照真值表(表 16-1)的要求来控制数据流向。方向控制是通过强制两个缓冲器中的一个进入高阻状态,而使另一个允许传送数据来实现的。为此采用了两个门电路。

表 16 1　真值表

$\overline{CS}$	$\overline{DIEN}$	数 据 流 向
1	X	高阻抗状态
0	0	DI→DB
0	1	DO←DB

2. 8080A CPU 与双向数据总线驱动器的连接

图 16-4 示出了 8080A CPU 与数据总线驱动器的连接线路,图中两片 8216 直接接到 8080A 数据总线($D_7 \sim D_0$)上,8080A CPU 模块送出的 DBIN 信号端接到 8216 的方向控制端($\overline{DIEN}$),以控制正确的数据流向。8216 的片选端($\overline{CS}$)连接到 $\overline{BUSEN}$(开放总线)。当 $\overline{BUSEN}$ 为高电平时,表示不选用数据总线驱动器并置 8216 的输出端为高阻抗状态。此时,可进行 DMA 传送。

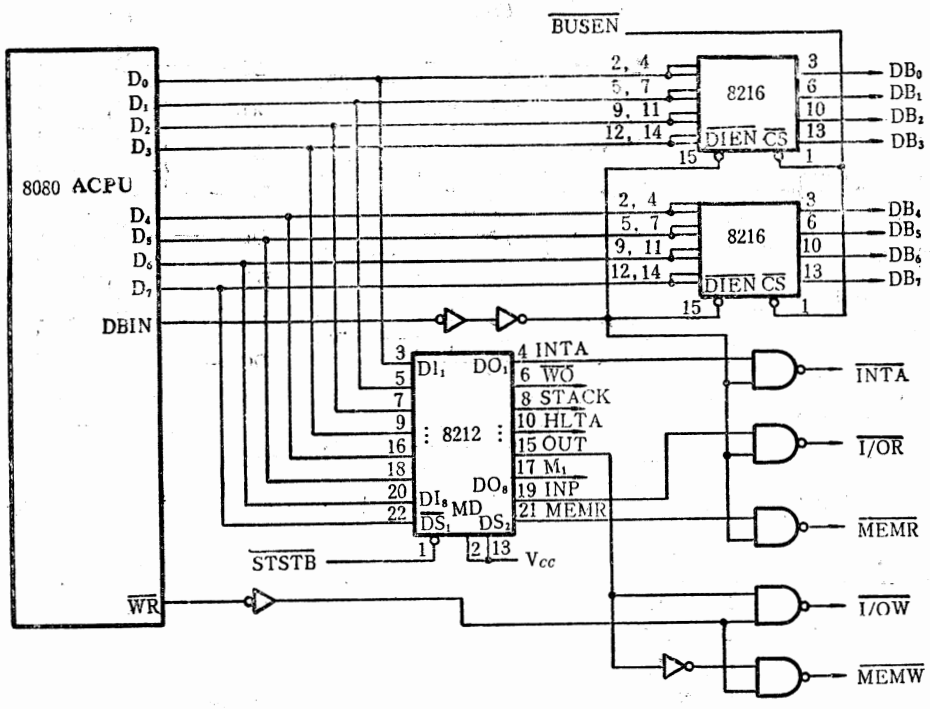


图 16-4　8080 A CPU 与双向数据总线驱动器的连接

图中 8212 用作 8080A CPU 的周期信息码锁存器。在每个机器周期开始时，利用时钟发生器(8224)来的周期选通($\overline{STSTB}$)信号，把周期信息码锁存至 8212 中。周期锁存器 8212 的输出和 8080A 的 DBIN 及 $\overline{WR}$ 信号组合成简单的门控电路，产生 5 个控制信号，即 $\overline{MEMR}$、$\overline{MEMW}$、$\overline{I/OR}$、$\overline{I/OW}$ 和 $\overline{INTA}$。这 5 个控制信号可直接连接到 MCS-80 的 ROM、RAM 和 I/O 接口等器件上。

Intel 8085A 和 Z80 CPU 与双向数据总线驱动器的连接的情况，与 Intel 8080A 相似。此时，8216 的片选端($\overline{CS}$)连接到 8085A 的保持响应(HLDA)或 Z80 的总线响应($\overline{BUSAK}$)的反相端。而 8216 的数据方向控制端($\overline{DIEN}$)可连接到 8085A 或 Z80 的读出控制信号($\overline{RD}$)的反相端。

在设计系统总线驱动器时，应根据微型计算机系统的实际情况(如有时是在原有的单板微计算机基础上增设总线驱动器)来正确地控制 8216 的 $\overline{CS}$ 和 $\overline{DIEN}$ 端，此时，考虑的因素要比上述情况稍许复杂些。

§16-2　Intel 8080A 微型计算机系统 (MCS-80) 的组成

图 16-5 和表 16-2 示出完整的 Intel 8080A 微型计算机系统框图及系统部件。它除了 8080A CPU 模块、ROM/RAM、I/O 并行接口和串行接口外，还包括处理中断优先权的电路芯片、DMA 控制器电路芯片、地址译码器及总线驱动器等。

在上述部件中，存贮器 EPROM/RAM 的数量，可根据程序和数据的容量选用，如果存贮器容量较大，则应增设总线驱动器并适当增加地址译码器。

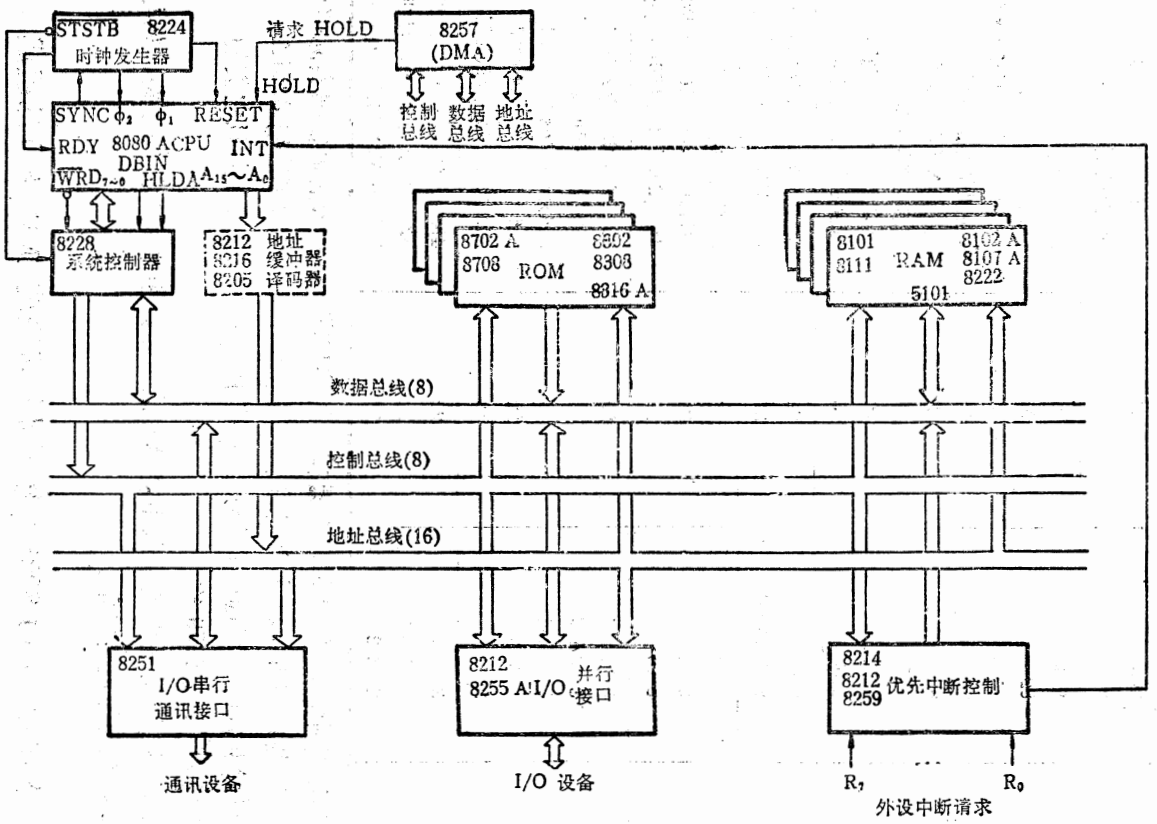


图 16-5 Intel 8080A 微型计算机系统框图

表 16-2 8080A 微型计算机系统的部件

功　能	型　号	引　线	名 称 和 规 格
CPU 模块	8080A	40	8 位 CPU
	8224	16	时钟发生器
	8228/38	28	系统控制器
输入/输出	8212	24	8 位 I/O 端口
	8251	28	可编程序串行通信接口
	8255	40	可编程序并行接口
外　围	8205	16	三-八译码器
	8214	24	优先权中断控制器
	8216	16	4 位双向总线驱动器(50mA)
	8226	16	4 位双向总线驱动器(50mA)、反相
	8222(3222)	22	动态 RAM 刷新控制器
	8253	24	可编程序定时器
	8257	40	可编程序 DMA 控制器
	8259	28	可编程序中断控制器
EPROM	8702	24	2K(256×8), 1.3μs 存取时间
	8708(2708)	24	8K(1024×8), 450ns 存取时间
ROM	8302	24	2K(256×8), 1μs 存取时间
	8308	24	8K(1024×8), 450ns 存取时间
	8316	24	16K(2048×8), 850ns 存取时间
NMOS 静态 RAM	8101A-4	22	256×4, 450ns 存取时间
	8102A-4	16	1024×1, 450ns 存取时间
	8111A-4	18	256×4, 450ns 存取时间
	5101L-1	22	256×4, 450ns 存取时间
	2114	18	1024×4, 450ns 存取时间
动态 RAM	8107B	22	4K(4096×1), 400ns 存取时间
	8107B-4	22	4K(4096×1), 470ns 存取时间
	2116-2	16	16K(16384×1), 350ns 存取时间

在 I/O 接口芯片中，8212 和 8255A 是基本的接口芯片，其余的可根据实际需要选用。当所设计的系统需要与电传打字机或盒式磁带机等串行外设连接时可选用 8251A；当系统中的外围设备需要定时装置时，可选用 8253；当系统中有多个外围设备并且它们的工作顺序有优先要求时，可选用 8214 或 8259 等。

除了上述芯片外，系统还需要一些中规模集成电路，用来作为辅助电路，如与非门、或非门、反相器和 D 触发器等逻辑电路。

§16-3 Intel 8085 A 微型计算机系统 (MCS-85) 的组成

一、基本的 8085A 微型计算机系统(MCS-85 系统)

基本的 8085A 微型计算机系统由三片 LSI 电路 (8085A、8155 和 8355/8755)组成。如图 16-6 所示。Intel 公司为 8085A 专门设计制造了把存贮器、I/O 接口和计时器等结合在一起的两种新型外围芯片。这些芯片能直接与 MCS-85 的多重总线接口。例如，8155/8156 为带

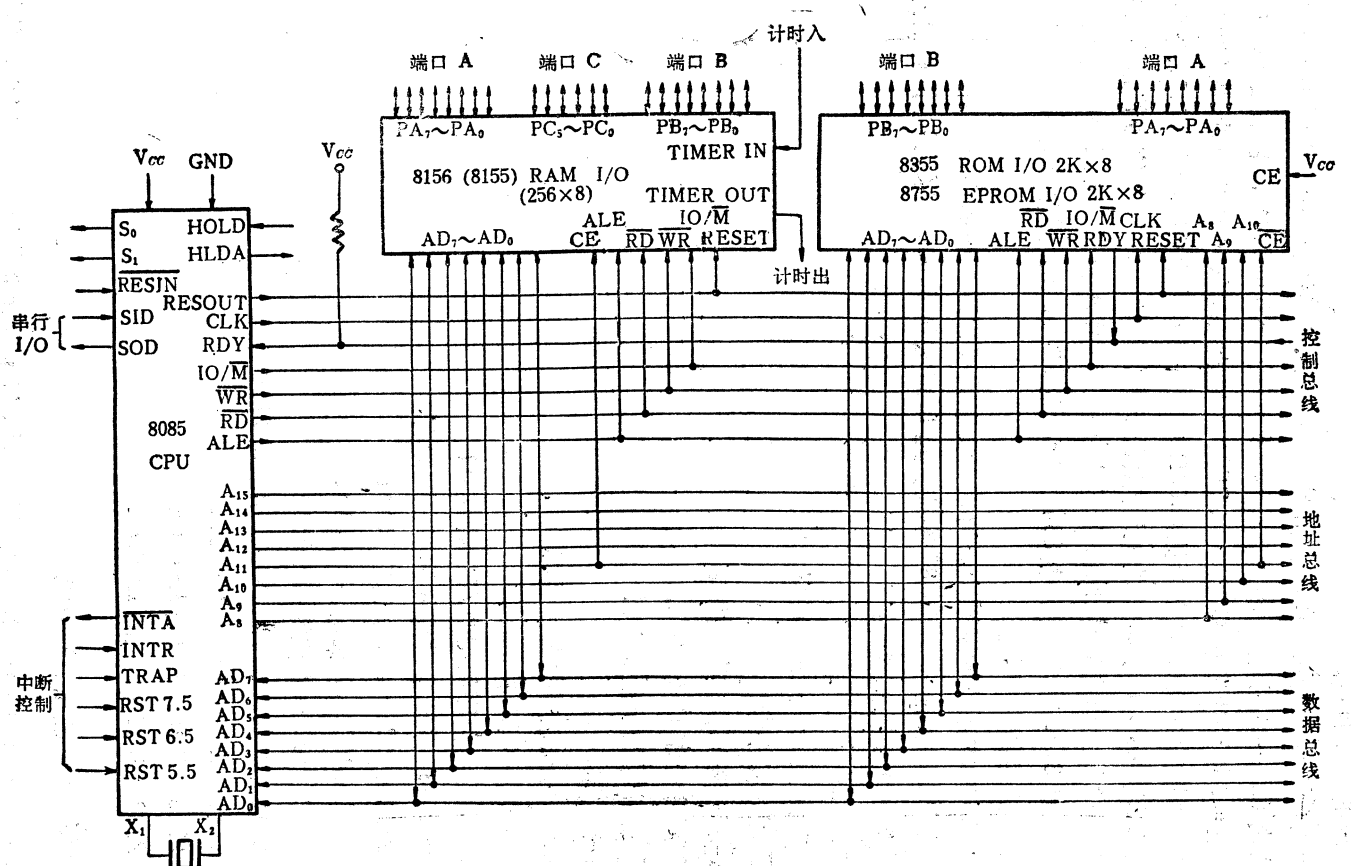


图 16-6 Intel 8085 A 微型计算机系统

有 I/O 端口和计时器的 2048 位(256×8 位)静态随机存贮器；8355 为带有 I/O 端口的 16384 位(2048×8 位)只读存贮器；8755 与 8355 基本相同，区别仅在于 8755 所带的是 2048×8 位的 EPROM。利用这些芯片，可以在不用附加缓冲器和译码器的情况下，构成各种中、小型系统，从而减少系统的组件数量。

下面分析一下系统的互连情况。由图 16-6 可知，该系统采用"线性选择"方法寻址，即利用高位地址线 (A_{15}～A_{11}) 作为芯片选择端。8355 有 2KB 的 ROM，它需要 11 位地址 (A_{10}～A_0)，故用 $A_{11}=0$ 接至片选端 $\overline{CE}$。这样，ROM 的地址空间总共 2KB 地址，从 0000H～07FFH，以便于开机后程序计数器(PC)立即从 0000H 单元开始执行程序。8156 有 256 个字节的RAM，需用 8 条线 (AD_7～AD_0) 与 CPU 的地址/数据多重总线连接；ALE 信号将指出：在特定的时间内，这条多重总线上出现的是地址，而不是数据。可用 $A_{11}=1$ 接至 8156 的片选端 CE，这样保证 CPU 在选择 8156 时，不会选中 8355。两片的 IO/$\overline{M}$ 线，用来选择存贮器或 I/O 端口。RESET 用来清除内部寄存器。

在 MCS-85 基本系统的基础上，可根据实际需要，适当增加各种外围组件和标准存贮器片，以构成具有不同性能的 MCS-85 系统。凡 Intel 公司 825X-5 系列的外围组件，都可以直接与 MCS-85 总线连接。例如：

通用外围电路：

8251A 可编程序串行通信接口

8253-5 可编程序计时器

8255A-5 可编程序的并行接口

8257-5 可编程序 DMA 控制器

8259-5 可编程序中断控制器

专用外围电路：

8271/8272 可编程序软盘控制器

8275 可编程序 CRT 控制器

8278/8279 可编程序的键盘/显示器接口

此外，MCS-85 系统中常用的标准存贮器有：

RAM：　2114A 1024×4 位的静态 RAM

　　2141A 4096×1 位的静态 RAM

　　2142　1024×4 位的静态 RAM

ROM：　2316　2048×8 位的 ROM、引线与 2716 兼容

EPROM：2758　1024×8 位的 EPROM

　　2716　2048×8 位的 EPROM

　　2732　4096×8 位的 EPROM

二、SDK-85 单板微型计算机

SDK-85(System Design Kit)系统设计套件，以配套元器件的方式为用户组装单板微型计算机，提供了完整的配套元器件，而且可以根据实际需要，在布线区或板外进一步扩展功能部件，以适应数据处理、工业控制等方面的需要。

1．SDK-85 系统的组成和功能特性

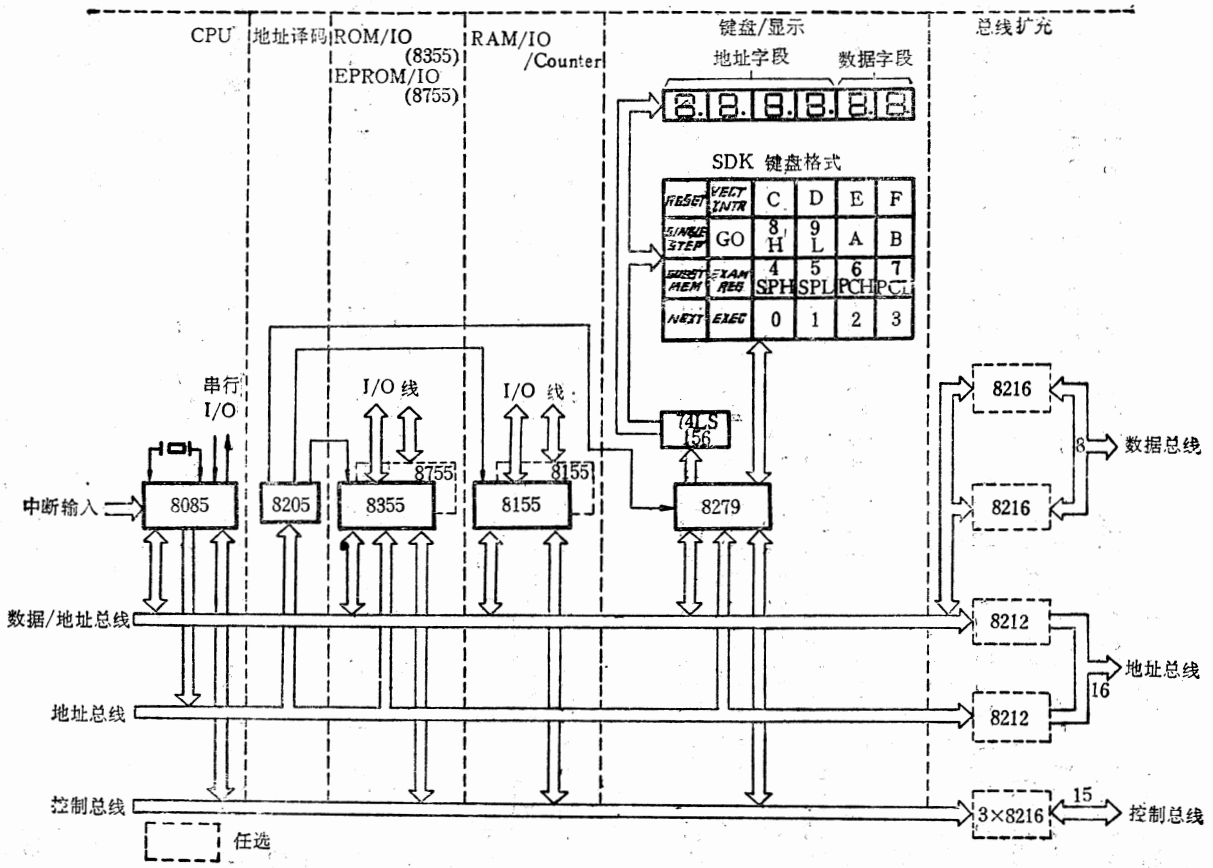


图 16-7　SDK-85 系统组成框图

(1) SDK-85 系统组成框图

图 16-7 是 SDK-85 系统的组成框图，虚线所示部件已在 SDK-85 的印刷板上留有位置，供总线扩展之用。

由图可知 SDK-85 系统设计套件是由 8085A CPU 和 8355/8755、8155、8279 及 8205 等主要器件组成的，并有 6 位发光二极管(LED)数码显示器和 24 个输入键构成的一台完整单板微型计算机。

① 8085A CPU 及系统总线

从系统框图可以看到 8085A 直接从一晶体取得它的时钟信号。8085A 的地址总线的低 8 位和数据总线采用多重总线方式。套件中的 8155 和 8355 (或 8755) 存贮器/输入输出器件能与这种总线兼容，而不需要附加总线锁存器。

在 MCS-85 中，除了 8080A 型的标准中断外，还有 4 个矢量中断输入，此外还有一对串行输入和串行输出数据线，可用于串行数据传输。

② 8155

8155 是一种能和 8085A 总线兼容的 LSI 芯片，它包含 256 字节的静态 RAM，22 条可编程序的 I/O 线，以及 14 位的定时器或计数器。

SDK-85 配套组件包括一片 8155，并且在印刷线路板上为另一片 8155 留有位置。8155 RAM 用来存放用户程序以及暂时存放系统程序所需要的信息。

在 SDK-85 监控程序中，单步程序利用 8155 计时器在每条指令执行后，向 CPU 发出中断请求。

③ 8355/8755

8355 包含 2048 字节的掩模编程只读存贮器(ROM)和 16 条 I/O 线。8755 有同 8355 相同的功能和引线，但其内部是可用紫外线抹除的可编程序只读存贮器。

SDK-85 包含一片 8355 或一片 8755。

④ 8279

8279 是一种键盘/显示器控制器芯片。它用来管理 8085A 同 SDK-85 印制板上的键盘及发光二极管显示器之间的接口。在扫描键盘以及检测键盘输入的同时，8279 根据其内部存贮器来更新显示。

⑤ 8205

8205 用来为 SDK-85 系统的存贮器地址译码，以供 8155、8355、8755 及 8279 作片选信

表 16-3 8205 的片选输出端

输出端	有效地址范围	选中器件
CS_0	0000～07FFH	8755 或 8355 监控程序 ROM(A14)
CS_1	0800～0FFFH	8755 或 8355 扩充 ROM(A15)
CS_2	1000～17FFH	N/C
CS_3	1800～1FFFH	8279 键盘/显示控制器(A13)
CS_4	2000～27FFH	8155 基本 RAM(A16)
CS_5	2800～2FFFH	8155 扩充 RAM(A17)
CS_6	3000～37FFH	N/C
CS_7	3800～3FFFH	N/C

N/C 表示用户扩充可用的未连接的引线。

号。

8205 译码片选输出信号的分配见表 16-3。

⑥ 8212

8212 是 8 位输入/输出接口芯片。在 SDK-85 系统中，它用作地址锁存器和缓冲器。8212 能分离 CPU 输出的多重地址/数据总线，使低 8 位地址线 A_7～A_0 经 8212 锁存后在整个机器周期内保持着供系统使用，而高 8 位地址线 A_{15}～A_8 则用 8212 增加总线驱动能力。

⑦ 8216

8216 是 4 位双向总线驱动器。在 SDK-85 系统中，它用于增加数据总线和控制总线的驱动能力，该系统共用 5 片 8216 芯片。

此外，图中 74LS156 是 TTL 系列电路中 2 输入端的双译码器，其功能与"3-8"译码器类似。在 SDK-85 系统中用于译码 8279 与键盘、显示器的接口。

(2) SDK-85 的主要性能

① 中央处理单元

CPU：8085A，时钟频率：3.072 兆赫，时钟周期：330 毫微秒，基本指令周期：1.3 微秒。

② 存贮器

ROM：8355 或 8755。

容量：2K 字节(可扩展到 4K 字节)。

RAM：8155。

容量：256 字节(可扩展到 512 字节)。

存贮器地址：

ROM 地址：从 0000～07FFH (外加 8355 或 8755 时，可扩到 0FFFH)。

RAM 地址：从 2000～20FFH (外加 8155 时，可用地址 2800～28FFH)。

③ 输入和输出

并行：38 条线(可扩到 76 根线)。

串行：通过 8085A 的 SID/SOD 引线，可直接与串行外设进行串行通信。

波特率：110。

④ I/O 接口

总线：全部信号与 TTL 兼容。

并行输入输出：全部信号与 TTL 兼容。

串行输入输出：可与电传打字机构成电流回路，产生 20mA 的电流。

⑤ 中断方式

8085A CPU 具有五个矢量中断输入线：TRAP、RST 7.5、RST 6.5、RST 5.5 及 INTR，其中优先权最高的是 TRAP，然后依次下降，INTR 优先权最低。

⑥ 数据通道方式(DMA)

"保持"请求：可以采用不同的跨接线来进行选择。输入信号与 TTL 兼容。

⑦ 软件

系统监控程序：放在可编程的 8755 或者 8355 ROM 中。

地址：0000～07FFH。

⑧ 输入输出方式

24 个键的输入键盘/6 位 LED 数码显示器或 TTY 电传机串行输入输出。它们可通过开关转接来实现。

⑨ 电气特性：

V_{CC}：　+5V±5%　　1.3A

V_{TTY}：　−10V±10%　0.3A (仅在 TTY 接在该单板机上时才需要)。

2. SDK-85 存贮器及 I/O 外设寻址

(1) SDK-85 存贮器寻址

在图 16-7 所示的基本 SDK-85 系统中，存贮器(包括 I/O 口)的寻址是由 8205 地址译码器输出的片选信号来实现的。表 16-3 列出了各个片选输出端以及它的各自的有效地址空间和被选中的器件。

在这里，8279 实际上是一个输入/输出器件，但在本系统中它被当作一个存贮器来寻址即存贮器映象 I/O 寻址。

SDK-85 存贮器的地址分配，如图 16-8 所示。

从图 16-8 可知，基本 SDK-85 套件提供了监控程序只读存贮器和基本随机存贮器的存贮区，用户必须把用户程序限制在基本随机存贮器的可用空间。这是因为基本随机存贮器中，有

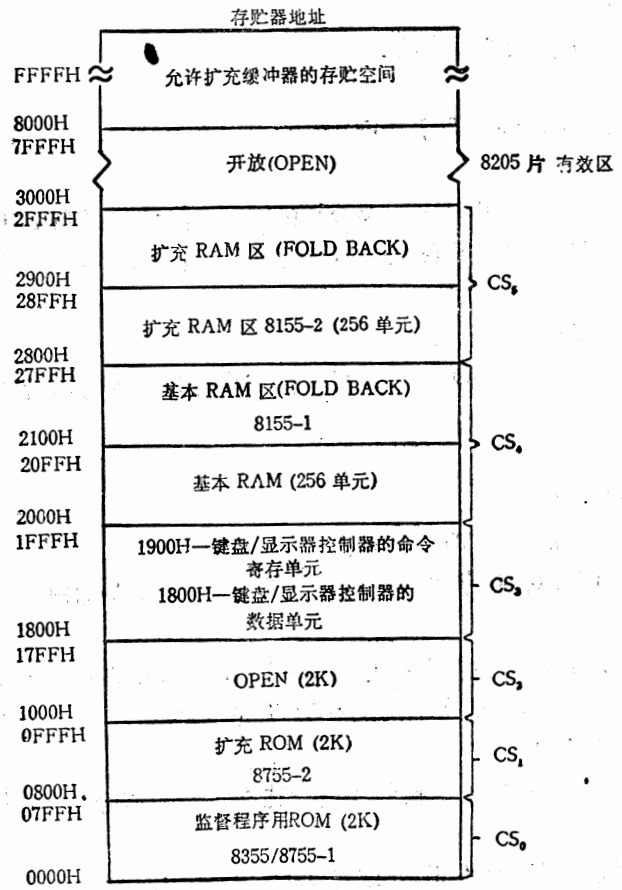


图 16-8　SDK-85 存贮空间地址分配图

一部分还要用作监控程序的存贮单元(20C2～20FFH)。

当用户在SDK-85板上留出的空位再装上一片8155时，图16-8中所示的扩展随机存贮器的RAM单元就可用于编程序。监控程序不占用扩充RAM，故其256个存贮单元均可用于编程序。

图16-8中所示的"OPEN"区是可以扩充的，用户可以在SDK-85板的布线区或其它线路板上安装附加的存贮器芯片。8205地址译码器尚有3条未用到的片选线，因此在不增加译码电路的条件下，可增加3×2048字节的存贮器。

为了驱动大的系统，在印刷线路板上还为另一个8212留有空位，以便用来缓冲高8位地址线。此外，为了缓冲数据总线和控制信号，还需要5个8216总线驱动器，在印刷板上也为它们留有空位。这些元器件的功能在前面章节均已叙述。

如图16-8所示，在SDK-85印刷板上的可选择的扩充缓冲器，其地址范围仅限于8000～FFFFH(后32K字节)。如果要利用图16-8所示的任何"OPEN"扩充区(由8205来选片)，应当先熟悉手册中的SDK-85的原理图，并应对原线路进行修改。

(2) I/O接口寻址

如前所述，8155和8355/8755本身带有输入/输出端口，这些端口可用8085A的IN和OUT指令来访问，它们的各个端口都有唯一的端口地址，表6-4列出有两个8155和两个8355/8755的扩充SDK-85的所有端口地址。关于表中所涉及到的各种特殊用途的寄存器(如方向寄存器，命令/状态寄存器等)，以及关于使存贮器I/O芯片工作的完整的指令，请参阅MCS-85用户手册。

第一个8155的定时器/计数器被用作键盘监控程序的定时器，它以8085A系统时钟(3.072 MHz)作为其计数输入。监控程序的单步功能需要用到这个定时器，因而，如果需要同时

表16-4 SDK-85输入输出端口对照表

端口(H)	功能	
00	监控程序ROM端口A	8355
01	监控程序ROM端口B	
02	监控程序ROM端口A数据方向寄存器	
03	监控程序ROM端口B数据方向寄存器	
08	扩充 ROM端口A	8755
09	扩充 ROM端口B	
0A	扩充 ROM端口A数据方向寄存器	
0B	扩充 ROM端口B数据方向寄存器	
20	基本 RAM命令/状态寄存器	8155-1
21	基本 RAM端口A	
22	基本 RAM端口B	
23	基本 RAM端口C	
24	基本 RAM定时器计数低位字节	
25	基本 RAM定时器计数高位字节	
28	扩充 RAM命令/状态寄存器	8155-2
29	扩充 RAM端口A	
2A	扩充 RAM端口B	
2B	扩充 RAM端口C	
2C	扩充 RAM定时器计数低位字节	
2D	扩充 RAM定时器计数高位字节	

计数和利用单步功能，应当设法避免在使用时间上发生冲突。

(3) 8279 键盘/显示控制器的寻址

前已提及，8279 是一个 I/O 外围接口片子，在 SDK-85 系统中，采用"存贮器映象" I/O 方式寻址。表 16-5 列出用作同 8279 通信的两个存贮单元。

表 16-5 8279 键盘显示控制器寻址

存贮单元(H)	读/写	功　　能
1800	读	读键盘"先进先出"(FIFO)存贮器
	写	把数据写到显示器
1900	读	读 8279 状态字
	写	写 8279 命令字

3．微处理器中断存贮单元的分配

8085A 除了有一个与 8080A 兼容的中断输入外，还有 4 个矢量中断输入引线，各个中断的名称及其在 SDK-85 系统中的功能列于表 16-6。

表 16-6 SDK-85 系统的中断分配

输　　入	功　　能
RST5.5	8279 专用
RST6.5	可用的用户中断
RST7.5	"矢量中断"按钮中断
TRAP	8155 定时器中断
INTR	可用的用户中断

4．串行数据接口

由于利用了 8085A 的串行输入(SID)、串行输出(SOD)数据线，因此，SDK-85 能够方便地同电传机进行通信。

三、MCS-85 的典型外围接口芯片

1．8155/8156——带有 I/O 端口及计时器的 2048 位静态 MOS 随机存贮器(RAM)

(1) 8155/8156 的结构框图

8155/8156 是一种多功能的 LSI 芯片。它的结构框图如图 16-9 所示。

由框图可知，它的内部结构主要包括：

① 256×8 位的静态 RAM；

② 两个可编程序的 8 位 I/O 端口 PA、PB；

③ 一个可编程序的 6 位 I/O 端口 PC；

④ 可编程序的 14 位二进制计数器/计时器；

⑤ 命令/状态寄存器 C/S (8 位)。

其中，C/S 寄存器分别由 8 位只写命令寄存器和 8 位只读状态寄存器组成，通过 C/S 寄存器可用程序编制 PA、PB、PC 端口的工作方式，并可读出 8155/8156 的工作状态。

14 位可编程序定时器/计数器，可为系统提供方波或终端计数脉冲。

8155/8156 内部译码器输出的通道地址分配见表 16-7。

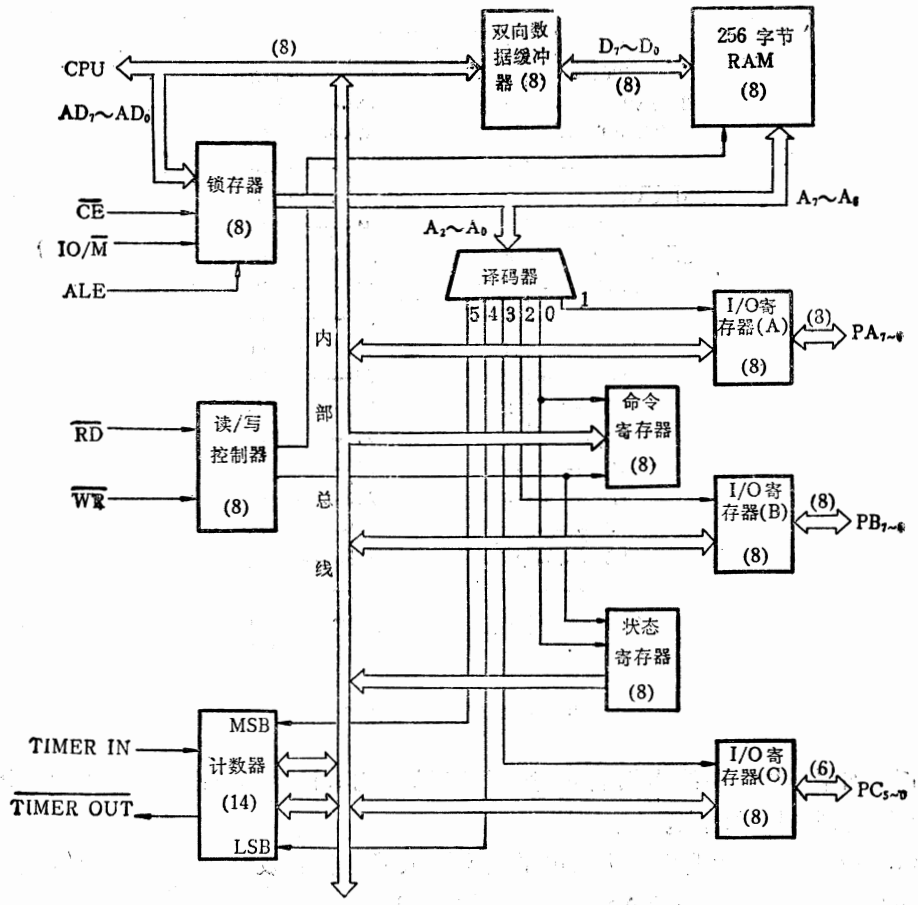


图 16-9　8155/8156 结构框图

表 16-7　地址分配表

IO/$\overline{M}$=1 时 A_7～A_0 状态	通　道　选　择	引　　线
××××× 0 0 0	C/S 寄存器	无　引　线
××××× 0 0 1	I/O 端口 PA	PA_7～PA_0
××××× 0 1 0	I/O 端口 PB	PB_7～PB_0
××××× 0 1 1	I/O 端口 PC	PC_5～PC_0
××××× 1 0 0	低位计时器	无　引　线
××××× 1 0 1	高位计时器及计时器方式	无　引　线

8155/8156 的引线布置如图 16-10 所示。

图中，8155 = $\overline{CE}$，8156 = CE

(2) 8155/8156 引线功能

① RESET——复位信号，它是由 8085A 提供将系统置于初始状态的脉冲信号。当这条线为高电平时，总清芯片，并把三个输入/输出端口置成输入方式。

② AD_7～AD_0——三态地址/数据总线，它与 CPU 的低 8 位地址/数据总线连接。

③ CE 或 $\overline{CE}$——芯片允许，在 8155 中，当此引线为低电平时，允许芯片操作；而在 8156 中，当此引线为高电平时才允许芯片操作。

④ $\overline{RD}$ 读控制——当此引线为低电平，且 $\overline{CE}$(或 CE)为有效时，AD_7～AD_0 缓冲器允许

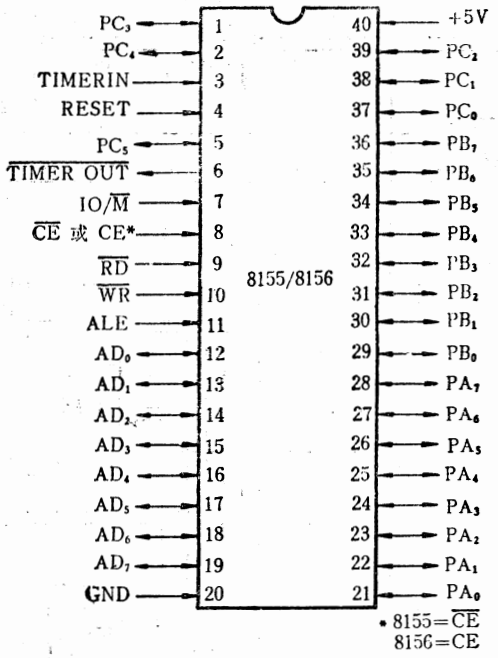


图 16-10　8155/8156 引线图

工作。此时，如果 IO/$\overline{M}$ 引线为低电平，则将 RAM 的内容读至 AD 总线上；反之，将选中的 I/O 端口内容读到 AD 总线上。

⑤ $\overline{WR}$ 写控制——当此引线为低电平且 $\overline{CE}$(或 CE)为有效时，将按照 IO/$\overline{M}$ 信号为“0”或“1”，把 AD 线上的数据写入 RAM 或写入 I/O 端口。

⑥ ALE——地址锁存允许。在 ALE 有效信号的后沿，把 AD_7～AD_0、$\overline{CE}$(CE) 和 IO/$\overline{M}$ 锁存至芯片中。

⑦ IO/$\overline{M}$——I/O 和 M 的选择。若此线为低电平时，则用于选择存贮器；若此线为高电平时，则选择 I/O 端口。

⑧ PA_7～PA_0(8)——通用的输入/输出线。由程序编制的命令/状态寄存器选择输入/输出的流向。

⑨ PB_7～PB_0(8)——通用的输入/输出线。由程序编制的命令/状态寄存器选择输入/输出的流向。

⑩ PC_5～PC_0(6)——可作为一个输入/输出端口，或者用作 PA 和 PB 的控制端口。可通过 C/S 寄存器实现程序控制。当 PC_5～PC_0 用作控制信号时，其作用如下：

PC_0——A INTR(端口 A 的中断)

PC_1——A BF(端口 A 的缓冲器“满”)

PC_2——$\overline{A\ STB}$(端口 A 的选通脉冲)

PC_3——B INTR(端口 B 的中断)

PC_4——B BF(端口 B 的缓冲器“满”)

PC_5——$\overline{B\ STB}$(端口 B 的选通脉冲)

⑪ TIMER IN——计时器/计数器的输入端。

⑫ $\overline{\text{TIMER OUT}}$——计时器的输出端。这个输出信号是矩形波还是脉冲，取决于计时器的方式。

⑬ V_{CC}——+5V 电源

V_{SS}——接地

(3) 8155/8156 的操作说明

8155/8156 的操作由 C/S 寄存器的程序编制来控制。C/S 寄存器由两个 8 位寄存器组成，它们共用一个端口地址(一个只读，另一个只写，故不会冲突)。写控制字时，CPU 执行 OUT ×××××000 指令，将程序员所设定的 CPU 中累加器的内容送往控制寄存器，此时，IO/$\overline{M}$ = 1, $\overline{WR}$ = 0, $\overline{CE}$ = 0。控制寄存器的内容用程序只能写入，不能读出。控制字的定义见图 16-11。读状态字时，CPU 执行 IN ××××××000 指令把该片 8155 的状态寄存器内容读入 CPU 的累加器 A。状态寄存器内容用程序只能读出而不能写入。

