8080
微型计算机系统
用户手册

上海电子计算机厂
翻 印

8080微型计算机系统用户手册

目　　录

* 译注：本章没有把本手册中的全部系列部件都译出，仅作部份选译。

8080微型计算机系统用户手册

引　言

§1. 概　述

数字式电子计算机自问世以来，不断更新，工作效率日益提高，随其每次的重大技术进展，其应用领域亦进一步扩展。由于小型计算机的出现，数字式电子计算机成为各式各样的过程控制系统中的一种常设部件。但是，小型计算机在"专用系统"的应用中体积大，价格高，其应用仍受到限制。如果不采用小型计算机，就采用"随机逻辑"(即，逻辑门、触发器、计数器等）凑成的专门定制系统。然而这种系统在其设计和调试中要耗费大量资金和漫长的研制时间，这仍然限制了其在大量的应用场合推广使用，因为要牵涉到许许多多的机器需要花费研制费用。

现在，英特尔公司(Intel Corp)给系统设计者提供了一种新的可供选择的办法，即：微型计算机。该公司是目前世界上供应大规模集成电路(LSI)存储器的最大厂商，他们利用在这方面取得的技术和经验以集成电路为基础发挥了计算机的功能。采用n-沟道硅栅MOS工艺在单块LSI片子上做出了快速（2微秒）、高性能(72种基本指令)的8080微处理器。此微处理器与存储器和I/O电路相连就可以组成完整的计算机。英特尔公司也生产各种各样的随机存取存储器(RAM)、只读存储器（ROM）和移位寄存器电路片子。这些片子与8080微处理器结合起来，就组成MCS-80型微型计算机系统，这种系统可以直接存取存在存储器中的数据达65,536字节。

8080微处理器用40线双列直插式封装件封装，很容易装接。8080有一条16位地址总线，一条8位双向数据总线，还有全译码的、与TTL相匹配的控制输出电路。除了RAM和ROM混合的存储器可提供多达64K的字节外，8080还能选择256个输入口和256个输出口，因此实际上允许无限制地扩充系统容量。8080指令系统包括条件转移、10进制和二进制运算、逻辑操作、寄存器与寄存器之间传输、堆栈控制和存储器访问这些指令。实际上8080指令系统是很强的，足以超过价钱高得多的许多小型计算机的功能；并且8080还与该公司早期生产的8008微处理器在软件上具有向上的兼容性（也就是说，为8008编制的程序能够在8080上汇编和执行)。

8080除了具有一组扩充的面向解决问题的指令系统以外，还有一个显著的特点——速度快。和往往以并行方式进行工作的随机逻辑相反，微型计算机是采取顺序地执行程序的工作方式，因此，微型计算机在给定的时间内能够执行的任务数与执行速度成正比。执行的速度成了微型计算机推广应用的限制因素。8080执行指令时间短到2 μS，比早期的微型计算机速度快一个数量级，所以其可能的应用领域会更广阔。

8080的总体结构较早期的微型计算机有显著的改进。8080有一个16位的堆栈指示器，用以控制对设在存储器中的外部堆栈寻址。可以通过专门的指令设置指示器的初始值，于是，外部存储器的任何一个区都可以用作后进先出堆栈，因而可以实现几乎无限制的子程序嵌套。堆栈指示器允许程序计数器、累加器、状态标志或任一数据寄存器的内容存入堆栈或从堆栈取出。另外，使用8080的堆栈控制指令可以处理多级中断。对中断服务时，处理器的状态就"推入"堆栈，相应中断服务完毕后，再从堆栈中"弹出"。即使中断服务例行程序本身被中断了，也可以保留处理器的寄存

器的内容。

§ 2. 采用微型计算机进行产品设计的优点

采用微型计算机几乎简化了产品研制的每一个阶段。象任何产品的研制步骤一样，第一步就是确定最终做出的系统要执行的各种功能。现在这些功能的实现不是采用门和触发器网络的办法，而是靠在存储器里编成适当的指令序列（程序）的方法来实现的。数据和某种类型的程序存储于RAM中，而基本程序则可以存储于ROM中。微处理器执行该系统的全部功能，靠在存储器中取指令、执行指令并通过微型计算机的输入/输出口输出结果。8080微处理器执行存放在一块2048字节ROM中的程序逻辑就等于以往需要1000个逻辑门才能执行的功能。