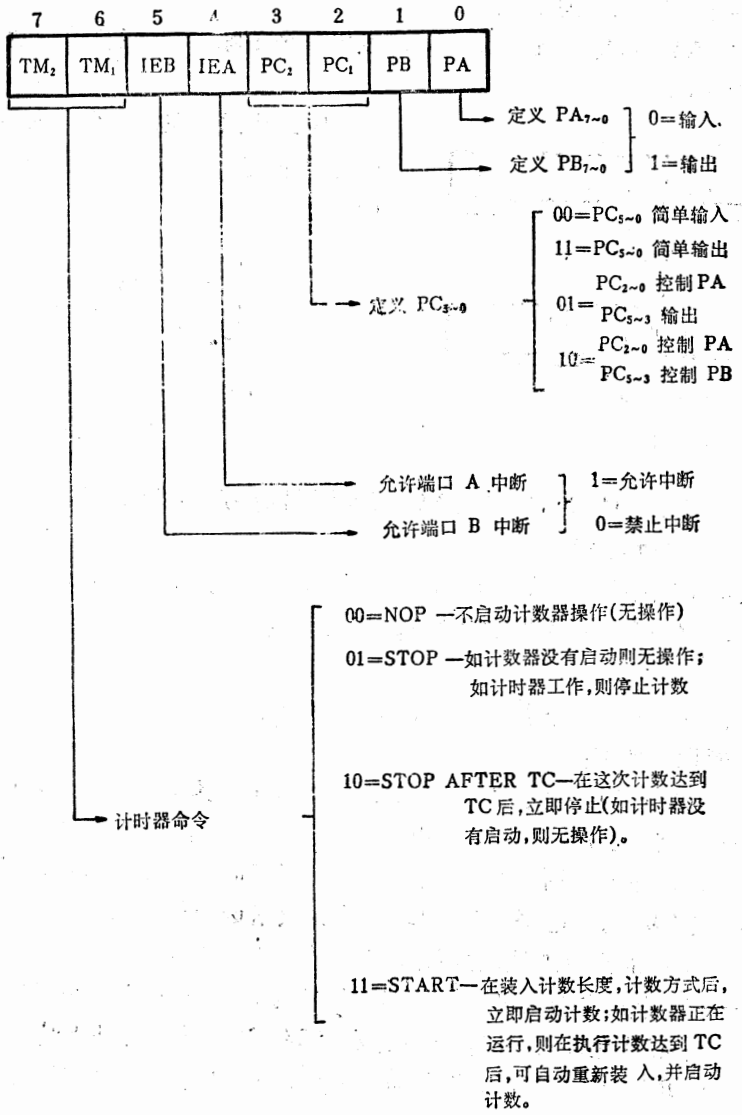


图 16-11 命令/状态寄存器位的定义

计时器是14位的计数器，它可根据程序编制对输入脉冲进行计数，当达到最后计数(TC)时，输出一个方波或脉冲。

计时器用I/O地址 ××××××100 作为寄存器的低位字节，I/O地址 ××××××101 作为寄存器的高位字节。当用程序控制计时器时，首先要通过选择计时器的地址装入计数长度，一次装入一个字节。0～13位将指定一个计数长度，14～15位将指定计时器输出的方式。在0～13位中装入计数长度的值可以从02H到3FFFH中的任意值。计时器格式见图16-12。

计时器的输出方式有4种可供选择，由 M_2、M_1 来定义。

M_2M_1

0 0 输出单个矩形波。

0 1 输出连续矩形波，即矩形波的周期等于计数值(TC)，计完后TC能重新自动装入，并开始计数。

1 0 计完TC值后，输出单脉冲。

1 1 输出连续脉冲，即每完成一次TC，就发出一个单脉冲，并能自动重新装入，而发出一串重复的单脉冲。

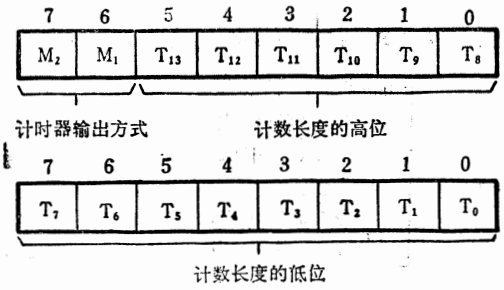
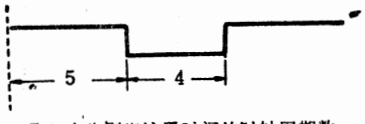


图16-12 计时器格式

应当注意：在每一个非对称计数(如奇数9)情况下，较大的一半将是高电平，如图16-13所示。



注：5和4分别指该段时间的时钟周期数。

图16-13 非对称的计数

2．8355带有I/O端口的2K字节的ROM(8755带有I/O端口的2K字节的EPROM)

(1) 8355/8755的功能和引线

8355掩摸式可编程序ROM和8755可擦除的PROM都是2048×8位的存贮器。这两种芯片都含有两个8位的可编程序的I/O端口。每条引线都可以分别用程序编制为输入或输出。

8355/8755的引线图和引线逻辑图如图16-14所示。

8355引线功能说明如下：

① ALE——当ALE为高电平时，$AD_{7\sim0}$、IO/$\overline{M}$、$A_{10\sim8}$、CE和$\overline{CE}$进入地址锁存器。

② $AD_{7\sim0}$——双向地址/数据总线。当ALE为高电平时，出现在总线上的是ROM的低8位地址或I/O地址。在I/O周期中，根据AD_0的锁存值来选择端口A或B。当锁存的CE为有效时，若$\overline{RD}$或$\overline{IOR}$为低电平，则输出缓冲器将数据送到总线上。

③ $A_{10\sim8}$——ROM的高位地址，它不影响输入/输出的操作。

④ $\overline{CE}$，CE——芯片允许输入端，仅当$\overline{CE}$信号为低电平，且CE是高电平时，才能允许8355工作。当任一个芯片允许无效时，$AD_{7\sim0}$和READY的输出将处于高阻抗状态。

⑤ IO/$\overline{M}$——当$\overline{RD}$为低电平时，若锁存的IO/$\overline{M}$为高电平，则输出数据来自输入/输出端口。若IO/$\overline{M}$为低电平，则输出数据来自ROM。

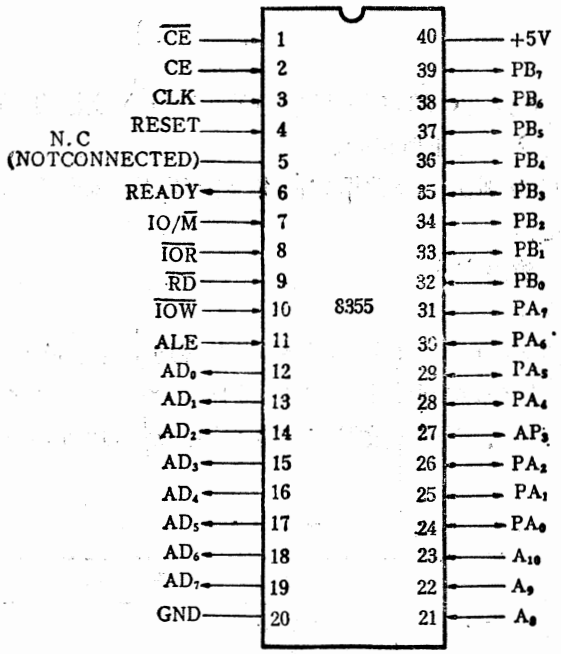


图 16-14 8355 引线图

⑥ $\overline{RD}$ 读控制——当 $\overline{RD}$ 为低电平，且锁存的 CE 信号有效时，则 $AD_{7\sim0}$ 输出缓冲器允许工作，且输出来自选中的 ROM 或 I/O 端口。当 $\overline{RD}$ I和 $\overline{OR}$ 均为高电平时，则 $AD_{7\sim0}$ 输出缓冲器处于高阻抗状态。

⑦ $\overline{IOW}$——如果锁存的 CE 信号有效，且 $\overline{IOW}$ 为低电平，将使 $AD_{7\sim0}$ 上的数据写入 8355 的端口 A 或 B (它是由 AD_0 的锁存值来确定的)。此时 IO/$\overline{M}$ 的状态不影响操作。

⑧ CLK——当 $\overline{CE}$ 低电平、CE 高电平和 ALE 高电平迫使 READY 为低电平后，用 CLK 使 READY 进入高阻抗状态。

⑨ READY——由 $\overline{CE}$、CE、ALE 和 CLK 控制的三态输出线。当 ALE 为高电平，且 CE 信号有效时，就迫使 READY 为低电平，一直到下一个 CLK 的上升沿为止。因此当 READY 被连接到 8085A 的 READY 线时，就能保证从 8085A 得到一个时钟周期的等待时间。

⑩ $PA_{7\sim0}$——通用的输入/输出引线，它们的输入/输出流向是由数据方向寄存器(DDR)的内容确定的。当 CE 信号有效，$\overline{IOW}$ 为低电平，且由 AD_0 预先锁存的是“0”时，端口 A 选定进行写操作。当 CE 有效，$\overline{IOR}$ 为低电平，且由 AD_0 预先锁存的是“0”时，端口 A 选定读操作。此外，当芯片被启用，且 AD_0 为低电平，可用 IO/$\overline{M}$ 为高电平和 $\overline{RD}$ 低电平来代替 $\overline{IOR}$，而允许从一个端口读出。

⑪ $PB_{7\sim0}$——通用的输入/输出引线，其功能与端口 PA 相似，区别仅在于，它是由 AD_0 锁存的“1”来选中通道工作的。

⑫ RESET——RESET 输入为高电平时，将使端口 A 和端口 B 中的所有引线都置于输入方式。

⑬ $\overline{IOR}$——当 CE 信号有效，且 $\overline{IOR}$ 为低电平时，将把选中的 I/O 端口送到 AD 总线上。当 $\overline{IOR}$ 为低电平时，执行的功能与 IO/$\overline{M}$ 为高电平和 $\overline{RD}$ 为低电平时所执行的功能相同。

⑭ V_{CC}——+5V 电源

V_{SS}——接地

(2) 8355 的操作说明

8355 的两个通用的 8 位 I/O 端口，可以单独编程为输入或输出，这使用户能从 16 个输入到 16 个输出之间选择 I/O 位的任何组合。

PA、PB 端口的各位分别由两个数据方向寄存器(DDR)来确定相应端口的每一 I/O 引线的数据流向。8355 规定：

DDRAi＝1　PAi 为输出方式

DDRAi＝0　PAi 为输入方式

8 位的数据方向寄存器 DDRA（或 DDRB）的内容均由程序员通过 CPU 对累加器 A 设置状态控制字，并执行输出指令，送到给定的 DDR 的地址 ×××××× AD_1 AD_0 来实现。

PA、PB 端口及 DDRA、DDRB 数据方向寄存器的通道地址由 AD_1、AD_0 的锁存值来寻址。地址分配如下：

表 16-8

AD_1	AD_0	通　道　选　择	引　线
0	0	端口 A	$PA_{7\sim0}$
0	1	端口 B	$PB_{7\sim0}$
1	0	PA 的数据方向寄存器(DDRA)	无
1	1	PB 的数据方向寄存器(DDRB)	无

例如，欲将 8355 的 PA、PB 编制为输出工作方式，则可执行下列指令：

```
MVI A,FFH;  将 8355 DDR 命令值置入 A
OUT 02H   ; 把命令——→PA 的 DDRA
OUT 03H   ; 把命令——→PB 的 DDRB
```

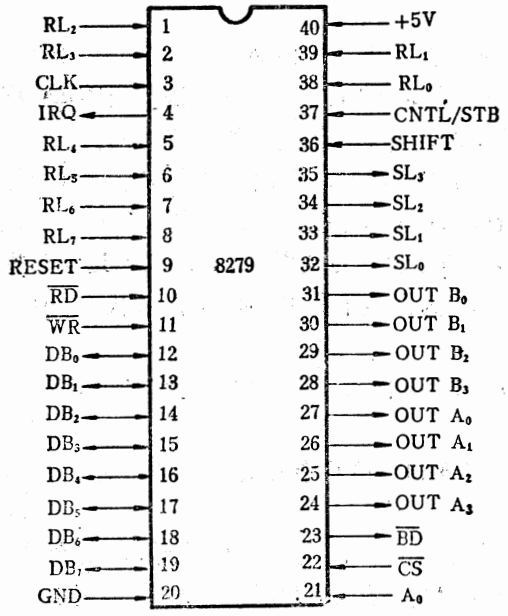


图 16-15(a)　8279 引线图

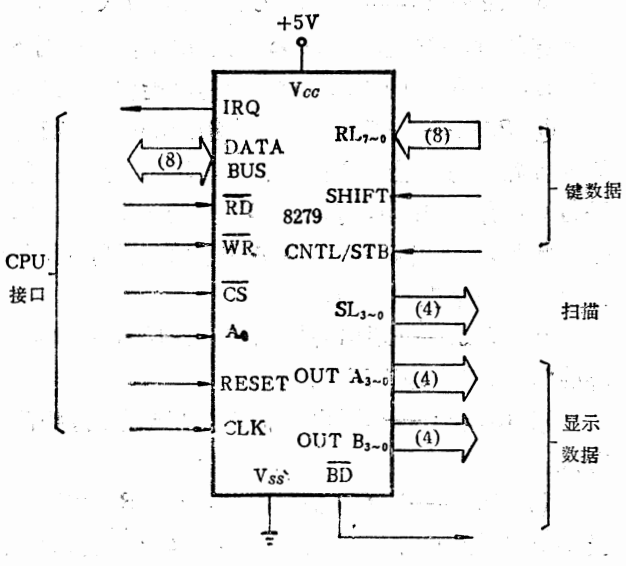


图 16-15(b)　8279 逻辑符号图

3．8279 可编程序的键盘/显示器控制器

8279 是 Intel 公司专为控制键盘和显示器而设计、研制的专用接口芯片——可编程序键盘/显示器控制器。它可直接与 64 键的键盘（可扩充为 128 键）和 16 个发光二极管显示器接口，并能自动地提供键盘去抖动和实现键扫描功能。这就使得键盘监控程序大为简化，工作效率显著提高。

(1) 8279 的结构和引线

8279 内部有键盘和显示器控制两个部分。其内部有一个 16×8 显示器随机存贮器和一个 8×8“先进先出”(FIFO) RAM、后者是一个把来自键盘的信息保存起来的等待表。当键盘送入信息时，8279 向 CPU 输出中断请求信号。

8279 的引线图和引线逻辑图如图 16-15。引线名称表见表 16-9。

表 16-9 引线名称

名　　称	I/O	功　　能
$DB_{7\sim0}$	I/O	双向数据总线
CLK	I	时钟输入
RESET	I	复位输入
$\overline{CS}$	I	片选
$\overline{RD}$	I	读输入端
$\overline{WR}$	I	写输入端
A_0	I	缓冲器地址
IRQ	O	中断请求输出端
$SL_{3\sim0}$	O	扫描线
$RL_{7\sim0}$	I	回送线
SHIFT	I	字型变换输入端
CNTL/STB	I	控制/选通输入端
$OUTA_{3\sim0}$	O	(A)显示器输出端
$OUTB_{3\sim0}$	O	(B)显示器输出端
$\overline{BD}$	O	空格显示输出端

(2) 键盘/显示器的操作原理

我们结合 8279 键盘/显示器接口芯片的应用来说明它的操作原理。8279 与 8085 A CPU 及键盘，显示器的接口框图如图 16-16 所示。

8085A CPU 与 8279 的连接信号有：

DB_7～DB_0　双向数据总线，用于在 8085A CPU 和 8279 之间传送数据和命令信息。

$\overline{CS}$　片选信号，当 $\overline{CS}=0$ 时，8279 被选中，否则不被选中，此时，$DB_{7\sim0}$ 为高阻抗状态。

$\overline{RD}$，$\overline{WR}$　读写控制信号，用于决定 I/O 的数据流向。

A_0　缓冲器地址。当 $A_0=0$ 时，表示 I/O 信息为数据。当 $A_0=1$ 时，表示 I/O 信息应为命令或状态。

CLK　系统时钟脉冲，用作 8279 内部的定时信号，以与系统同步。

RESET　系统总清。当其为高电平时，使 8279 复位。

IRQ　中断请求线。在键盘工作方式时，当在 FIFO RAM 中有数据时，此中断请求线变为高电平，向 CPU 发出中断请求。在 FIFO RAM 每次读出时，中断请求线就变为低电平。如果 RAM 中还有信息，则此线重新变为高电平。

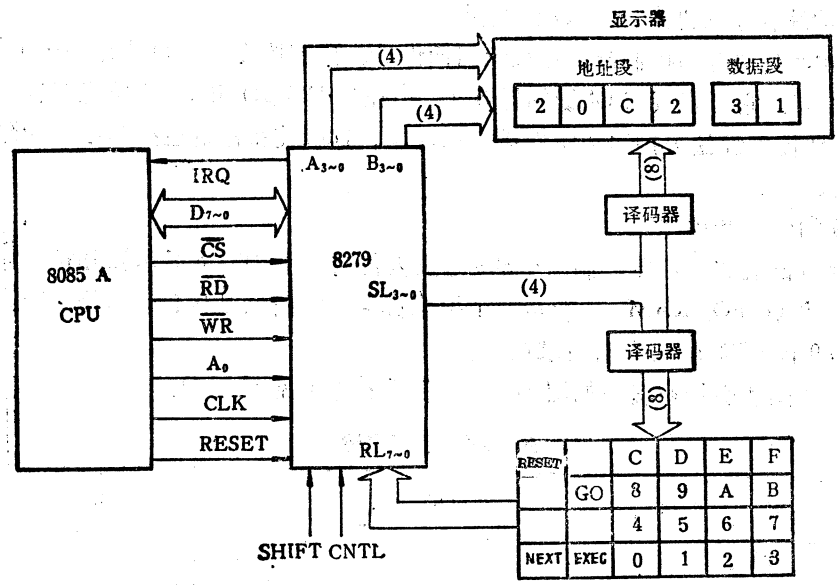


图 16-16　8279 键盘、显示器接口逻辑框图

8279 有键盘、显示器两部分接口电路。现键盘部分接有 24 个接触式按键(最多可扩展到 128 个键)。显示器部分接有发光二极管(LED)组成的 6 位数码显示器,其中 4 位为地址段,2 位为数据段(最多可显示 16 位)

8279 所有的操作方式都是由 CPU 送入的命令字来设定的。它的工作过程说明如下:

首先由 8279 的定时控制器通过 4 条扫描线 $SL_{3\sim0}$, 经译码器译码输出最多可达 16 条扫描线。现选 3 条 $SL_{2\sim0}$, 可译出 8 条扫描线,用作键盘行扫描线,并作为 8 根扫描回送线 $RL_{7\sim0}$ 的列扫描线。如图 16-16 所示。

当某一键闭合时,其中 $RL_{7\sim0}$ 回送线中必有一条线变成低电平,其余线都为高电平。由于在键盘工作方式时,这几条回送线被逐排扫描,以寻出在哪排键中的哪个键闭合。当反跳电路探测到某一闭合键时,它就等待 10 毫秒,然后再重核此键是否仍然闭合;如果仍闭合,那末该键在按键阵列中的地址信息(表示按键的 6 位编码)、字型变换 (SH FT) 和控制状态 (CONT/STB) 被送入 FIFO 存贮器,每按一次键,就送入一次。FIFO RAM 里最多能放 8 个字节(8×8 位二进制数),它的状态跟踪 FIFO 中的字符数量。判定是"满"还是"空",写入或读出过多均被看作出错。当使 $\overline{RD}$、$\overline{CS}$ 为低电平且 A_0 为高电平时,就可把此状态读出,在 FIFO 非"空"时,状态逻辑还提供一个中断请求信号(IRQ),以把 FIFO 中的内容读入 CPU,同时在软件控制下,将该字符显示在显示器上。假如所按的一个键是"2",显示器就显示一个字符"2";若按"20C2"4 个键,则在地址段显示出"20C2"4 个字符。

显示器显示原理是:8279 的扫描线 $SL_{3\sim0}$ 输出 (现选 3 条 $SL_{2\sim0}$) 经译码器译码后,译出 8 条扫描线,可控制 8 位字符显示器(现只用 6 位显示器)。8279 的显示器地址寄存器保存正由 CPU 读或写的那个字符的地址,经显示器 RAM 取出要显示字符的两个 4 位二进制字段,(A 字段和 B 字段)。A 和 B 两字段可以分别送入,也可作为一个字符送入,数据送入显示器的工作方式可以由左端送入或由右端送入,这由 CPU 所设置的工作方式而定。

显示每一字符需要一个字节,即 8 位二进制数据,其数据格式如下:

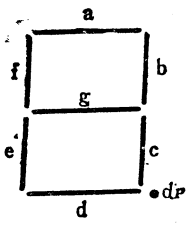


图 16-17 八段显示器的结构

最高位							最低位	
A_3	A_2	A_1	A_0	B_3	B_2	B_1	B_0	←8279 的显示器 RAM 输出
d	c	b	a	d.p.	g	f	e	←显示器发光二极管的段

SDK-85 所用的 8 段数码显示器的结构如图 16-17 所示。

要显示的字符与扫描线 $SL_{3\sim0}$ 同步送到显示器。每一位显示器由 7 段发光二极管组成一个字符，故一个字符所用的 8 位二进制数，其中 7 位分别对应 7 段发光二极管中的一段，另一位则对应小数点。

显示器硬件已设计成把一个“0”写入某一位，就点亮对应段的发光二极管。例如，显示“4”时，b、c、f、g 段应点亮，各相应位为“0”，其余位则为“1”，此时，显示数据的格式为 10011001 = 99H。

这样，在 CPU 编程的控制下，8279 就能实现同时控制键盘和显示器的操作。

16 字符显示器 RAM 除了用按键送入显示外，CPU 中的随机存贮器 RAM 的数据也可通过 8279 接口直接进行显示。

6 位显示器的排列如下所示。

地址段				数据段	
1	2	3	4	1	2

显示的数字存放在 8279 的显示器 RAM 单元中列于表 16-10。

表 16-10

8279 的显示器 RAM 单元	显 示 器 位 置
0	地 址 位 1
1	地 址 位 2
2	地 址 位 3
3	地 址 位 4
4	数 据 位 1
5	数 据 位 2
6	未 用
7	未 用

§16-4 Z80 微型计算机系统

一、基本的 Z80 微型计算机系统

图 16-18 示出基本的 Z80 微型计算机系统，它由 Z80 CPU、存贮器和 PIO 接口芯片，再加上电源、振荡器和复位电路等组成。

振荡器要求产生 5V 的方波，可用晶体振荡器。若系统不要求全速工作，则可用简单的 RC 振荡器。

外部存贮器可以由标准的 RAM、ROM 或 EPROM 混合构成。在本例中只用一个 8K 位的 ROM(1K 字节)存贮器。

在本例中用 Z80-PIO 作 I/O 接口电路。在这里只用了三块 LSI 电路，一个简单的振荡器

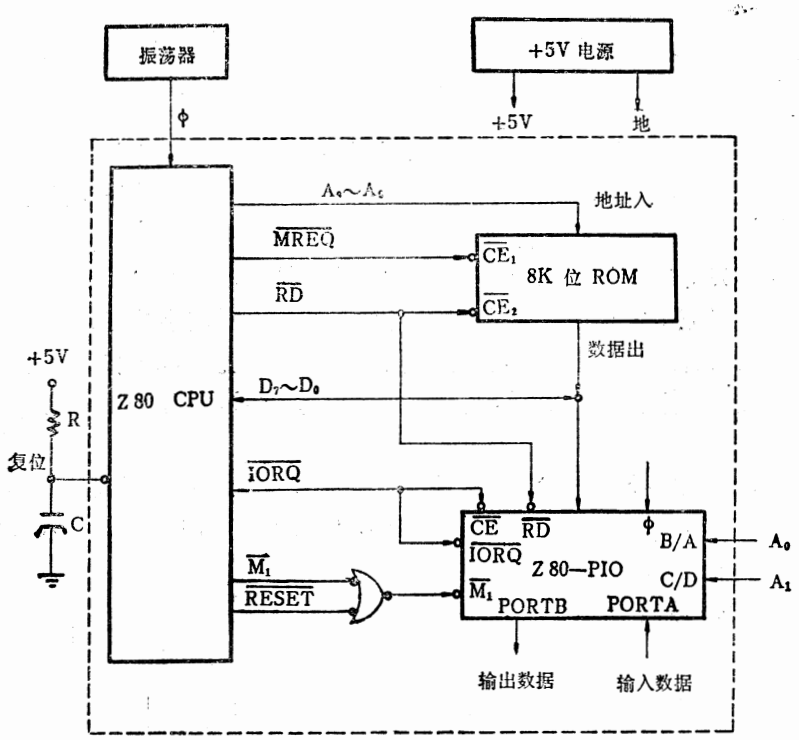


图 16-18 基本的 Z80 微型计算机系统

和一个 5 伏电源，就组成了一台具备初步功能的微型计算机系统。

如果再利用 Z80 系统的 4 种基本外围接口电路芯片 Z80 CTC、Z80 PIO、Z80 SIO 及 Z80 DMA 等，则可构成较高性能的微型计算机系统。

二、Z80 STARTER SYSTEM KIT 单板微型计算机

Z80 STARTER SYSTEM KIT 是美国 S. D SYSTEMS 公司生产的单板微型计算机。自 1980 年引进以来，许多单位承接装配，有的还对其监控程序进行若干修改，出现了象 DBJ-Z80，TP801-Z80 等拥有广大用户的单板微型计算机。该机结构简单，功能齐全。它可用于工业控制、数据处理等，尤其适合于作为初学者学习微型计算机的硬件和软件知识的工具，而且该机还具有简易的开发功能：如单步、断点和对 EPROM 进行编程等，故也可用于调试样机。

1．系统的组成和功能特性

(1) 系统逻辑框图

Z80 STARTER SYSTEM KIT 单板微型计算机的原理框图如图 16-19 所示。

由框图可知，该机由以下的三类功能部件组成：

① Z80 CPU；

② 存贮器：有 ROM、$PROM_1$、$PROM_2$、RAM；

(3) I/O 接口电路：有 Z80 PIO、Z80 CTC、键盘/显示器接口和录音机接口。

此外，还有译码电路、时钟和复位电路等。

(2) 主要功能特性

① 中央处理单元为高性能的 Z80 CPU，具有 150 条基本指令。

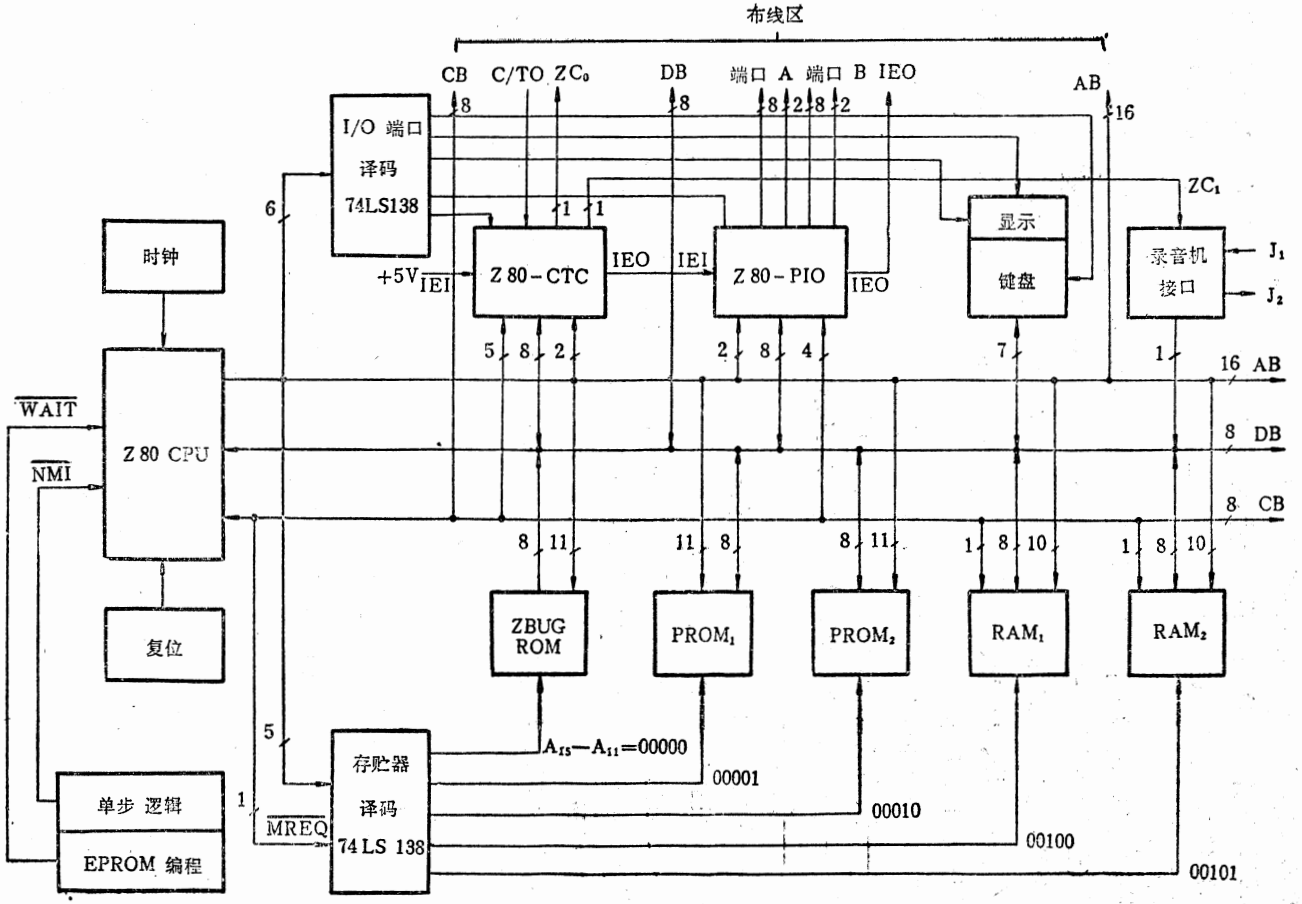


图 16-19　Z80 STARTER SYSTEM KIT 单板微型计算机原理框图

时钟(φ)频率为1.9968MHz，以便于使用8080A的接口电路。晶体振荡器的频率为3.9936MHz。

② RAM为4K字节的2114静态随机存贮器，也可只使用2K字节。

③ ROM为2K字节，编入监控程序ZBUG。

④ PROM插座两个，可插入4K字节的EPROM(2716)。

⑤ Z80 PIO并行I/O接口芯片一片，它有两个8位的可编程I/O口，供用户使用。

⑥ Z80 CTC计数器/定时器芯片一片，它有四个通道，其中通道0可供用户使用，其余的供监控程序使用。

⑦ 按键共28个，16个为十六进制数字键(其中有8个兼作CPU寄存器寻址用)。命令键为12个(当采用TPBUG-A时，12个命令键中有8个为双挡命令键)。

采用ZBUG或TPBUG时，命令键为：存贮单元检查、通道检查、寄存器检查、辅助寄存器检查、连续执行程序、单步执行程序、设置断点、磁带输入、信息转贮磁带、EPROM写入和MON(监控)及NEXT键。

采用TPBUG-A时，除保持上述12个命令功能外，还增设了以下8个命令功能键：

MON′——交换键。用于双功能键的上下挡更换。

LAST——前一个存贮单元地址访问键(与NEXT对应)。

MOVE——数据块移动操作键。

DISP——相对转移指令偏移量计算操作键。

四个用户自己定义的命令键(2FB8H, 2FBAH, 2FBCH, 2FBEH)。

⑧ 显示器有六位LED数码显示，通常左四位显示地址，右两位显示数据。

⑨ 配有音频盒式磁带机接口。可将RAM中的信息转贮到盒式磁带，或将盒式磁带中的信息输入到RAM。

⑩ 电源为+5V±5%，1A。若要对EPROM进行写入，尚须接入+25V±1V，30mA的直流电源。

2．存贮器寻址

(1) 存贮空间分配

存贮空间分配及片选信号见表16-11。

表16-11 Z80 STARTER的存贮空间分配表

地　　址	器　　件	A_{15}～A_{11}	A_{10}～A_0	译码器输出
3800～3FFFH	N/C	00111	可　变	$\overline{Y_7}=\overline{CS_7}$
3000～37FFH	N/C	00110	可　变	$\overline{Y_6}=\overline{CS_6}$
2800～2FFFH	2KB RAM_2	00101	可　变	$\overline{Y_5}=\overline{CS_5}=\overline{RAM_2\ SEL}$
2000～27FFH	2KB RAM_1	00100	可　变	$\overline{Y_4}=\overline{CS_4}=\overline{RAM_1\ SEL}$
1800～1FFFH	N/C	00011	可　变	$\overline{Y_3}=\overline{CS_3}$
1000～17FFH	2KB $PROM_2$	00010	可　变	$\overline{Y_2}=\overline{CS_2}=\overline{PROM_2\ SEL}$
0800～0FFFH	2KB $PROM_1$	00001	可　变	$\overline{Y_1}=\overline{CS_1}=\overline{PROM_1\ SEL}$
0000～07FFH	2KB ROM	00000	可　变	$\overline{Y_0}=\overline{CS_0}=\overline{MON\ SEL}$

表中，0000～07FFH为2K字节的ROM，用来存放ZBUG监控程序。

0800～0FFFH为2K的$PROM_1$，可存放用户程序。

1000～17FFH为2K的$PROM_2$，也可存放用户程序。此外，ZBUG还可对$PROM_2$插座

中的 EPROM 进行编程。

1800～1FFFH 为 2KB，未被使用。

2000～27FFH 为 2KB 的 RAM_1。

2800～2FFFH 为 2KB 的 RAM_2。

其中 2F90～2FFFH 为监控程序的堆栈缓冲区。

3000～37FFH 为 2KB，未被使用。

3800～3FFFH 为 2KB，未被使用。

(2) 存贮器地址译码

该机存贮器的译码是由 3—8 译码器 74LS138 来完成的。如图 16-20 所示。

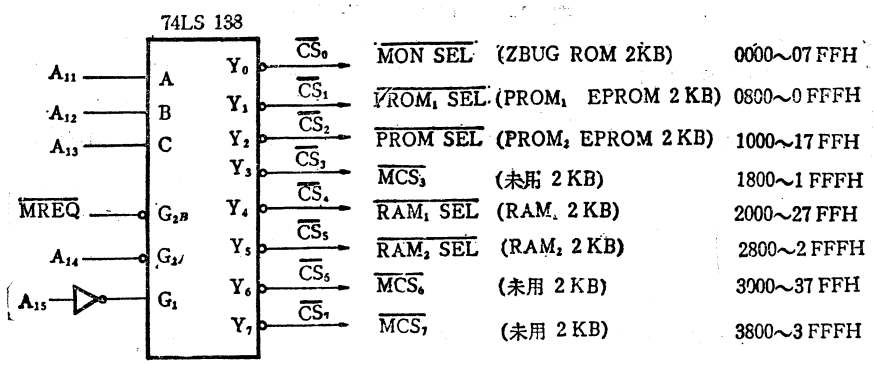


图 16-20 Z80 STARTER SYSTEM KIT 存贮器地址译码器

译码器的每条输出线接至 2KB 存贮器的 $\overline{CE}$ 端或 $\overline{CS}$ 端，故该机在不另加译码器情况下，可配 16KB 存贮器。由于规定 A_{15}、A_{14} 为 0，因此，16KB 存贮器的地址空间为 0000～3FFFH。

该机存贮器与 CPU 之间的连接。可参考图 19-48。

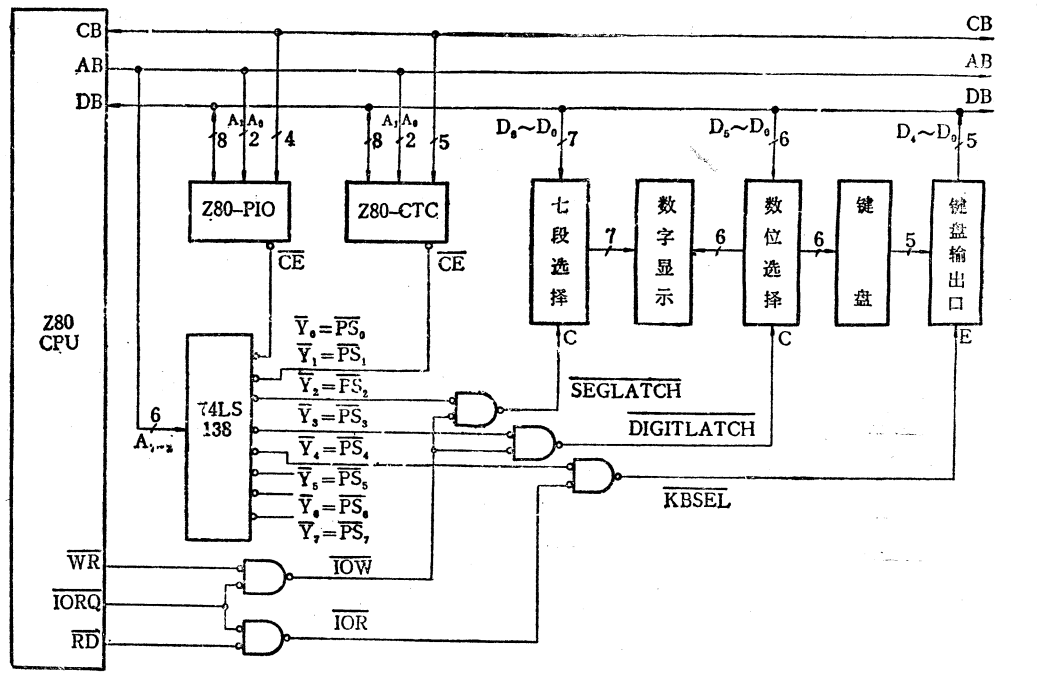


图 16-21 I/O 接口地址译码电路

3．I/O接口寻址

(1) I/O接口地址译码及地址空间分配

I/O接口地址译码也是用3-8译码器74LS138来完成的。如图16-21所示。3-8译码器将I/O接口中的每4个端口地址作为一组，每次选中一组如表16-12所示。

(2) Z80 PIO

Z80 PIO的功能已在§13-3叙述过。在该机中，Z80 PIO的两个端口全部可供用户使用。

(3) Z80 CTC

表16-12 I/O端口分配表

A_7-A_2	译码器输出	器件	A_1A_0	通道	通道地址
100000	$\overline{Y_0}=\overline{PS_0}=\overline{PIO\ SEL}$	Z80PIO	0 0	端口A数据寄存器	80H
			0 1	端口B数据寄存器	81H
			1 0	端口A控制寄存器	82H
			1 1	端口B控制寄存器	83H
100001	$\overline{Y_1}=\overline{PS_1}=\overline{CTC\ SEL}$	Z80CTC	0 0	通道0	84H
			0 1	通道1	85H
			1 0	通道2	86H
			1 1	通道3	87H
100010	$\overline{Y_2}=\overline{PS_2}=\overline{SEG\ LH}$	74 LS 273 8位锁存器	××	字模选择(7段选择)(只写)	88H～8BH
100011	$\overline{Y_3}=\overline{PS_3}=\overline{DIG\ LH}$	74 LS 273 8位锁存器	××	数值选择(只写)	8CH～8FH
100100	$\overline{Y_4}=\overline{PS_4}=\overline{KB\ SEL}$	74 LS 244 8位缓冲器	××	读键值(只读)	90H～93H
100101	$\overline{Y_5}=\overline{PS_5}$	N/C			94H～97H
100110	$\overline{Y_6}=\overline{PS_6}$	N/C			98H～9BH
100111	$\overline{Y_7}=\overline{PS_7}$	N/C			9CH～97H

Z80 CTC的功能已在§13-2中作过叙述。在该机中，Z80 CTC的4个通道的作用如下：

通道0——供用户使用，中断服务程序的入口地址为2FD6H，用户可在2FD6H处放置一条转移指令，使程序转移到中断服务程序。

通道1——用于将RAM中信息转贮到音频盒式磁带中。

通道2——用于单步逻辑和EPROM编程器。

通道3——用于将盒式磁带中的信息装入到RAM中。

(4) 键盘和显示器

这一部分内容将在第十七章中详细介绍。

第十七章　微型计算机应用系统设计导论

本章将着重介绍微型计算机应用系统，特别是微型计算机控制系统设计中的一些共性问题。这里所涉及的技术问题是多方面的，其中合理地解决 I/O 接口技术和编制应用程序等问题是设计微型计算机应用系统的关键。

§17-1　微型计算机应用系统设计概述

由于微型计算机具有价格低、体积小，性能可靠和使用灵活等特点，因此在许多应用系统中，正在取代尚未充分应用的小型计算机和组合逻辑硬件。这一类的应用在微型计算机应用中大约占 60%～70%，取代后的系统成本可降低 15%～60%。研制周期可缩短 1/3～2/3。微型计算机应用系统的共同特点是：系统的控制功能是通过软件来实现的。即在不更换硬件系统情况下，仅通过改变软件，就能更改系统的控制功能以适应新的要求。

目前，微型计算机应用方式主要可分以下两种：一种是直接购买单板微型计算机或微型机系统装置，在此基础上，根据实际需要扩展内存贮器和 I/O 接口，并编制相应的应用程序；另一种是根据用户的需要购买微处理器和必要的存贮器片以及 I/O 接口芯片，自行设计微型计算机应用系统的硬件和软件。两种方式各有利弊，究竟采用哪一种方式，应根据实际需要和条件来定。

一、微型计算机应用系统的设计步骤

在设计微型计算机应用系统之前，应当首先明确在该应用系统（或称目标系统）中引入微型计算机的必要性。这就必须在对系统性能、可靠性、可维护性和经济效果等因素进行综合考虑后，才能决定在应用系统中是否采用微型计算机。通常在论证一个应用系统是采用组合逻辑结构还是采用微型计算机时，可参考图 17-1 所示的分析方法。

当决定在应用系统中采用微型计算机之后，其设计步骤大致如下：

1．系统设计任务的确定

在进行微型计算机应用系统设计之前，必须对应用对象的工作过程进行详尽的调查分析，以确定微型计算机在应用系统中应承担的功能。这一步骤的注意力应集中在微型计算机需完成的各种功能和需满足的定时关系上。可用时间和控制流程图来详尽地描述系统的工作过程。进一步还要明确需要何种类型的微型计算机和外围设备，最后画出系统框图作为应用系统设计的依据。

2．系统功能部件的初估和选择

系统设计任务确定以后，应根据应用系统的规模、输入输出方式、外围设备的类型和数量、应用软件占用的内存贮器容量以及成本等因素，初估和选择所需要的功能部件，并对各种选择方案进行比较。主要应对构成微型计算机系统的三个主要功能部件——微处理器、存贮器和 I/O 接口电路进行初估和选择。