采用微型计算机来设计一个系统，其好处决不仅仅是简化了产品研制过程，还可以得到用微型计算机代替专门设计的随机逻辑的好处。最明显的好处就是大大节省硬件的成本。一组微型计算机芯片可以代替几十个随机逻辑元件，因而不仅缩小了系统体积，而且降低了成本。此外，随着要处理的单个元件数量的减少，生产成本降低了，复杂的印制电路板（其布线、测试和修改都是很困难的）的数量也会大大减少。微型机最大的好处莫如改变的灵活性，如果要对系统加以改变，只对存储器重编程序就行了，不必要对整个的系统重新设计。如果要想改进一个系统，可以设想一下会节省多少时间和资金。提高系统的可靠性是选用微型计算机而不用随机逻辑的又一个理由。随着元件数量的减少，发生故障的可能性亦会减少。从前用许多硬件元件执行的全部逻辑控制功能，现在都可以用几个ROM电路来执行，这种ROM是不易失的，就是说，ROM的内容是决不会消失的，即使发生了电源故障亦然。表0-1总结了使用微型计算机的许多优点。

表 0-1 采用微型计算机的优点

	常规系统	程序逻辑
产品性能		由于结合的简易性而简化了
系统和逻辑设计	用逻辑图进行	可用设计工具（编译程序，汇编程序、编辑程序）编制程序。
调试	用常规实验室仪器进行	用软件和硬件辅助工具可减少时间
印制板布线		需布线的板子较少
编制文件		需编文件的硬件较少
冷却和封装		因系统体积缩小功耗降低，减轻了工作任务
功率分配		要分配的功率较少
工程改变	用改线完成	改变程序

§ 3. 微型计算机设计工具

对于习惯于逻辑设计的人来说，不论程序逻辑优点如何，用程序逻辑进行设计似乎是变化太大了。现在却没有顾虑的必要，因为英特尔公司已经做了大量的基础工作。“Intellec 8研制系统”提供了配套件产品（OEM Product）*研制用的方法，灵活、便宜、简单。“Intellec 8”包括有：RAM程序存储器，使得程序输入和修改都比较容易；显示器和控制台，用于系统监控和调试；标准电传打字机接口；具有编程序能力的可编程序只读存储器（PROM）；标标的软件包（系统管理程序、汇编程序和测试编辑程序）。除了“Intellec 8”有的标准软件包以外，英特尔还供应一种PL/M编译程序，即一种交叉汇编程序和一种用FORTRANIV写的并且设计在任何大型计算机上运行的模拟程序。这些程序可以直接由英特尔公司和由许多全国性计算机分时服务公司取得。英特尔公司的“微型计算机系统小组”在用户的产品研制的每个阶段都可以一直提供帮助。

英特尔公司还提供其软硬件产品的成套文件。除了本“用户手册”以外，还有：

* OEM Product：是专供某些成套设备配套用的产品——译注。

- PL/M 语言参考手册
- 8080 汇编语言程序计设手册
- Intellec 8 /MOD 80 操作员手册
- Intellec 8 /MOD 80 硬件参考手册
- 8080 用户程序库

§ 4. 应用举例

8080 可以用作各式各样的运算和控制系统的基本部件。各种特殊应用的系统结构将随所使用的外部设备的性质和所需要的存储器的容量和类型而有所不同。这一部分所介绍的应用实例和解决办法主要表明如何利用微型计算机来解决设计问题，不应该认为 8080 所能发挥的作用或性能仅限于这里所列出的使用范围。

这里所举的例子是将一部 8080 微型计算机用在另售商店所使用的自动秤中。基本机器有两个输入部件：称量部件和键盘，后者用以选择功能和输入单位重量的价格。唯一的输出部件就是一个显示器，显示总的价钱，当然也可以加上一台票据打印机，作为一种选用输出部件。

控制部件必须从称量部件接受重量信息，从键盘接收功能和数据输入，然后发出显示。要执行的唯一操作就是一项简单的重量乘价格的乘法。

控制部件当然也可以用另外的办法，即用标准的 TTL 逻辑实现。可以划出各部分的状态图，设计出一个乘法器。然后将整个的设计联在一起，最后简化成一组精选的组件和一个印制电路板布线图。实际上，当采用象 TTL 这样的一个逻辑系列进行设计时，就是靠选择组件和逻辑连接，实现"专门"的设计。

然而，如采用 8080 微型计算机来实现控制部件（如图 0-1 所示），唯一的"专门"的逻辑部分也就是接口电路那一部分。这些电路通常是十分简单的，为输入和输出信号提供电气缓冲作用。

系统设计者就不需要划状态图进而形成逻辑，而只要准备一个流程图，表示哪些输入信号必须读，需要做什么处理和运算，必须产生什么输出信号。从流程图写成程序。该程序再汇编成二进位排列格式，送入程序存储器。因此该系统主要靠程序存储器的内容而专门化了。

就本自动秤而言，程序可以存于 ROM

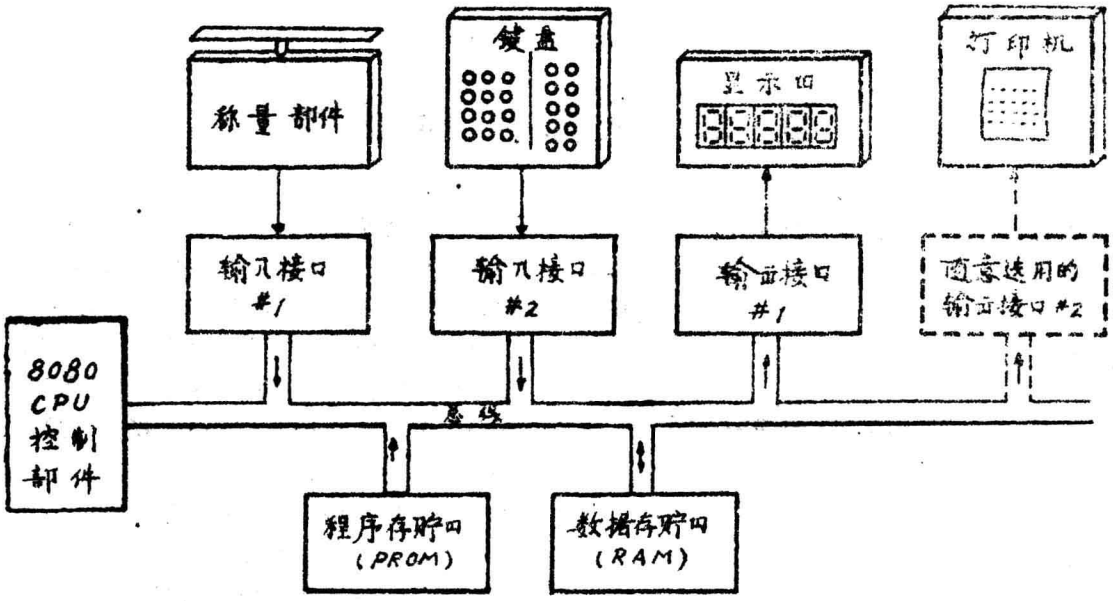


图 0-1 微型计算机应用——自动秤

中，因为本微型计算机将总是执行同一个程序，即实现称量功能的程序。处理器不断地监控着键盘和称量部件，任何时候都根据实况变换显示内容。该自动秤需要的存储容量很少，只需要存储价格、中间结果和显示内容的副本。

若一个产品的控制部分用一组微型计算机芯片来实现的话，只要改变存储器的程序，就可以改变功能和增加性能。而利用 TTL 系统，改变就需要广泛的改变联线，还要改印制电路板等。

微型计算机应用广度只限于设计人员想象的深度如何。表 0-2 列出了一些可能的应用项目，以及每个项目通常使用的外部设备种类。

§ 5. 应 用 表

表 0-2 微型计算机应用

应用种类	所需的外围设备
智能终端	阴极射线管显示器 打印机 同步和异步数据线路 盒式磁带机 键盘
博奕论机	键盘，按钮和开关 各种显示设备 硬币接受器 硬币配出器
出纳机	键盘或输入开关阵列 零钱配出器 数字显示器 票据打印机 磁卡片阅读机 通讯接口
会计机 (记帐)	键盘 打印机 盒式磁带或其他磁带机 软磁盘
电话交换控制机	电话线路扫描器 模拟交换网络 拨号记发器等
数控机床	步进马达 磁(纸)带阅读机 光轴编码器
过程控制	模-数转换器 数-模转换器 控制开关 显示器