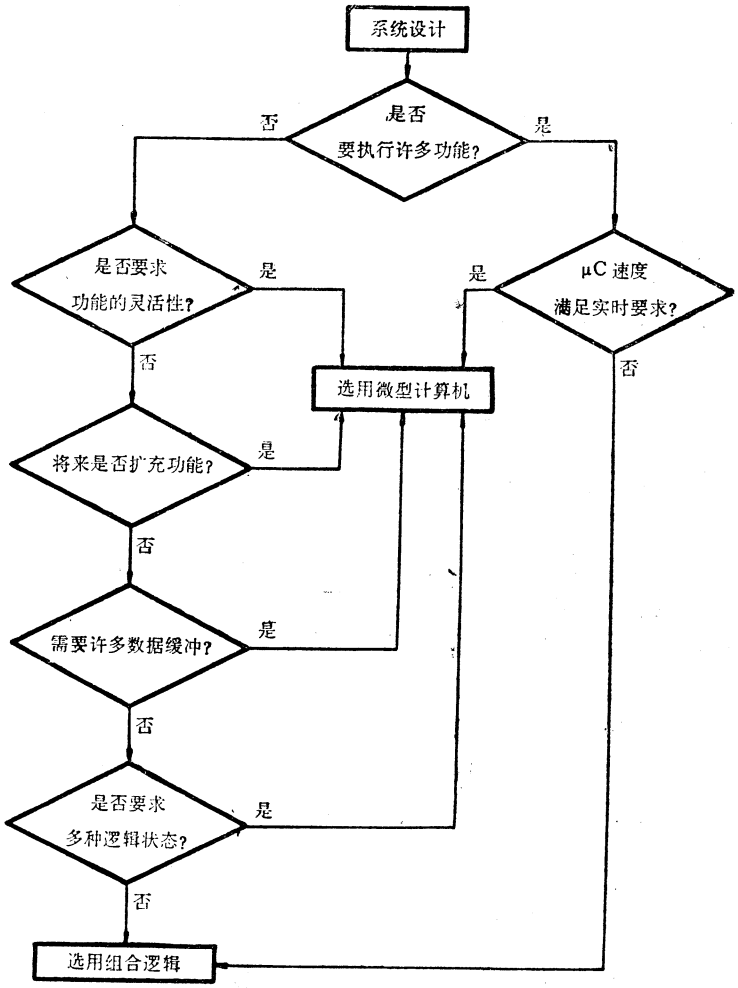


图 17-1　微型计算机或组合逻辑的选用

(1) 微处理器的选择

微处理器是微型计算机应用系统的核心部件，它的合理选择是整个系统设计的关键所在。设计人员应根据不同的应用要求，选择合适的微处理器，例如，用于数据处理任务的系统，应选用面向计算的微处理器；用于工业控制的系统则应选用面向控制的微处理器。一般在估计和选择微处理器时可从以下几个方面考虑：

① 字长的选择

微型计算机字长直接影响到应用系统的精度、功能和速度。一般而言，字长越长，微处理器功能越完备，所以希望字长能长点。但从减少辅助电路的复杂性和降低成本来考虑，则字长短一点为宜。究竟选择多少字长为宜，应根据具体情况来定。表 17-1 列出了目前不同字长的微处理器的性能特点和应用范围的比较。

② 处理速度(或指令执行时间)

执行速度的选择需视具体情况而定。若应用对象对速度要求不高，例如对于温度、压力等变化较慢的变量的控制，采用 4 位、8 位微处理器是比较合适的。对于有大量实时处理任务的生产过程控制系统，应选择速度较高的 8 位微处理器或 16 位微处理器。如果应用系统要求速

表 17-1 不同字长的微处理器性能比较表

字　长	4　位	8　位	16　位
成　本	低	中	较　高
特点与功能	适宜于10进制运算，指令功能简单，速度低，内存容量很小，有限的I/O能力，不可扩展性。	指令功能较丰富，速度有较大提高，内存容量可扩至64KB，扩大的I/O能力，有中断、DMA功能、可扩展性。	指令功能丰富，速度有很大提高，内存容量大于64KB，很强的I/O能力，具有中断、DMA功能，扩展性好。
应用范围	适用于精度、速度要求不高的场合，一般面向消费品，如家用电器、计算器、简单控制器、娱乐游戏用品等。	适用于精度、速度要求一般的场合，一般面向工业控制、智能仪器仪表、智能终端等。	适用于精度、速度要求较高的场合，一般面向实时数据处理、实时控制和科学计算。如数据库、企业管理、多微机系统等。

度很高，即响应时间为微秒数量级时，应考虑选用位片式微处理器或高性能的小型计算机。

③ 支援芯片和软件兼容

目前有三种有代表性的8位微处理器，在国内外得到广泛应用。一种是Intel公司的8080A及其支援芯片，它相当于国产的050系列；另一种是Motorola公司的MC6800及其支援芯片，它相当于国产的060系列；第三种是Zilog公司的Z80及其支援芯片。比较而言，Intel系列是应用得最广的微处理器。这是由于Intel公司最先占领市场，产品丰富，支援芯片完备，软件支持能力强，产品第二来源较多等，尤其是在8080A的基础上，又试制成功高性能的8位微处理器Intel 8085A、16位微处理器Intel 8086以及单片微型计算机Intel 8048/8748等，这更增强了Intel 8080系列的竞争能力。Z80和M6800系列也在相当广泛的范围内获得了应用。在我国，目前Z80系列的应用范围最为广泛。

④ 加载(备用扩充容量)

在进行初步系统设计时，一般加载大约50%。例如，若完成某一功能需要10 ms时间，则希望微处理器执行任务的时间为5 ms，留有足够的容量以备将来扩充之用。

(2) 存贮器容量的初估和确定

应用系统存贮容量的估算取决于整个系统所需要的程序和数据的容量。如果系统所需的存贮器容量较大，则应增设相应的总线驱动器。此外，还应确定是否需要外存贮器。

(3) I/O接口电路的选择

在估计与选择I/O接口电路时，应考虑如下几个问题：

① 数据采集和传输需要多少个I/O通道？

② 是否所有的I/O通道都使用同样的数据传输率？

③ I/O通道是串行通信还是并行通信？

④ I/O通道是随机选择还是按某种预定的顺序工作？

⑤ I/O数据的字长为多少位？

以Intel 8080A系列为例，8212、8251、8255A是最基本的I/O接口芯片。当系统中的微型计算机需要与电传打字机或盒式磁带机连接时，可选用可编程序的串行通信接口8251或Z80-SIO；当微型计算机的外围设备需要有定时装置时，可选用8253或Z 80 CTC；当所设计的应用系统有多个外围设备，而且它们的工作顺序有规定的要求时，可选用8214或8259芯片实现优先中断。

除了上述芯片外，系统还需要一些中、小规模集成电路芯片，以用于设计辅助电路，此类芯片的品种繁多，可查阅有关产品手册。

3．系统设计

微型计算机应用系统设计分硬件设计和软件设计两大部分。通常软、硬件应同时地进行设计。系统设计步骤示意图如图 17-2 所示。

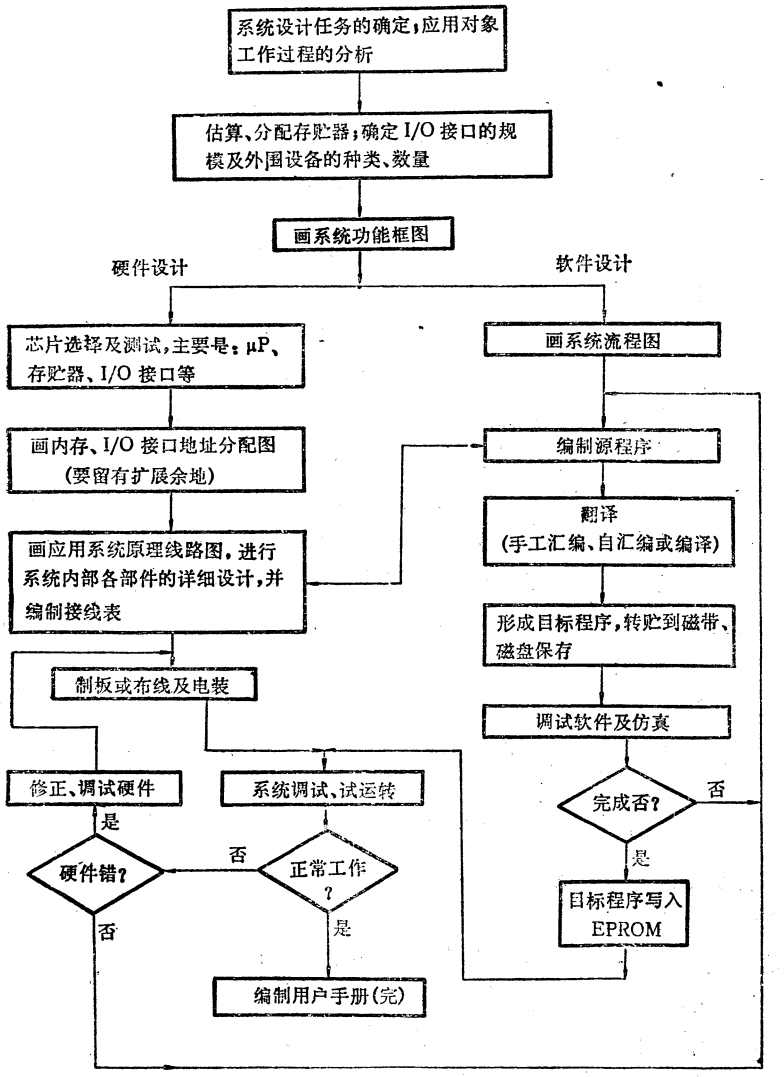


图 17-2　系统设计步骤示意图

(1) 硬件设计

由于在微型计算机系统中，各个功能部件大部分使用 LSI 电路芯片，不但组件数量减少了，而且设计者所需要的线路技术也比使用中、小规模组件设计容易。其设计步骤如图 17-2 左边部分所示。但是，在输入输出设计方面，具体如何连接不同的外围设备和控制对象(例如键盘、打印机、磁带机，A/D 和 D/A 转换器等)，给系统带来了复杂性。输入输出接口及其控制程序的设计，通常是系统设计中最重要，最复杂的部分。当然，当硬件采用单板微型计算机

或微型计算机系统装置时，它的设计就稍许简单些。

硬件设计结束后，就进入电装阶段，在电装中应重点注意系统的可靠性，排除早期损坏的组件和克服外来的电气干扰等。

(2) 软件设计

如图 17-2 右边所示。首先根据系统框图画出控制流程图，然后用合适的程序设计语言如汇编语言编制应用程序。编成的源程序可采用手工汇编或机器汇编，翻译成目标程序。微型计算机应用系统的性能，在很大程度上取决于软件设计的质量。如果有条件的话，使用微型机开发系统 MDS(Micro Computer Development System) 进行设计，那是比较理想的。

4．系统调试

系统硬件、软件设计完成之后，必须对系统进行调试，即将应用程序输入到样机系统试运转，以检查系统是否正常工作。若工作不正常，就需要分析原因，排除故障。如果是硬件发生故障，则应修正硬件设计，重新制板或布线电装；如果是软件发生故障，则应修改程序，重新调试，直至完全符合系统设计的要求为止。最后将调试好的应用程序固化于 EPROM 中。

二、微型计算机应用系统的研制工具

在微型计算机应用系统的硬件设计和软件设计过程中，整个设计过程不但并行进行，而且在软件调试中还需要硬件的密切配合，可是此时硬件又往往正处于研制过程中，因而，为了缩短系统的研制周期，微型计算机开发系统便成为理想的研制工具。对于小型的应用系统而言，也可在具有简易开发功能的单板微型计算机或微型计算机系统装置上进行软件调试。

所谓微型计算机开发系统就是用来研制微型计算机系统和微型计算机应用系统的通用的设计和调试工具。它是一种综合性的软件和硬件的研制工具。通常，它是由常规计算机的主要部件(如微处理器、存贮器、输入/输出设备)，一些软件研制工具(如编辑程序、调试程序、跟踪程序和模拟程序等)以及一些硬件研制工具(如联机仿真器 ICE、PROM 编程器等) 组成的。它能辅助设计人员设计和调试程序，能仿真所设计的应用系统的中央处理器、存贮器以及输入/输出系统，并且可把硬件和软件综合起来进行调试和研制。

微型计算机开发系统一般应具备以下性能：

(1) 在面板上显示工作状态，以便于设计人员观察总线和存贮单元的内容。

(2) 应能方便地改变存贮单元的内容，以便于系统的调试。

(3) 设有“复位”控制，能使微处理器从某个已知状态启动、运行。

(4) 设有“单步”控制功能，允许程序每次执行一条指令，并可显示每步的工作状态，以便于系统调试和诊断。

(5) 具有运行控制功能，允许程序从某个特定的存贮单元开始执行。

(6) 有一定容量的 RAM/PROM (或 EPROM) 作为程序存贮器。

(7) 具有通用 PROM 编程器(UPP)，以便将调试好的程序固化于 EPROM 中。

(8) 配有比较齐全的外围设备接口，最简单的 MDS 主要配有电传打字机接口，较完善的 MDS 还配备有纸带输入机、键盘、CRT 显示器、行式打印机、盒式磁带机以及软、硬磁盘系统等。

(9) 配有丰富的系统软件，它主要包括有操作系统、监督程序、编辑程序、汇编程序、编译程序、仿真程序、诊断程序和调试程序等。

MDS 的硬件系统通常由中央处理器插件板（内部包括 32～64KB RAM、4KB ROM）、输入输出控制板、显示器、键盘以及可存放操作系统的软盘驱动器等组成。

这种系统的组成，如图 17-3、图 17-4 所示。它大致可分为两种工作方式，即主从处理器方式和单处理器方式。

所谓主从处理器方式，就是在 MDS 中使用联机仿真器作为从处理器。代替应用系统内选用的微处理器。例如用 ICE-80 中的 8080A 取代应用系统中的 Intel 8080A 就是一例。使用时将它通过电缆插头插入应用系统中，进行应用系统的仿真、调试、修正应用程序的错误和硬件故障。在联机仿真器外，还有一个主处理器，它在调试中提供系统软件，起着支援作用。

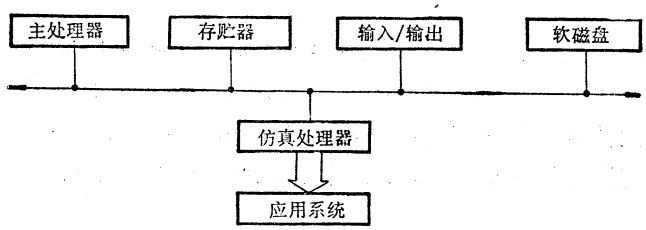


图 17-3　主从处理器方式微型计算机开发系统

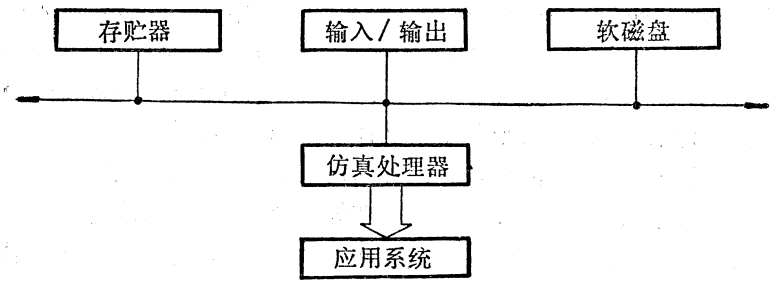


图 17-4　单处理器方式微型计算机开发系统

单处理器方式，即仿真器兼有主处理器职能。此时，虽然通用性差些，但结构要比前者简单经济。

设计人员用 MDS 作为研制工具，可以使一个系统的软、硬件研制工作同时并进，进一步还可以为设计者积累研制各种应用系统的经验。这样，不仅可以缩短整个系统的研制周期，而且可对系统进行全面和详细的测试和调试，有助于提高整个应用系统的质量和可靠性。

近年来，MDS 产品逐渐丰富起来，有的开发系统是针对研制某种微处理器应用系统设计的。例如，1978 年 Intel 公司研制的 Intellec 系列 II 中的三种机型（210、220、230）就是为 MCS-80、MCS-85、MCS-48 微处理器系列设计的。Zilog 公司研制的 ZDS-1/4 型的 MDS 是为开发 Z80、Z80A、Z8、Z8000 微处理器系列设计的。Motorola 公司研制的 EXORciser I、I_a、II 型，EXORterm 200、220 及 M68ADS2A 型的 MDS 是用于开发 M6800、6801、6802 微处理器的。近年来，由国际数据服务公司研制的 MUDS-II 型开发系统是一种多用户的通用开发系统，它不仅能开发 Intel 8080A/8085A、8048、Zilog 的 Z80、Motorola 的 6800 以及 F8、6502 等多种 8 位微处理器，而且能支持 Z8000、Intel 8086 和 MC 68000 等 16 位微处理器的开发。

§17-2 微型计算机的监控程序

I/O 接口设计和应用程序的设计，是微型计算机应用系统设计中最重要的部分。微型计算机应用系统中的应用程序与应用对象的性能、工作特点密切相关，并且随着应用场合的不同有很大的差异。从程序设计的角度看，应用程序一般包括启动程序(初始化程序)、I/O 程序、控制程序、定时程序、中断程序以及自诊断和故障处理程序等。其中最重要的，是 I/O 程序和控制程序。通常，微型计算机的监控程序集中地反映了应用程序和 I/O 程序的各种编程技术和实用技巧。因而，解剖一台简易的单板微型计算机的监控程序，是学习和掌握编程技巧的重要途径，也是学习微型计算机应用技术的关键之一。为此，本节将详细剖析 Z80 STARTER SYSTEM K T 单板微型计算机的监控程序 ZBUG Monitor 的主要程序段。读者可以从中体会 I/O 程序和控制程序设计的基本方法和技巧。

一、ZBUG Monitor 的功能

ZBUG Monitor 用 16 进制数字键，12 个功能键作为数据和程序的输入，用 6 个七段数码显示器作为输出显示。此外，它还能与盒式磁带机接口，以及配有写 2716/2758 EPROM 的功能。

1．初始引导

在上电复位或按 RESET 按钮复位后，ZBUG 可由一个开关控制，或转向监控程序，或转向 PROM1 (用户程序——入口地址为 0800H) 执行。

2．能接受、分析、执行键盘输入的命令。

3．能接收键盘输入的数据和程序，或对已输入的数据和程序进行修改。

4．能显示寄存器、存贮器和外设端口的内容，并能修改寄存器、RAM 和 I/O 端口的内容。

5．能控制执行在 RAM，或 ROM，或 EPROM 中的用户程序。

6．设有诊断功能：如设置多重断点、单步执行等。

7．能用盒式磁带录音机把存贮器中的信息转贮到磁带上，或把磁带上的信息装入 RAM 中。

8．配有一个可编程的 2716/2758 EPROM 芯片的编程器。

二、ZBUG 键盘/显示器的布局和定义

键盘/显示器是微型计算机应用系统中最常用的输入/输出设备。操作人员通过键盘可以向微型计算机应用系统发布各种命令，以完成预期的任务。同时，通过显示器可以随时了解系统运行的状态。为键盘/显示器编制的一套专用的控制程序是 ZBUG Monitor 的主要组成部分。

Z80 STARTER SYSTEM KIT 的键盘/显示器布局图如图 17-5 和图 17-6 所示。

1．复位按钮(RESET)

复位按钮(RESET)强迫 CPU 复位并开始执行起始地址为 0000H 的程序。然后，由一个开关 S_2 控制转向执行监控程序或转向执行 $PROM_1$ 中的用户程序(入口地址为 0800H)。

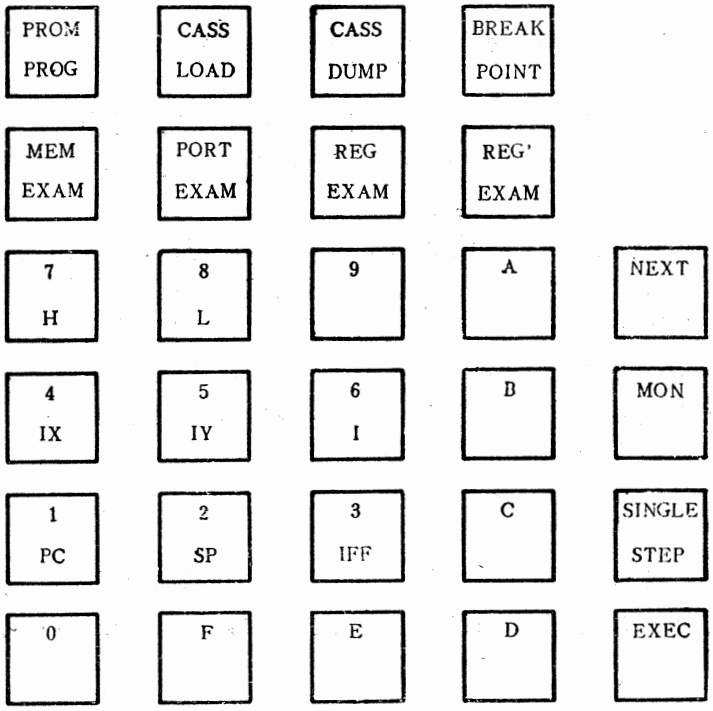


图 17-5 ZBUG 键盘布局图

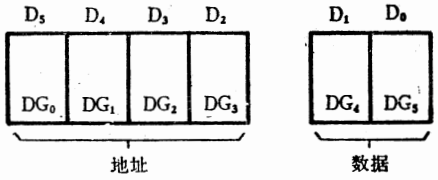


图 17-6 ZBUG 显示器布局图

2．12 个命令键

(1) MEM EXAM (Memory Examine)——存贮单元检查键

操作时先输入要显示的存贮单元的地址 (4 位十六进制数)，立即在左面 4 个显示器显示出地址，然后按 MEM EXAM 键，就在右面两个显示器显示出该地址单元的内容。

若要改变该存贮单元内容，只要再输入两位数 (16 进制) 即可。

若按 NEXT 键，可使存贮单元地址加 1，而显示器将显示新地址及其单元中的内容。

(2) PORT EXAM (Port Examine)——端口检查键

先输入用两位十六进制数表示的端口地址，然后按 PORT EXAM 键，则所指定端口地址的寄存器的内容即显示出来(两位数)。

若要改变它的内容，只要再输入两位数即可。

若按 NEXT 键，可使端口地址加 1，且显示出相应的内容。

(3) REG EXAM (Register Examine)——寄存器检查键

若先按下任一寄存器号 A、B、C、D、E、H (即数字键 7)、L (即数字键 8) 再按下此键，则可显示出相应的寄存器内容(右面两个显示器显示两位 16 进制数)。

若先按下 IX、IY、PC、SP 键，再按下此键，则可显示相应的寄存器内容（4 位十六进制数）。

若要改变寄存器的内容，则只要再输入 2 位（或 4 位）数即可。

(4) REG′ EXAM(Alternate Register Examine)——辅助寄存器检查键

可用与 (3) 中同样方法，显示和修改 CPU 中一组辅助寄存器的内容，只是功能键改为 REG′ EXAM 键。

(5) BREAK POINT——设置断点键

为便于调试程序，通常需要设置断点，即使程序运行到指定的地址停止执行，以便检查内存和各个寄存器的内容。设置断点的方法如下：

先输入断点地址（4 位十六进制数），然后再按此键，即可设置一个断点。ZBUG 在所设断点处用一条 RST 08H 指令（在程序运行时）代替用户的操作码（将它保留在堆栈），当运行到断点时，保留现场，然后返回 ZBUG，即可对断点处的各种内容进行检查。

若所设置的断点处为一条多字节指令，则断点必须设在指令的第一字节。

STARTER KIT 最多可允许设置 5 个断点。

在断点处，将所需检查的内容检查完后，可按 EXEC 键继续运行，遇到新的断点再停止。

设置的断点可用以下三种方法之一予以撤消：

① 在没有输入完 4 个十六进制数的断点地址前，按 BP 键；

② 按单步(SINGLE STEP)键；

③ 按 RESET 按钮。

(6) SINGLE STEP——单步执行程序键

此键用来每次执行程序中的一条指令，执行完后，显示 PC（即下一条指令的地址）和 A 的内容。此时，用户可使用 MON 键来返回 ZBUG，再用它的按键来检查或修改存贮单元、I/O 端口或 CPU 寄存器的内容。

连续按单步键，程序就可以一条指令一条指令地执行下去。

(7) EXEC(Execute)——连续执行程序键

此键用来连续执行存在 RAM、ROM 或 EPROM 中的程序。它有两种使用方法：

① 先输入要执行的程序起始地址（4 位数），然后按此键，即从此地址开始执行程序；

② 如果在连续执行程序时遇到断点，或是使用单步方式后暂停执行程序，此后要求连续执行程序，则只需按 EXEC 键即可。

(8) MON (Monitor)——监控键

按下此键，即可产生一个不可屏蔽中断请求，保存寄存器状态并返回监控程序。它有以下两个功能：

① 用于中止或退出当前的命令状态或输入数据，使控制返回监控程序，等待输入新的命令或数字。

② 用于中止现行程序的执行，保护 CPU 各寄存器的状态后返回监控程序，这一功能在排除程序故障中非常有用。

(9) NEXT 键

NEXT 键用于下述三种操作方式：检查存贮器，检查通道，检查 EPROM 编程中的下一个单元的错误。在存贮器检查和通道检查方式中，NEXT 键用于选择顺延的下一个存贮单元或

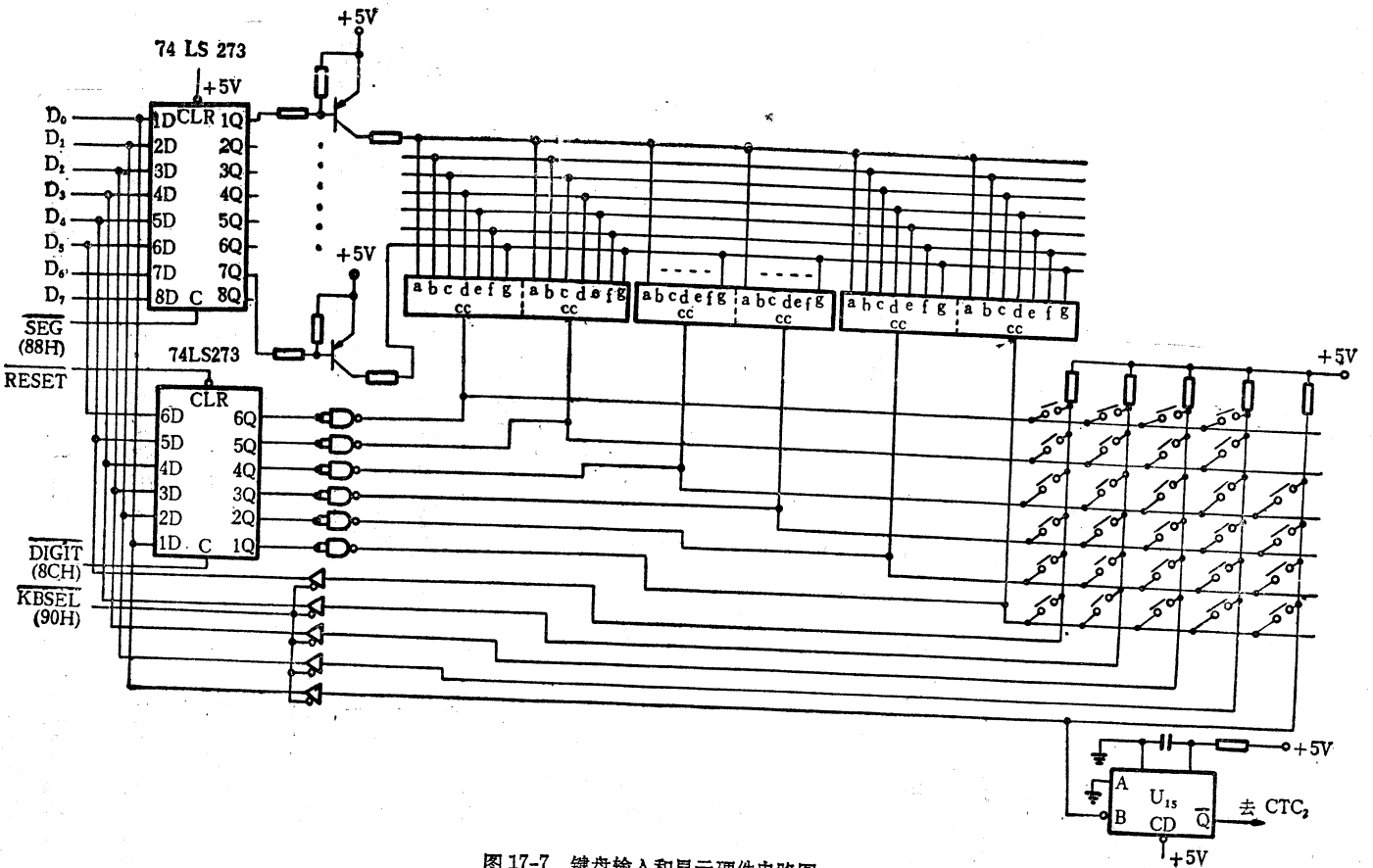


图 17-7　键盘输入和显示硬件电路图

通道单元，并自动显示该单元的内容。

(10) PROM PROG (EPROM Programmer)——EPROM 写入键

(11) CASS LOAD(Cassette Load)——磁带信息装入键

(12) CASS DUMP(Cassette Dump)——信息转贮磁带键。

ZBUG 显示器由 6 个七段 LED 显示器组成。其中，左四位为地址段，右二位为数据段。

Z80 STARTER KIT 的键盘和显示器电路如图 17-7 所示。

三、ZBUG 监控程序的组成和分析

ZBUG 监控程序固化在可擦除的可编程序只读存贮器 (EPROM) 中，存贮空间地址为 0000～07 FFH，此外 ZBUG 还使用了 112 个 RAM 单元。

ZBUG 监控程序总框图如图 17-8 所示。

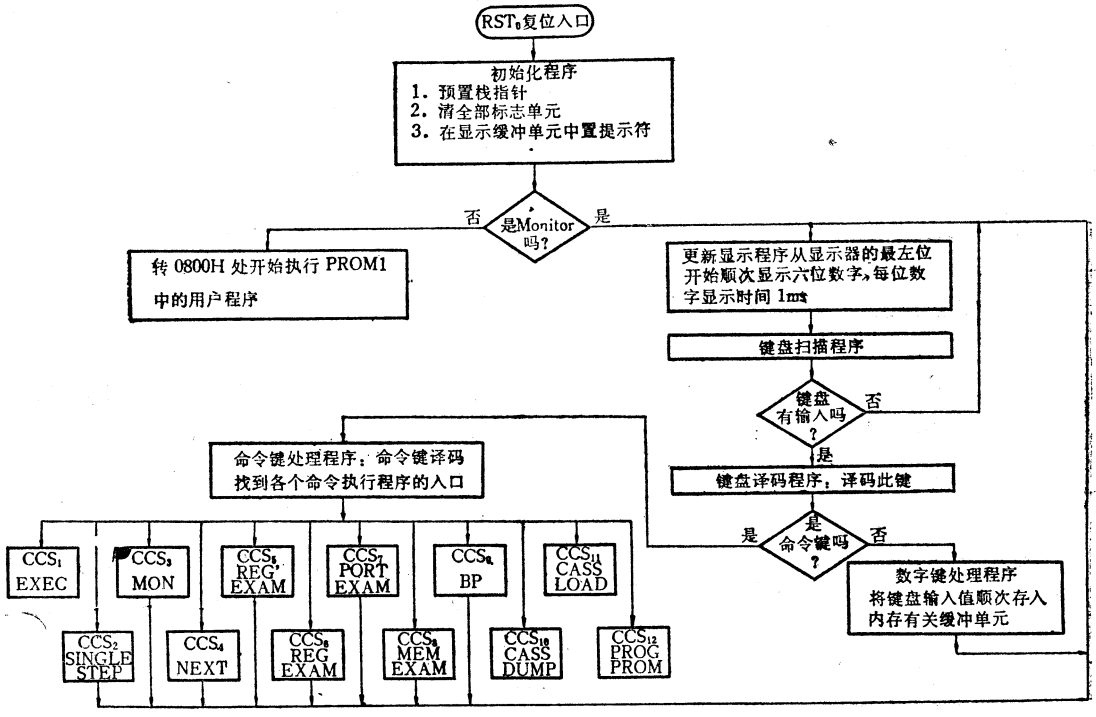


图 17-8 ZBUG 监控程序总框图

它由下列四个程序组成：

(1) 初始化、显示及键盘扫描译码程序；

(2) 键盘命令程序；

(3) 实用子程序；

(4) 表格。

在上述程序中，键盘扫描译码程序是整个监控程序的核心。它用来识别按键位置(即由键盘矩阵的行与列所确定的唯一的位置)，接着通过查表法把按键位置转换为相应的按键识别码，然后进入检测键释放程序，根据键值大小进行判别之后，分别转入数字键或命令键处理程序进行处理。

下面分别予以讨论：

1．初始化程序(启动程序)(0000～00F3H)

所谓初始化程序，系指微型计算机应用系统加电以后，系统程序执行以前，需要进行的一系列准备操作的程序。初始化程序流程图如图17-9所示。

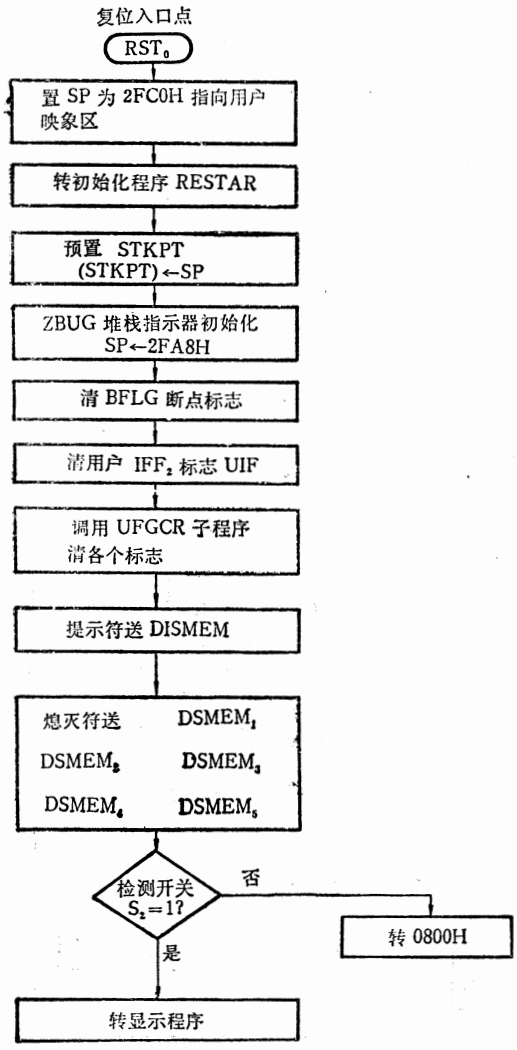


图 17-9　初始化程序流程图

(1) 设定堆栈指针

它设定状态堆栈指针 SP = 2FC0H，并将其保存在 2FF2～2FF3H 单元，然后设定 ZBUG 工作堆栈指针 SP = 2FA8H。其中，工作堆栈(2F90H～2FA7H)用来存放 ZBUG 中子程序调用的返回地址或中断后的返回地址；状态堆栈(2FA8H～2FBFH)又称寄存器映象单元(MEMORY MAP)，用来保留 CPU 中各寄存器的状态。如果改变映象单元的内容，就相当于改变相应寄存器的内容。

(2) 清除各标志单元

ZBUG 监控程序设置有缓冲区(2FC0H～2FFFH)，包含显示缓冲区单元、各种标志单元以及断点表等，初始化程序将所有标志单元全部清除。

(3) 在最左端显示器所对应的显示单元 DISMEM 中填入“－”的代码 11H，其余填入空格的代码 10H。即在显示器最左端显示待命提示符“－”，表示机器等待接收命令。

(4) 上电后，按复位按钮(RESET)，程序由开关 S_2 控制或转移到执行监控程序，或转移到执行 PROM1 的用户程序(入口地址 0800H)。

(5) 重新启动指令处理

RST 0 用于复位，RST 08 用于断点，其它所有的 RST 指令如 RST 16、RST 24、RST 32、RST 40、RST 48 和 RST 56 都在 RAM 中设置了转移指令，可转移到存在 RAM 中的中断服务程序。

初始化程序

```
        ORG     0000H
        LD      SP,     2FC0H    ; 设置状态堆栈指针
        JP      RESTAR           ; 转至重新启动程序
重新启动程序
        ORG     009FH
RESTAR: LD      (STKPT), SP      ; 保护状态堆栈指针
        LD      SP,     2FA8H    ; 设置监控程序的工作堆栈指针
```

```
        LD      A,00H
        LD      (BFLG),   A          ; 清 BFLG (断点标志)
        LD      (UIF),    A          ; 清中断请求(IFF2 标志)
RESTR1: CALL    UFGCR                ; 调用清标志子程序
        LD      A,        11H        ; 置提示符"-"
        LD      (DISMEM), A          ; 送至显示缓冲区
        LD      A,        10H        ; 送熄灭符
        LD      (DSMEM1), A
        LD      (DSMEM2), A
        JR      RESTR2-$
        JR      RESTR3-$             ; 转至相对转移偏移量计算程序的入口
RESTR2: LD      (DSMEM3), A
        LD      (DSMEM4), A
        LD      (DSMEM5), A
        IN      A,        (KBSEL)    ; 输入开关 S2 的状态
        BIT     5,        A          ; 位测试,检查开关 S2 的位置
        JP      NZ,       DISUP      ; 若开关 S2 倒向 MON RST,则转至显示程序
        JP      0800H                ; 否则,转至 PROM1 的起始地址
KBSEL:  EQU     90H                  ; MONITOR/PROM 检测端口
```

2. Z80 STARTER SYSTEM KIT 显示器接口电路和显示程序

(1) 显示器接口电路

该单板微型计算机有 6 个七段 LED 显示器，可以显示十六进制数、部分英文字母及一些特定的字符，如在 ZBUG 监控程序中可显示"-""/"和"空白"等。

Z80 STARTER 中所用的七段 LED 数码显示器，每一段为一发光二极管，其结构及原理如图 17-10 所示。

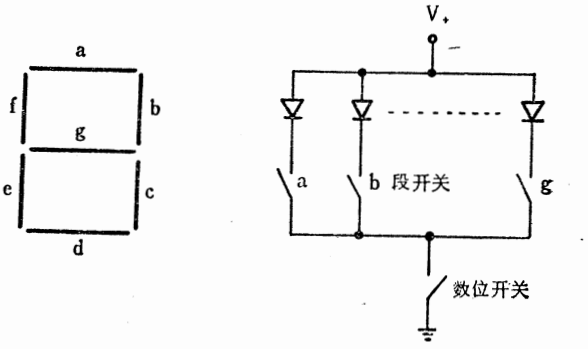


图 17-10 七段 LED 显示器的结构及原理

如图所示，七段 LED 显示器共有 7 个段控制信号，当其为"0"电平，并经反相驱动器变为"1"电平时，相应的段开关闭合；但该段能否发光，还取决于该显示器的数位(位选)开关是否闭合。数位开关分别由相应的数位控制信号来控制，当其为"1"电平，并经反相驱动器变为"0"电平时，该显示器的数位开关闭合。此时，相应的段才会点亮，显示出相应的数字或字母。

显示器接口逻辑电路如图 17-11 所示并参阅图 17-7。

所要显示的数据(或字符)由 CPU 通过数据总线 D_6～D_0 写入字模锁存器 $\overline{SEG}$ (端口地址为 88H)，经三极管 Q_7～Q_1 反向驱动后，送至所有数码管的相应段。至于要显示哪一位则由位选锁存器 $\overline{DIGIT}$ (端口地址为 8CH) 来控制。反向驱动器(一般用晶体管)用来给 LED 提供足够的驱动能力。

七段 LED 显示器所显示的字符以及段号与数据位的对应关系如图 17-10 及表 17-2。

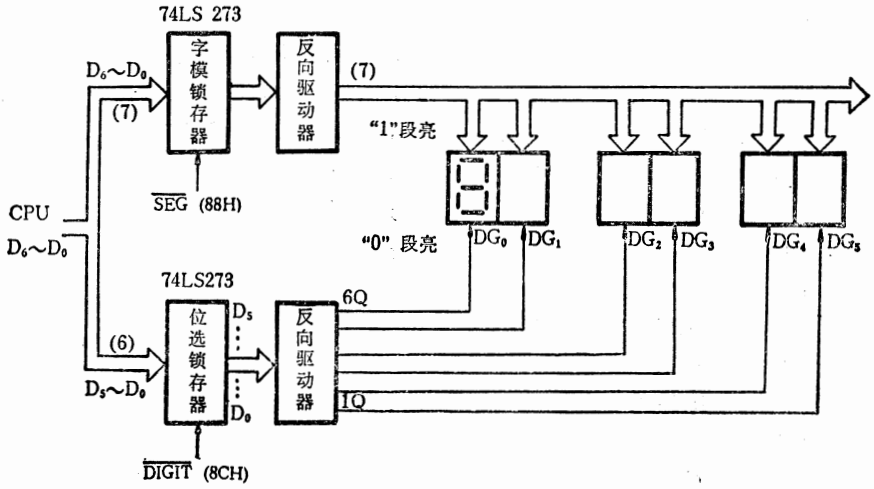


图 17-11　显示器接口逻辑电路图

表 17-2　七段显示的代码与字模的编码表

数字(符号)	16 进 制 (H)	$D_6D_5D_4D_3D_2D_1D_0$ g f e d c b a	字模
0	40	1 0 0 0 0 0 0	0
1	79	1 1 1 1 0 0 1	1
2	24	0 1 0 0 1 0 0	2
3	30	0 1 1 0 0 0 0	3
4	19	0 0 1 1 0 0 1	4
5	12	0 0 1 0 0 1 0	5
6	02	0 0 0 0 0 1 0	6
7	78	1 1 1 1 0 0 0	7
8	00	0 0 0 0 0 0 0	8
9	18	0 0 1 1 0 0 0	9
A	08	0 0 0 1 0 0 0	A
B	03	0 0 0 0 0 1 1	B
C	46	1 0 0 0 1 1 0	C
D	21	0 1 0 0 0 0 1	D
E	06	0 0 0 0 1 1 0	E
F	0E	0 0 0 1 1 1 0	F
空白	7F	1 1 1 1 1 1 1	
提示符	3F	0 1 1 1 1 1 1	-
右撇号	7D	1 1 1 1 1 0 1	'