第一章 计 算 机 的 功 能

本章列出了计算机的一些基本概念，提供一些基础知识和定义，对于以后各章的了解很有用处。对计算机已经很熟悉的，可以按自己意愿跳过这一部分阅读。

§ 1. 典型的计算机系统

一部典型的数字计算机由以下三部分组成：

a) 中央处理部件(CPU)

b) 存储器

c) 输入/输出口

存储器就是存放“指令”和“数据”的地方，“指令”就是指挥 CPU 操作的代码信息，“数据”就是由 CPU 处理的代码信息。一组存放在存储器中的逻辑上有关联的指令叫做“程序”。CPU按照逻辑规定的顺序从存储器“读出”每条指令，并用这条指令启动处理操作。假若程序列是连贯的和有逻辑联系的，处理该程序就会产生出可以理解的和有用的结果。

存储器不仅存放指挥 CPU 处理操作的指令，还用来存放待处理的数据。程序的组成必须符合下列要求，即：当 CPU 希望调用一条

指令时，它不能读出一个非指令的字。CPU能够迅速地访问存放在存储器的任何一个数据；但存储器的容量通常不是大到足以存放某一特殊应用所需要的全部数据组，这个问题的解决办法是使计算机备有一个或多个"输入口"。计算机可以对这些口进行寻址，并输入存放在那里的数据。增加输入口能够使计算机以高速度和大量地从外部设备（例如纸带机或软盘）接收信息。

一部计算机还需要一个或多个输出口，使CPU将其处理结果送出去。输出可以联到显示器，供操作员使用；可以联到产生人可理解的复制品的外部设备，例如行式打印机；可以联到外存，例如软盘。或者，输出可以形成过程控制信号，指挥另一个系统的操作，例如一条自动装配线。像输入口一样，输出口亦是可以寻址的，输入口和输出口一起使处理机和外界进行通讯。

CPU统一指挥整个系统。它控制着其他部件所执行的功能。它必须能够由存储器取出指令，对指令的二进制内容译码，并且加以执行。它还必须能够按照指令的需要访问存储器和输入/输出口。此外，CPU还应能识别和响应某些外部控制信号，例如"中断"请求和"等待"请求。

下面介绍CPU赖以执行各种功能的功能部件。

§ 2 CPU的总体结构

一部典型的中央处理部件(CPU)由下列的功能部件互相连接起来组成的：

- 寄存器
- 算术/逻辑部件(ALU)
- 控制电路

寄存器是CPU内部的临时存储单元。有些寄存器，例如程序计数器和指令寄存器，是专用的。而另一些寄存器，譬如累加器则是通用的。

累加器　累加器通常存放ALU要处理的一个操作数。一条典型指令可以指挥ALU将某些其他寄存器的内容和累加器的内容相加，并将结果存放在累加器本身。所以，累加器既是操作数寄存器，也是结果寄存器。

CPU常常包含许多额外的通用寄存器，可以用来存放操作数和中间结果。采用了通用寄存器就避免了中间结果在存储器和累加器之间来回传送，因而提高了处理的速度和效率。

程序计数器（转移、子程序和堆栈）构成程序的指令是存放在系统的存储器中的。中央处理器要取出存储器的内容，才能确定适当的操作。这就是说，处理器必须知道哪一个存储单元存放着下一条指令。

存储器的每一个存储单元都是编有号码的，以便和存储器的所有其它存储单元区别开来。表示存储单元的数码叫做"地址"。

处理器有一个计数器，存放着下一条指令的地址。此寄存器叫做"程序计数器"。处理器每取一条指令都要给计数器加"1"，用以更新程序计数器的内容，所以程序计数器总是表示现行状态（指向下一条指令）。

因而，程序员总是将其指令存放在数字相近的地址里．若是低数地址存放要执行的第一条指令，高一个数的地址就存放着它的下一条指令。程序员也可以违反这种顺序编排，这就是在存储器某一区的一条指令正好是一条"转移"到存储器另一个区的指令的时候。

转移指令中有跟在其后面要操作的那一条指令的地址。只要程序转移规定出正确的地址，下一条指令可以存放在任一存储单元。当执行一条转移指令时，处理器将程序计数器的内容代之以转移指令中存放的地址，因而保持着程序的逻辑连续性。

当存储器的程序"调用"一个子程序时，就要发生一种特殊的程序转移。在这种转移中，处理器需要"记住"发生转移时的程序计数器的内容。这就使处理机能够在子程序完成最后一条指令后恢复执行主程序。

子程序就是程序中的程序。它通常是在主程序执行中必须反复执行的一组通用指令。最好的例子如计算乘方、计算正弦或一个顺序变

量的对数等，通常都是编成子程序的。其它的例子还有，如用来将数据输入或输出于某一特定外部设备的程序。

处理器有一种专门处理子程序的方法，以保证有秩序地返回主程序。当处理器收到某一条“调用”指令时，它就将程序计数器加“1”，并将其内容存放于一个保留的存储区，叫做“堆栈”。因此，堆栈就保留着子程序完成以后要执行的那一条指令的地址。然后处理器就将“调用”指令所规定的地址送入程序计数器。于是取出的下一条指令将是被调子程序的第一条。

任何子程序的最后一条指令都是一个“返回”操作。这一条指令不需要指明地址，当处理器取出一条“返回”指令时，它只将程序计数器的现行内容代之以存放在堆栈顶端的地址。就能使处理器在原来调用程序执行完毕后立即恢复执行主程序。

子程序常常是“嵌套”的，也就是，有时一个子程序会调用第二个子程序。第二个子程序还可以调用第三个子程序，依此类推。这是完全可以解决的，只要处理器具有足以存储必要的返回地址的容量和备有这样做的逻辑条件就行了。换句话说，嵌套的深度取决于堆栈本身的深度。假若堆栈有存放三个返回地址的空间，那末，子程序嵌套可以达到三级。

各种处理器设置堆栈的方法不同。有些处理器将存放返回地址的地方就设在处理器内部。另一些处理器则在外部存储器中开辟一个指定区作为堆栈，处理器内只设置一个存放堆栈最近输入地址的“指示器”寄存器。外部堆栈理论上可以做到无限止的子程序嵌套。另外，假若处理器设有指令，能把累加器和其他通用寄存器的内容推入堆栈或者通过堆栈指示器中存放的地址从堆栈弹出，那末可以实现多级中断处理（本章后面还要介绍）。处理器的状态（即，所有寄存器的内容）可以保留在堆栈里，然后接受中断，在中断处理以后，再恢复。即使中断服务例行子程序本身被中断了，也有保留处理器当时的状态的能力。

指令寄存器和译码器　每部计算机都有一个这部机器特征的字长。计算机的字长通常由其内部存储器的字长和互连通路（叫做“总线”）的宽度决定；例如，一部计算机，其寄存器能存放8位信息，其总线能传送8位信息，这部计算机就有一个表示其特征的8位字长，并且中央处理部件被称为8位并行处理器。8位并行处理器通常处理8位二进制字段是最有效的。与这样的处理器相连的存储器也要组织得在每个可寻址的存储单元里能存放8位信息。数据和指令也要作为一个8位二进制数，或者作为8位的整数倍数，如16位，24位等，存放在存储器里。这一个表示机器特征的8位字段经常被叫做“字节”。

计算机能够执行的每一项操作都由一个特定的字节数据来确定，每个特定的字节数据，叫做“指令码”或“操作码”。用作指令码的一个8位字能够区分出256种不同的操作，对于大多数的处理器都足足有余。

处理器取一条指令有两个不同的操作。首先，处理器将其程序计数器的地址送到存储器。然后，存储器将被寻址的字节送回处理器。CPU将此指令字节存放在一个寄存器里，该寄存器叫做“指令寄存器”，并且用来指挥本指令执行过程的余下的各步动作。

要把处理器赖以把指令码翻译成特定处理动作的机理说清楚还需要更多篇幅，此处不再赘述，这一概念对于任何一个逻辑设计者都应该是一目了然的。存放在指令寄存器里的8位码可以加以翻译并用来有选择地启动许多条（此处可多达256条）输出线中的一条。每条线各表示与执行某一特定指令码有关的一组动作。被启动的这条线能够与指定的定时脉冲相配合产生出电信号，此电信号可用来启动特定的操作。将指令码转换成操作这个过程是由指令译码器和有关的控制电路执行的。