若把 LED 显示器最左边的一位命名为 DG_0 位，其余的依次为 DG_1、DG_2、DG_3、DG_4、DG_5（见图 17-6）。例如，要在最左边的 LED 显示器 DG_0 位上显示"8"，可以通过下列程序段来完成：

```
LD    A,    00H   ;  } 段选
OUT   (88H), A     ;  }
LD    A,    20H   ;  } 位选
OUT   (8CH), A     ;  }
```

其中 00H 的二进制表示为

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	0	0	0	0	0	0
	g	f	e	d	c	b	a

D_7 不起作用，对 D_6～D_0 来说“0”为有效，故 a、b、c、d、e、f、g 段全亮，显示“8”。

数 20H 的二进制数表示：

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	1	0	0	0	0	0
		↑ DG_0					↑ DG_5

D_7、D_6 不起作用，对 D_5～D_0 来说，“1”为有效，故显示最左边的 DG_0 位。

实际的显示过程要稍许复杂些。下面介绍其实际的更新显示程序。

(2) 显示程序 DISUP(00F4～0122H)

显示程序 DISUP 的功能是依次将欲显示的显示缓冲区 (DISMEM～DSMEM5) 的数码取出，并转换成字模码自左向右轮流送入七段 LED 显示器显示，每位保持显示约 1 毫秒。

显示程序所使用的 RAM 单元及显示器与显示缓冲单元的对应关系如图 17-12 所示。

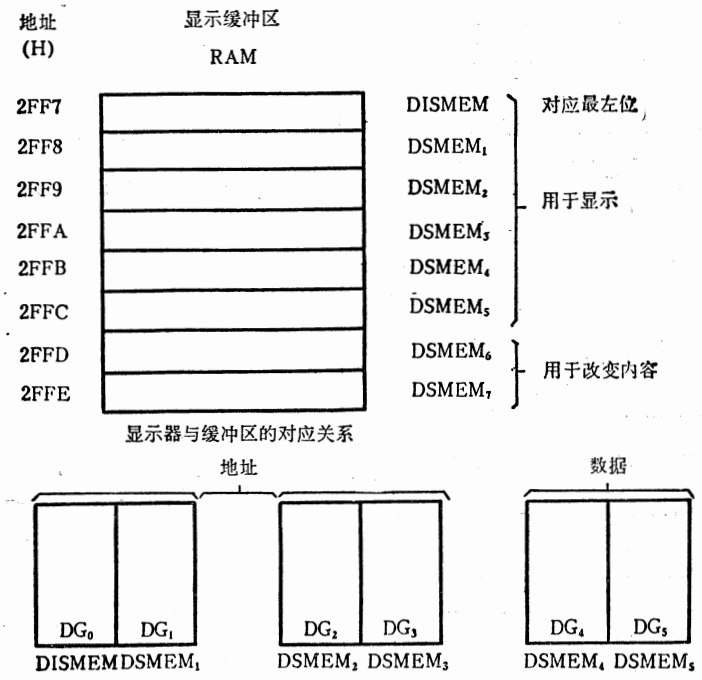


图 17-12　显示器与显示缓冲区的对应关系图

要显示的数码应先送入以 DISMEM 单元开始的显示缓冲区中。显示程序必须将这个数码转换为显示器段形的编码(字模码)，这种转换是由软件通过查阅七段字模表来实现的。七段字模码是根据如图 17-10 和图 17-11 的硬件电路来编排的。我们将字模码按十六进制数的次序编成字模表(见表 17-3)，并将字模表的起始地址送 IX，以要显示的数作为偏移量，然后以字模表首址 IX 加偏移量为指针，通过查字模表后就可取出相应的数的字模码。

显示程序流程图及延时程序流程图如图 17-13 和图 17-14 所示。

表 17-3　显示程序使用的字模表

地 址 (H)	ROM 内容(H)	显　　示	偏移量(H)	地 址 (H)	ROM 内容(H)	显　　示	偏移量(H)
07A6	40	0	0	07B0	08	A	A
07A7	79	1	1	07B1	03	b	B
07A8	24	2	2	07B2	46	c	C
07A9	30	3	3	07B3	21	d	D
07AA	19	4	4	07B4	06	E	E
07AB	12	5	5	07B5	0E	F	F
07AC	02	6	6	07B6	7F	熄 灭 码	10
07AD	78	7	7	07B7	3F	提 示 符	11
07AE	00	8	8	07B8	7D	右撇号标记	12
07AF	18	9	9				

显示程序入口：将要显示的 6 位数放入显示缓冲区的 6 个单元内，起始单元地址为 DISMEM(2FF7H)。

出口：更新 6 位七段数码显示数据。

使用的寄存器：

IX——字模表指针

HL——显示缓冲区当前单元的指针

B——现行数位指示器

E——保持被显示的位的数据

A、D——暂存寄存器

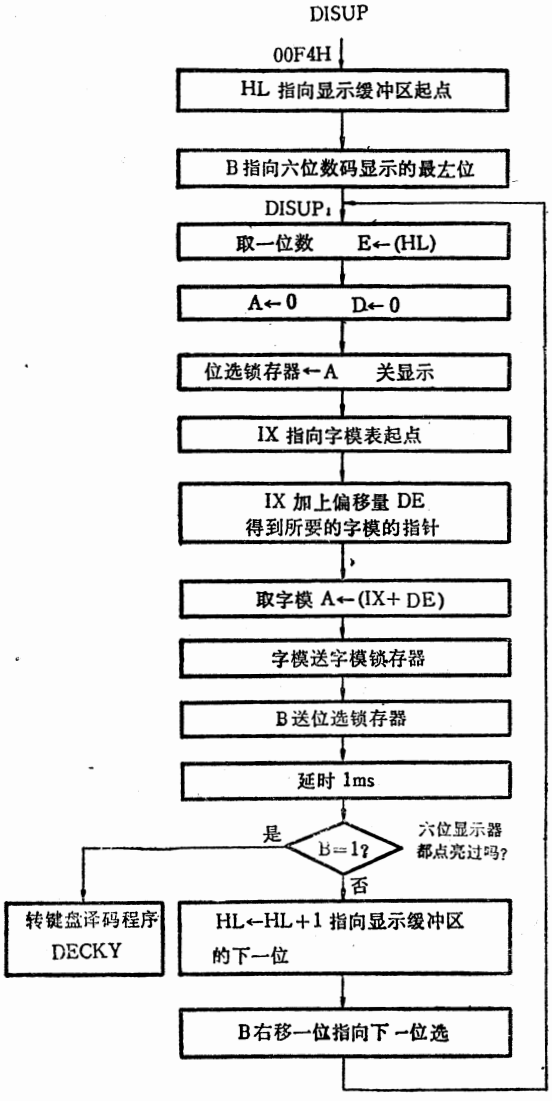


图 17-13　显示程序流程图

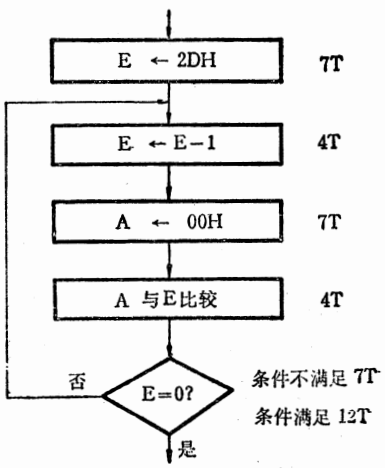


图 17-14　延时程序流程图

显示程序

```
        ORG     00F4H
DISUP:  LD      HL,     DISMEM      ; 指向显示缓冲区的首地址
        LD      B,      20H         ; 指向最左边的显示位
DISUP1: LD      E,      (HL)        ; 取第一个字节
        LD      D,      00H         ; D 寄存器清零
        LD      A,      00H         ; A 累加器清零
        OUT     (DIGLH), A          ; 关显示
        LD      IX,     SEGPT       ; IX 指向字模表的首地址
        ADD     IX,     DE          ; 指向要显示的数的字模
        LD      A,      (IX+0)      ; 取出字模码→A
        OUT     (SEGLH), A          ; 输出至字模锁存器
        LD      A,      B           ; 取出位选码→A
        OUT     (DIGLH), A          ; 输出到数位锁存器
        LD      E,      2DH         ; 设置 1 毫秒延时
DISUP2: DEC     E
        LD      A,      00H         ; 延时循环
        CP      E
        JR      NZ,     DISUP2-$
        LD      A,      01H
        CP      B                   ; B=1?(即是否已移至最右边的显示位)
        JR      Z,      DISUP3-$    ; 是,转出口
        INC     HL                  ; 否,指向下一个要显示的数
        SRL     B                   ; 移至下一个显示位
        JR      DISUP1-$            ; 循环
DISUP3: JP      DECKY               ; 转至键盘分析程序
DIGLH:  EQU     8CH                 ; 只写数位选择
SEGLH:  EQU     88H                 ; 只写字模选择
        ORG     07A6H
SEGPT:  DB      40H                 ; 0
        DB      79H                 ; 1
        DB      24H                 ; 2
        DB      30H                 ; 3
        DB      19H                 ; 4
        DB      12H                 ; 5
        DB      02H                 ; 6
        DB      78H                 ; 7
        DB      00H                 ; 8
        DB      18H                 ; 9
        DB      08H                 ; A
        DB      03H                 ; B
        DB      46H                 ; C
        DB      21H                 ; D
```

```
DB      06H             ; E
DB      0EH             ; F
DB      7FH             ; 空白
DB      3FH             ; 提示符
DB      7DH             ; 右撇号标记
```

3. Z80 STARTER SYSTEM KIT 键盘接口电路

Z80 STATER SYSTEM KIT 键盘是一种非编码键盘，其硬件框图如图 17-15 和图 17-7 所示。

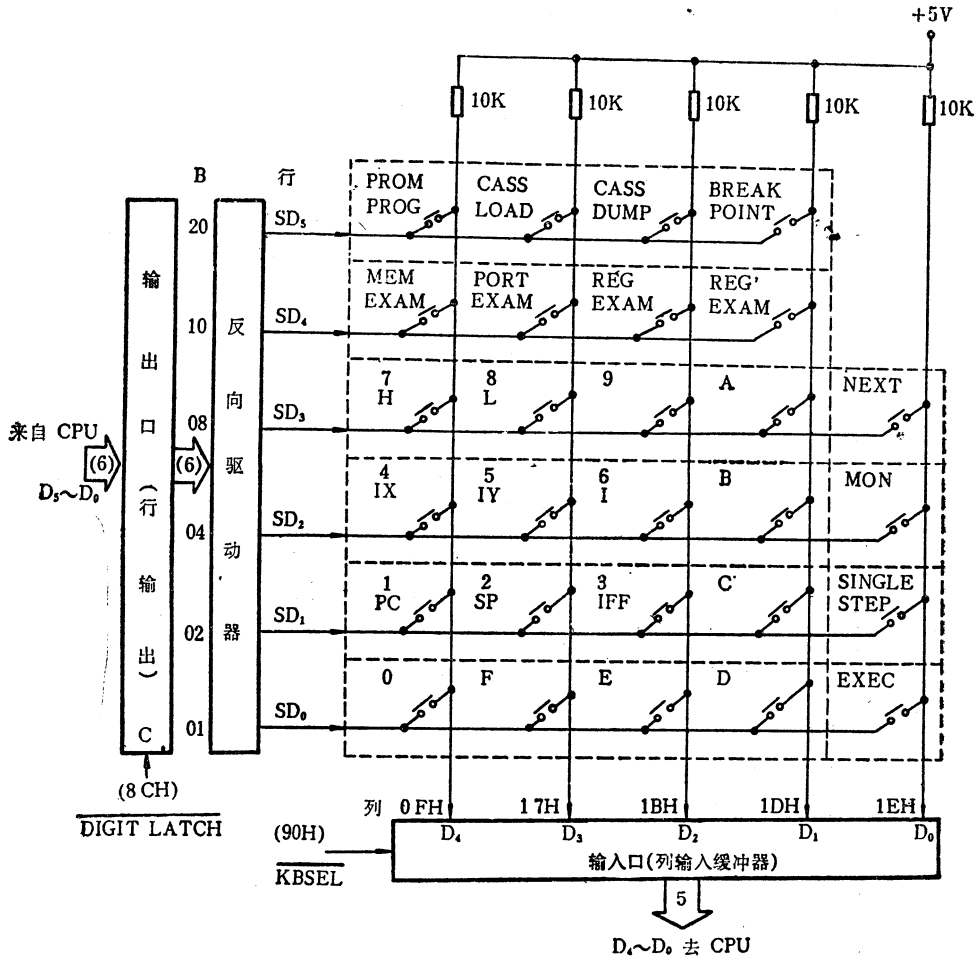


图 17-15 Z80 STARTER SYSTEM KIT 键盘接口图

这种键盘仅简单地提供 6 行×5 列的键盘矩阵，各键的定义已如上述，其全部工作都靠软件去完成。即用软件去解决键的识别、产生相应的键识别码，去抖动及防止串键等技术问题。它借用位选锁存器 (74LS273) 作行扫描输出口，以节省器件，并用一个 8 位缓冲器 (74LS244) (仅用其中 5 位)作为列输入口。

如图 17-7 中所示，数据总线 D_5～D_0 经反相后输出作为行线(键盘的输入信号线)，它由端口 $\overline{\text{DIGIT}}$ (地址为 8CH)控制；5 条列线由 +5V 电源经 10 kΩ 电阻引出 (键盘的输出信号线)，通过三态缓冲器引至 D_4～D_0，由端口 $\overline{\text{KBSEL}}$ (地址为 90H)控制。

在每一行与每一列之间接有一个键。

在工作时，程序按行扫描键盘，检查列的输出，由行信号与列信号的组合来确定哪一个键闭合。若此行中有键闭合，则相应的列输出为“0”电平，而其余列都为“1”电平；若此行中没有键闭合(即所有的列全为“1”电平)，则再逐行扫描相邻高位的一行，直至找到有闭合的键的一行为止。这样逐行扫描，由行信号与列信号的组合，便可确定是哪一个键闭合。

4．Z80 STARTER SYSTEM KIT 键盘扫描译码程序(0123H～01CAH)

该程序首先检测是否有任一键按下，若无任一键按下则转至显示程序。当检测到有键闭合时，就调用延时 20 ms 子程序，以消除键抖动。然后按行扫描键盘，并检测列值以确定此行中是否有键闭合，若有键闭合，则转至键盘译码程序以确定此键值；若无键闭合，则逐行自下而上(见图 17-15) 继续扫描。

键盘扫描译码程序

入口：在显示程序之后执行

出口：如按命令键，在执行命令之后转出。如按十六进制数字键，在数据送入显示缓冲区

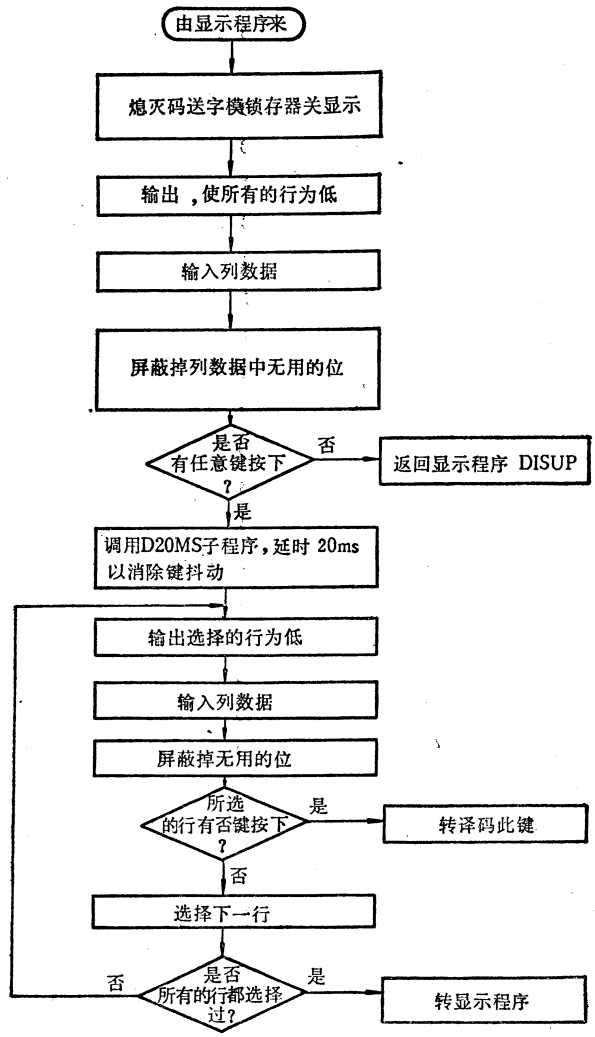


图 17-16　键盘扫描程序流程图

之后转出。

所用的寄存器：

A——列数据(暂存键值)

B——扫描的行数据，键译码后为键值

C——暂存由行和列决定的唯一字

HL——(1)键值查阅表(KYTBL)指针

(2) 显示缓冲区指针(KEYPTR)

BC——键译码时计算出来的偏移量

EXX——用于 EPROM 编程在校验期间保存指针

(1) 键盘扫描程序

键盘扫描程序 DECKY 流程图如图 17-16 所示。

键盘扫描程序

```
          ORG   0123H
DECKY:    LD    A,        7FH
          OUT   (SEGLH),  A          ; 关显示
          LD    A,        3FH
          OUT   (DIGLH),  A          ; 输出使所有行为低
          IN    A,        (KBSEL)    ; 输入列数据
          AND   1FH                  ; 屏蔽掉无用位(D7 D6 D5)
          CP    1FH                  ; 是否有键按下?
          JP    Z,        DISUP      ; 否,返回显示程序 DISUP
          CALL  D20MS                ; 是,调用 20ms 延时以消除键抖动
          LD    C,        DIGLH      ; 端口地址→C
          LD    B,        01H        ; 设置第一行为低值
KEYDN1:   OUT   (C),      B          ; 使所选择的行为低
          IN    A,        (KBSEL)    ; 输入列数据
          AND   1FH                  ; 屏蔽无用的位(D7 D6 D5)
          CP    1FH                  ; 是否有键按下?
          JR    NZ,       KEYDN2-$   ; 是,扫描结束,转至译码
          SLA   B                    ; 否,选择下一行
          LD    A,        40H
          CP    B                    ; 所有行都选择过吗?
          JR    NZ,       KEYDN1-$   ; 否,返回循环,继续扫描
          JP    DISUP                ; 是,返回显示程序。
```

(2) 键盘译码程序

当键盘扫描程序检测到闭合键时，经过 20 ms 的延时去抖动后，就一次扫描键矩阵的一行，同时检测所有的列，直至确定由行和列所决定的唯一的按键位置后，程序就进入键译码程序。此时，行存于B寄存器，列(已被 1 FH 屏蔽过)存于A累加器。

下面先对照图 17-15 所示键盘接口电路图来分析键值。如表 17-4 所示。

按键被识别后，如何把由行和列所确定的按键位置转换为相应的代码呢？这是通过“查表”方法来实现的。这需要预先将 28 个键所对应的键识别码以表格形式存入 ROM 中，然后

表 17-4 用于键译码的键值转换表

标号	地址 (H)	ROM的内容 (H)	键值	键的意义	B(存行)	A(存列)
KYTBL:	70B9	0FF	0	0	01	0F
	07BA	0EF	1	1	02	0F
	07BB	0F7	2	2	02	17
	07BC	0FB	3	3	02	1B
	07BD	0DF	4	4	04	0F
	07BE	0E7	5	5	04	17
	07BF	0EB	6	6	04	1B
	07C0	0CF	7	7	08	0F
KYTBL:	07C1	0D7	8	8	08	17
	07C2	0DB	9	9	08	1B
	07C3	0DD	A	A	08	1D
	07C4	0ED	B	B	04	1D
	07C5	0FD	C	C	02	1D
	07C6	0D	D	D	01	1D
	07C7	0B	E	E	01	1B
	07C8	07	F	F	01	17
	07C9	0E	10	EXEC(执行)	01	1E
	07CA	0FE	11	SS(单步)	02	1E
	07CB	0EE	12	MON	04	1E
	07CC	0DE	13	NEXT	08	1E
	07CD	0CD	14	REG' EXAM(替换)	10	1D
	07CE	0CB	15	REG EXAM(主寄)	10	1B
	07CF	0C7	16	PORT EXAM(通道)	10	17
	07D0	0BF	17	MEM EXAM(存贮器)	10	0F
	07D1	0BD	18	BP(断点)	20	1D
	07D2	0BB	19	DUMP(转贮)	20	1B
	07D3	0B7	1A	LOAD(装入)	20	17
	07D4	0AF	1B	PROG(编程)	20	0F

把行扫描中所得到的键识别码与之一一比较，直至两者相符为止。

表 17-4 键值表中 ROM 的键识别码是如何确定的呢？原来它是采用由列号和经过左移 4 次的行号补码所组合起来的键识别码，用此法可保证每个按键只有唯一的键识别码。我们可用表 17-5 加以说明：

表 17-5

行号 (行值) B		C (行号补码)	左移4次后行号补码	键识别码				
6	20	0FA	0A0	0AF	0B7	0BB	0ED	0BE
5	10	0FB	0B0	0BF	0F7	0CB	0CD	0CE
4	08	0FC	0C0	0CF	0D7	0DB	0DD	0DE
3	04	0FD	0D0	0DF	0E7	0EB	0ED	0EE
2	02	0FE	0E0	0EF	0F7	0FB	0FD	0FE
1	01	0FF	0F0	0FF	07	0B	0D	0E
				0F	17	01B	01D	01E (列值)
				D_4	D_3	D_2	D_1	D_0

例如，设“0”键位置为行号 1、列号 0FH $\left(000\boxed{0}1111\,B\right)$；将行号 1 的补码左移 4 次后为 0F0H，再与列号相加得 0FFH，即为键值查阅表 ROM 中“0”键的键识别码。其它键的识别码，也是根据同样道理获得的。

键译码程序流程图如图 17-17 所示。

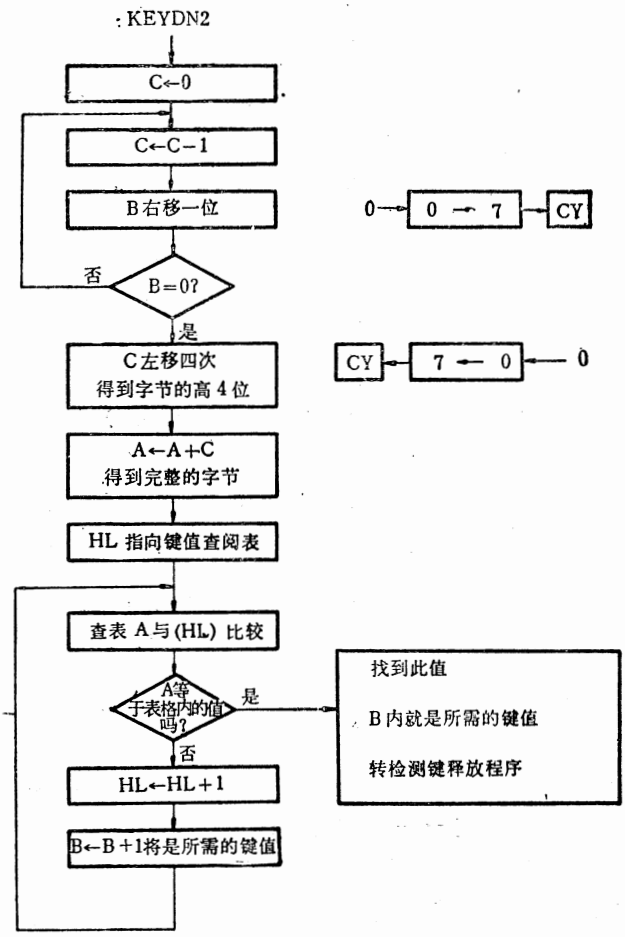


图 17-17　键译码程序流程图

(3) 检测键释放程序

当按下的键经过键译码程序而找到相应的键值(即存于 B 内的内容)后，就转入检测键释放程序，其作用有三点：

① 判别此键值是否小于 10H，若小于 10H，则该按键即为数字键；反之则为命令键。随后就转入相应的处理程序；

② 延时 20 ms 消除放开按键时的抖动；

③ 等待键释放，只有该键释放，才显示数据或执行命令。

检测键释放程序流程图如图 17-18 所示。

(4) 命令键分析程序

当程序判别该键是命令键(即键值≥10H)时，就进入命令键分析程序。其作用是：在 A 累

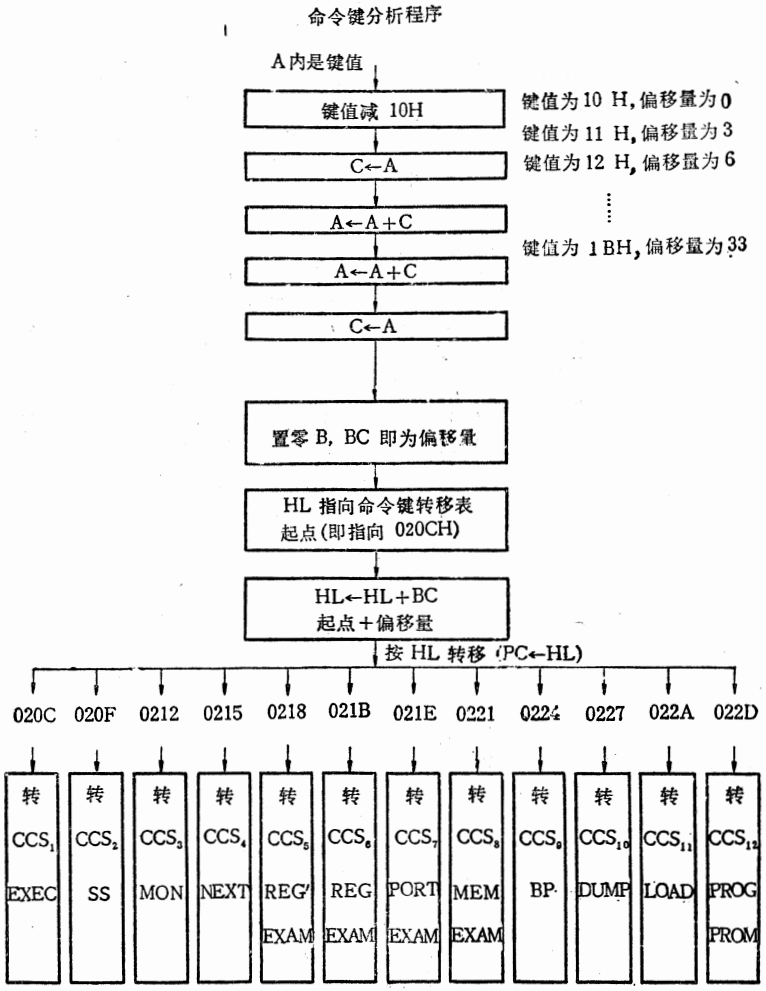


图 17-18　检测键释放程序流程图

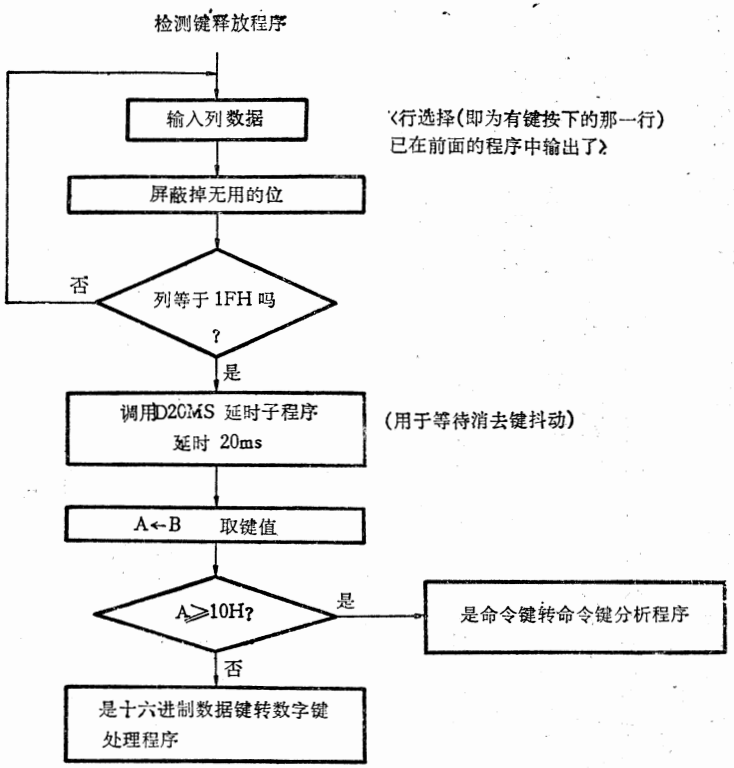


图 17-19　命令键分析程序流程图

加器中处理键值，找到相应命令键处理程序的入口地址。其程序流程图如图 17-19 所示。命令键转移表见表 17-6。

键盘命令处理程序的入口地址表的起始地址为 JPTAB，其中每一个入口都存放一条三字节的转移指令，故每一个入口之间差 3 个存贮单元。因而，可以用某命令按键之值减去 10 H，再将所得的差值倍乘以 3，以形成命令表中的偏移量。然后，将 JPTAB 表的入口地址送 HL，三倍后的偏移量送 BC，则 HL←HL＋BC，即为该命令键处理程序的入口地址，执行 JP(HL) 指令就能实现转至相应的命令键处理程序。

表 17 6　命令键转移表 JPTAB

地址 (H)	命令键	
020C	转 CCS1	; EXEC(执行)
020F	转 CCS2	; SS(单步)
0212	转 CCS3	; MON(监控)
0215	转 CCS4	; NEXT(下一个)
0218	转 CCS5	; REG′ EXAM(辅助寄存器检查)
021B	转 CCS6	; REG EXAM(主寄存器检查)
021E	转 CCS7	; PORT EXAM(通道检查)
0221	转 CCS8	; MEM EXAM(存贮器检查)
0224	转 CCS9	; BP (设置断点)
0227	转 CCS10	; DUMP (转贮磁带)
022A	转 CCS11	; LOAD (磁带装入)
022D	转 CCS12	; PROG PROM (EPROM 编程)

(5) 数字键处理程序(0179～01BAH)

当键盘输入的是数字键时，程序立即转入数字键处理程序进行处理。数字键处理程序的流程图如图 17-20 所示。

其工作过程是：当按下数字键后，相应的键值被监控程序接收，送至显示缓冲区指针 KEYPTR 所指向的显示缓冲单元 DISMEM～DSMEM5，它们分别对应数码显示器自左至右的六位。由于按 RESET 或 MON 键后显示缓冲区指针已初始化，每按一次数字键，KEYPTR 逐次加 1，相应的键值依次存到显示缓冲单元 DISMEM～DSMEM7 中。当数字键的输入个数大于 8 时，程序重新启动，显示器不再显示数字而显示提示符和空格。如继续按数字键，则重复上述过程。如果数字键的操作是属于某个命令键的修改操作时，则显示指针 KEYPTR，将由相应的命令键处理程序控制指向 DSMEM6(双字节修改时指向 DSMEM 4)。从而数字键输入值被直接送往显示缓冲单元 DSMEM6～7 (双字节时送往 DSMEM4～7) 作为修改值。然后处理程序按相应命令键的规定将此修改值送相应的映象单元(或内存单元)实现修改。与此同时，该修改值也被送往显示单元 DSMEM4～5 (双字节时送往 DSMEM2～5) 进行数字显示。

如上所述的键盘译码程序、检测键释放程序、命令键分析程序和数字键处理程序可以通过下列的程序(统称为键译码程序)来实现：

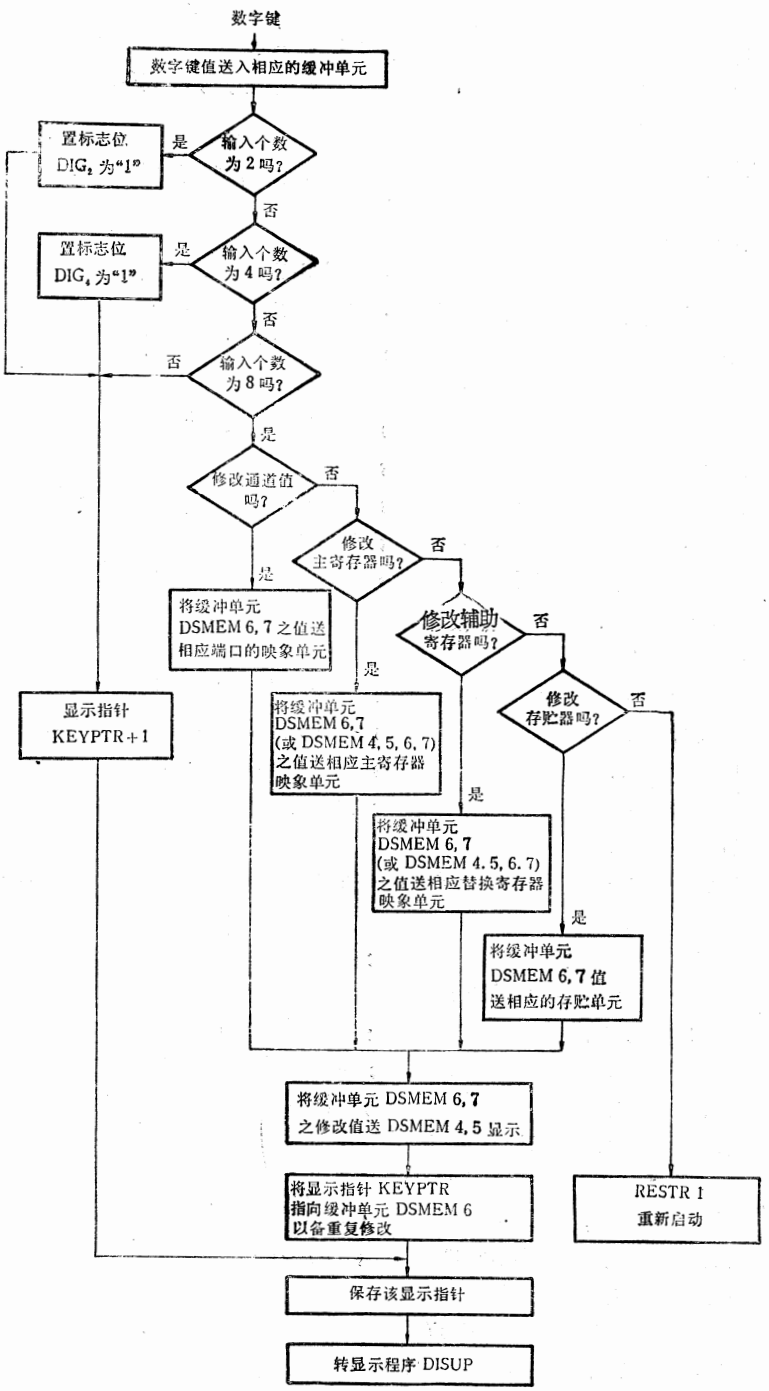


图 17-20　数字键处理程序的流程图

```
；键译码程序
                                        ；B中为行值,A中为列值(已用1FH屏蔽过)
KEYDN2: LD      C,      00H
KEYDN3: DEC     C                       ；寄存器C内容减1
        SRL     B                       ；B右移一位
        JR      NZ,     KEYDN3-$        ；若B未全移为0,则使C减1,直至B=0
        SLA     C                       ；
        SLA     C                       ；  左移四次,把C的低4位移至高4位
        SLA     C                       ；
        SLA     C                       ；
        ADD     A,      C               ；与A中的内容相结合,形成键值识别码
        LD      HL,     KYTBL           ；设置键值转换表指针
KEYDN4: CP      (HL)                    ；查表找到等于A的项
        JR      Z,      KEYDN5-$        ；是,找到此项
        INC     HL                      ；否,修改地址指针
        INC     B                       ；修改键值
        JR      KEYDN4-$                ；继续查找,当在表格中找到相应的编码时,在寄存器B
                                        ；中得到键值
KEYDN5: IN      A,      (KBSEL)         ；输入列值检测键是否释放
        AND     1FH                     ；屏蔽其它输入位
        CP      1FH
        JR      NZ,     KEYDN5-$        ；若未释放,则循环等待
        CALL    D20MS                   ；已释放,调用延时子程序以消除键抖动
        LD      A,B                     ；键值→A,此时B中存有键值
        CP      10H                     ；是命令键吗?
        JR      NC,     KEYDN6-$        ；是,转至命令键分析程序
        LD      HL,     (KEYPTR)        ；是16进制数字键,则存入显示缓冲区
        LD      (HL),   B
        OR      A                       ；清CY
        LD      BC,     DSMEM1
        SBC     HL,     BC              ；输入的是第2位数吗?
        JR      Z,      KEYDNA-$        ；是, DIG2标志置"1"
        OR      A                       ；清CY
        LD      BC,     DSMEM3
        LD      HL,     (KEYPTR)
        SBC     HL,     BC              ；输入的是第4位数吗?
        JR      Z,      KEYDN8-$        ；是, DIG4标志置"1"
        OR      A
        LD      BC,     DSMEM7
        LD      HL,     (KEYPTR)
        SBC     HL,     BC              ；显示缓冲区已写入8位数吗?
        JR      Z,      KEYDN9-$        ；是,转至KEYDN9,置输入8位数标志
KEYDN7: LD      HL,     (KEYPTR)        ；取键指针
```

```
        INC     HL                      ; 指针加1
        LD      (KEYPTR),HL             ; 保存指针
        JP      DISUP                   ; 转至显示程序入口,等待输入新的键值
KEYDNA: LD      HL,     DIG2
        INC     (HL)                    ; 设置输入2位数标志
        JR      KEYDN7-$
KEYDN8: LD      HL,     DIG4
        INC     (HL)                    ; 设置输入4位数标志
        JR      KEYDN7-$
KEYDN9: CALL    ALTER                   ; 输入修改数据到存贮器、I/O端口或寄存器
        LD      HL,     (KEYPTR)
        DEC     HL                      ; 将指针修改为进入6位数时的位置
        LD      (KEYPTR),HL             ; 保存HL
        JP      DISUP
; 从命令键转移表中,找到命令键处理程序的
; 入口地址,此时键值在A中。
KEYDN6: SUB     10H                     ; 键值减去10H,求偏移量;键值为16偏移量为0
        LD      C,      A               ; 保存偏移量
        ADD     A,      C
        ADD     A,      C               ; 偏移量乘以3
        LD      C,      A
        LD      B,      00H             ; BC为3倍的偏移量
        LD      HL,     JPTAB           ; 取命令键转移表首地址
        ADD     HL,     BC              ; 查表找到该命令键的入口地址
        JP      (HL)                    ; 转至相应命令键的入口地址(首地址加偏移量)
; 命令键转移表
        ORG     020CH
JPTAB:  JP      CCS1                    ; EXEC
        JP      CCS2                    ; SS
        JP      CCS3                    ; MON
        JP      CCS4                    ; NEXT
        JP      CCS5                    ; REG' EXAM
        JP      CCS6                    ; REG EXAM
        JP      CCS7                    ; PORT EXAM
        JP      CCS8                    ; MEM EXAM
        JP      CCS9                    ; BP(设置断点)
        JP      CCS10                   ; DUMP TAPE(信息转贮磁带)
        JP      CCS11                   ; LOAD TAPE(磁带装入)
        JP      CCS12                   ; PROM 编程
; 键值转换表
        ORG     07B9H
KYTBL:  DB      0FFH                    ; 0  B=01, A=0F
        DB      0EFH                    ; 1  B=02, A=0F
        DB      0F7H                    ; 2  B=02, A=17
```

```
DB      0FBH            ; 3  B=02, A=1B
DB      0DFH            ; 4  B=04, A=0F
DB      0E7E            ; 5  B=04, A=17
DB      0EBH            ; 6  B=04, A=1B
DB      0CFH            ; 7  B=08, A=0F
DB      0D7H            ; 8  B=08, A=17
DB      0DBH            ; 9  B=08, A=1B
DB      0DDH            ; A  B=08, A=1D
DB      0EDH            ; B  B=04, A=1D
DB      0FDH            ; C  B=02, A=1D
DB      0DH             ; D  B=01, A=1D
DB      0BH             ; E  B=01, A=1B
DB      07H             ; F  B=01, A=17
DB      0EH             ; EXEC  B=01, A=1E
DB      0FEH            ; SS  B=02, A=1E
DB      0EEH            ; MON  B=04, A=1E
DB      0DEH            ; NEXT  B=08, A=1E
DB      0CDH            ; REG' EXAM  B=10, A=1D
DB      0CBH            ; REG EXAM  B=10, A=1B
DB      0C7H            ; PORT EXAM  B=10, A=17
DB      0BFH            ; MEM EXAM  B=10, A=0F
DB      0BDH            ; 断点(BP)  B=20, A=1D
DB      0BBH            ; 写带(DUMP)  B=20, A=1B
DB      0B7H            ; 读带(LOAD)  B=20, A=17
DB      0AFH            ; 编程(PROG)  B=20, A=0F
```

(6) 命令键处理程序(0230～0633H)

当按键被用作为命令键时，程序立即转入命令分析程序，找到相应命令键处理程序的入口地址，然后，按相应命令键处理程序进行处理。本机共设置 12 个命令键，每个命令键分别对应一个命令处理程序。这些命令处理程序按其功能可分为检查型及控制型两种。

① 检查型命令处理程序

这类命令程序的功能为检查内存单元、寄存器和输入/输出口的内容。这类命令的操作往往要结合数字键进行，其步骤如下：首先用数字键输入被检查单元的地址，输入数字键的数量视各命令键的内容而定，如存贮器检查为 4 个数字键；I/O 口检查为 2 个数字键；寄存器检查为一个数字键；设置断点为 4 个数字键；然后，按相应的命令键，该命令键将启动相应的命令处理程序使显示器显示所需要的内容。如果需要对该内容进行修改，只要继续按两个代表相应修改值的数字键即可。

现以 MEM EXAM 命令键处理程序为例加以说明：

显示存贮器内容

通过键盘先输入表示地址的 4 个 16 进制数字。键盘输入程序和显示程序将这 4 位数存入以 D SMEM 为首地址的连续的 4 个显示缓冲单元中。每送入一个数字，就立即显示在显示器上，最高位显示在最左端，然后，按下 MEM EXAM 键，键盘译码程序对此键进行译码，程序