一组8位指令码通常是足够用来规定某一特殊处理操作的。不过，有时执行指令需要的信息比8位能传递的信息要多，那就要增加次数。

这方面的一个实例就是当指令访问某一存

储单元时，基本指令码确定要执行的操作，但还不能规定目标地址。这时就必须采用2字节或3字节指令。连续的指令字节存放在顺序排列的相邻的存储单元里。处理器要连续取两或三次指令，以取出全部指令。由存储器取出的第一个字节放在处理器的指令寄存器里，以后又取出的字节放在临时使用的暂存器里。然后处理器才进行指令的执行阶段。这样一种指令被叫做"可变字长"指令。

地址寄存器　CPU可以利用一个寄存器或寄存器对来保存要被访问的某一存储单元的地址。假若地址寄存器是"可编程序的"（即，假若有允许程序员改变寄存器内容的指令），那末，程序就能够在执行一条访问内存指令（即一条从存储器取数据，将数据写入存储器或对存放在存储器的数据进行操作的指令）之前在地址寄存器对内"建立"一个地址。

算术/逻辑部件(ALU)　所有的处理器都有一个算术/逻辑部件，通常简称为ALU。ALU。正如其名称意味的那样，就是GPU对二进制数据执行算术和逻辑操作的那一部分硬件。

ALU必须有一个"加法器"能够按照二进制算术的逻辑将两个寄存器的内容加在一起。这一措施就使处理器对由存储器或其他输入途径取得的数据执行算术运算。

只使用这个基本加法器，一个有能力的程序员就能够编写出执行加、减、乘、除的例行程序，赋予机器以完全的算术运算的能力。不过实际上多数的处理器还有其他的内在功能，包括硬件减法，布尔逻辑操作和移位能力。

ALU要有"标志位"，用以规定在算术和逻辑操作过程中产生的某些状态。标志一般包括"进位"、"零"，"符号"和"奇偶位"这些。往往有条件地根据一个或多个标志发生程序转移。举例来讲，程序可以这样来编制：假若在一条加法指令执行以后"进位"位置位，就转移到一个专用例行程序。

控制电路　控制电路是CPU内的主要功能部件。使用时钟输入，控制电路可以维持各种处理任务所需要的事件发生的适当顺序。在一条指令取出并译码之后，控制电路就（对CPU的内部和外部的部件）发出相应的信号，以启动适当的处理操作。通常控制电路还可响应外部信号，比如中断或等待请求。"中断"请求会使控制电路临时中断主程序的执行，转移到某一个专用、例行程序为要求中断的设备服务，然后再自动地返回主程序。"等待"请求通常由操作比CPU慢的存储器或输入/输出设备发出。控制电路将使CPU空转，直到存储器或输入/输出口数据准备就绪为止。

§3　计算机操作

有一些操作几乎对任何一种计算机都是基本的。对这些基本操作的完全理解是考察某一部计算机的特定操作的必要前提。

定时　中央处理器的活动是周期性的。处理器取出一条指令，执行所需要的操作，再取下一条指令等等。这种事件发生的顺序需要精确的定时。因而CPU需要有一个自激振荡运行的时钟发生器，为处理器的一切动作提供一个基准。一条指令的取出和执行结合在一起叫做一个"指令周期"。与某一明确规定的动作有关的一个周期部分叫做一个"机器周期"。定时振荡器的脉冲间的间隔叫做一个"时钟周期"。完成一个机器周期*通常必须一个或多个时钟周期，并且一个指令周期又有若干个机器周期*。

取指令　任一指令周期的第一个机器周期*都用来取本条指令。CPU发出一个读信号并且程序计数器的内容送到存储器，存储器就将本条指令字送回作为响应。指令的第一个字节存放在指令寄存器。假若指令由一个以上的字节组成，就需要更多的机器周期*，以便取指令的各个字节。当整条指令都取到CPU中时，程序计数器就加"1"（以备取下一条指令）并把指令加以译码。指令中规定的操作将在本指令周期的其他机器周期*里执行。指令可以要求一次存储器读或写，一次输入或输出"与"/"或"一