就转入显示存贮器内容程序段。首先检查是否已输入了4个数，若没有，则转显示程序；若已输入，则设置存贮器检查标志。然后调用子程序UFOR2把已输入的4个数形成地址送入HL中，再用HL间址从内存中取出要显示的内容，接着调用子程序UFOR1，把此内容放入显示缓冲区，程序转向显示程序就可把内容显示出来了。整个程序段的程序如下：

```
；显示存贮器内容程序段
CCS8:   LD      A,      (DIG4)
        OR      A                       ；验证是否输入了4位数？
        JR      Z,      CCS8A-$         ；否，转显示程序
        LD      (MFLG), A               ；是，设置存贮器检查标志
        CALL    UFOR2                   ；把输入的4个数形成地址→HL
        LD      A,      (HL)            ；取出内容
        LD      IX,     DSMEM4          ；显示缓冲区指针→IX
        CALL    UFOR1                   ；把内容写入显示缓冲区
        INC     IX
        INC     IX
        LD      (KEYPTR), IX            ；修改缓冲区指针为修改存贮器内容作准备
CCS8A:  JP      DISUP
```

子程序UFOR2

在上述显示存贮器内容的程序中，调用了子程序UFOR2，它把已输入到显示缓冲区中的数取出来，并且把两个数拼成一个字节，高位字节送至寄存器H中，低位字节送至寄存器L中。程序如下：

```
；将键盘输入的在显示缓冲区前4位的4个数，变为16位地址送HL
UFOR2:  LD      IX,     DISMEM          ；IX作为缓冲区指针
        LD      A,      (IX)            ；取出第一位数
        SLA     A
        SLA     A
        SLA     A
        SLA     A                       ；在A内移至高四位
        OR      (IX+1)                  ；与第二位数组合
        LD      H,      A               ；形成高位字节送H
        LD      A,      (IX+2)          ；取第三位数
        SLA     A
        SLA     A
        SLA     A
        SLA     A                       ；移至高四位
        OR      (IX+3)                  ；与第四位数组合
        LD      L,      A               ；形成低位字节送L
        RET
```

子程序UFOR1

在显示内存内容程序中，还调用了子程序UFOR1，它以IX为显示缓冲区指针，把要显示的内容的高4位与低4位分开，然后分别送入显示缓冲区。其程序为：

```
UFOR1:  LD      B,      A               ；保存A的内容
```

```
        AND     0FH                     ; 屏蔽字节的高4位
        LD      (IX+1),     A           ; 写入字节的低4位
        LD      A,          B           ; 恢复A
        SRL     A
        SRL     A
        SRL     A
        SRL     A                       ; 字节的高4位移至低4位
        LD      (IX+0),     A           ; 高4位写入显示缓冲区
        RET
```

修改存贮器内容

MEM EXAM 键用来检查或修改存贮器 RAM 单元的内容，并用来输入程序或数据。在完成检查存贮器单元内容的操作之后，显示器上有 6 个数字，左边 4 个数字为地址，右边 2 个数字为存贮单元的内容。如果此时再输入 2 个数字，则程序就转至修改存贮器、I/O 端口或寄存器内容的程序 ALTER。在 ALTER 程序中，首先检查的是修改通道，或是修改主寄存器，或是修改替换寄存器。在确定是修改存贮器值之后，就调用子程序 UFOR2，把输入的 4 个数形成为地址送 HL，然后从显示缓冲区中取出最后输入的两个数，把它们拼成一个字节，写入指定的内存单元。再从此单元读回，调用 UFOR1，将它们写入显示缓冲区并加以显示。其程序如下：

```
        ; 修改存贮器内容程序
ALTER:  LD      IX,         DISMEM      ; IX指向显示缓冲区
        LD      A,          (PFLG)      ; 取通道检查标志
        OR      A                       ; 设置标志位
        JR      NZ,         ALTER2-$    ; 是通道修改，则转至ALTER2
        LD      A,          (RFLG)      ; 取寄存器检查标志
        OR      A                       ; 设置标志位
        JR      NZ,         ALTR3-$     ; 是寄存器修改，则转至ALTR3
        LD      A,          (ARFLG)     ; 取备用寄存器检查标志
        OR      A
        JP      NZ,         ALTR4       ; 是备用寄存器检查，转至ALTR4
        LD      A,          (MFLG)      ; 取存贮器检查标志
        OR      A                       ; 设置标志位
        JP      Z,          RESTR1      ; 若不是存贮器修改，则转回RESTR1
        CALL    UFOR2                   ; 是存贮器检查，调用UFOR2
                                        ; 输入的地址→HL
        LD      A,          (IX+6)      ; 取出写入的第一位数
        CALL    ALTER7                  ; 左移4位
        OR      (IX+7)                  ; 与第二位数组合，得到需要修改的输入值
        LD      (HL),       A           ; 写入内存指定单元
        LD      A,          (HL)        ; 再读回A
ALTER1: LD      IX,         DSMEM4
        CALL    UFOR1                   ; 把新的数写入显示缓冲区(DSMEM4)
                                        ; (DSMEM5)
        RET
```

```
; 子程序 ALTER7
ALTER7: SLA    A
        SLA    A
        SLA    A
        SLA    A
        RET
```

② 控制型命令处理程序

这类命令为控制单板微型计算机进行某种操作的命令，如启动用户程序运行，启动用户程序进行单步操作，启动盒式磁带机进行读或写以及将调试好的程序写入 EPROM 等。

有关各命令键的处理程序以及监控程序中的实用子程序，可参阅 Z80 STARTER SYSTEM KIT ZBUG 监控程序及流程图。ZBUG 监控程序功能比较完善，只要读者仔细地阅读并掌握监控程序具有模块结构的特点，就一定会从中学会输入输出程序设计的基本方法和技巧，并且还可以直接移植若干程序段作为用户程序的组成部分。

TPBUG-A 和 ZBUG-Y 监控程序，就是在原 ZBUG 监控程序的基础上，在不更改原机器硬件的情况下，利用双档键原理，新增了若干个实用的命令键功能，并对 ZBUG 监控程序的个别错误进行了修改。经过修改后的新监控程序的功能确实比原监控程序增强了。

§17-3 I/O 传送程序

一、用中断方式得到延时的 I/O 程序

前已述及，可用 CPU 执行一段程序来实现延时，称之为软件延时，其缺点是在延时期间，CPU 不能干别的工作，利用率低。因而常用外部定时器如 Intel 8253、Z80 CTC 等，将其设置为定时器工作方式，经过一个整定的延时后，向 CPU 发出一个中断请求，由 CPU 加以处理。这种用中断方式得到延时的方法，可实现 CPU 与外部定时器并行工作，提高 CPU 的利用率。

下面以 Z80 STARTER SYSTEM KIT 机上的 Z80 CTC 产生定时中断，执行一个简单的表演程序为例加以说明。先设置 Z80 CTC 的通道 0 工作在定时器方式，每隔 33 毫秒向 CPU 发出一个中断请求。改变输入到 CTC 的时间常数，可以改变请求中断的时间间隔。

在执行此程序后，字符“8”快速地从显示器右端移到左端，经过十次循环之后，立刻变为较慢的一次移动，接着又是十次较快的循环移动，又接着一次慢速移动……，如此循环不已。根据需要可以方便地改变显示的字符和循环的速度。按下 RESET 键，CTC 随即复位，移动也随之停止，程序回到 ZBUG 控制。

本表演程序的流程图如图 17-21 所示。

表演程序如下：

```
内存地址(H)  目标代码(H)  标号:  指令(助记符)          注释
                                ORG   2000H
2000         3E21          LD    A,    21H
2002         ED47          LD    I,    A          ; 设定中断矢量高位字节
2004         3E00          LD    A,    00H
2006         D384          OUT   (84H), A         ; 设定外设提供的中断矢量低位字节
2008         3EA5    ST:   LD    A,    0A5H
```

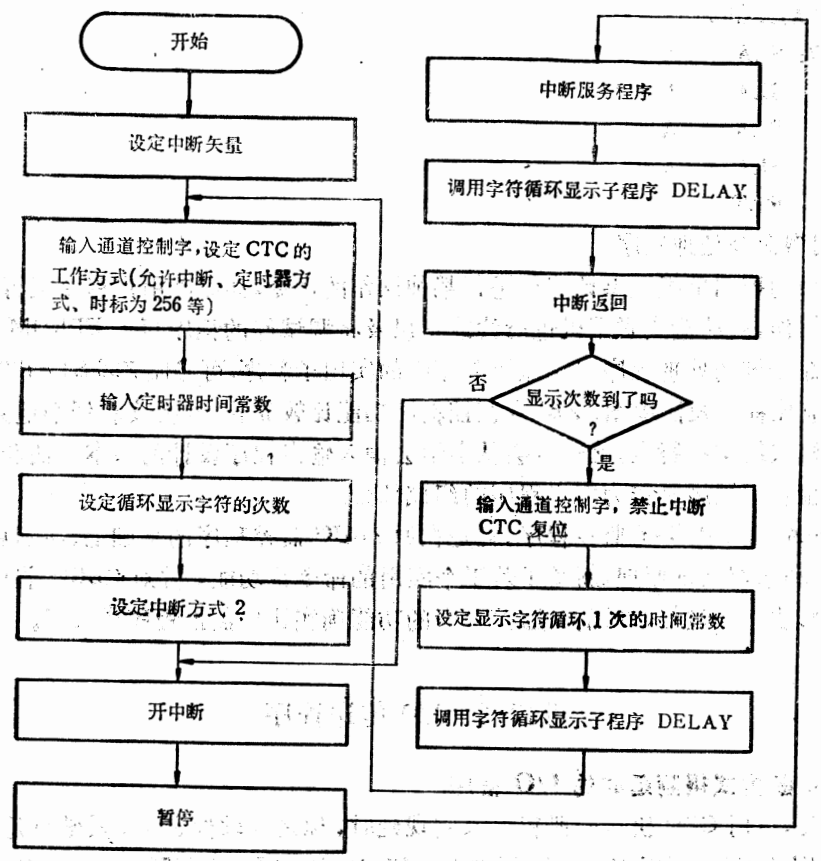


图 17-21 用中断方式延时程序流程图

```
200A   D384          OUT   (84H),  A        ; 设定 CTC 工作方式
200C   3EFF          LD    A,      0FFH
200E   D384          OUT   (84H),  A        ; 输入时间常数
2010   0EF6          LD    C,      0F6H     ; 设定快速循环次数(10 次)
2012   ED5E          IM    2                ; 设定中断方式 2
2014   FB      LOOP: EI                     ; 开中断
2015   76            HALT
2016   0C            INC   C
2017   20FB          JR    NZ,     LOOP-$   ; C≠0 循环
2019   3E03          LD    A,      03H
201B   D384          OUT   (84H),  A        ; 禁止通道中断
201D   1E5F          LD    E,      5FH      ; 设定字符循环一次的时间常数
201F   CD5020        CALL  DELAY            ; 调用子程序
2022   C30820        JP    ST               ; 循环返回重新工作
    ; 中断服务程序入口地址表
2100   0022
    ; 中断服务程序
2200   1E0A          LD    E,      0AH      ; 设定字符循环速度的时间常数
2202   CD5020        CALL  DELAY
```

```
2205   ED4D              RETI
   ; 子程序
2050   3E00    DELAY:    LD    A,    00H      ; 选定字符"8"
2052   D388              OUT   (88H),A
2054   3E01              LD    A,    01H
2056   D38C    LOOP1:    OUT   (8CH),A
2058   43                LD    B,    E
2059   CD4F06  LOOP2:    CALL  D20MS           ; 调用延时 20ms 子程序
205C   10FB              DJNZ, LOOP2-$         ; 若 B≠0, 则转 LCOP2, 若 B=0,
                                               ; 则继续
205E   CB6F              BIT   5,    A         ; 测试 A 的第 5 位 Z←Ā5
2060   2003              JR    NZ,   LOOP3-$   ; 若 A5=1, 则转 LOOP3
2062   07                RLC   A               ; A 累加器循环左移一位
2063   18F1              JR    LOOP1-$
2065   C9      LOOP3:    RET
```

若需显示别的字符，可参考 §17-2 的表 17-3，修改 DELAY 子程序的第一条指令 LD A，00H 中的立即数即可。

二、实时时钟中断 I/O 程序

实时时钟用来产生一系列相等间隔的脉冲序列，可利用这些脉冲序列定时地向 CPU 发出中断请求。对中断次数加以计数即可实现程序的日历钟——软件时钟计时。其基本方法是先将小时、分钟和秒存于不同的存贮单元中。其基本的计数间隔以一秒钟内发生的中断次数为准。当计满要求的中断次数时，秒计数器就加 1。当秒计数器为 60 时(日历钟是每秒 60 次)，它就被置"0"，同时分计数器加 1。当分计数为 60 时，它就被置"0"，同时小时计数器加 1。这种计数的累加值可继续到所需要的任何时间长度。

下面以 Z80 CTC 产生实时时钟中断为例加以说明。

1．通过程序设定 Z80 CTC 为定时器工作方式，并由 CTC 的 ZC0 提供一个周期为 1/100 秒的脉冲序列。每隔 1/100 秒，CTC 的 INT 端发出一个中断请求信号，使之转入中断服务程序。中断服务程序以 1/100 秒、秒、分、小时对实时时钟进行计数(即 1/100 秒计数器计到100，清"0"并向秒计数器进位；秒计数器计到 60 后清"0"，向分计数器进位。分计数器计满 60 后清"0"，向小时计数器进位；小时计数器计满 24，复位为"0")。同时在单板机的 LED 数码显示器上显示时、分、秒计数值。

2．实时时钟时间常数的计算

当设定 CTC_0 为定时器工作方式时，减 1 计数器的回零脉冲周期 $t = t_C \times P \times TC(s)$。

其中，t 是作为计时基准脉冲周期，取 1/100 秒值。

t_C 是系统的时钟周期，若主频为 1.9968 MHz，则 $t_C = \dfrac{1}{1.9968 \times 10^6}$ (s)

P 是 CTC 定标器系数，当通道控制字的第 5 位 $D_5 = 1$ 时，P = 256；当 $D_5 = 0$ 时，P = 16，本例取 P = 256。TC 是时间常数。当通道控制字的第 2 位置"1"时，下一条指令将时间常数输入到时间常数寄存器后，CTC 就开始工作。

现要求 CTC 每隔 1/100 秒申请一次中断，经过计算，时间常数 TC=78，即 TC=78=4EH。

应当注意，必须使每次中断服务时间小于 CTC 减 1 计数器的回零时间，否则程序不能正常工作。

3．实时时钟程序操作步骤

假设在 Z80 STARTER SYSTEM KIT 单板机上执行实时时钟程序，其操作步骤如下：

(1) 选定计数时、分、秒和 1/100 秒的存贮单元；

(2) 根据脉冲周期为 10 毫秒，选取时标 P=256，计算时间常数（由公式 $t=t_c\times P\times TC$）；

(3) 编制程序；

(4) 输入程序；

(5) 将实时时钟的初值存入时、分、秒 1/100 秒单元；

(6) 按 RESET 键，使 CTC 复位；

(7) 从程序首址开始执行程序，在 6 个显示器上从左到右分别显示时、分、秒值。

4．实时时钟程序流程图和程序

实时时钟主程序和中断服务程序如图 17-22 和图 17-23 所示。

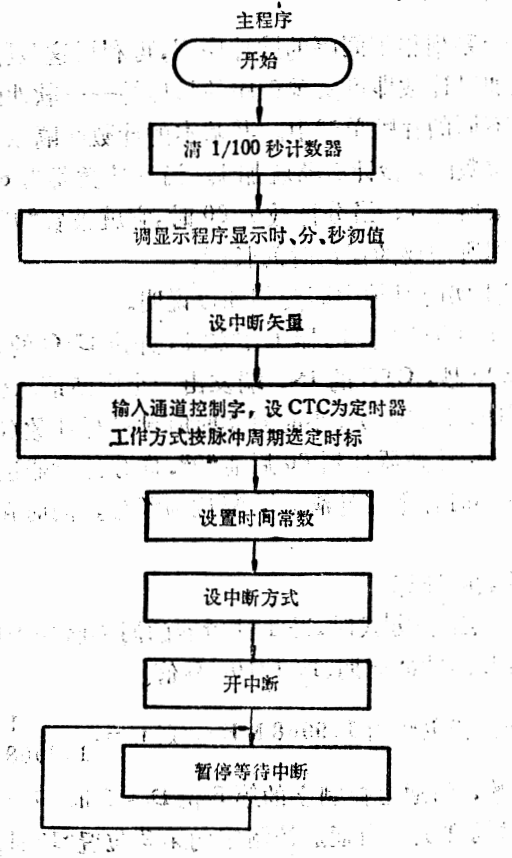


图 17-22 实时时钟主程序流程图

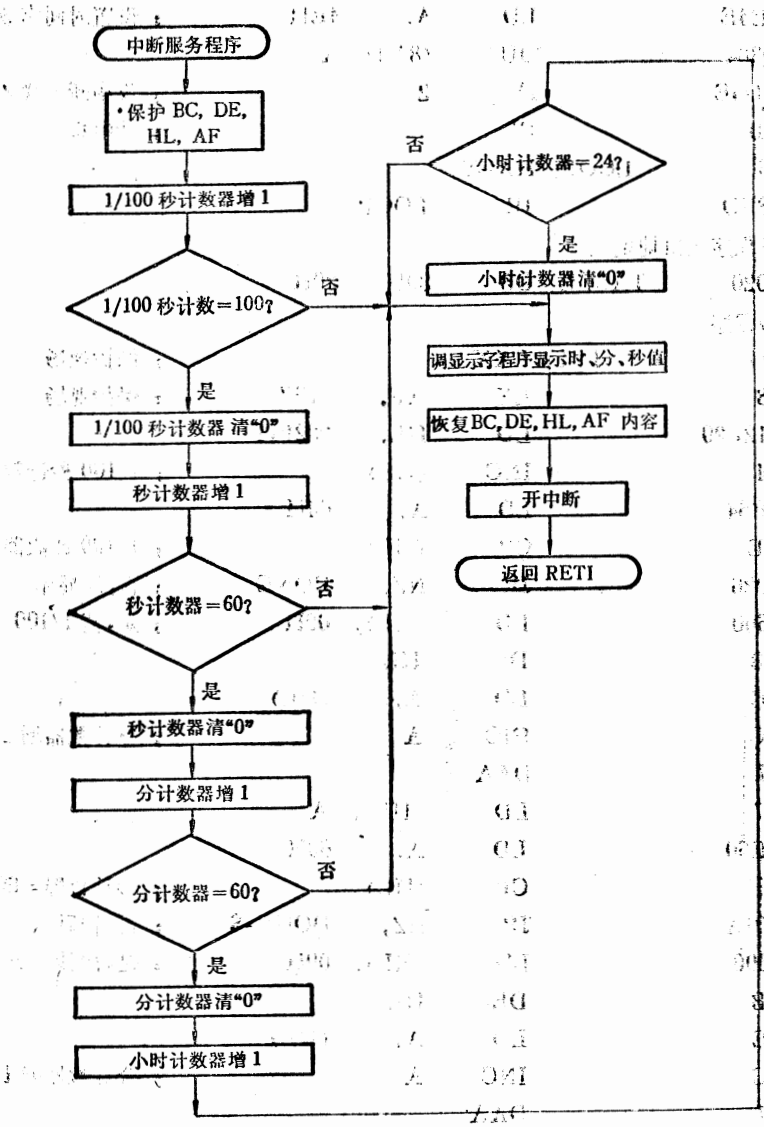


图 17-23 实时时钟中断服务程序流程图

实时时钟程序

内存地址 (H)	目标代码 (H)	标号	指令 (助记符)			注释
			ORG	2000H		
2000	3E00		LD	A,	00H	; 清 1/100 秒单元
2002	322F20		LD	(202FH),	A	
2005	CD7020		CALL	DISUP0		; 调用显示子程序
2008	3E20		LD	A,	20H	
200A	ED47		LD	I,	A	
200C	3E20		LD	A,	20H	; 设置中断矢量
200E	D384		OUT	(84H),	A	
2010	3EA5		LD	A,	10100101B	; 输入控制字,设置 CTC$_0$
2012	D384		OUT	(84H),	A	; 通道为定时器方式

```
2014    3E4E          LD    A,    4EH       ; 设置时间常数 TC=78
2016    D384          OUT   (84H), A
2018    ED5E          IM    2               ; 设中断方式 2
201A    FB            EI                    ; 开中断
201B    76      LOOP: HALT
201C    18FD          JR    LOOP
    ; 中断服务程序入口地址表
2020    3020  TABLE: DB    30H,  20H
    ; 中断服务程序
2030    D9      INT:  EXX                   ; 保护现场
2031    08            EX    AF,   A'F'      ; 保护现场
2032    212F20        LD    HL,   202FH
2035    34            INC   (HL)            ; 1/100 秒计数器增 1
2036    3E64          LD    A,    64H
2038    BE            CP    (HL)            ; 1/100 计数器 = 100?
2039    2026          JR    NZ,   DONE      ; 否,转显示
203B    3600          LD    (HL), 00H       ; 是,清 1/100 秒计数器
203D    2B            DEC   HL
203E    7E            LD    A,    (HL)
203F    3C            INC   A               ; 秒计数器增 1
2040    27            DAA
2041    77            LD    (HL), A
2042    3E60          LD    A,    60H
2044    BE            CP    (HL)            ; 秒计数器 = 60?
2045    201A          JR    NZ,   DONE-$    ; 否,转显示
2047    3600          LD    (HL), 00H       ; 是,清秒计数器
2049    2B            DEC   HL
204A    7E            LD    A,    (HL)
204B    3C            INC   A               ; 分计数器增 1
204C    27            DAA
204D    77            LD    (HL), A
204E    3E60          LD    A,    60H
2050    BE            CP    (HL)            ; 分计数器 = 60?
2051    200E          JR    NZ,   DONE-$    ; 否,转显示
2053    3600          LD    (HL), 00H       ; 是,清分计数器
2055    2B            DEC   HL
2056    7E            LD    A,    (HL)
2057    3C            INC   A               ; 小时计数器增 1
2058    27            DAA
2059    77            LD    (HL), A
205A    3E24          LD    A,    24H
205C    BE            CP    (HL)            ; 小时计数器 = 24?
205D    2002          JR    NZ,   DONE-$    ; 否,转显示
```

```
205F   3600              LD    (HL),   00H        ; 是,清小时计数器
2061   CD7020   DONE:    CALL  DISUP0
2064   D9                EXX                      ; 恢复现场
2065   08                EX                       ; 恢复现场
2066   FB                EI                       ; 开中断
2067   ED4D              RETI                     ; 返回
    ; 显示子程序
2070   DD21F72F DISUP0:  LD    IX,     DISMEM
2074   3A2C20            LD    A,      (202CH)    ; 时计数器置数
2077   CD3C06            CALL  UFOR1              ; A->(IX),(IX+1)
207A   DD21F92F          LD    IX,     DSMEM2     ; 分计数器置数
207E   3A2D20            LD    A,      (202DH)
2081   CD3C06            CALL  UFOR1              ; A->(IX),(IX+1)
2084   DD21FB2F          LD    IX,     DSMEM4     ; 秒计数器置数
2088   3A2E20            LD    A,      (202EH)
208B   CD3C06            CALL  UFOR1              ; A->(IX),(IX+1)
208E   21F72F            LD    HL,     DISMEM
2091   0620              LD    B,      20H
2093   5E       DISUP1:  LD    E,      (HL)
2094   1600              LD    D,      00H
2096   3E00              LD    A,      00H
2098   D38C              OUT   (DIGLH),A
209A   DD21 A607         LD    IX,     SEGPT
209E   DD19              ADD   IX,     DE
20A0   DD7E00            LD    A,      (IX+0)
20A3   D388              OUT   (SEGLH),A
20A5   78                LD    A,      B
20A6   D38C              OUT   (DIGLH),A
20A8   1E70              LD    E,      70H
20AA   1D       DISUP2:  DEC   E
20AB   3E00              LD    A,      00H
20AD   BB                CP    E
20AE   20FA              JR    NZ,     DISUP2-$
20B0   3E01              LD    A,      01H
20B2   B8                CP    B
20B3   2805              JR    Z,      DISUP3-$
20B5   23                INC   HL
20B6   CB38              SRL   B
20B8   18D9              JR    DISUP1
20BA   C9       DISUP3:  RET
                         ORG   2FF7H
2FF7           DISMEM:   DS    1
2FF8           DSMEM1:   DS    1
2FF9           DSMEM2:   DS    1
2FFA           DSMEM3:   DS    1
```

```
2FFB        DSMEM4:     1
2FFC        DSMEM5:     1
          UFOR1: EQU    063CH
          SEGLH: EQU    88H
          DIGLH: EQU    8CH
          SEGPT: EQU    07A6H
                 END
```

§17-4 微型计算机控制系统的实现

一、工业控制用计算机的组成和特点

1. 组成

工业控制用计算机称为工业控制计算机(简称控制机),其基本组成框图如图 17-24 所示。

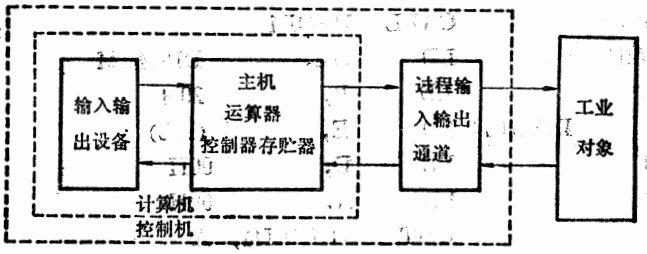


图 17-24 工业控制计算机组成框图

工业控制机由计算机和外围设备两大部分组成。在这里,外围设备由许多与生产过程相互联系的装置组成,故又称为过程输入/输出通道。它们一方面把工业控制对象的生产过程参数取出,转换成电子计算机能够接收的二进制代码送入计算机进行处理;另一方面把计算机发出的控制命令和其它数据信息,转换成改变工业控制对象的控制变量的信号,再送往工业控制对象。

2. 特点

控制机与一般计算机相比,具有下列特点:

(1) 要求有比较完善的过程通道,以便能实现各种形式的信息变换,如数/模转换,模/数转换,检测仪表和常规调节仪表等。

(2) 要求有比较完善的中断系统和高速数据通道,以便能迅速响应生产过程和各种外设发出的中断请求,并能与生产过程实时交换信息。

(3) 要求有高度的可靠性。因为许多生产过程是昼夜连续工作的,不能中途停顿,所以要求计算机控制系统有高度的可靠性,故障率低(一般允许几千小时出一次故障),并且平均故障时间要短(一次故障时间不超过几分钟)。

(4)要求有方便的人机联系措施,以便实现人机通信,及时地对生产过程进行必要的干预。

(5) 要求有正确反映生产过程规律的数学模型,以实现生产过程的最优控制,达到增加产量,提高质量,降低成本的目的。

(6)要求有比较完善的软件,包括具有比较完整的操作系统和应用程序,以提高控制质量。

总之,生产过程的数学模型、控制算法、传感器、控制元件、计算机系统和生产过程等组合

在一起，就组成了一个生产过程的计算机控制系统。

二、微型计算机控制系统的一些概念

1. 微型计算机开环控制(Microcomputer Open-Loop Control)系统

若μC控制系统的输出量对系统的控制作用没有影响，则称之为μC开环控制系统。在μC开环控制系统中，μC不接受来自受控过程输出的反馈而仅仅按操作者的要求计算出控制方案，或者说受控过程的输出不直接控制受控过程而仅将其结果提供操作人员参考，由操作人员决定控制方案并执行控制作用。μC开环控制系统的示意图如图17-25所示。

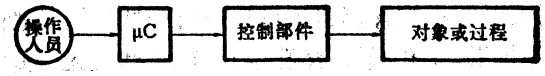


图17-25 微型计算机开环控制系统

2. 微型计算机闭环控制(Microcomputer Close-Loop Control)系统

若μC控制系统的输出量对控制作用产生直接影响，则称之为μC闭环控制系统或称μC反馈控制系统。在μC闭环控制系统中，μC在操作人员的监督下，自动地接收测量结果、计算控制方案并直接指挥控制部件的动作以控制生产过程。如图17-26所示。

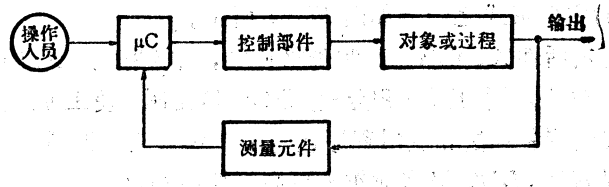


图17-26 微型计算机闭环控制系统

在此系统中，受控过程一方面接受计算机和控制部件的控制，另一方面又将自己的输出经过测量元件反馈给μC作为其输入，从而使受控过程、测量元件、μC、控制部件构成一个闭合环路。这样的控制称为微型计算机闭环控制。

μC闭环控制系统的特点是利用输入信号与反馈信号之间的偏差信号来减小系统的误差，并使系统的输出量趋于所希望的最佳值。

3. 在线控制与离线控制

(1) 在线控制(On-Line Control)

所谓"在线"控制是指对象或过程在μC直接参与下进行操作，不需要人工直接干预，故又称为联机操作。

(2) 离线控制(Off-Line Control)

所谓"离线"控制是指对象或过程并非在μC直接参与下进行操作，它要求人工进行直接干预，如定期或不定期地为该系统进行计算和控制，故又称为脱机操作。

4. 实时系统(Real-Time System)

所谓"实时"是指μC对外来的信息能以足够快的速度进行处理，并在规定的时间内迅速地作出响应或控制。这个规定的时间范围，就是最长的允许时间，若超过这个允许时间，就会失去控制的时机。

一个在线系统并不一定是一个实时系统。但是一个实时系统必定同时具有在线的功能和设备。例如，一个仅用于数据记录的计算机系统是一个在线系统，但不是一个实时系统。一个计算机直接控制系统必定是一个在线实时控制系统。

三、计算机控制系统的分类

1．按计算机参与控制的方式分类

(1) 数据采集系统(DAS-Data Acquisition System)

数据采集系统是微型计算机在工业控制中最广泛最简单的应用之一。在第十三章中，我们曾列举过DAS的实例。简单地讲，DAS是将系统所需要采集的过程变量经过ADC转换后送入微型机的内存。需要时，微型计算机还可以进行一些数据处理。计算的结果由微型计算机输出，记录在诸如打印机纸、穿孔纸带、穿孔卡片、磁带和磁盘等媒介物上，以备查用。

DAS是微型计算机控制系统的不可缺少的组成部分。其目的是实时地采集生产过程的各种参数，送入CPU处理，然后再去控制生产过程，或者是记录生产过程的各种工况，积累资料，供研究生产过程的数学模型用。

(2) 直接数字控制(DDC——Direct Digital Control)系统

微型计算机可以取代常规的模拟式调节仪表而直接对生产过程进行控制。由于它的控制信号是数字量，故称为DDC系统。DDC系统本质上是属于离散的采样控制系统，它已成为当前计算机控制系统的主要形式之一。如图17-27所示。

DDC的优点是：灵活性大、可靠性高和价格便宜，它能用数字运算的方式，对若干回路甚至数十个回路的生产过程，进行比例-积分-微分(PID)控制，使工业对象的各项参数保持在给定值上，而且只要改变控制算法和应用程序便可实现较复杂的控制，如串级控制、前馈控制和最佳控制等。一般情况下，DDC级多数作为更高级控制的执行级。

(3) 监督控制(SCC——Supervisory Computer Control)系统

SCC系统的功能是根据生产过程的各种状态，按数学模型计算出生产过程系统的最佳给定值，以作为过程控制的依据；然后，自动地或人工地对模拟式调节仪表进行整定，或者由下一级计算机进行直接数字控制。如图17-28所示。

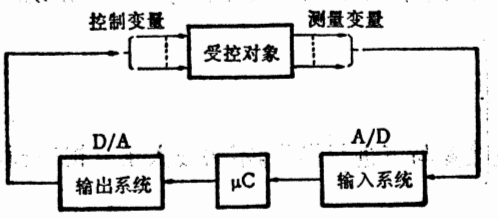


图17-27 DDC系统框图

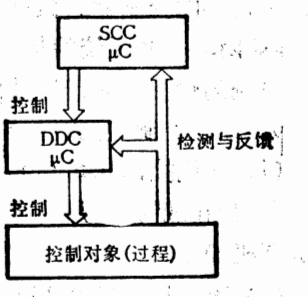


图17-28 DDC与SCC的结构示意图

SCC系统的优点是能保证生产过程始终处于最佳状态下运行，以获得最大的效益。直接影响SCC效果优劣的是它的数学模型、控制算法和应用软件。

通常，要求SCC具有较强的科学计算能力，有较大的内存、外存容量和较丰富的软件支持。

(4) 分级控制(Multi-level Control)或递阶控制(Hierarchical Control)系统

在生产规模较大、生产过程比较复杂的情况下，不仅需要解决过程控制问题，而且还要求

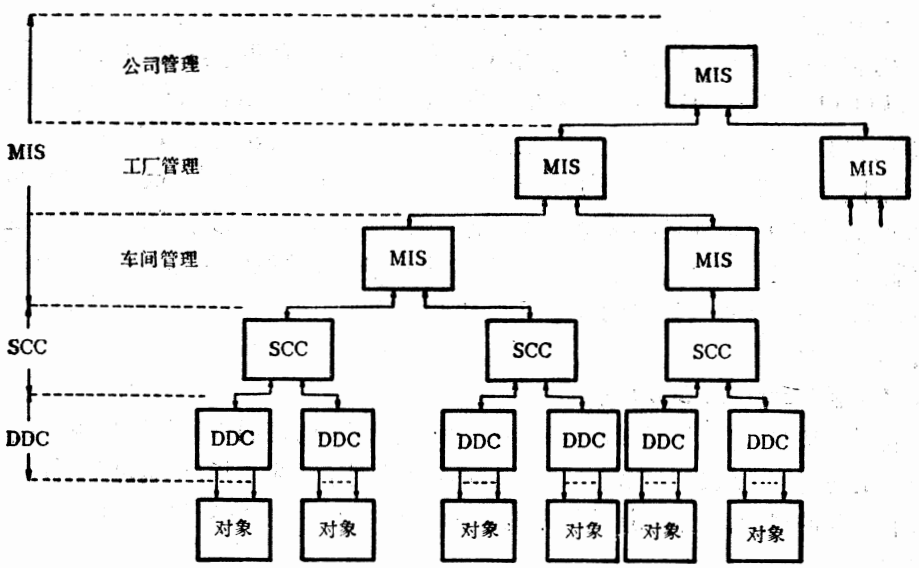


图 17-29　多级计算机控制系统的示意图

解决生产管理问题，于是出现了分级(递阶)控制系统(分级集成控制信息系统)。如图 17-29 所示。

图中，DDC 级主要用于直接控制生产过程，进行 PID 或前馈控制；SCC 级主要用于进行最佳控制或自适应控制的计算，以指挥 DDC 级工作，并向管理信息系统(MIS)级汇报情况。

MIS(Management Information Systems)级主要用于实现信息的实时处理，为各级决策者提供有用的信息，作出关于生产计划、调度和管理的决策，使计划协调和经营管理处于最优状态。并指挥 SCC 级工作。这一级又可视企业的规模和管理范围的大小分成若干级。例如可分成车间管理级、工厂管理和公司管理级。其中，公司管理级的主要功能是进行产品预测计算，制订长期发展规划，并根据定货要求、原料供应情况和企业的生产情况，进行最优化计算，制订出整个企业较长时期(月或旬)的生产计划、销售计划，并向各工厂管理级下达任务。

工厂管理级的主要功能是根据上级机下达的生产任务和本厂的实际情况，进行最优化计算，制订出本厂生产计划和短期(旬或日)安排，然后给车间管理级下达任务。

车间管理级的主要功能是根据厂级下达的命令和收集上来的生产过程的各种信息，随时进行最优调度，指挥 SCC 级工作。

通常，公司或工厂级的 MIS 的计算机都是中、大型的通用计算机，它要求具有很强的科学计算和数据处理能力以及大容量的内存和外存，并且一般都有一个数据库。

车间级的 MIS 的计算机通常是中、小型计算机或高档微型计算机。

SCC 级通常是小型计算机或高档微型计算机。

DDC 级通常是微型计算机。

(5) 分布式控制(Distributed Control)或分散控制(Decentralized Control)系统

分布式控制或分散控制，是指由若干个独立的子系统(分散控制器)来共同完成整个控制系统的总任务。微型机的出现和迅速发展，为实现分散控制提供了物质技术基础，近年来，分散控制得到了蓬勃的发展，而且成为计算机控制发展的重要方向。

七十年代出现的分散型综合控制系统(TDCS——Total Distributed Control System)，

简称集散系统，便是采用分散控制的新型的计算机控制系统。

我们知道，传统的计算机控制系统是集中型的，其特点是信息集中，控制集中，一旦计算机的公用部分（如 CPU，过程 I/O 接口等）发生故障，就会影响整个系统的正常工作，即所谓“危险集中”，这样，势必降低系统的可靠性。与之相比，分散控制采用控制分散化，使“危险分散”，这样，如果个别子系统发生故障，不致影响整个系统的正常工作。由于分散控制系统可靠、灵活，以及微型机成本低廉，因而近年来越来越引起人们的关注，有着极其广阔的发展前途。

分散控制和递阶控制的区别在于：分散控制指的是作为递阶控制的局部控制级；递阶控制则是在分散控制基础上配以协调控制级而组成的多级控制。

2．按调节规律分类

(1) 程序控制(Program Control)

要求控制系统按照预先规定的时间函数进行变化，这样的控制称为程序控制。例如要求某炉温按照一定的时间曲线进行控制就属于程序控制的例子。应当指出，这里的所谓“程序”，其含义仅具有时间的概念，而不是指计算机的程序。因为早在电子计算机出现前，就已经有程序控制的概念了。

(2) 顺序控制(Sequence Control)

顺序控制是程序控制发展的产物。这种控制系统每一时刻的给定值的确定，不仅取决于时间还取决于此时刻以前的控制结果的逻辑判断。

(3) 比例-积分-微分 PID(Proportional-integral-derivative)控制

PID 控制的输出信号不仅与输入偏差和偏差积分成正比例，而且与偏差的微分即其变化速度成正比例。由于微分作用的存在，这种控制能超前动作，克服对象的动态滞后而改善调节品质。

(4) 前馈控制(Feedforward Control)

在反馈控制系统中，必须在干扰造成一定的后果后，才能反馈过来产生抑制干扰的控制作用。而前馈控制不需要等干扰作用影响到被控变量(这总是需要一定时间的)，而是在测得干扰量的大小以后，经过适当的延时，就在干扰点的前方，加入一个前馈控制作用，使它正好能够完全抵消干扰对被控变量的影响，故又称为扰动补偿。

(5) 最优控制，(最佳控制) (Optimum Control)

所谓最优控制，就是恰当地选择控制规律，在控制系统规定的工作条件的限制下，使得系统的某种性能指标(或目标函数)最小(或最大)。例如，要求在规定的限制条件下，某个综合指标最好，或完成一定的产量和质量所消耗的燃料最少等等。

(6) 自适应控制(Adaptive Control)

上述最优控制系统，当其工作条件或限制条件改变时，就不能获得最优的控制效果了。如果在工作条件已经改变了的情况下，仍能使系统具有适应环境的变化并始终保持最优性能指标的功能，这样的控制称为自适应控制。

(7) 自学习控制(Learning Control)

如果控制系统能随着运行条件的改变，而不断地积累经验，并能根据所积累的经验，自动地改变控制器本身的结构和参数，使控制效果越来越好，这样的控制系统称为自学习控制系统。

应当指出，最优控制、自适应控制和自学习控制等复杂的控制系统，都涉及到复杂的数学

计算，因此，必须要有很强科学计算能力的计算机才能实现。

四、微型计算机基本控制系统的组成

通常用于DDC级的微型计算机，要求结构简单、运行可靠、易于实现、成本低廉，因此，目前在工业控制中，单板微型计算机获得了广泛的应用。

图17-30示出了一个微型计算机基本控制系统的框图。

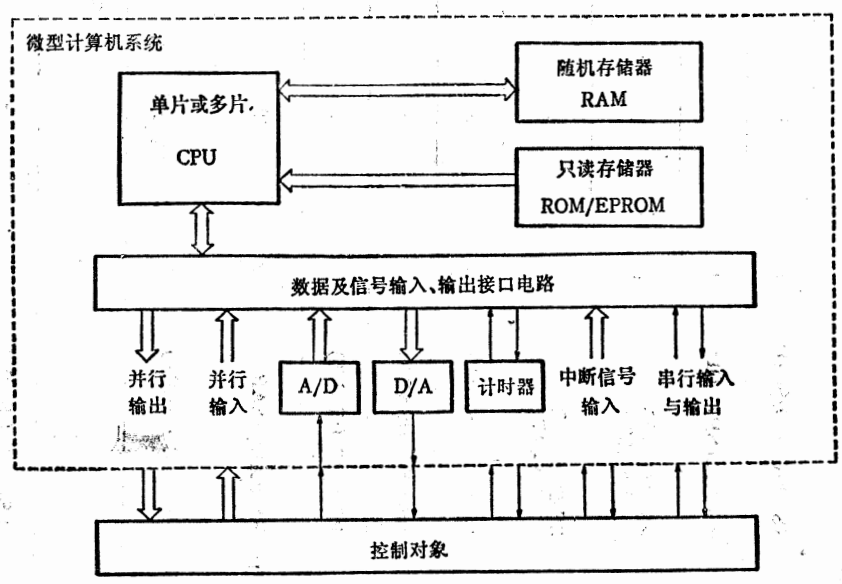


图17-30　微型计算机基本控制系统的框图

在这种μC基本控制系统中，一般是以一台单板微型计算机为核心，根据控制系统的实际需要，扩展必要的RAM、EPROM、并行I/O接口、串行I/O接口以及计时器等组成的。

为了进一步扩展单板微型计算机的功能，许多厂家设计了各种不同功能的扩展板，如存贮器扩展板，I/O扩展板、通信控制扩展板等，从而构成积木式的单板微型计算机系列。这样，不但主机板可以根据需要选用，而且扩展板也可根据需要选用，具有灵活、通用、可扩等特点。用户可以根据实际需要，组成实用的微型计算机应用系统。

下面列举一些常用的典型产品加以说明：

(一) ISBC系列单板微型计算机

美国Intel公司的ISBC系列单板微型机就是上述的积木式单板微型机，只要将各种功能的插件板插在总线上，即可进行系统扩展。为了方便使用，还提供通用机箱和工业控制用的机箱、信号条件板和各种端子板。

1．主机板

ISBC系列的单板微型机的品种、型号及功能，见表17-7。

2．扩展板

(1) 存贮器扩展板

这类扩展板有ISBC016、032、048、064，ISBC028A、056A、012B，分别有16、32、48、64、128、256、512 KB动态RAM；还有ISBC416、464的16、64 KB EPROM扩展板。

(2) 外围控制器插件板

表 17-7 ISBC 系列单板微型计算机的品种、型号及功能

品　种	CPU	EPROM(用 2716) 容量(字节)	SRAM (静态)或 DRAM(动态)容量(字节)	可编程序 并行 I/O	串行 I/O 能力	定时品数目	中 断 级 数	MULTIBUS (多总线能力)	有无 ISBX 板连接插座
80/04	8085	EPROM 4K	SRAM 256	22	RS-232C(SID/SOD)	1	4	无	无
80/05	8085	EPROM 4K	SRAM 512	22	RS-232C(SID/SOD)	1	4	多主	无
80/10A	8080A	EPROM 4K	SRAM 1K	48	RS-232C/TTY (USART)	0	1	单主	无
80/10B	8080A	EPROM 16K	SRAM 4K	48	RS-232C/TTY (USART)	1	1	单主	1
80/20	8080A-2	EPROM 8K	SRAM 2K	48	RS-232C(USART)	2	8	多主	无
80/20-4	8080A-2	EPROM 8K	SRAM 4K	48	RS-232C(USART)	2	8	多主	无
80/24	8085A	EPROM 32K	SRAM 4K	48	RS-232C(USART)	2	12	多主	2
80/30	8085A	EPROM 8K	DRAM 16K 双口存取	24+UPI*	RS-232C(USART)	2	12	多主	无
86/12A	8086	EPROM 4～16K	DRAM 32K	24	RS-232C(USART)	2	9 (可扩到65 级)	多主	无
88/40	8088	EPROM 16～32K	SRAM 4K 其中1K 双口存取	24	无	3	9	多主，独立控制 智能从机	3
86/14	8086-2	EPROM 8～64K	SRAM 32K	24	RS-232C(USART)	2	9	多主	2
86/30	8086-2	EPROM 8～64K	DRAM 128K 双口存取	24	RS-232C(USART)	2	9	多主	2

* UPI-Universal Peripheral Interface (通用外围接口电路)

这类插件板有 ISBC 202、204、208、215 A/215B、220 等 5 种。如 ISBC 208 板有通用单双密度软磁盘控制器，能支持 4 个 5″ 或 8″ 的单双密度软磁盘控制器等。

(3) 数字量 I/O 扩展板

这类扩展板，有 ISBC 501、508、517、519、530、556、569 等 7 种。如 ISBC 519 有 72 条可编程序的并行 I/O 线，有插座提供连接端和驱动、8 级中断控制以及可选的 0.5 ms、1 ms、2 ms 或 4 ms 间隔的定时器。

(4) 模拟量 I/O 扩展板

这类扩展板，有 ISBC 711、724、732 等 3 种。

ISBC 711 是模拟量输入板，包括 8 路微分和 16 个单终点输入多路转换器，可编程序增益放大器、采样/保持器、12 位 A/D 转换器、定时时钟及接口逻辑等。

ISBC 724 是模拟量输出板，包括 4 个独立的 12 位 D/A 转换器(电压输出) 及接口逻辑。备有单极性或双极性 0～+10V、0～+5V 及 ±10V 量程。

(5) 通信控制器插件板

这类插件板，有 ISBC 534、544 两种。如 ISBC 544 是智能通信控制器，板上装有 8085 微处理器，8K EPROM，16K 的双口存取 RAM，12 级中断控制，10 条可编程序的并行 I/O 线，4 个可编程序通信接口 8251，3 个可编程序定时器。它既能作为独立的单板通信控制器，也可以在通信网络中作为智能从机，起前端处理机作用。

3．外设配置

TTY、CRT、行打印机、键盘等

4．MULTI 总线

MULTI 总线即 MULTI BUS，也称为 SBC 多总线，是 Intel 公司为 SBC 系列插件板的开发而生产的。可以通过 MULTI 总线方便地构成各种不同规模的微型计算机系统。

MULTI 总线把地址总线、数据总线、控制总线和电源分别集中安排在一个区域内，便于印刷板的布线和查找，有利于减小相互之间的干扰。该总线已被 IEEE 委员会定义为工业标准 IEEE-796。

5．软件支持

PL/M-80，FORTRAN-80、BASIC-80 等。

目前，国内已生产出一部分与 ISBC 80 系列单板微型机及扩展板相兼容的产品。

其它系列的单板微型机就不一一列举了。

(二) 工业控制微型计算机系统

这是一种完整的工业控制微型计算机系统，适用于信息处理要求高、希望系统有较大的存贮容量和比较齐全的外围设备的生产过程控制系统。

这类 μC 基本系统的主要性能特点如下：

(1) CPU：Intel 8080A/8085，Z80(Z80A)，MC 6800；

(2) 存贮容量：RAM 4～16KB，EPROM 4K～16KB；

(3) 并行 I/O 接口：8 位，4～6 个；

(4) 串行 I/O 接口：2 个，RS-232(RS-232C)或 RS-422；

(5) 定时/计数器：4 个；

(6) 中断方式：根据 CPU 种类，设置优先中断控制电路，如选用可编程序中断控制器

Intel 8259;

(7) 模拟量输入通道：16路，10或12位A/D转换器；

(8) 模拟量输出通道：8路，10或12位D/A转换器；

(9) 人-机联系接口(两种可选)

① DEP数据输入操作面板；

② CRT终端；

③ 行打印机1台：用于定时地或随机地打印；

(10) 实时时钟及报警；

(11) 盒式磁带机1台：用于记录组态信息及数据；

(12) 模拟型回路操作面板(可选)：指示和设定给定值、测量值及阀位；手/自动切换开关；手操旋钮等；

(13) 系统软件(全部固化于EPROM中)

① 监控程序(管理程序)；

② 控制算法软件包。

(14) 电源系统：交流220 V、50 Hz、配有隔离变压器和滤波器，并具有上电自启动和掉电保护的功能。

这类基本控制系统被设计成积木形式，可根据系统的实际需要扩充成为不同规模的应用系统。通常在基本系统的基础上可作如下扩充：

(1) 存贮器容量：最大可扩至64 KB；

(2) 开关量输入：24点，光电隔离式；

(3) 开关量输出：32点，继电器输出；

(4) 开关量输出切换操作面板；

(5) EPROM编程器；

(6) 软磁盘：5″双面双密度。

(三) 分散型综合控制系统TDCS

从七十年代初期起，国外就开始了TDCS的研制和开发工作。美国Honeywell于1975年11月制成了TDC-2000系统。接着美国、日本等国的一些公司也相继制成了TDCS。

TDCS是以满足现代化大型工业生产的要求为前提，实现过程综合自动化为目标而研制、开发出来的。它是以微型机为核心，与数据通信系统、CRT显示装置、以及用于数据采集和控制的过程I/O接口相结合的新型控制系统。

TDCS的特征是采用"危险分散"的设计思想，将系统的控制任务分散给各个子系统独立地完成，而将操作、显示部分高度集中于监控级和管理级的计算机，以便操作人员实现集中监控和生产管理工作。具体地说，集散系统包括以微型计算机为核心的分散控制级(即以μC为核心的基本控制器)、数据通道、CRT操作站、监控级和管理级微型计算机等几个主要组成部分，再加上强大的软件支持系统。其系统结构框图如图17-31所示。

由图可知，集散系统属于分层(或分级)结构。其中，分散控制级包括若干台基本控制器，分别由一台微型计算机为核心，扩展必要的RAM、EPROM和I/O接口，再加上过程通道(ADC和DAC)以及常用的应用程序所组成。它们分别并行地独立完成对生产过程的数据采集和实时控制的功能。

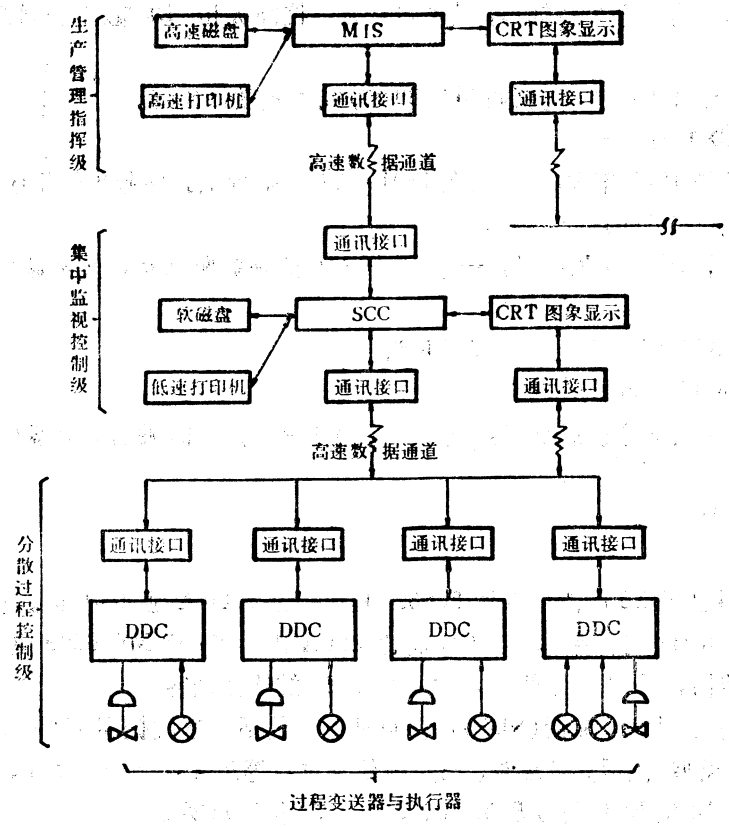


图 17-31 集散型微型计算机控制系统框图

监控级通常以一台具有磁盘的微型计算机系统为核心，来完成生产过程的数据处理和最优控制，并利用 CRT 操作站对生产过程实现集中显示、报警和人机联系等监视操作。

管理级是各级指挥中枢，其功能如前所述，这里不再重复。数据通道是 TDCS 的支柱，它将各个基本控制器、监控计算机和 CRT 操作站有机地连接起来，以实现综合控制。

（四）IBM 个人计算机系统

IBM 个人计算机(IBM Personal Computer，简称 IBM-PC)是美国 IBM 公司于 1981 年底制成的。它是一种可以广泛用于科学计算、商业、管理和工业控制等方面的微型计算机系统。

目前 IBM 公司提供 IBM-PC1 和 IBM-PCXT 两种型号。

1．IBM-PC1 的组成

(1) CPU：Intel 8088 是一种高性能的 16 位微处理器，指令系统与有 99 条指令的 8086 兼容。

(2) 动态 RAM 64 KB，可扩展至 640 KB。

(3) ROM 40 KB。

(4) 键盘：83 键，10 个数字键、10 个功能键（供用户定义）以及其它标准打字键。键盘内含有 8048/8748 单片 μC。

(5) 显示连接器(黑白/彩色)。

(6) 2 个 5″ 软盘驱动器。

(7) 1 个扬声器。

(8) 主机板上有5个扩展插口，其中包括显示控制板和FDC磁盘控制板，还余下3个插口可供扩展。

(9) 可外接标准点阵打印机(如MT-80等)或盒式磁带机等。

2．IBM-PCXT的组成

IBM-PCXT是1983年IBM公司在IBM-PC1的基础上扩展起来的系统(XT意即extended)。它主要作了如下的扩充：

(1) 设置一个5″软盘驱动器，一个5″ 10 MB的硬盘驱动器，使磁盘总容量达到10.6 MB。

(2) 动态RAM 128 KB可扩至640 KB。

(3) ROM 40 KB，可扩至256 KB。

(4) 主机板上有8个扩展插口，其中4个用于显示/打印、硬盘和软盘驱动板、异步通信板。

3．系统的特点

(1) 高性能

① 采用高性能的16位微处理器8088，其功能比8080A强4～6倍。它采用HMOS工艺制造，在4.77 MHz时钟频率下工作，将比Z80A快4倍；比MC 6809快2～3倍。它具有16位内部体系结构，1MB寻址能力，并且与8086的软件兼容。

② 当配上数值运算处理器(NDP)后，可使系统的运算速度提高100倍左右。8087总共可处理7种数据类型，包括算术运算、三角函数及对数等运算，能处理8位、16位、32位、64位、二进制整数、18位BCD码以及32位、64位、80位浮点实数。

③ 当配上总线控制器8288后，可以使系统适用于多处理机系统。

④ 8088可采用与8080A、8085A同样的存贮器，而且在执行中没有等待状态，提高了性能价格比。

(2) 易于扩充

① 专设一个扩展箱，该箱自备电源和时钟，与主机箱联接，可以扩展固定式或盒式磁盘，内设8个扩展槽、全部供用户选用。

② 有数量众多的扩展板，例如有：

通信用：

同步数据连接(SDLC)接口板

二进制同步通信(BSC)接口板

系统扩充：

内存扩展板(512KB)

盒式5 MB、10 MB硬盘扩展板

I/O扩展板

过程控制：

A/D、D/A扩充板

X-Y传感接口板

等等。

(3) 丰富的软件

目前 IBM-PC 已开发的软件达 2000 多种，包括操作系统、语言、通信、专业、文字处理、商业、教育、工业控制、网络、游戏娱乐等几大类。

语言有：BASIC、FORTRAN、COBOL、Macro、Assember、PASCAL、ALOGO、APL、LISP、C 等。

操作系统有：PC-DOS、CP/M-86、UCSD-P-SYSTEM 等。

(4) 兼容性好

IBM-PC 能容易地与 IBM 公司的大中型计算机联机，也能与任何拥有 IBM 程序转换软件包的其他公司的大中型计算机联机。共享硬盘和打印机的 OMNINET 和低中速的 ETHERNET 局部网络都能支持 IBM-PC 共享资源。

以上特点使得 IBM-PC 成为当前颇受欢迎的机种之一。特别是在企业管理、办公室自动化方面，IBM-PC 将成为一种比较理想的工具。

五、微型计算机控制系统的实现

1．以 SDK-85 为核心组成的微型计算机控制系统

SDK-85 单板微型计算机的系统结构和各部件的工作原理，我们已在第十六章作过介绍。8085A CPU 的独特优点是将时钟发生器、系统控制器及中断优先权控制器等功能都集中到 CPU 芯片上，Intel 公司还专门为 8085A CPU 设计了把存贮器和 I/O 接口电路集成在一个芯片上的外围接口芯片，这就使 SDK-85 单板机在器件数量上减到最小的限度，为形成最小系统提供了可能。因此，我们只要在 SDK-85 单板机上扩充 A/D、D/A 转换电路，就可以构成微型计算机实时控制系统，如图 17-32 所示。

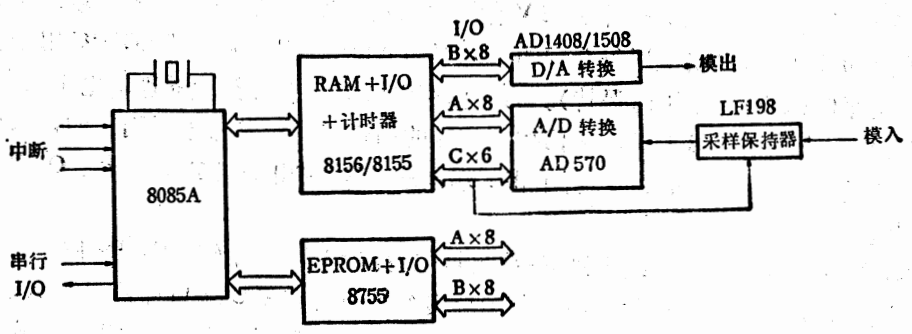


图 17-32　以 SDK-85 为核心的微型计算机控制系统

图中除 AD 570 和 LF 198 外，其余的器件都分别在第十五章、第十六章中作过介绍，这里不再重复。下面简要介绍 AD 570 和 LF 198 的功能。

(1) AD 570 是单片 8 位模数转换器。它是由数/模转换电路、电压基准、脉冲源、比较器、逐次逼近寄存器和输出缓冲器等组成的。转换时间为 25 μs，供电电源为 +5 V 和 −15 V，AD 570 允许 0 到 10 伏单极性输入或 −5 到 +5 V 双极性输入，输入阻抗为 5 kΩ 左右。

AD 570 电路如图 17-33 所示。

图中，AIN 为模拟信号的输入端，为了避免地线对模拟信号的干扰，模拟输入地线 A．COM 与数字输出的地线 D．COM 分别设置。$D_7 \sim D_0$ 为三态数字输出端，当转换信号 $\overline{\text{CONVER}}$ 有效时(低电平)，电路开始 A/D 转换，直到转换结束发出 $\overline{\text{DATA READY}}$ (低电平)

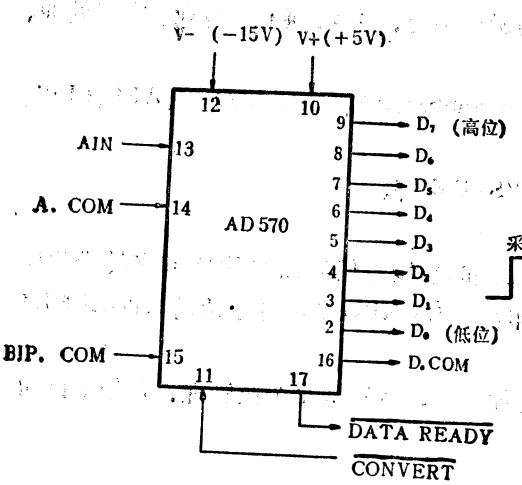


图 17-33 AD570 电路图

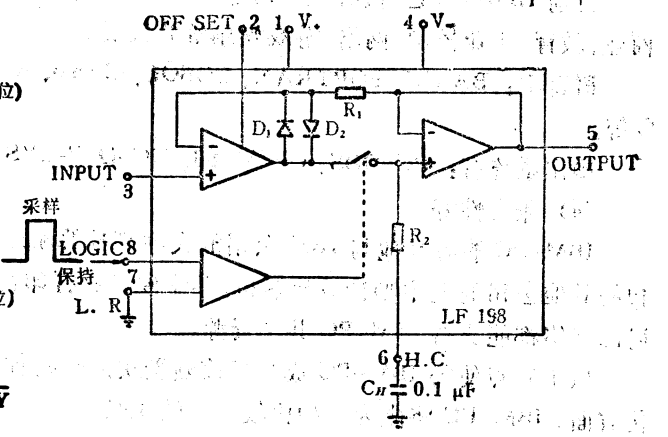


图 17-34 LF198 采样保持电路原理图

之前，D_7～D_0 输出端一直保持高阻抗状态。BIP.CON 端用于选择输入信号的极性类别，当此端接地时，允许输入信号范围为 0 伏到 +10 伏；若将此端浮空，则允许输入信号为双极性，其范围从 −5 伏到 +5 伏，此时与 −5 伏对应的输出信号为 00H，+5 伏则对应 FFH。

A/D 转换器在转换期间（25 μs），要求输入信号保持稳定或在小范围内（20 mV 内）变化，否则将导致很大的转换误差。因而在工程中通常在模拟信号与 A/D 转换器之间接入采样保持电路。

(2) LF198 采样保持器的原理如图 17-34 所示。

LF198 允许供电电源在 ±5 V 到 ±18 V 范围内波动。保持电容 C_H 的数值取决于保持时间的长短，$C_H = 0.1$ μF 时输出电压每分钟的下降率为 5 mV。逻辑输入端 LOGIC 用于控制采样或保持（高电平时采样；低电平时保持）。为了使逻辑端能与多种类型的逻辑电平兼容，电路中设置了逻辑电平参考输入端 L.R，因而控制信号的高低是相对参考电平而言的，只要差值在 1～4 伏以上，便确认为高电平。如图所示将 7 端接地，则控制电平与 TTL 兼容。OFFSET 端用于零位调整。

现对本控制系统的工作过程说明如下：系统复位后向 CPU 发出 $\overline{\text{RESET IN}}$ 信号，使 CUP 从 0000H 地址单元开始执行初始化程序，其主要内容是使所有 I/O 端口都处于输入工作方式，避免在系统开始运行时受控对象接收错误信息。各 I/O 端口及计时器的工作方式由初始化程序来设置。由于 8155 的 I/O 端口与 D/A 和 A/D 转换电路相接，因此它的各个端口的工作方式应规定为：端口 A——输入；端口 B——输出；端口 C 的低位 (PC_0, PC_1, PC_2) 用于控制 PA，高位 (PC_3, PC_4, PC_5) 用于输出。此时控制字的低 4 位应设置为 $D_3D_2D_1D_0 = 0110$。从端口 C 的 $PC_5/PC_4/PC_3$ 位发出“0”信号以后，LF 198 处于信息保持状态，同一信号启动 A/D 转换器；当 A/D 转换结束并将数据置入输出缓冲器时，通过 PC_2 端向 8155 发出选通信号 STB；当端口 A 接收数据后，通过 PC_1 端向外部电路发出 BF（缓冲器接收信息结束）信号，接着通过 PC_0 端向 CPU 的 RST 6.5 端发出中断请求信号，形成重新启动指令，其执行结果将使 CPU 自动转向 0034 H，并作为 A/D 转换服务程序的入口地址。于是通过执行中断服务程序将端口 A 的数据读入 CPU。CPU 对输入的数据进行处理后，通过 8155 的端口 B 输至 AD 1408 进行 D/A 转换，然后再将转换后的模拟量输出至受控对象，以实现微型计算机控制系统的控制功

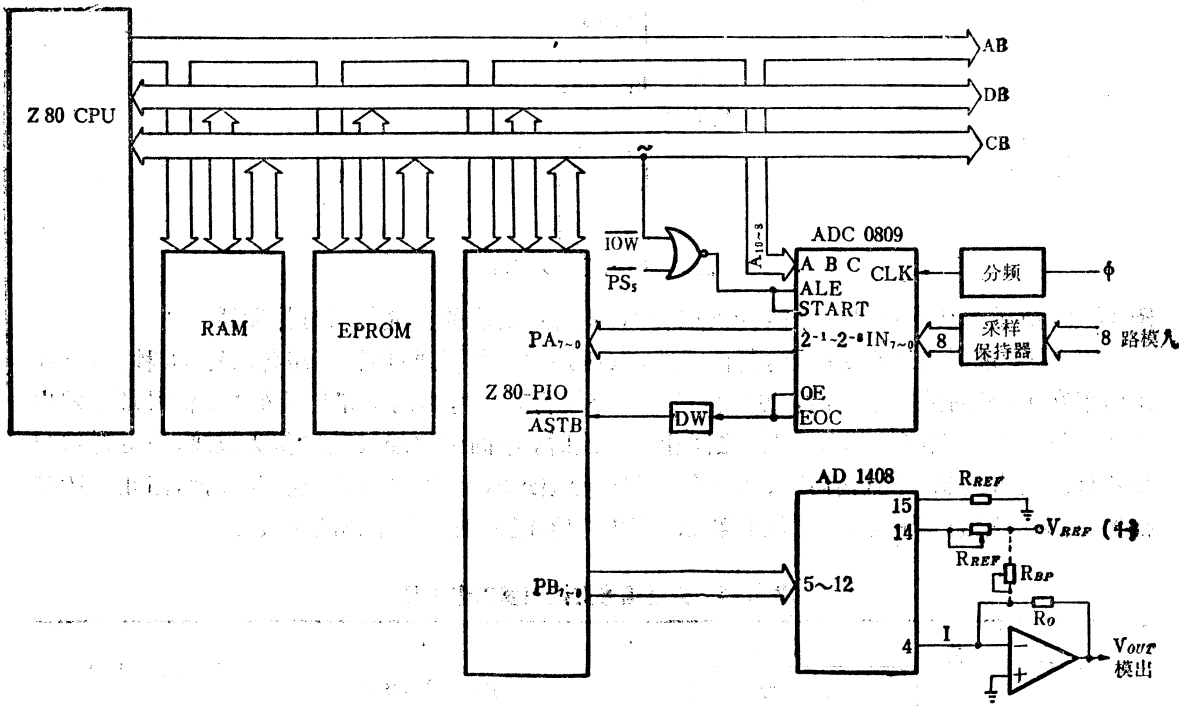


图 17-35　以 Z80 STARTER SYSTEM KIT 为核心组成的微型计算机控制系统

能。

2．以 Z80 STARTER SYSTEM KIT 为核心组成的微型计算机控制系统

如图 17-35 所示，利用一台 Z80 STARTER SYSTEM KIT 单板微型计算机，配备以 ADC 0809 和 AD 1408 等器件为基础的过程通道，即可方便地组成微型计算机控制系统。

下面简要说明本控制系统的工作过程：首先设置初始化程序预置 Z80 PIO 各通道的工作方式，如图 17-35 所示，应规定：端口 A——输入；端口 B——输出。本控制系统中，用于A/D、D/A 转换器接口的 Z80 PIO，是利用系统 I/O 译码器空余的译码输出端 $\overline{PS_6}$ 进行片选的，故 PIO 中的 4 个通道号为 98 H～9BH，其中 98H、99H 分别对应端口 A 及端口 B 的数据寄存器，9AH、9BH 分别为端口 A 及端口 B 的控制寄存器的端口地址。然后，按照 PIO 的编程格式设置初始化程序。关于 ADC 0809 和 AD 1408 的工作过程已在第十五章介绍过，不再重复。本例把 ADC 0809 作为单通道的 A/D 转换器使用，如果 8 路模拟输入共用一个采样/保持器，则在采样/保持器前面需要配置多路转换开关(如 CD 4501)。此外，应当指出，当把 ADC 0809 作为 8 通道的 A/D 转换器使用时，为了保证可靠的工作，宜将 ALE、START 输入端分别加以控制。

§17-5　微型计算机应用实例

一、应用微型计算机的交通信号灯控制器

1．设计要求

南北路 N/S(A 道)，东西路 E/W(B 道)两道交叉口，设有车辆检测器 A_1、A_2、B_1、B_2，如

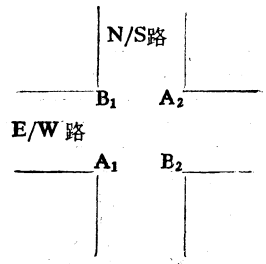


图 17-36 十字交叉路口

图 17-36 所示。要求设计一个能提供如表 17-8 所示功能的交通信号灯控制器。

表 17-8 说明在正常情况下，若南北路，东西路均有车要通过时，每路各放行 18 秒。若一路上的车在 18 秒内放行不完时，也应让给另一路放行 18 秒；当两条路均无车时，信号灯不发生转换，保持原状态不变；当红、绿灯之间相互转换时，中间必须经过黄灯作为过渡，黄灯每次只亮 2 秒。当有紧急车辆(消防车、救护车、警车等)通过时，两路红灯均应亮 2 分钟，阻止一般车辆通行，只让紧急车辆通过(假设有紧急车辆检测器能检测紧急车辆的出现)。

表 17-8 交通信号灯控制器工作情况

项目 / 说明	工作状态						状态时间	
	南北 N/S(A 道)			东西 E/W(B 道)				
	D_5 红(R)	D_4 黄(A)	D_3 绿(G)	D_2 红(R)	D_1 黄(A)	D_0 绿(G)	代码(H)	时间(秒)
任意起始状态，保持此状态一直到 PN 出现	1	0	0	0	0	1	21	18
按次序进行状态转换，此间，若出现 PE 则不计	1	0	0	0	1	0	22	6
	1	0	0	1	0	0	24	2
	1	1	0	1	0	0	34	6
保持此状态直到 PE 出现	0	0	1	1	0	0	0C	18
按次序进行状态转换，此间，若出现 PN 则不计	0	1	0	1	0	0	14	6
	1	0	0	1	0	0	24	2
	1	0	0	1	1	0	26	6
保持此状态直到 PN 出现	1	0	0	0	0	1	21	18
紧急车辆(救护车、消防车、警车)通过 PF 有效	1	0	0	1	0	0	24	120

2．系统方案设计

用 SDK-85 单板微型机来处理这个交通信号灯控制器，其系统框图如图 17-37 所示。

其中 A_1、A_2 表示南北路(N/S)上的车辆检测器，若 $\overline{PN}=\overline{A}_1+\overline{A}_2=$"0"，则表示南北路有车要通过。$B_1$、$B_2$ 表示东西路(E/W)上的车辆检测器，若 $\overline{PE}=\overline{B}_1+\overline{B}_2=$"0"，则表示东西路有车要通过。PF：表示紧急车辆检测器，当有紧急车辆要通过，由 PF 向 RST 7.5 端发一正脉冲信号，以请求中断。

R_A、A_A、G_A：当它们分别为“1”时，表示南北路为红、黄、绿灯亮。

R_B、A_B、G_B：当它们分别为“1”时，表示东西路为红、黄、绿灯亮。

现设 SDK-85 机上的 8155 作为 I/O 接口电路，其端口 A 的 PA_5～PA_0 作为信号灯状态输出线，用以控制交叉路口的信号灯。端口 B 的 PB_0、PB_7 作为车辆阻塞信号的输入线，用以检测来车的信号。

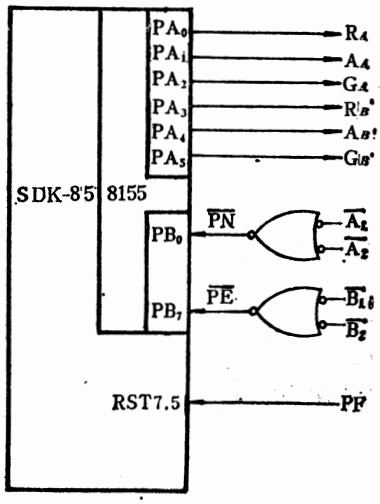


图 17-37　交通信号灯控制器系统框图

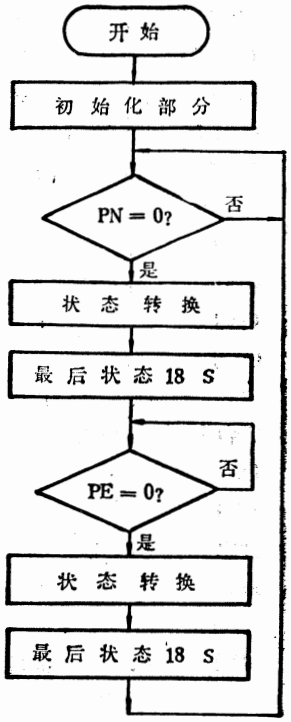


图 17-38　主控制程序流程图

3．程序设计

(1) 程序流程图

主控制程序流程图如图 17-38 所示。

(2) 控制程序

① 主程序

```
标号      指令(助记符)          注释
          ORG   2000H
          LXI   SP,   20C2H   ; 设置推栈指针
START:    MVI   A,    01H     ; 设置8155的工作方式
          OUT   20H
          MVI   A,    08H     ; 设置解除中断屏蔽字
          SIM
          EI                  ; 开中断
          MVI   A,    21H     ; 设置起始工作状态
          OUT   21H
SEQ1:     IN    22H           ; 测检 PN = 0?
          ANI   01H           ; 屏蔽无用位
```

```
        JNZ     SEQ1            ; 若 PN≠0, 则转 SEQ1
        MVI     A,      22H     ; 若 PN = 0, 则状态转换
        OUT     21H
        MVI     E,      06H     ; 延时 6 秒
        CALL    CLOCK           ; 调用 1 秒子程序
        MVI     A,      24H
        OUT     21H
        MVI     E,      02H     ; 延时 2 秒
        CALL    CLOCK
        MVI     A,      34H
        OUT     21H
        MVI     E,      06H     ; 延时 6 秒
        CALL    CLOCK
        MVI     A,      0CH
        OUT     21H
        MVI     E,      12H     ; 延时 18 秒
        CALL    CLOCK
SEQ2:   IN      22H             ; 检测 PE = 0?
        ANI     80H             ; 屏蔽无用位
        JNZ     SEQ2            ; 若 PE≠0, 则转 SEQ2
        MVI     A,      14H
        OUT     21H
        MVI     E,      06H
        CALL    CLOCK
        MVI     A,      24H
        OUT     21H
        MVI     E,      02H
        CALL    CLOCK
        MVI     A,      26H
        OUT     21H
        MVI     E,      06H
        CALL    CLOCK
        MVI     A,      21H
        OUT     21H
        MVI     E,      12H
        CALL    CLOCK
        JMP     SEQ1
```

② 1 秒子程序

```
        ORG     2070H
CLOCK:  MVI     B,      04H
CLK1:   MVI     C,      D4H
CLK2:   MVI     D,      00H
CLK3:   DCR     D
```

```
        JNZ     CLK3
        DCR     C
        JNZ     CLK2
        DCR     B
        JNZ     CLK1
        DCR     E
        JNZ     CLOCK
        RET
```

③ 中断服务程序

```
        ORG     2090H
INT:    PUSH    B           ;
        PUSH    D           ;   保护现场
        PUSH    H           ;
        PUSH    PSW         ;
        MVI     A,  24H     ;
        OUT     21H
        MVI     E,  78H     ;
        CALL    CLOCK
        POP     PSW         ;   恢复现场
        POP     H           ;
        POP     D
        POP     B
        EI
        RET
        ORG     20CEH
20CEH   JMP     INT         ;  当 RST 7.5 中断时，控制转移到 20CEH。
```

顺便指出，本例中的 RST 7.5 中断借用 SDK-85 机上的矢量中断键 VECT INTR 来实现。当按下 VECT INTR 键立即产生 RST 7.5 中断，使程序转到 003CH 单元，003 CH 单元中存放了无条件转移指令，使程序转移到用户随机存贮器中的 20CEH 单元。本例在 20CE～20DOH 单元内放一无条件转移指令，使程序转入中断服务程序的入口 INT，并执行中断服务程序。

二、应用微型计算机控制的直流调速系统

用微型计算机实现调速系统的直接数字控制，这是当前国内外电气自动化领域中普遍关注的研究课题之一。它与模拟式调节系统相比，最主要的优点就是控制的灵活性。由于系统的控制方案是由软件来实现的，因此它不仅能实现 PID 控制，而且还能引入其它多种先进的控制规律，诸如非线性控制、前馈控制、最优控制和自适应控制等。

图 17-39 示出微型计算机控制的可控硅双闭环直流调速系统的框图。

1. 系统的组成

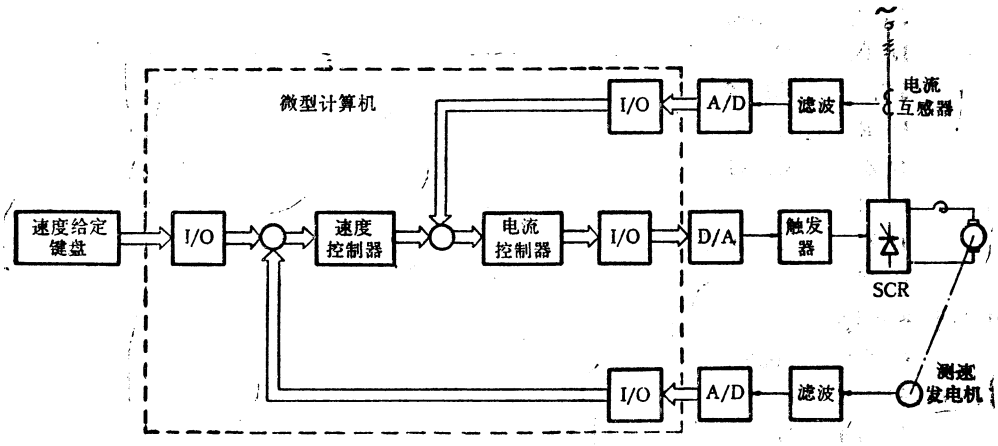


图 17-39　用 μC 实现 DDC 调节的可控硅双闭环直流调速系统的框图

系统由以下五部分组成：

(1) SDK-85 或 Z80 STARTER SYSTEM KIT 单板微型计算机。可根据需要扩充一定容量的 RAM 和 EPROM 以及 I/O 接口。

(2) 微型计算机控制系统的实时外围设备及其接口。在本系统中主要的是 A/D 和 D/A 转换器以及一个数字量的速度给定键盘。在选择这些外围设备时，应当着重考虑所选择的 A/D 和 D/A 转换器，其精度和速度是否满足控制系统的要求。

(3) 正弦波或锯齿波作为同步电压的可控硅触发器。

(4) 检测电路。其中电流检测器使用的是交流互感器，速度检测使用的是测速发电机。

(5) 主回路。主回路由交流电源、三相桥式全波可控硅变流器及它激直流电动机组成。

2．采样周期的选择和采样过程

在本系统中，微型机起着数字控制器的作用，它代替了速度、电流的模拟 PI 调节器。设计这样一个微型计算机控制系统的主要问题是采样周期的选择和确定数字控制器的实现方案。

(1) 采样周期的选择

采样周期是指两次采样之间的时间间隔。显而易见，采样周期 T 越小，精确度越高，但却增加了不必要的计算负担；而 T 过长，又会造成很大的误差，使系统的动态品质变坏，甚至导致系统的不稳定。究竟如何选择合适的采样周期 T 呢？

采样定理只给我们指出确定采样周期的原则，并未给出解决实际问题的条件公式。因而目前选择采样周期，大都还是参考经验数据，在实践中依靠试验来确定。

对于一个控制回路的采样周期可以采用该回路自然振荡周期的 1/8～1/100。

对于一般的闭环调速系统，可取采样周期为被控制对象的时间常数的 1/10。由于速度环的时间常数为 100～200 ms，因此，速度控制器的采样周期选为 $T_1 = 10 \sim 20$ ms(取 10ms)；而电流环的时间常数为 10～20 ms，故电流控制器的采样周期选为 $T_2 = 1 \sim 2$ ms (取 2 ms)。

(2) 采样过程

可以采用定时器(如 Z80 CTC) 实现电流及速度的定时采样，并用优先级中断的方法实现分时控制。

电流环的采样每隔 1 ms 进行 1 次，这就要求定时器每隔 1 ms 发出一个有效的脉冲信号，启动 A/D 转换器开始转换，待转换结束后，由 A/D 转换器的转换结束信号通过单稳触发

器，形成有效的脉冲信号送至可编程序的 I/O 接口(如 Z 80 PIO)的选通端(如 $\overline{\text{ASTB}}$)，于是 I/O 接口电路向 CPU 发出中断请求，执行电流环中断服务程序 INT1，把电流采样值取入 CPU，送到内存保存。

同样地，速度环采样每隔 10 ms 进行 1 次，CPU 通过执行速度环中断服务程序 INT2 把速度采样值取入 CPU，送到内存。

3．计算机的分时控制与数字滤波

本控制系统采用离散化的 PID 调节算法，取代速度及电流的模拟量的 PI 调节器。经过采样将反馈值取入 CPU，经 PID 调节后，再经 D/A 转换器及保持器，控制可控硅的触发角，从而调节直流电机的转速。由于计算机既要进行电流环的 PID 运算，又要进行速度环的 PID 运算，同时，还要控制电流及速度的采样，A/D 及 D/A 转换等多项工作，因此，采用优先级中断的方式实现分时控制。

在计算机控制系统中，抗干扰问题是一个必须解决的技术关键之一。通常采用硬件滤波(如用RC 低通滤波器等)和软件数字滤波(如取一定个数采样值的平均值等)的方法，来抑制或消除干扰。数字滤波实质上是一种程序滤波，其稳定性高，易于多路复用，近年发展快，用途广。

4．微型机控制的直流调速系统的研究动向

双闭环直流调速系统成功地解决了系统的无差调速和起动电流的控制问题。近年来，人们正在探索改变传统的双闭环直流调速方案，采用微机全数字化直流调速方案，在控制算法上，采用最小拍控制和反电势跟踪控制，已取得初步成果。在提高系统精度方面，采用光电脉冲发生器取代测速电机，利用同步变压器提供同步脉冲，以减少积累误差。

关于计算机控制系统的具体设计及实施技术将由“微型计算机控制技术”课程来解决，本书就不作详细介绍了。

结　束　语

展望八十年代，微型计算机的发展正方兴未艾，前途无量。近年来，全世界微型计算机的产量以每年 40% 的速度递增，这种发展速度在人类科学技术史上是空前的。随着超大规模集成电路技术的飞速发展，目前，国外市场上已出现了 32 位微处理器，如 Intel 432 和 NS 16000 系列，其集成度为每片 45 万个晶体管，其性能足以同高档小型计算机相媲美。功能极强的 64 位微处理器也即将问世。国外有人预言，本世纪末将出现毫微处理器(Nanoprocessor)，即单片集成度为 1 亿个晶体管。将来还可能出现微微处理器(Picoprocessor)，电路将达到分子电路的水平。

随着微型计算机功能的不断增强，它正在逐步取代小型计算机，并且进而逼近中、大型计算机，这已成为当代计算机工业发展的必然趋势。

目前，微型计算机的应用范围，几乎遍及人类社会的所有领域。据统计，到目前为止，国外微型计算机的应用已有 6000 多种。估计本世纪末，全世界将有 50～100 亿台微型计算机在各行各业中应用。随着微型计算机应用的大普及、大发展，当前在全世界范围内，一场以微型计算机应用为标志的新技术革命正在蓬勃兴起。可以预料，在我国四个现代化建设中，微型计算机应用技术必将日益发挥出强大的威力。

附　录

附录1　Z80指令的寻址方式和操作码

表A1-1　8位传送指令组(LD)

目的地 \ 源		隐含		寄存器							寄存器间接			变址		扩展寻址	立即
		I	R	A	B	C	D	E	H	L	(HL)	(BC)	(DE)	(IX+d)	(IY+d)	(nn)	n
寄存器	A	ED 57	ED 5F	7F	78	79	7A	7B	7C	7D	7E	0A	1A	DD 7E d	FD 7E d	3A n n	3E n
	B			47	40	41	42	43	44	45	46			DD 46 d	FD 46 d		06 n
	C			4F	48	49	4A	4B	4C	4D	4E			DD 4E d	FD 4E d		0E n
	D			57	50	51	52	53	54	55	56			DD 56 d	FD 56 d		16 n
	E			5F	58	59	5A	5B	5C	5D	5E			DD 5E d	FD 5E d		1E n
	H			67	60	61	62	63	64	65	66			DD 66 d	FD 66 d		26 n
	L			6F	68	69	6A	6B	6C	6D	6E			DD 6E d	FD 6E d		2E n
寄存器间接	(HL)			77	70	71	72	73	74	75							36 n
	(BC)			02													
	(DE)			12													
变址	(IX+d)			DD 77 d	DD 70 d	DD 71 d	DD 72 d	DD 73 d	DD 74 d	DD 75 d							DD 36 d n
	(IY+d)			FD 77 d	FD 70 d	FD 71 d	FD 72 d	FD 73 d	FD 74 d	FD 75 d							FD 36 d n
扩展寻址	(nn)			32 n n													
隐含	I			ED 47													
	R			ED 4F													

注：粗线框内为与8080A/8085A相同的操作码。

表 A1-2　16 位传送指令组(LD, PUSH, POP)

目的地			源 寄存器 AF	BC	DE	HL	SP	IX	IY	立即扩充 nn	扩展地址 (nn)	寄存器间接 (SP)
寄存器		AF										F1
		BC								01 n n	ED 4B n n	C1
		DE								11 n n	ED 5B n n	D1
		HL								21 n n	2A n n	E1
		SP				F9		DD F9	FD F9	31 n n	ED 7B n n	
		IX								DD 21 n n	DD 2A n n	DD E1
		IY								FD 21 n n	FD 2A n n	FD E1
扩展地址		(nn)		ED 43 n n	ED 53 n n	22 n n	ED 73 n n	DD 22 n n	FD 22 n n			
寄存器间接		(SP)	F5	C5	D5	E5		DD E5	FD E5			

推入指令 →　　↑ 弹出指令

注：每次执行之后，推入与弹出指令调整 SP(堆栈指示器)。

表 A1-3(a)　交换指令组(EX 和 EXX)

		隐含寻址				
		AF′	BC′,DE′和HL′	HL	IX	IY
隐含	AF	08				
	BC DE & HL		D9			
	DE			EB		
寄存器间接	(SP)			E3	DD E3	FD E3

表 A1-3(b)　数据块传送指令组

目的地	源		
		寄存器间接	
		(HL)	
寄存器间接	(DE)	ED A0	'LDI':—(DE) ←传送 (HL) HL 和 DE 增 1,BC 减 1
		ED B0	'LDIR':—(DE) ←传送 (HL) HL 和 DE 增 1,BC 减 1 如此重复到 BC=0
		ED A8	'LDD':—(DE) ←传送 (HL) HL 和 DE 减 1,BC 减 1
		ED B8	'LDDR':—(DE) ←传送 (HL) HL 和 DE 减 1,BC 减 1 如此重复到 BC=0

HL 指示源地址
DE 指示目的地地址
BC 指示字节计数器

表 A1-3(c)　数据块搜索指令组

查找单元	
寄存器间接	
(HL)	
ED A1	'CPI' HL 增 1,BC 减 1
ED B1	'CPIR'—HL 增 1, BC 减 1,重复直到 BC=0 或找到需要的字符
ED A9	'CPD' HL 和 BC 减 1
ED B9	'CPDR'—HL 和 BC 减 1,重复直到 BC=0 或找到需要的字符

HL 指示其内容应与累加器内容相比较的存贮单元地址
BC 指示字节计数器

表 A1-4　8位算术和逻辑指令组

源

	寄存器寻址							寄存器间接	变址		立即
	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)	n
加 (ADD)	87	80	81	82	83	84	85	86	DD 86 d	FD 86 d	C6 n
带进位加 (ADC)	8F	88	89	8A	8B	8C	8D	8E	DD 8E d	FD 8E d	CE n
减 (SUB)	97	90	91	92	93	94	95	96	DD 96 d	FD 96 d	D6 n
带进位减 (SBC)	9F	98	99	9A	9B	9C	9D	9E	DD 9E d	FD 9E d	DE n
'与' (AND)	A7	A0	A1	A2	A3	A4	A5	A6	DD A6 d	FD A6 d	E6 n
'异' (XOR)	AF	A8	A9	AA	AB	AC	AD	AE	DD AE d	DD AE d	EE n
'或' (OR)	B7	B0	B1	B2	B3	B7	B5	B6	DD B6 d	FD B6 d	F6 n
比较 (CP)	BF	B8	B9	BA	BB	BC	BD	BE	DD BE d	FD BE d	FE n
增量 (INC)	3C	04	0C	14	1C	24	2C	34	DD 34 d	FD 34 d	
减量 (DEC)	3D	05	0D	15	1D	25	2D	35	DD 35 d	FD 35 d	

表 A1-5　16 位算术指令

目的地		源 BC	DE	HL	SP	IX	IY
加(ADD)	HL	09	19	29	39		
	IX	DD 09	DD 19		DD 39	DD 29	
	IY	FD 09	FD 19		FD 39		FD 29
带进位加和置位标志(ADC)	HL	ED 4A	ED 5A	ED 6A	ED 7A		
带进位减和置位标志(SBC)	HL	ED 42	ED 52	ED 62	ED 72		
增量(INC)		03	13	23	33	DD 23	FD 23
减量(DEC)		0B	1B	2B	3B	DD 2B	FD 2B

表 A1-6(a)　通用运算指令

累加器十进制调整(DAA)	27
累加器变反(CPL)	2F
累加器变补(NEG)(2 的补码)	ED 44
进位标志变反(CCF)	3F
进位标志置位(SCF)	37

表 1-6(b)　其它 CPU 控制指令

空操作(NOP)	00	
暂停(HALT)	76	
禁止中断(DI)	F3	
开放中断(EI)	FB	
置中断方式 0(IM0)	ED 46	IM0 8080A 方式
置中断方式 1(IM1)	ED 56	IM1 转向 0038 H 单元重新启动
置中断方式 2(IM2)	ED 5E	IM2 利用寄存器 I 的内容和请求中断的设备提供的低 8 位中断矢量作指针的间接调用

表 A1-7　循环和移位指令

循环或移位的类型	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)
'RLC'	CB 07	CB 00	CB 01	CB 02	CB 03	CB 04	CB 05	CB 06	DD CB d 06	FD CB d 06
'RRC'	CB 0F	CB 08	CB 09	CB 0A	CB 0B	CB 0C	CB 0D	CB 0E	DD CB d 0E	FD CB d 0E
'RL'	CB 17	CB 10	CB 11	CB 12	CB 13	CB 14	CB 15	CB 16	DD CB d 16	FD CB d 16
'RR'	CB 1F	CB 18	CB 19	CB 1A	CB 1B	CB 1C	CB 1D	CB 1E	DD CB d 1E	FD CB d 1E
'SLA'	CB 27	CB 20	CB 21	CB 22	CB 23	CB 24	CB 25	CB 26	DD CB d 26	FD CB d 26
'SRA'	CB 2F	CB 28	CB 29	CB 2A	CB 2B	CB 2C	CB 2D	CB 2E	DD CB d 2E	FD CB d 2E
'SRL'	CB 3F	CB 38	CB 39	CB 3A	CB 3B	CB 3C	CB 3D	CB 3E	DD CB d 3E	FD CB d 3E
'RLD'								ED 6F		
'RRD'								ED 67		

	A
'RLCA'	07
'RRCA'	0F
'RLA'	17
'RRA'	1F

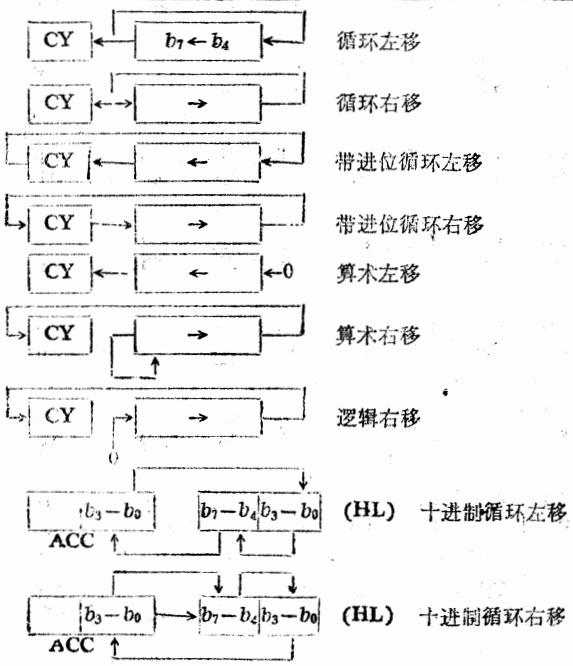


表 A1-8 位操作指令组

	位	寄存器寻址 A	B	C	D	E	H	L	寄存器间接 (HL)	变址 (IX+d)	(IY+d)
位测试(BIT)	0	CB 47	CB 40	CB 41	CB 42	CB 43	CB 44	CB 45	CB 46	DD CB d 46	FD CB d 46
	1	CB 4F	CB 48	CB 49	CB 4A	CB 4B	CB 4C	CB 4D	CB 4E	DD CB d 4E	FD CB d 4E
	2	CB 57	CB 50	CB 51	CB B2	CB 53	CB 54	CB 55	CB 56	DD CB d 56	FD CB d 56
	3	CB 5F	CB 58	CB 59	CB 5A	CB 5B	CB 5C	CB 5D	CB 5E	DD CB d 5E	FD CB d 5E
	4	CB 67	CB 60	CB 61	CB 62	CB 63	CB 64	CB 65	CB 66	DD CB d 66	FD CB d 66
	5	CB 6F	CB 68	CB 69	CB 6A	CB 6B	CB 6C	CB 6D	CB 6E	DD CB d 6E	FD CB d 6E
	6	CB 77	CB 70	CB 71	CB 72	CB 73	CB 74	CB 75	CB 76	DD CB d 76	FD CB d 76
	7	CB 7F	CB 78	CB 79	CB 7A	CB 7B	CB 7C	CB 7D	CB 7E	DD CB d 7E	FD CB d 7E
位复位(RES)	0	CB 87	CB 80	CB 81	CB 82	CB 83	CB 84	CB 85	CB 86	DD CB d 86	FD CB d 86
	1	CB 8F	CB 88	CB 89	CB 8A	CB 8B	CB 8C	CB 8D	CB 8E	DD CB d 8E	FD CB d 8E
	2	CB 97	CB 90	CB 91	CB 92	CB 93	CB 94	CB 95	CB 96	DD CB d 96	FD CB d 96
	3	CB 9F	CB 98	CB 99	CB 9A	CB 9B	CB 9C	CB 9D	CB 9E	DD CB d 9E	FD CB d 9E
	4	CB A7	CB A0	CB A1	CB A2	CB A3	CB A4	CB A5	CB A6	DD CB d A6	FD CB d A6
	5	CB AF	CB A8	CB A9	CB AA	CB AB	CB AC	CB AD	CB AE	DD CB d AE	FD CB d AE
	6	CB B7	CB B0	CB B1	CB B2	CB B3	CB B4	CB B5	CB B6	DD CB d B6	FD CB d B6
	7	CB BF	CB B8	CB B9	CB BA	CB BB	CB BC	CB BD	CB BE	DD CB d BE	FD CB d BE

续表 A1-8

	位	寄存器寻址 A	B	C	D	E	H	L	寄存器间接 (HL)	变址 (IX+d)	(IY+d)
位置位(SET)	0	CB C7	CB C0	CB C1	CB C2	CB C3	CB C4	CB C5	CB C6	DD CB d C6	FD CB d C6
	1	CB CF	CB C8	CB C9	CB CA	CB CB	CB CC	CB CD	CB CE	DD CB d CE	FD CB d CE
	2	CB D7	CB D0	CB D1	CB D2	CB D3	CB D4	CB D5	CB D6	DD CB d D6	FD CB d D6
	3	CB DF	CB D8	CB D9	CB DA	CB DB	CB DC	CB DD	CB DE	DD CB d DE	FD CB d DE
	4	CB E7	CB E0	CB E1	CB E2	CB E3	CB E4	CB E5	CB E6	DD CB d E6	FD CB d E6
	5	CB EF	CB E8	CB E9	CB EA	CB EB	CB EC	CB ED	CB EE	DD CB d EE	FD CB d EE
	6	CB F7	CB F0	CB F1	CB F2	CB F3	CB F4	CB B5	CB F6	DD CB d F6	FD CB d F6
	7	CB FF	CB F8	CB F9	CB FA	CB FB	CB FC	CB FD	CB FE	DD CB d FE	FD CB d FE

表 A1-9 转移指令组

			（条件）									
			无条件	进位	无进位	零	非零	校验偶	校验奇	符号为负	符号为正	寄存器 B≠0
转移(JP)	立即扩展	nn	C3 n n	DA n n	D2 n n	CA n n	C2 n n	EA n n	E2 n n	FA n n	F2 n n	
转移(JR)	相对	PC+e	18 e−2	38 e−2	30 e−2	28 e−2	20 e−2					
转移(JP)	寄存器间接	(HL)	E9									
转移(JP)		(IX)	DD E9									
转移(JP)		(IY)	FD E9									
B减1,非零转(DJNZ)	相对	PC+e										10 e−2

表 A1-10(a) 调用和返回指令组

			条件									
			无条件	进位	无进位	零	非零	校验偶	校验奇	符号为负	符号为正	
调用(CALL)	立即扩展	nn	CD n n	DC n n	D4 n n	CC n n	C4 n n	EC n n	E4 n n	FC n n	F4 n n	
返回(RET)	寄存器间接	(SP) (SP+1)	C9	D8	D0	C8	C0	E8	E0	F8	F0	
从中断返回(RETI)	寄存器间接	(SP) (SP+1)	ED 4D									
从不可屏蔽中断返回(RETN)	寄存器间接	(SP) (SP+1)	ED 45									

表 A1-10(b) 重新启动指令组

		操作码	
调用地址	0000H	C7	'RST0'
	0008H	CF	'RST8'
	0010H	D7	'RST16'
	0018H	DF	'RST24'
	0020H	E7	'RST32'
	0028H	EF	'RST40'
	0030H	F7	'RST48'
	0038H	FF	'RST56'

表 A1-11(a) 输入指令组

输入目的地				源端口地址：立即	源端口地址：寄存器间接
				n	(c)
输入(IN)	寄存器寻址	A		DB n	ED 78
		B			ED 40
		C			ED 48
		D			ED 50
		E			ED 58
		H			ED 60
		L			ED 68
输入，HL 增 1，B 减 1 (INI)	寄存器间接	(HL)			ED A2
输入，HL 增 1，B 减 1，如 B≠0 则重复 (INIR)					ED B2
输入，HL 减 1，B 减 1 (IND)					ED AA
输入，HL 减 1，B 减 1，如 B≠0 则重复 (INDR)					ED BA

数据块输入指令

表 A1-11(b) 输出指令组

输出目的地端口地址			源：寄存器 A	B	C	D	E	H	L	源：寄存器间接 (HL)
输出(OUT)	立即	n	D3 n							
	寄存器间接	(c)	ED 79	ED 41	ED 49	ED 51	ED 59	EE 61	ED 69	
输出，HL 增 1，B 减 1(OUTI)	寄存器间接	(c)								ED A3
输出，HL 增 1，B 减 1，如 B≠0 则重复(OTIR)	寄存器间接	(c)								ED B3
输出，HL 减 1 B 减 1(OUTD)	寄存器间接	(c)								ED AB
输出，HL 减 1，B 减 1，如 B≠0 则重复(DTDR)	寄存器间接	(c)								ED BB

数据块输出指令

表 A1-12 标志操作小结

指令	D7 S	D6 Z	D5	D4 H	D3	D2 P/V	D1 N	D0 C	说明
ADD A,s;ADCA,s	↕	↕	X	↕	X	V	0	↕	8 位加或带进位加
SUB s;SBC A,s;CPs;	↕	↕	X	↕	X	V	1	↕	8 位减、带进位减和比较
AND s	↕	↕	X	1	X	P	0	0	逻辑运算
OR s;XOR s	↕	↕	X	0	X	P	0	0	
INC s	↕	↕	X	↕	X	V	0	·	8 位增量
DEC s	↕	↕	X	↕	X	V	1	·	8 位减量
ADD HL,ss	·	·	X	X	X	·	0	↕	16 加
ADC HL,ss	↕	↕	X	X	X	V	0	↕	16 位带进位加
SBC HL,ss	↕	↕	X	X	X	V	1	↕	16 位带进位减
RLA;RLCA;RRA;RRCA	·	·	X	0	X	·	0	↕	累加器循环移位
RLs;RLCs;RRs;RRCs SLAs;SRAs;SRLs	↕	↕	X	0	X	P	0	↕	存贮单元循环和移位
RLD;RRD	↕	↕	X	0	X	P	0	·	BCD 码向左和向右循环移位
DAA	↕	↕	X	↕	X	P	·	↕	累加器十进制调整
CPL	·	·	X	1	X	·	1	·	累加器变反
NEG	↕	↕	X	↕	X	V	1	↕	累加器变补
SCF	·	·	X	0	X	·	0	1	进位位置位
CCF	·	·	X	X	X	·	0	↕	进位位变反
IN r,(C)	↕	↕	X	0	X	P	0	·	输入寄存器间接寻址
INI;IND;OUTI;OUTD	X	↕	X	X	X	X	1	·	数据块输入和输出
INIR;INDR;OTIR;OTDR;	X	1	X	X	X	X	1	·	如 B≠0，则 Z=0;否则 Z=1
LDI;LDD	X	X	X	0	X	↕	0	·	数据块转移指令
LDIR;LDDR	X	X	X	0	X	0	0	·	如 BC≠0，则 P/V=1；否则 P/V=0
CPI;CPIR;CPD;CPDR	X	↕	X	X	X	↕	1	·	数据块搜索指令如 A=(HL)，则 Z=1;否则 Z=0.如 BC≠0，则 P/V=1;否则P/V=0
LD A,I;LD A,R	↕	↕	X	0	X	IFF	0	·	中断允许触发器 (IFF) 的内容复制到 P/V 标志
BIT b,s	X	↕	X	1	X	X	0	·	存贮单元 s 的 b 位状态反映到 Z 标志

注：

符号　　操　　作

CY　进位/连接标志。若操作数或结果的最高位产生进位，则 CY=1。

Z　零标志。若操作的结果为零，则 Z=1。

S　符号标志。若操作结果的最高位为 1，则 S=1。

P/V	奇偶或溢出标志。奇偶(P)和溢出(V)共用这个标志。当逻辑操作位时，P/V 反映奇偶性，若操作结果中 1 的个数为偶数，则 P/V=1；结果中 1 的个数为奇数，则 P/V=0。当算术运算时，P/V 反映溢出，若操作结果产生溢出，则 P/V=1；否则，P/V=0。
H	半进位标志。若在加法或减法时，从累加器的位 3 产生进位或借位，则 H=1。
N	加/减标志。若前一操作为减法，则 N=1。
↕	根据操作的结果来影响标志状态。
。	标志不受操作的影响。
0	标志由操作复位。
1	标志由操作置位。
X	任意
V	P/V 标志反映操作结果是溢出。
P	P/V 标志反映操作结果的奇偶性。
r	CPU 寄存器 A、B、C、D、E、H、L 中的任一个。
s	由特定指令所允许的所有寻址方式的任一个 8 位单元。
ss	由特定指令所允许的所有寻址方式的任一个 16 位单元。
ii	两个变址寄存器中的任一个。
R	刷新计数器。
n	8 位数，其范围为(0～255)。
nn	16 位数，其范围为(0～65535)。

附录 2　Z-80 指令系统的功能表

表 A2-1　8 位传送指令组(21 条)

助记符	用符号表示的操作	标志 S	Z		H		P/V	N	C	操作码 76 543 210	Hex	字节数	M 周期数	T 时态数	说明
LD r_1,r_2	$r\leftarrow r_2$	·	·	X	·	X	·	·	·	01 r s		1	1	4	r,s 寄存器
LD r,n	r←n	·	·	X	·	X	·	·	·	00 r 110		2	2	7	000 B
										← n →					001 C
LD r,(HL)	r←(HL)	·	·	X	·	X	·	·	·	01 r 110		1	2	7	010 D
LD r,(IX+d)	r←(IX+d)	·	·	X	·	X	·	·	·	11 011 101	DD	3	5	19	011 E
										01 r 110					100 H
										← d →					101 L
LD r,(IY+d)	r←(IY+d)	·	·	X	·	X	·	·	·	11 111 101	FD	3	5	19	111 A
										01 r 110					
										← d →					
LD (HL),r	(HL)←r	·	·	X	·	X	·	·	·	01 110 r		1	2	7	
LD (IX+d),r	(IX+d)←r	·	·	X	·	X	·	·	·	11 011 101	DD	3	5	19	
										01 110 r					
										← d →					
LD (IY+d),r	(IY+d)←r	·	·	X	·	X	·	·	·	11 111 101	FD	3	5	19	
										01 110 r					
										← d →					
LD (HL),n	(HL)←n	·	·	X	·	X	·	·	·	00 110 110	36	2	3	10	
										← n →					
LD (IX+d),n	(IX+d)←n	·	·	X	·	X	·	·	·	11 011 101	DD	4	5	19	
										00 110 110	36				
										← d →					
										← n →					
LD (IY+d),n	(IY+d)←n	·	·	X	·	X	·	·	·	11 111 101	FD	4	5	19	
										00 110 110	36				
										← d →					
										← n →					
LD A,(BC)	A←(BC)	·	·	X	·	X	·	·	·	00 001 010	0A	1	2	7	
LD A,(DE)	A←(DE)	·	·	X	·	X	·	·	·	00 011 010	1A	1	2	7	
LD A,(nn)	A←(nn)	·	·	X	·	X	·	·	·	00 111 010	3A	3	4	13	
										← n →					
										← n →					
LD (EC),A	(BC)←A	·	·	X	·	X	·	·	·	00 000 010	02	1	2	7	
LD (DE),A	(DE)←A	·	·	X	·	X	·	·	·	00 010 010	12	1	2	7	
LD (nn),A	(nn)←A	·	·	X	·	X	·	·	·	00 110 010	32	3	4	13	
										← n →					
										← n →					
LD A,I	A←I	↕	↕	X	0	X	IFF	0	·	11 101 101	ED	2	2	9	
										01 010 111	57				
LD A,R	A←R	↕	↕	X	0	X	IFF	0	·	11 101 101	ED	2	2	9	
										01 011 111	5F				
LD I,A	I←A	·	·	X	·	X	·	·	·	11 101 101	ED	2	2	9	
										01 000 111	47				
LD R,A	R←A	·	·	X	·	X	·	·	·	11 101 101	ED	2	2	9	
										01 001 111	4F				

注　r_1、r_2 表示 A、B、C、D、E、H、L 寄存器中的任何一个，IFF 表示允许中断触发器的内容复制到 P/V 标志内。标志记号同前。

表 A2-2 16 位传送指令组(20 条)

助记符	用符号表示的操作	标志 S	Z		H		P/V	N	C	操作码 76 543 210	Hex	字节数	周期数	T时态数	说明
LD dd, nn	dd←nn	·	·	X	·	X	·	·	·	00 dd0 001 ← n → ← n →		3	3	10	dd 寄存器对 00 BC 01 DE 10 HL 11 SP
LD IX, nn	IX←nn	·	·	X	·	X	·	·	·	11 011 101 00 100 001 ← n → ← n →	DD 21	4	4	14	
LD IY, nn	IY←nn	·	·	X	·	X	·	·	·	11 111 101 00 100 001 ← n → ← n →	FD 21	4	4	14	
LD HI, (nn)	H←(nn+1) L←(nn)	·	·	X	·	X	·	·	·	00 101 010 ← n → ← n →	2A	3	5	16	
LD dd, (nn)	dd_H←(nn+1) dd_L←(nn)	·	·	X	·	X	·	·	·	11 101 101 01 dd1 011 ← n → ← n →	ED	4	6	20	
LD IX(nn)	IX_H←(nn+1) IX_L←(nn)	·	·	X	·	X	·	·	·	11 011 101 00 101 010 ← n → ← n →	DD 2A	4	6	20	
LD IY(nn)	IY_H←(nn+1) IY_L←(nn)	·	·	X	·	X	·	·	·	11 111 101 00 101 010 ← n → ← n →	FD 2A	4	6	20	
LD (nn), HL	(nn+1)←H (nn)←L	·	·	X	·	X	·	·	·	00 100 010 ← n → ← n →	22	3	5	16	
LD (nn), dd	(nn+1)←dd_H (nn)←dd_L	·	·	X	·	X	·	·	·	11 101 101 01 dd0 011 ← n → ← n →	ED	4	6	20	
LD (nn), IX	(nn+1)←IX_H (nn)←IX_L	·	·	X	·	X	·	·	·	11 011 101 00 100 010 ← n → ← n →	DD 22	4	6	20	
LD (nn), IY	(nn+1)←IY_H (nn)←IY_L	·	·	X	·	X	·	·	·	11 111 101 00 100 010 ← n → ← n →	FD 22	4	6	20	
LD SP, HL	SP←HL	·	·	X	·	X	·	·	·	11 111 001	F9	1	1	6	
LD SP, IX	SP←IX	·	·	X	·	X	·	·	·	11 111 101 11 111 001	DD F9	2	2	10	
LD SP, IY	SP←IY	·	·	X	·	X	·	·	·	11 111 101 11 111 001	FD F9	2	2	10	qq 寄存器对 00 BC 01 DE 10 HL 11 AF
PUSH qq	(SP−2)←qq_L (SP−1)←qq_H SP←SP−2	·	·	X	·	X	·	·	·	11 qq1 101		1	3	11	
PUSH IX	(SP−2)←IX_L (SP−1)←IX_H SP←SP−2	·	·	X	·	X	·	·	·	11 011 101 11 100 101	DD E5	2	4	15	
PUSH IY	(SP−2)←IY_L (SP−1)←IY_H SP←SP−2	·	·	X	·	X	·	·	·	11 100 101 11 100 101	FD E5	2	4	15	
POP qq	qq_H←(SP+1) qq_L←(SP) SP←SP+2	·	·	X	·	X	·	·	·	11 qq0 001		1	3	10	
POP IX	IX_H←(SP+1) IX_L←(SP) SP←SP+2	·	·	X	·	X	·	·	·	11 011 101 11 100 001	DD E1	2	4	14	
POP IY	IY_H←(SP+1) IY_L←(SP) SP←SP+2	·	·	X	·	X	·	·	·	11 111 101 11 100 001	FD E1	2	4	14	

注：dd 是 BC、DE、HL 和 SP 寄存器对中的任一对；qq 是 AF、BC、DE 和 HL 寄存器中的任一对；$(PAIR)_H$、$(PAIR)_L$ 分别表示寄存器对的高八位和低八位。如 BC_H=C, AF_H=A。

标志记号同前。

表 A2-3　交换指令组和数据块传送和搜索指令组(14 条)

助记符	用符号表示的操作	标志 S	Z		H		P/V	N	C	操作码 76 543 210	Hex	字节数	M周期数	T时态数	说明
EX DE,HL	DE↔HL	·	·	X	·	X	·	·	·	11 101 011	EB	1	1	4	
EX AF,AF	AF↔AF′	·	·	X	·	X	·	·	·	00 001 000	08	1	1	4	
EXX	BC↔BC′ DE↔DE′ HL↔HL′	·	·	X	·	X	·	·	·	11 011 001	D9	1	1	4	寄存器组和辅助寄存器组交换
EX (SP),HL	H↔(SP+1) L↔(SP)	·	·	X	·	X	·	·	·	11 100 011	E3	1	5	19	
EX (SP), IX	IXH↔(SP+1) IXL↔(SP)	·	·	X	·	X	·	·	·	11 011 101 11 100 011	DD E3	2	6	23	
EX(SP),IY	IYH↔(SP+1) IYL↔(SP)	·	·	X	·	X	·	·	·	11 111 101 11 100 011	FD E3	2	6	23	
LDI	(DE)←(HL) DE←DE+1 HL←HL+1 BC←BC−1	·	·	X	0	X	① ↕	·	·	11 101 101 10 100 000	ED A0	2	4	16	(HL)送到(DE),指针增1,字节计数器(BC)减1
LDIR	(DE)←(HL) DE←DE+1 HL←HL+1 BC←BC−1 重复直到 BC=0	·	·	X	0	X	0 ①	0	·	11 101 101 10 110 000	ED B0	2 2	5 4	21 16	如 BC≠0 如 BC=0
LDD	(DE)←(HL) DE←DE−1 HL←HL−1 BC←BC−1	·	·	X	0	X	① ↕	0	·	11 101 101 10 101 000	ED A8	2	4	16	
LDDR	(DE)←(HL) DE←DE−1 HL←HL−1 BC←BC−1 重复直到 BC=0	·	·	X	0	X	0 ①	0	·	11 101 101 10 111 000	ED B8	2 2	5 4	21 16	如 BC≠0 如 BC=0
CPI	A−(HL) HL←HL+1 BC←BC−1	↕	② ↕	X	↕	X	① ↕	1	·	11 101 101 10 100 001	ED A1	2	4	16	
CPIR	A−(HL) HL←HL+1 BC←BC−1 重复直到 A=(HL)或BC=0	↕	② ↕	X	↕	X	① ↕	1	·	11 101 101 10 110 001	ED B1	2 2	5 4	21 16	如 BC≠0 和 A≠(HL) 如 BC=0 或 A=(HL)
CPD	A−(HL) HL←HL−1 BC←BC−1	↕	② ↕	X	↕	X	① ↕	1	·	11 101 101 10 101 001	ED A9	2	4	16	
CRDR	A−(HL) HL←HL−1 BC←BC−1 重复直到 A=(HL)或BC=0	↕	② ↕	X	↕	X	① ↕	1	·	11 101 101 10 111 001	ED B9	2 2	5 4	21 16	如 BC≠0 和 A≠(HL) 如 BC=0 或 A(HL)

注：① 如 BC−1=0,P/V 标志为 0;否则 P/V=1,② 如 A=(HL),Z 标志为 1,否则 Z=0。
标志记号同前。

表 A 2-4　8 位算术和逻辑指令组(17 条)

助记符	用符号表示的操作	标志 S	Z	H	P/V	N	C	操作码 76	543	210	Hex	字节数	M周期数	T时态数	说明
ADD A,r	A←A+r	↕	↕	X	↕	X	V	0	↕	10	[000]	r		1	1	4	r 寄存器 000 B 001 C 010 D 011 E 100 H 101 L 111 A
ADD A,n	A←A+n	↕	↕	X	↕	X	V	0	↕	11	[000]	110		2	2	7	
										←	n	→					
ADD A,(HL)	A←A+(HL)	↕	↕	X	↕	X	V	0	↕	10	[000]	110		1	2	7	
ADD A,(IX+d)	A←A+(IX+d)	↕	↕	X	↕	X	V	0	↕	11	011	101	DD	3	5	19	
										10	[000]	110					
										←	d	→					
ADD A,(IY+d)	A←A(IY+d)	↕	↕	X	↕	X	V	0	↕	11	111	111	FD	3	5	19	
										10	[000]	110					
										←	d	→					
ADC A,s	A←A+S+CY	↕	↕	X	↕	X	V	0	↕		[001]						同ADD指令那样，s为r、n、(HL)、(IX+d)、(IY+d)中的任意一个，用方框所标出的位代替ADD指令系列中的[000]
SUB s	A←A−S	↕	↕	X	↕	X	V	1	↕		[010]						
SBC A,s	A←A−S−CY	↕	↕	X	↕	X	V	1	↕		[011]						
AND s	A←A∧S	↕	↕	X	1	X	P	0	0		[100]						
OR s	A←A∨S	↕	↕	X	0	X	P	0	0		[110]						
XOR s	A←A⊕S	↕	↕	X	0	X	P	0	0		[101]						
CP s	A−S	↕	↕	X	↕	X	V	1	↕		[111]						
INC r	r←r+1	↕	↕	X	↕	X	V	0	·	00	r	[100]		1	1	4	象INC指令那样，s是r、(HL)、(IX+d)、(IY+d)中的任意一个。DEC指令的格式和状态同INC。在操作码中用101代替100
INC (HL)	(HL)←(HL)+1	↕	↕	X	↕	X	V	0	·	00	110	[100]		1	3	11	
INC (IX+d)	(IX+d)←	↕	↕	X	↕	X	V	0	·	11	011	101	DD	3	6	23	
	(IX+d)+1									00	110	[100]					
										←	d	→					
INC (IY+d)	(IY+d)←	↕	↕	X	↕	X	V	0	·	11	111	101	FD	3	6	23	
	(IY+d)+1									00	110	[100]					
										←	d	→					
DEC s	S←S−1	↕	↕	X	↕	X	V	1	·			[101]					

注：P/V 标志一列中 V 的符号指明，P/V 标志包含着运算结果的溢出情况。类似地，P 符号指明奇偶性，V=1表示溢出，V=0 表示没有溢出；P=1 表示结果中 1 的个数为偶数，P=0 表示结果中 1 的个数为奇数。
标志记号同前。

表 A 2-5　16 位算术指令组(11 条)

助记符	用符号表示的操作	标志 S	Z		H		P/V	N	C	操作码 76	543	210	Hex	字节数	M 周期数	T 时态数	说明
ADD HL, ss	HL←HL+ss	·	·	X	X	X	·	0	↕	0 0	ss 1	001		1	3	11	ss 寄存器
																	00 BC
ADC HL, ss	HL←HL+ss+CY	↕	↕	X	X	X	V	0	↕	1 1	101	101	ED	2	4	15	01 DE
										0 1	ss 1	010					10 HL
																	11 SP
SBC HL, ss	HL←HL−ss−CY	↕	↕	X	X	X	V	1	↕	1 1	101	101	ED	2	4	15	
										0 1	ss 0	010					
ADD IX, pp	IX←IX+pp	·	·	X	X	X	·	0	↕	1 1	011	101	DD	2	4	15	pp 寄存器
										0 0	pp 1	001					00 BC
																	01 DE
																	10 IX
																	11 SP
ADD IY, rr	IY←IY+rr	·	·	X	X	X	·	0	↕	1 1	111	101	FD	2	4	15	rr 寄存器
										0 0	rr 1	001					00 BC
																	01 DE
																	10 IY
																	11 SP
INC ss	ss←ss+1	·	·	X	·	X	·	·	·	0 0	ss 0	011		1	1	6	
INC IX	IX←IX+1	·	·	X	·	X	·	·	·	1 1	011	101	DD	2	2	10	
										0 0	100	011	23				
INC IY	IY←IY+1	·	·	X	·	X	·	·	·	1 1	111	101	FD	2	2	10	
										0 0	100	011	23				
DEC ss	ss←ss−1	·	·	X	·	X	·	·	·	0 0	ss 1	011		1	1	6	
DEC IX	IX←IX−1	·	·	X	·	X	·	·	·	1 1	011	101	DD	2	2	10	
										0 0	101	011	2B				
DEC IY	IY←IY−1	·	·	X	·	X	·	·	·	1 1	111	101	FD	2	2	10	
										0 0	101	011	2B				

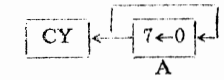
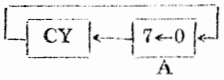
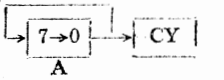
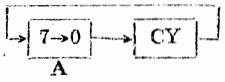
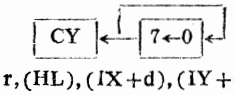
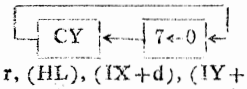
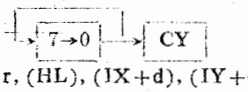
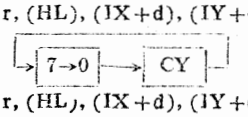
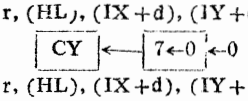
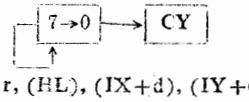
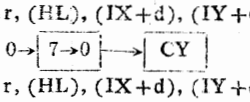
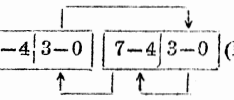
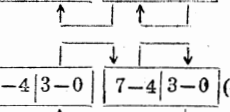
注：ss 是寄存器对 BC、DE、HL、SP 中的任意一个。
pp 是寄存器对 BC、DE、HL、SP 中的任意一个。
rr 是寄存器对 BC、DE、HL、SP 中的任意一个。
标志记号同前。

表 A 2-6　通用算术和 CPU 控制指令(12 条)

助记符	用符号表示的操作	标志								操作码				字节数	M周期数	T时态数	说明
		S	Z		H		P/V	N	C	7 6	5 4 3	2 1 0	Hex				
DAA	把累加器内容调整为二-十进制数	↕	↕	X	↕	X	P	·	↕	0 0	1 0 0	1 1 1	27	1	1	4	累加器+进制调整
CPL	$A \leftarrow \overline{A}$	·	·	X	1	X	·	1	·	0 0	1 0 1	1 1 1	2F	1	1	4	累加器变反(1 的补码)
NEG	$A \leftarrow \overline{A}+1$	↕	↕	X	↕	X	V	1	↕	1 1	1 0 1	1 0 1	ED	2	2	8	累加器变补(2 的补码)
										0 1	0 0 0	1 0 0	44				
CCF	$CY \leftarrow \overline{CY}$	·	·	X	X	X	·	0	↕	0 0	1 1 1	1 1 1	3F	1	1	4	进位标志变反
SCF	CY←1	·	·	X	0	X	·	0	1	0 0	1 1 0	1 1 1	37	1	1	4	进位标志置 1
NOP	空操作	·	·	X	·	X	·	·	·	0 0	0 0 0	0 0 0	00	1	1	4	
HALT	CPU 暂停	·	·	X	·	X	·	·	·	0 1	1 1 0	1 1 0	76	1	1	4	
DI*	IFF←0	·	·	X	·	X	·	·	·	1 1	1 1 0	0 1 1	F3	1	1	4	
EI*	IFF←1	·	·	X	·	X	·	·	·	1 1	1 1 1	0 1 1	FB	1	1	4	
IM0	置中断方式 0	·	·	X	·	X	·	·	·	1 1	1 0 1	1 0 1	ED	2	2	8	
										0 1	0 0 0	1 1 0	46				
IM1	置中断方式 1	·	·	X	·	X	·	·	·	1 1	1 0 1	1 0 1	ED	2	2	8	
										0 1	0 1 0	1 1 0	56				
IM2	置中断方式 2	·	·	X	·	X	·	·	·	1 1	1 0 1	1 0 1	ED	2	2	8	
										0 1	0 1 1	1 1 0	5E				

注　IFF 表示允许中断触发器。
CY 表示进位位触发器。
标志记号同前。

表 A2-7　循环和移位指令组（16 条）

助记符	用符号表示的操作	标志 S	Z		H		P/V	N	C	操作码 76 543 210	Hex	字节数	M周期数	T时态数	说明
RLCA		.	.	X	0	X	.	0	↕	00 000 111	07	1	1	4	累加器循环左移
RLA		.	.	X	0	X	.	0	↕	00 010 111	17	1	1	4	累加器带进位循环左移
RRCA		.	.	X	0	X	.	0	↕	00 001 111	0F	1	1	4	累加器循环右移
RRA		.	.	X	0	X	.	0	↕	00 011 111	1F	1	1	4	累加器带进位循环右移
RLC r		↕	↕	X	0	X	P	0	↕	11 001 011 00 000 r	CB	2	2	8	寄存器 r 循环左移
RLC (HL)		↕	↕	X	0	X	P	0	↕	11 001 011 00 000 110	CB	2	4	15	r 寄存器 000 B 001 C
RLC (IX+d)	r,(HL),(IX+d),(IY+d)	↕	↕	X	0	X	P	0	↕	11 011 101 11 001 011 ← d → 00 000 110	DD CB	4	6	23	010 D 011 E 100 H 101 L 111 A
RLC (IY+d)		↕	↕	X	0	X	P	0	↕	11 111 101 11 001 011 ← d → 00 000 110	FD CB	4	6	23	指令格式和时态同 RLCs 为形成新的操作码，用新操作码代替 RLC中的 000
RL s	 S=r,(HL),(IX+d),(IY+d)	↕	↕	X	0	X	P	0	↕	010					
RRC s	 S=r,(HL),(IX+d),(IY+d)	↕	↕	X	0	X	P	0	↕	001					
RR s	 S=r,(HL),(IX+d),(IY+d)	↕	↕	X	0	X	P	0	↕	011					
SLA s	 S=r,(HL),(IX+d),(IY+d)	↕	↕	X	0	X	P	0	↕	100					
SRA s	 S=r,(HL),(IX+d),(IY+d)	↕	↕	X	0	X	P	0	↕	101					
SRL s	 S=r,(HL),(IX+d),(IY+d)	↕	↕	X	0	X	P	0	↕	111					
RLD		↕	↕	X	0	X	P	0	·	11 101 101 01 101 111	ED 6F	2	5	18	在累加器和存贮单元(HL)间进行十进位循环左移和右移
RRD		↕	↕	X	0	X	P	0	·	11 101 101 01 100 111	ED 67	2	5	18	累加器高4位内容不受影响

注：标志记号同前。

表 A 2-8　位操作(置位、复位和测试)指令组(9 条)

助记符	用符号表示的操作	标志 S	Z	H	P/V	N	C	操作码 76 543 210	Hex	字节数	M 周期数	T 时态数	说明
BIT b,r	$Z\leftarrow\overline{r_b}$	X	↕	X	1	X	X	0	·	11 001 011 01 b r	CB	2	2	8	
BIT b,(HL)	$Z\leftarrow\overline{(HL)_b}$	X	↕	X	1	X	X	0	·	11 001 011 01 b 110	CB	2	3	12	
BIT b,(IX+d)	$Z\leftarrow\overline{(IX+d)_b}$	X	↕	X	1	X	X	0	·	11 011 101 11 001 011 ← d → 01 b 110	DD CB	4	5	20	
BIT b,(IY+d)	$Z\leftarrow\overline{(IY+d)_b}$	X	↕	X	1	X	X	0	·	11 111 101 11 001 011 ← d → 01 b 110	FD CB	4	5	20	
SET b,r	$r_b\leftarrow 1$	·	·	X	·	X	·	·	·	11 001 011 [11] b r	CB	2	2	8	
SET b,(HL)	$(HL)_b\leftarrow 1$	·	·	X	·	X	·	·	·	11 001 011 [11] b 110	CB	2	4	15	
SET b,(IX+d)	$(IX+d)_b\leftarrow 1$	·	·	X	·	X	·	·	·	11 011 101 11 001 011 ← d → [11] b 110	DD CB	4	6	23	
SET b,(IY+d)	$(IY+d)_b\leftarrow 1$	·	·	X	·	X	·	·	·	11 111 101 11 001 011 ← d ← [11] b 110	FD CB	4	6	23	
RES b,s	$S_b\leftarrow 0$ S≡r,(HL), (IX+d), (IY+d)	·	·	X	·	X	·	·	·	[10]					为形成新的操作码，用[10]代替置位指令 SET b, s 中的[11]标志和时态同 SET 指令

r	寄存器
000	B
001	C
010	D
011	E
100	H
101	L
111	A

b	测试位
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

注：记号 S_b 指明单元 S 的位 b(0 到 7)。
标志记号同前。

表 A2-9 转移指令组(11条)

助记符	用符号表示的操作	标志 S	Z	H	P/V	N	C	操作码 76 543 210	Hex	字节数	M周期数	T时态数	说明
JP nn	PC←nn	·	·	X	X	·	·	·	11 000 011	C3	3	3	10	
									← n →					cc 条件
									← n →					
JP cc、nn	若条件cc满足，则	·	·	X	X	·	·	·	11 cc 010		3	3	10	000 NZ非零
	PC←nn，否则继续								← n →					001 Z零
									← n →					010 NC无进位
														011 C进位
														100 PO奇偶奇
														101 PE奇偶偶
														110 P正号
JR e	PC←PC+e	·	·	X	X	·	·	·	00 011 000	18	2	3	12	111 M负号
									← e−2 →					
JR C、e	若C=1，PC←PC+e	·	·	X	X	·	·	·	00 111 000	38	2	3	12	若条件满足
	若C=0，则继续								← e−2 →		2	2	7	若条件不满足
JR NC, e	若C=0，PC←PC+e	·	·	X	X	·	·	·	00 110 000	30	2	3	12	若条件满足
	若C=1，则继续								← e−2 →		2	2	7	若条件不满足
JR Z, e	若Z=1，PC←PC+e	·	·	X	X	·	·	·	00 101 000	28	2	3	12	若条件满足
	若Z=0，则继续	·	·	X	X	·	·	·	← e−2 →		2	2	7	若条件不满足
JR NZ, e	若Z=0，PC←PC+e	·	·	X	X	·	·	·	00 100 000	20	2	3	12	若条件满足
	若Z=1，则继续								← e−2 →		2	2	7	若条件不满足
JP (HL)	PC←HL	·	·	X	X	·	·	·	11 101 001	E9	1	1	4	
JP (IX)	PC←IX	·	·	X	X	·	·	·	11 011 101	DD	2	2	8	
									11 101 001	E9				
JP (IY)	PC←IY	·	·	X	X	·	·	·	11 111 101	FD	2	2	8	
									11 101 001	E9				
DJNZ, e	B←B−1	·	·	X	X	·	·	·	00 010 000	10	2	2	8	若B=0
	若B=0，则继续								← e−2 →					
	若B≠0										2	2	13	若B≠0
	PC←PC+e													

注：e表示相对寻址方式中地址的位移量。e为带符号的对2补码数，其范围为(−126, +129)。操作码中的e−2提供有效地址PC+e，因为在加e之前PC已经加2。
标志记号同前。

表 A2-10 调用和返回指令组（7 条）

助记符	用符号表示的操作	标志 S	Z		H		P/V	N	C	操作码 76	543	210	Hex	字节数	M周期数	T时态数	说明
CALL nn	$(SP-1)\leftarrow PC_H$	·	·	X	·	X	·	·	·	11	001	101	CD	3	5	17	
	$(SP-2)\leftarrow PC_L$									←	n	→					
	$PC\leftarrow nn$									←	n	→					
CALL cc, nn	若条件 cc 是假，则继续；否则，同 CALL nn	·	·	X	·	X	·	·	·	11	cc	100		3	3	10	若 cc 是假
										←	n	→		3	5	17	若 cc 是真
										←	n	→					
RET	$PC_L\leftarrow(SP)$	·	·	X	·	X	·	·	·	11	001	001	C9	1	3	10	
	$PC_H\leftarrow(SP+1)$																
RET cc	若条件 cc 是假，则继续；否则，同 RET	·	·	X	·	X	·	·	·	11	cc	000		1	1	5	若 cc 是假
														1	3	11	若 cc 是真
RETI	从中断返回	·	·	X	·	X	·	·	·	11	101	101	ED	2	4	14	
										01	001	101	4D				
RETN[注]	从不可屏蔽中断返回	·	·	X	·	X	·	·	·	11	101	101	ED	2	4	14	
										01	000	101	45				
RST p	$(SP-1)\leftarrow PC_H$	·	·	X	·	X	·	·	·	11	t	111		1	3	11	
	$(SP-2)\leftarrow PC_L$																
	$PC_H\leftarrow 0$																
	$PC_L\leftarrow p$																

cc	条件
000	NZ 非零
001	Z 零
010	NC 无进位
011	C 进位
100	PO(奇偶)奇
101	PE(奇偶)偶
110	P 正号
111	M 负号

t	p
000	00H
001	08H
010	10H
011	18H
100	20H
101	28H
110	30H
111	38H

注：RETN 指令将 IFF_2 传送到 IFF_1($IFF_2\rightarrow IFF_1$)。

标志记号同前。

表 A2-11 输入和输出指令组(12 条)

助记符	用符号表示的操作	标志 S	Z	H	P/V	N	C	操作码 76 543 210	Hex	字节数	M周期数	T时态数	说明
IN A, (n)	A←(n)	·	·	X	·	X	·	·	11 011 011	DB	2	3	11	n 至 A_0～A_7
								← n →					累加器至 A_8～A_{15}
IN r, (c)	r←(c)	↕	↕	X	0	X	P	0	·	11 101 101	ED	2	3	12	C 至 A_0～A_7
	若 r=110, 仅标志受影响								01 r 000					B 至 A_8～A_{15}
			①										
INI	(HL)←(C)	X	↕	X	X	X	X	1	·	11 101 101	ED	2	4	16	C 至 A_0～A_7
	B←B−1								10 100 010	A2				B 至 A_8～A_{15}
	HL←HL+1												
INIR	(HL)←(c)	X	1	X	X	X	X	1	·	11 101 101	ED	2	5	21	C 至 A_0～A_7
	B←B−1								10 110 010	B2		(若 B≠0)		B 至 A_8～A_{15}
	HL←HL+1										2	4	16	
	一直重复到 B=0		①									(若 B=0)		
IND	(HL)←(C)	X	↕	X	X	X	X	1	·	11 101 101	ED	2	4	16	C 至 A_0～A_7
	B←B−1								10 101 010	AA				B 至 A_8～A_{15}
	HL←HL−1													
INDR	(HL)←(C)	X	1	X	X	X	X	1	·	11 101 101	ED	2	5	21	C 至 A_0～A_7
	B←B−1								10 111 010	BA		(若 B≠0)		B 至 A_8～A_{15}
	HL←HL−1										2	4	16	
	一直重复到 B=0											(若 B=0)		
OUT(n), A	(n)←A	·	·	X	·	X	·	·	·	11 010 011	D3	2	3	11	n 至 A_0～A_7
									← n →					累加器至 A_8～A_{15}
OUT(c), r	(C)←r	·	·	X	·	X	·	·	·	11 101 101	ED	2	3	12	C 至 A_0～A_7
									01 r 001					B 至 A_8～A_{15}
			①											
OUTI	(C)←(HL)	X	↕	X	X	X	X	1	·	11 101 101	ED	2	4	16	C 至 A_0～A_7
	B←B−1								10 100 011	A3				B 至 A_8～A_{15}
	HL←HL+1													
OTIR	(C)←(HL)	X	1	X	X	X	X	1	·	11 101 101	ED	2	5	21	C 至 A_0～A_7
	B←B−1								10 110 011	B3		(若 B≠0)		B 至 A_8～A_{15}
	HL←HL+1										2	4	16	
	一直重复到 B=0											(若 B=0)		
			①											
OUTD	(C)←(HL)	X	↕	X	X	X	X	1	·	11 101 101	ED	2	4	16	C 至 A_0～A_7
	B←B−1								10 101 011	AB				B 至 A_8～A_{15}
	HL←HL−1													
OTDR	(C)←(HL)	X	1	X	X	X	X	1	·	11 101 101	ED	2	5	21	C 至 A_0～A_7
	B←B−1								10 111 011	BB		(若 B≠0)		B 至 A_8～A_{15}
	HL←HL−1										2	4	16	
	一直重复到 B=0											(若 B=0)		

注：①若 B−1 的结果为 0，Z 标志置 1；否则置 0。
标志记号同前。

附录 3　Intel 8080A/8085A 十六进制指令码

表 A3-1　按指令功能分类

MOVE

MOV	A,A	7F
	A,B	78
	A,C	79
	A,D	7A
	A,E	7B
	A,H	7C
	A,L	7D
	A,M	7E
MOV	B,A	47
	B,B	40
	B,C	41
	B,D	42
	B,E	43
	B,H	44
	B,L	45
	B,M	46
MOV	C,A	4F
	C,B	48
	C,C	49
	C,D	4A
	C,E	4B
	C,H	4C
	C,L	4D
	C,M	4E
MOV	D,A	57
	D,B	50
	D,C	51
	D,D	52
	D,E	53
	D,H	54
	D,L	55
	D,M	56
MOV	E,A	5F
	E,B	58
	E,C	59
	E,D	5A
	E,E	5B
	E,H	5C
	E,L	5D
	E,M	5E
MOV	H,A	67
	H,B	60
	H,C	61
	H,D	62
	H,E	63
	H,H	64
	H,L	65
	H,M	66

MOVE (cont)

MOV	L,A	6F
	L,B	68
	L,C	69
	L,D	6A
	L,E	6B
	L,H	6C
	L,L	6D
	L,M	6E
MOV	M,A	77
	M,B	70
	M,C	71
	M,D	72
	M,E	73
	M,H	74
	M,L	75

XCHG　EB

MOVE Immediate

MV1	A, byte	3E
	B, byte	06
	C, byte	0E
	D, byte	16
	E, byte	1E
	H, byte	26
	L, byte	2E
	M, byte	36

LOAD Immediate

LX1	B, dble	01
	D, dble	11
	H, dble	21
	SP, dble	31

Load/Store

LDAY	B	0A
LDAX	D	1A
LHLD	adr	2A
LDA	adr	3A
STAX	B	02
STAX	D	12
SHLD	adr	22
STA	adr	32

Add①

ADD	A	87
	B	80
	C	81
	D	82
	E	83
	H	84
	L	85
	M	86
ADC	A	8F
	B	88
	C	89
	D	8A
	E	8B
	H	8C
	L	8D
	M	8E

Sobtract①

SUB	A	97
	B	90
	C	91
	D	92
	E	93
	H	94
	L	95
	M	96
SBB	A	9F
	B	98
	C	99
	D	9A
	E	9B
	H	9C
	L	9D
	M	9E

Double Add③

DAD	B	09
	D	19
	H	29
	SP	39

Increment

②INR	A	3C
	B	04
	C	0C
	D	14
	E	1C
	H	24
	L	2C
	M	34
INX	B	03
	D	13
	H	23
	SP	33

Decrement

②DCR	A	3D
	B	05
	C	0D
	D	15
	E	1D
	H	25
	L	2D
	M	35
DCX	B	0B
	D	1B
	H	2B
	SP	3B

Specials

DAA①		27
CMA		2F
STC		37
CMC		3F

Rotate③

RLC		07
RRC		0F
RAL		17
RAR		1F

Logical①

ANA	A	A7
	B	A0
	C	A1
	D	A2
	E	A3
	H	A4
	L	A5
	M	A6
XRA	A	AF
	B	A8
	C	A9
	D	AA
	E	AB
	H	AC
	L	AD
	M	AE
ORA	A	B7
	B	B0
	C	B1
	D	B2
	E	B3
	H	B4
	L	B5
	M	B6
CMP	A	BF
	B	B8
	C	B9
	D	BA
	E	BB
	H	BC
	L	BD
	M	BE

Arith & Logical① Immediate

ADI	byte	C6
ACI	byte	CE
SUI	byte	D6
SBI	byte	DE
ANI	byte	E6
XRI	byte	EE
ORI	byte	F6
CPI	byte	FE

JUMP

JMP	adr	C3
JNZ	adr	C2
JZ	adr	CA
JNC	adr	D2
JC	adr	DA
JPo	adr	E2
JPE	adr	EA
JP	adr	F2
JM	adr	FA
PCHL		E9

CALL

CALL	adr	CD
CNZ	adr	CA
CZ	adr	CC
CNC	adr	DA
CC	adr	DC
CPo	adr	EA
CPE	adr	EC
CP	adr	FA
CM	adr	FC

Return

RET	C9
RNZ	C0
RZ	C8
RNC	D0
RC	D8
RPO	E0
RPE	E8
RP	F0
RM	F8

Restart

RST	0	C7
	1	CF
	2	D7
	3	DF
	4	E7
	5	EF
	6	F7
	7	FF

Stack Ops

PUSH	B	C5
	D	D5
	H	E5
	PSW	F5
POP	B	C1
	D	D1
	H	E1
	PSW①	F1
XTHL		E3
SPHL		F9

Input/Output

OUT	byte	D3
IN	byte	DB

Control

DI	F3
EI	FB
NOP	00
HLT	76

New Instructions (8085A Only)

RIM	20
SIM	30

注：① 全部标志 (CY、Z、S、P、AC) 均受影响。

② 除 CY 外，全部标志受影响(但 INX 与 DCX 不影响标志位)。

③ 仅 CY 受影响。

④ byte 表示 8 位数据；dble 表示 16 位数据；adr 表示 16 位地址。

表 A 3-2 按指令操作码顺序编排（8080A/8085A）

00	NOP	2B	DCX H	56	MOV D,M	81	ADD C	AC	XRA H	D7	RST 2				
01	LXI B, dble	2C	INR L	57	MOV D,A	82	ADD D	AD	XRA L	D8	RC				
02	STAX B	2D	DCR L	58	MOV E,B	83	ADD E	AE	XRA M	D9	...				
03	INX B	2E	MVI L, byte	59	MOV E,C	84	ADD H	AF	XRA A	DA	JC adr				
04	INR B	2F	CMA	5A	MOV E,D	85	ADD L	B0	ORA B	DB	IN byte				
05	DCR B	30	SIM	5B	MOV E,E	86	ADD M	B1	ORA C	DC	CC adr				
06	MVI B, byte	31	LXI SP, dble	5C	MOV E,H	87	ADD A	B2	ORA D	DD	...				
07	RLC	32	STA adr	5D	MOV E,L	88	ADC B	B3	ORA E	DE	SBI byte				
08	...	33	INX SP	5E	MOV E,M	89	ADC C	B4	ORA H	DF	RST 3				
09	DAD B	34	INR M	5F	MOV E,A	8A	ADC D	B5	ORA L	E0	RPO				
0A	LDAX B	35	DCR M	60	MOV H,B	8B	ADC E	B6	ORA M	E1	POP H				
0B	DCX B	36	MVI M, byte	61	MOV H,C	8C	ADC H	B7	ORA A	E2	JPO adr				
0C	INR C	37	STC	62	MOV H,D	8D	ADC L	B8	CMP B	E3	XTHL				
0D	DCR C	38	...	63	MOV H,E	8E	ADC M	B9	CMP C	E4	CPO adr				
0E	MVI C, byte	39	DAD SP	64	MOV H,H	8F	ADC A	BA	CMP D	E5	PUSH H				
0F	RRC	3A	LDA adr	65	MOV H,L	90	SUB B	BB	CMP E	E6	ANI byte				
10	...	3B	DCX SP	66	MOV H,M	91	SUB C	BC	CMP H	E7	RST 4				
11	LXI D, dble	3C	INR A	67	MOV H,A	92	SUB D	BD	CMP L	E8	RPE				
12	STAX D	3D	DCR A	68	MOV L,B	93	SUB E	BE	CMP M	E9	PCHL				
13	INX D	3E	MVI A, byte	69	MOV L,C	94	SUB H	BF	CMP A	EA	JPE adr				
14	INR D	3F	CMC	6A	MOV L,D	95	SUB L	C0	RNZ	EB	XCHG				
15	DCR D	40	MOV B,B	6B	MOV L,E	96	SUB M	C1	POP B	EC	CPE adr				
16	MVI D, byte	41	MOV B,C	6C	MOV L,H	97	SUB A	C2	JNZ adr	ED	...				
17	RAL	42	MOV B,D	6D	MOV L,L	98	SBB B	C3	JMP adr	EE	XRI byte				
18	...	43	MOV B,E	6E	MOV L,M	99	SBB C	C4	CNZ adr	EF	RST 5				
19	DAD D	44	MOV B,H	6F	MOV L,A	9A	SBB D	C5	PUSH B	F0	RP				
1A	LDAX D	45	MOV B,L	70	MOV M,B	9B	SBB E	C6	ADI byte	F1	POP PSW				
1B	DCX D	46	MOV B,M	71	MOV M,C	9C	SBB H	C7	RST 0	F2	JP adr				
1C	INR E	47	MOV B,A	72	MOV M,D	9D	SBB L	C8	RZ	F3	DI				
1D	DCR E	48	MOV C,B	73	MOV M,E	9E	SBB M	C9	RET	F4	CP adr				
1E	MVI E, byte	49	MOV C,C	74	MOV M,H	9F	SBB A	CA	JZ adr	F5	PUSH PSW				
1F	RAR	4A	MOV C,D	75	MOV M,L	A0	ANA B	CB	...	F6	ORI byte				
20	RIM	4B	MOV C,E	76	HLT	A1	ANA C	CC	CZ adr	F7	RST 6				
21	LXI H, dble	4C	MOV C,H	77	MOV M,A	A2	ANA D	CD	CALL adr	F8	RM				
22	SHLD adr	4D	MOV C,L	78	MOV A,B	A3	ANA E	CE	ACI byte	F9	SPHL				
23	INX H	4E	MOV C,M	79	MOV A,C	A4	ANA H	CF	RST 1	FA	JM adr				
24	INR H	4F	MOV C,A	7A	MOV A,D	A5	ANA L	D0	RNC	FB	EI				
25	DCR H	50	MOV D,B	7B	MOV A,E	A6	ANA M	D1	POP D	FC	CM adr				
26	MVI H, byte	51	MOV D,C	7C	MOV A,H	A7	ANA A	D2	JNC adr	FD	...				
27	DAA	52	MOV D,D	7D	MOV A,L	A8	XRA B	D3	OUT byte	FE	CPI byte				
28	...	53	MOV D,E	7E	MOV A,M	A9	XRA C	D4	CNC adr	FF	RST 7				
29	DAD H	54	MOV D,H	7F	MOV A,A	AA	XRA D	D5	PUSH D						
2A	LHLD adr	55	MOV D,L	80	ADD B	AB	XRA E	D6	SUI byte						

助记符	操作码		M1					M2		
	$D_7D_6D_5D_4$	$D_3D_2D_1D_0$	T1	T2	T3	T4	T5	T1	T2	T3
MOV r1, r2	0 1 D D	D S S S	PCOUT STATUS	PC←PC +1	INST→ TMP/IR	SSS→TMP	TMP→DDD			
MOV r, M	0 1 D D	D 1 1 0				×		HL OUT STATUS		DATA→DDD
MOV M, r	0 1 1 1	0 S S S				SSS→TMP		HL OUT STATUS		TMP→DATA BUS
SPHL	1 1 1 1	1 0 0 1				HL ⟶ SP				
MVI r, data	0 0 D D	D 1 1 0				×		PC OUT STATUS	PC←PC+1	B2→DDD
MVI M, data	0 0 1 1	0 1 1 0				×			PC←PC+1	B2→TMP
LXI rp, data	0 0 R P	0 0 0 1				×			PC←PC+1	B2→Z
LDA addr	0 0 1 1	1 0 1 0				×			PC←PC+1	B2→Z
STA addr	0 0 1 1	0 0 1 0				×			PC←PC+1	B2→Z
LHLD addr	0 0 1 0	1 0 1 0				×			PC←PC+1	B2→Z
SHLD addr	0 0 1 0	0 0 1 0				×		PC OUT STATUS	PC←PC+1	B2→Z
LDAX rp	0 0 R P	1 0 1 0				×		rp OUT STATUS		DATA→A
STAX rp	0 0 R P	0 0 1 0				×		rp OUT STATUS		A→DATA BUS
XCHG	1 1 1 0	1 0 1 1				×			HL⟷DE	
ADD r	1 0 0 0	0 S S S				SSS→TMP A→ACT			ACT+TMP→A	
ADD M	1 0 0 0	0 1 1 0				A→ACT		HL OUT STATUS		DATA→TMP
ADI data	1 1 0 0	0 1 1 0				A→ACT		PC OUT STATUS	PC←PC+1	B2→TMP
ADC r	1 0 0 0	1 S S S				SSS→TMP A→ACT			ACT+TMP+CY→A	
ADC M	1 0 0 0	1 1 1 0				A→ACT		HL OUT STATUS		DATA→TMP
ACI data	1 1 0 0	1 1 1 0				A→ACT		PC OUT STATUS	PC←PC+1	B2→TMP
SUB r	1 0 0 1	0 S S S				SSS→TMP A→ACT			ACT−TMP→A	
SUB M	1 0 0 1	0 1 1 0				A→ACT		HL OUT LTATUS		DATA→TMP
SUI data	1 1 0 1	0 1 1 0				A→ACT		PC OUT STATUS	PC←PC+1	B2→TMP
SBB r	1 0 0 1	1 S S S				SSS→TMP A→ACT			ACT−TMP−CY→A	
SBB M	1 0 0 1	1 1 1 0				A→ACT		HL OUT STATUS		DATA→TMP
SBI data	1 1 0 1	1 1 1 0				A→ACT		PC OUT STATUS	PC←PC+1	B2→TMP
INR r	0 0 D D	0 1 0 0				DDD→TMP TMP+1→ALU	ALU→DDD			
INR M	0 0 1 1	0 1 0 0				×		HL OUT STATUS		DATA→TMP TMP+1→ALU
DCR r	0 0 D D	0 1 0 1				DDD→TMP TMP−1→ALU	ALU→DDD			
DCR M	0 0 1 1	0 1 0 1				×		HL OUT STATUS		DATA→TMP TMP−1→ALU
INX rp	0 0 R P	0 0 1 1				RP+1 ⟶ RP				
DCX rp	0 0 R P	1 0 1 1				RP−1 ⟶ RP				
DAD rp	0 0 R P	1 0 0 1				×		r1→ACT	L→TMP ACT+TMP→ALU	ALU→L.CY
DAA	0 0 1 0	0 1 1 1				60→ACT A→TMP			DAA→A FLAGS	
ANA r	1 0 1 0	0 S S S				SSS→TMP A→ACT			ACT+TMP→A	
ANA M	1 0 1 0	0 1 1 0	PC OUT STATUS	PC→PC +1	INST→ TMP/IR	A→ACT		HL OUT STATUS		DATA→TMP

指令各时态的功能表

M3			M4			M5				
T1	T2	T3	T1	T2	T3	T1	T2	T3	T4	T5
HL OUT STATUS		TMP→ DATA BUS								
PC OUT STATUS	PC←PC+1	B3→rh								
	PC←PC+1	B3→W	WZ OUT STATUS	DATA ⟶ A						
	PC←PC+1	B3→W	WZ OUT STATUS	A ⟶	DATA BUS					
	PC←PC+1	B3→W	WZ OUT STATUS	DATA ⟶ L WZ=WZ+1		WZ OUT STATUS	DATA→H			
PC OUT STATUS	PC←PC+1	B3→W	WZ OUT STATUS	L ⟶ WZ=WZ+1	DATA BUS	WZ OUT STATUS	H⟶	DATA BUS		
	ACT+TMP→A									
	ACT+TMP→A									
	ACT+TMP+CY→A									
	ACT+TMP+CY→A									
	ACT−TMP→A									
	ACT−TMP→A									
	ACT−TMP−CY→A									
	ACT−TMP−CY→A									
HL OUT STATUS		ALU→ DATA BUS								
HL OUT STATUS		ALU→ DATA BUS								
(rh)→ACT	H→TMP ACT+TMP+CY→ALU	ALU→H, CY								
	ACT+TMP→A									

助记符	操作码		M1					M2		
	$D_7D_6D_5D_4$	$D_3D_2D_1D_0$	T1	T2	T3	T4	T5	T1	T2	T3
ANI data	1 1 1 0	0 1 1 0	PC OUT STATUS	PC=PC +1	INST→ TMP/IR	A→ACT		PC OUT STATUS	PC=PC+1	B2→TMP
XRA r	1 0 1 0	1 S S S				A→ACT SSS→TMP			ACT+TPM→A	
XRA M	1 0 1 0	1 1 1 0				A→ACT		HL OUT STATUS		DATA→TMP
XRI data	1 1 1 0	1 1 1 0				A→ACT		PC OUT STATUS	PC←PC+1	B2→TMP
ORA r	1 0 1 1	0 S S S				A→ACT SSS→TMP			ACT+TMP→A	
ORA M	1 0 1 1	0 1 1 0				A→ACT		HL OUT STATUS		DATA→TMP
ORI data	1 1 1 1	0 1 1 0				A→ACT		PC OUT STATUS	PC←PC+1	B2→TMP
CMP r	1 0 1 1	1 S S S				A→ACT SSS→TMP			ACT−TMP, FLAGS	
CMP M	1 0 1 1	1 1 1 0				A→ACT		HL OUT STATUS		DATA→TMP
CPI data	1 1 1 1	1 1 1 0				A→ACT		PC OUT STATUS	PC←PC+1	B2→TMP
RLC	0 0 0 0	0 1 1 1				A→ALU ROTATE			ALU→A, CY	
RRC	0 0 0 0	1 1 1 1				A→ALU ROTATE			ALU→A, CY	
RAL	0 0 0 1	0 1 1 1				A, CY→ALU ROTATE			ALU→A, CY	
RAR	0 0 0 1	1 1 1 1				A, CY→ALU ROTATE			ALU→A, CY	
CMA	0 0 1 0	1 1 1 1				A→ALU COMPLEMENT			ALU→A	
CMC	0 0 1 1	1 1 1 1				CY→ALU COMPLEMENT			ALU→CY	
STC	0 0 1 1	0 1 1 1				1→ALU			ALU→CY	
JMP addr	1 1 0 0	0 0 1 1				×		PC OUT STATUS	PC←PC+1	B2→Z
Jcond addr	1 1 C C	C 0 1 0				JUDGE CONDITION		PC OUT STATUS	PC←PC+1	B2→Z
CALL addr	1 1 0 0	1 1 0 1				SP←SP−1		PC OUT STATUS	PC←PC+1	B2→Z
Ccond addr	1 1 C C	C 1 0 0				JUDGE CONDITION IF TRUE, SP=SP−1		PC OUT STATUS	PC←PC+1	B2→Z
RET	1 1 0 0	1 0 0 1				×		SP OUT STATUS	SP←SP+1	DATA→PCL
Rcond addr	1 1 C C	C 0 0 0			INST→ TMP/IR	JUDGE CONDITION		SP OUT STATUS	SP←SP+1	DATA→PCL
RST n	1 1 N N	N 1 1 1			0→W INST→TMP/IR	SP←SP−1		SP OUT STATUS	SP←SP−1	PCH→DATA BUS
PCHL	1 1 1 0	1 0 0 1			INST→ TMP/IR	(HL)——→PC				
PUSH rp	1 1 R P	0 1 0 1				SP←SP−1		SP OUT STATUS	SP←SP−1	rh→DATA BUS
PUSH PSW	1 1 1 1	0 1 0 1				SP←SP−1		SP OUT STATUS	SP←SP−1	A→DATA BUS
POP rp	1 1 R P	0 0 0 1				×		SP OUT STATUS	SP←SP+1	DATA→rl
POP PSW	1 1 1 1	0 0 0 1				×		SP OUT STATUS	SP←SP+1	DATA→FLAGS
XTHL	1 1 1 0	0 0 1 1				×		SP OUT STATUS	SP←SP+1	DATA→Z
IN port	1 1 0 1	1 0 1 1				×		PC OUT STATUS	PC←PC+1	B2→Z, W
OUT port	1 1 0 1	0 0 1 1				×		PC OUT STATUS	PC←PC+1	B2→Z, W
EI	1 1 1 1	1 0 1 1				×		SET INTE F/F		
DI	1 1 1 1	0 0 1 1				×		RESET INTE F/F		
HLT	0 1 1 1	0 1 1 0				×		PC OUT STATUS	HALT MODE	
NOP	0 0 0 0	0 0 0 0	PC OUT STATUS	PC←PC +1	INST→ TMP/IR	×				

（续表）

M3			M4			M5						
T1	T2	T3	T1	T2	T3	T1	T2	T3	T4	T5		
	ACT+TMP→A											
	ACT+TMP→A											
	ACT+TMP→A											
	ACT+TMP→A											
	ACT+TMP→A											
	ACT−TMP; FLAGS											
	ACT−TMP; FLAGS											
PC OUT STATUS	PC←PC+1 B3→W										WZ OUT STATUS	WZ+1 →PC
PC OUT STATUS	PC←PC+1 B3→W										WZ OUT STATUS	WZ+1 →PC
PC OUT STATUS	PC←PC+1 B3→W		SP OUT STATUS	PCH→ SP←SP−1	DATA BUS	SP OUT STATUS	PCL→	DATA BUS			WZ OUT STATUS	WZ+1 →PC
PC OUT STATUS	PC←PC+1 B3→W		SP OUT STATUS	PCH→ SP←SP−1	DATA BUS	SP OUT STATUS	PCL→	DATA BUS			WZ OUT STATUS	WZ+1 →PC
SP OUT STATUS	SP←SP+1 DATA→PCH											
SP OUT STATUS	SP←SP+1 DATA→PCH											
SP OUT STATUS	TMP=00NNN000→Z PCL→DATA BUS										WZ OUT STATUS	WZ+1 →PC
SP OUT STATUS	r1→DATA BUS											
SP OUT STATUS	FLAGS→DATA BUS											
SP OUT STATUS	SP←SP+1 DATA→rh											
SP OUT STATUS	SP←SP+1 DATA→A											
SP OUT STATUS	DATA→W		SP OUT STATUS	H→ SP←SP−1	DATA BUS	SP OUT STATUS	L→	DATA BUS	WZ→	HL		
WZ OUT STATUS	DATA→A											
WZ OUT STATUS	A→DATA BUS											

附录 5　Intel 微处理器和单片微型机系列表

表 A5-1　Intel 微处理器系列表

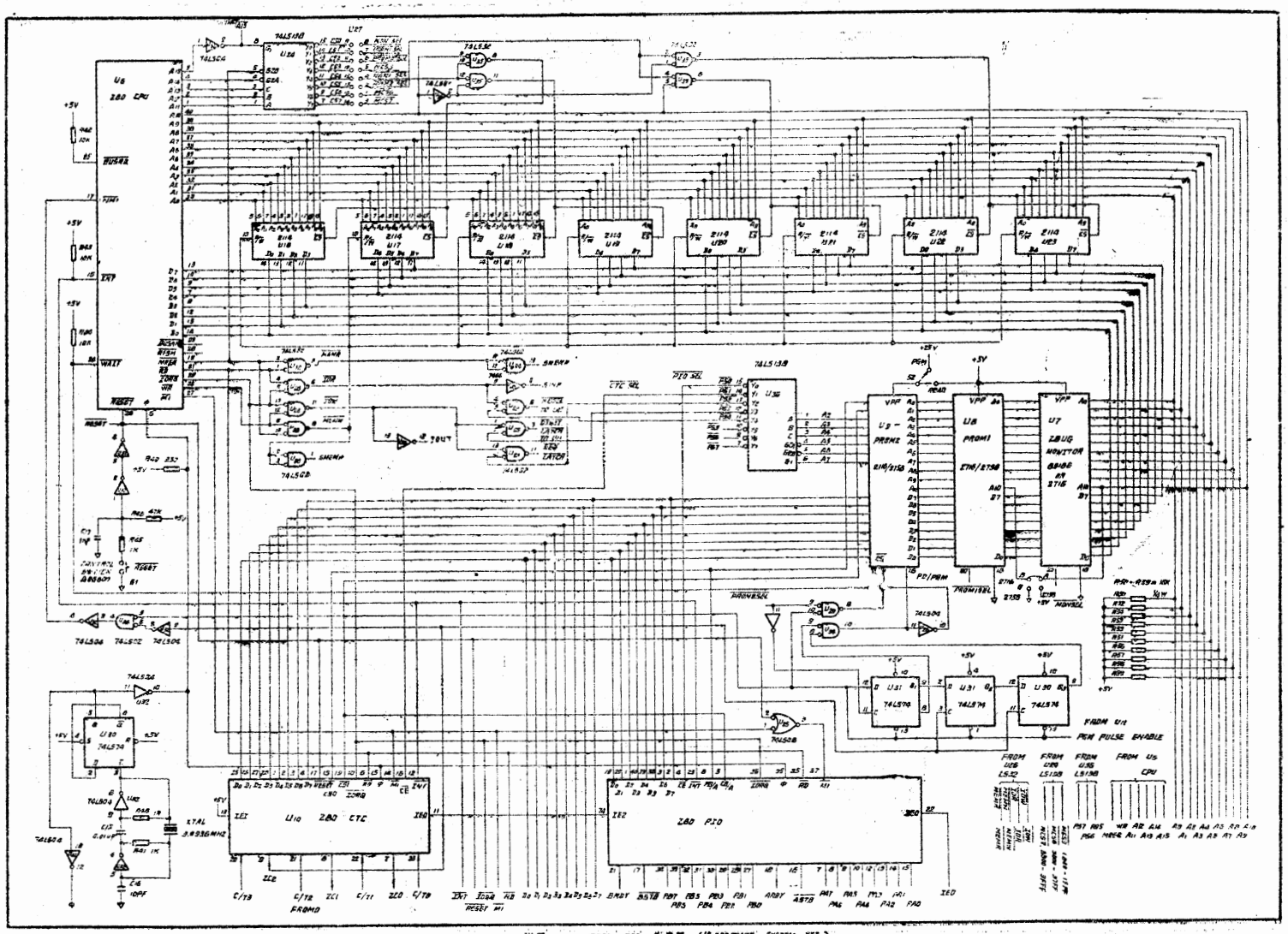
特　　点	8080A (MCS-80)	8085AH (MCS-85)	8086 (iAPX86)	8088 (iAPX88)	80186 (iAPX186)	80188 (iAPX188)	80286 (iAPX286)
1. 结构							
总线接口(位)	8	8	16	8	16	8	16
内部数据宽度(位)	8	8	16	16	16	16	16
时钟频率(MHz)	2, 2.6, 3	3, 5, 6	5, 8, 10	5, 8	6, 8	6, 8	4, 6, 8, 10
内存寻址范围	64KB	64KB	1MB	1MB	1MB	1MB	16MB
I/O 寻址范围	256B	256B	64KB	64KB	64KB	64KB	64KB
寻址方式	5	5	24	24	24	24	24
运算器	1	1	8	8	8	8	8
通用寄存器	6	6	8	8	8	8	8
2. 软件支持高级语言			Assembler, PASCAL 86/88, FORTRAN86/88, Microfocus Cobol, BASIC-86, C-86, PL/M 86,				
操作系统			iRMX86, CP/M86, UNIX V7				
3. 外围支持芯片						片	
时钟发生器	8224	片上	8284A	8284A	片上	片上	82284
系统控制器	8228	片上	8288	8288	片上	片上	82288
中断控制器	8259A	8259A	8259A	8259A	片上	片上	8259A
DMA 控制器	8257	8237A	8089	8237/8039	片上	片上	8089
定时/计数器	8253	8253/8254	8253/8254	8253/8254	片上	片上	8253/8254
总线驱动器	8216/8226	8286/8287	8286/8287	8286/8287	片上	片上	8286/8287
数字数据处理器			8087	8087	8087	8087	80287
4. 其它							
引线	40	40	40	40	68	68	68
封装型式	DIP	DIP	DIP	DIP	LCC	LCC	LCC
电源	±5V, 12V	5V	5V	5V	5V	5V	5V

表 A5-2 通用 8 位单片微型计算机

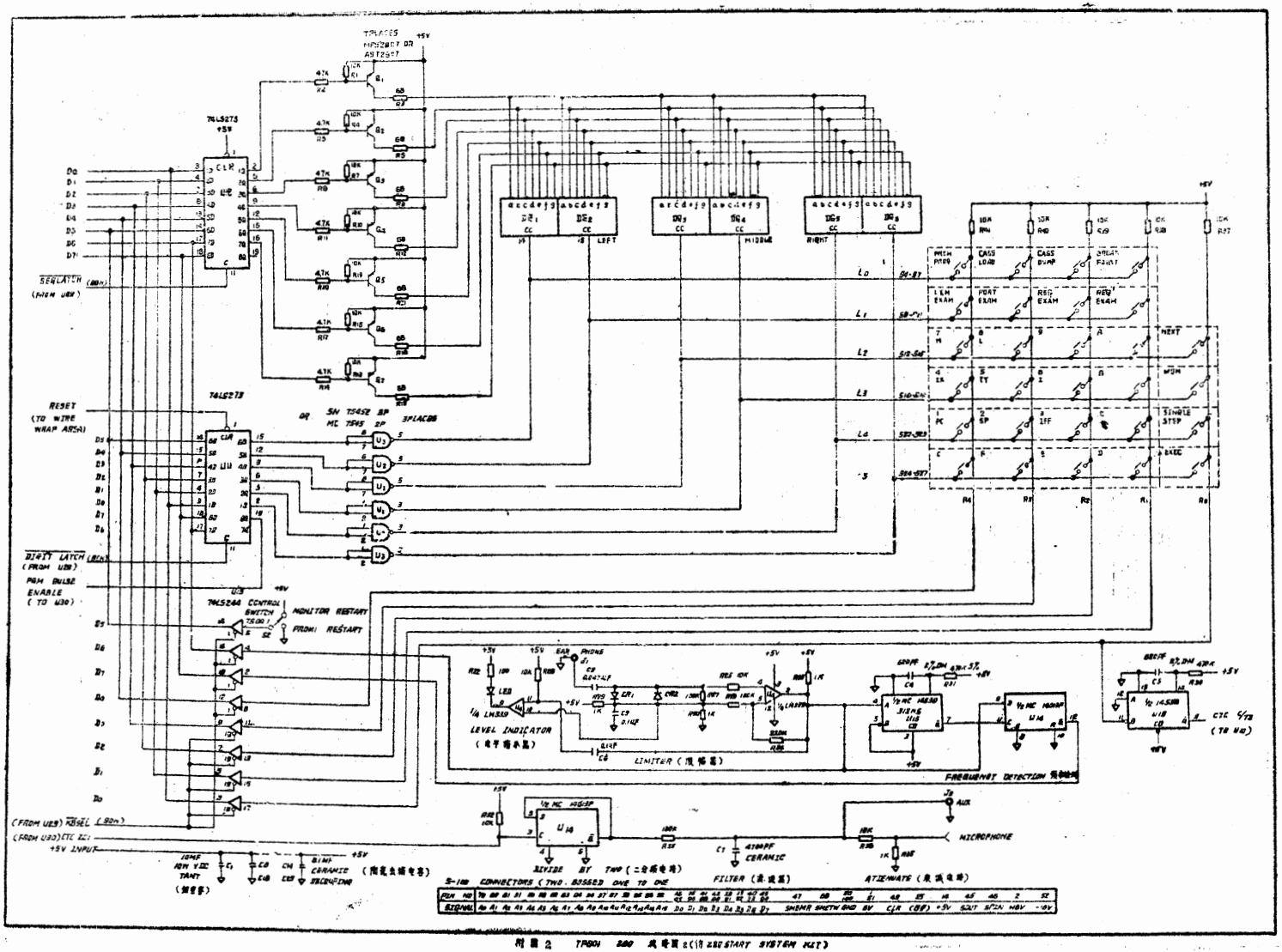
带 ROM	8020 H	8021 H	8022	8048 AH	8049 AH	8050 AH
带 EPROM	—	—	—	8748 H	8749 H	—
CPU/RAM/I/O	—	—	—	8035 AHL	8039 AHL	8040 AHL
基本周期	8.38 μs	8.38 μs	8.38 μs	1.36 μs	1.36 μs	1.36 μs
RAM 存贮器(Bytes)	64	64	64	64	128	256
程序存贮器 (Bytes)	1K	1K	2K	1K	2K	4K
I/O 线	13	21	28	27	27	27
定时/计数器	1	1	1	1	1	1
A/D	—	—	8 位	—	—	—
中断	—	—	2	2	2	2

表 A5-3 新型的 8 位单片微型计算机

特　　点	8051	80C51	8751	8031	80C31	8052	8032	8044+
程序存贮器(Byte)	4K	4K	4K EPROM	—	—	8K	—	4K
RAM 存贮器(Byte)	128	128	128	128	128	256	256	192
程序存贮器扩展(片外)(Byte)	64K	64K	64K	64K	64K	64K	64K	64K
数据存贮器扩展(片外)(Byte)	64K	64K	64K	64K	64K	64K	64K	64K
最大时钟频率(MHz)	12	12	12	12	12	12	12	12
标准指令时间(μs)	1	1	1	1	1	1	1	1
16 位定时/计数器	2	2	2	2	2	3	3	2
串行通信	同步/异步方式 9 或 10 位可编程							HDLC SDLC
I/O 线	32	32	32	16	16	32	16	32
中断源	5	5	5	5	5	6	6	5



TP801 Z80 系統圖 (SMART SYSTEM KIT)



附圖 2 TP801 280 系統圖 2(TP 280 START SYSTEM KIT)