* 原文中此处的"State"系"Machine Cycle"之误。——译注

次 CPU 的内部操作，譬如一次寄存器到寄存器之间的传送或寄存器加法操作。

存储器读　“取指令”只不过是一次特殊的存储器读操作而已，这个操作将指令送到 CPU 的指令寄存器。取出的这条指令还可能要求将数据由存储器再读出，读进 CPU。CPU 就再发出一次读信号并送出适当的存储器地址；存储器的响应就是送回所要的字。收到的数据存放到累加器里，或存放到一个其他的通用寄存器（而不是指令寄存器）里。

存储器写　存储器写操作除了数据流向以外和读操作相似。CPU先发出一个写信号，再发出适当的存储器地址，然后就把要写的数据字送入被寻址的存储单元。

等待(存储器同步)　如前所述，处理器活动是由主时钟振荡器定时的。时钟周期决定所有处理动作的时序。

不过，处理周期的速度要受到存储器的“存取时间”的限制。一旦处理器向存储器送出了一个该地址的信号，直到存储器有足够时间，作出回报时，处理器才能进行处理。大多数的存储器能够以比处理周期所需要的快得多的速度作出回报。然而，也有一些存储器不能够在处理器的时钟建立的最短时间内供给被寻址的字节。

因此处理器应该具有同步措施，让存储器请求一个“等待状态”。当存储器收到一个读或写启动信号时，它就在处理器的 READY（准备就绪）线上置一个请求信号，使处理器暂时空转。在存储器已够时间回报时，它就释放处理机的 READY 线，指令周期就进行下去。

输入/输出　输入和输出操作和存储器的读写操作类似，所不同的就是被寻址的对象是某一外部输入/输出设备，而不是某一存储单元。CPU发出相应的输入或者输出控制信号，送出适当的设备地址，然后不是收到输入的数据，就是送出输出的数据。

数据的输入或输出能够以并行或串行方式进行。数字计算机的所有数据都是以二进制代码形式表示的。一个二进制的数据字是由一组二进位组成，每一位不是“1”就是“0”。并行的输入/输出是把一个字的所有位同时传送，每一根线送一位。串行的输入/输出是在一根线上每次送一位。串行输入/输出自然是慢得多，但都比并行输入/输出硬件少得多。

中断　在许多中央处理器上都设有“中断”措施，作为提高处理机效率的一种手段。这里举一个例子：计算机正在处理大量数据，其中部分数据又要输出到打印机。CPU 能够在一个机器周期内输出一个字节的数据，但打印机却要花许多个机器周期才能打印完这个数据字节所规定的字符。于是 CPU 只能空等，直到打印机能接受下一个数据字节为止。假若计算机设有中断装置，CPU 能够在输出一个字节后，就回到数据处理。当打印机准备好接收下一个字节时，它再请求中断。当 CPU 响应中断时，就暂停执行主程序，并自动地转移到一个例行程序，输出下一个字节。等该字节输出以后，CPU 继续返回执行主程序。这在原则上和一次调用子程序十分相似，不同之处只在于转移的启动来自外部，而不是来自程序。

还可能有更复杂的中断结构。在这种结构中有几个要求中断的设备公用一个处理器，但要分出不同的中断级别。中断处理是一项重要的技术性能，它可以最大限度地发挥处理器的能力，使处理器具有高的吞吐率。

保持　另一个提高处理器吞吐率的重要特性就是“保持”。保持措施能够容许“直接存取存储器(DMA)”操作。

在一般的输入和输出操作中，处理器本身监控着全部的数据传送。存入到存储器去的信息要从输入设备送到处理器，然后再由处理器送到指定的存储单元。同样，由存储器送到输出设备的信息也要经过处理器。

要是有一些外部设备将信息直接送往或取之于存储器，就要比经过处理器传送快得多。当有相当量的数据需送往或取自于这样的外部设备，就可采用由这些设备直接传送的方法，这将会提高“系统吞吐率”。进行这样的传送时处理器必须暂停其操作，以避免处理器和外围

设备试图同时访问存储器而产生的矛盾。正是由于这个理由，在某些处理器中要有“保持”措施。

第二章 8080 中央处理器

§ 1. 引 言

8080 是用于通用数字计算机系统的完整 8 位并行的中央处理器（即 CPU），它是用 n 沟道硅栅 MOS 工艺做成的单片大规模集成电路。8080 通过一个 8 位双向的 3 状态数据总线（D_0～D_7）来传递数据和内部的周期信息，又通过一个 16 位的 3 状态地址总线（A_0～A_{15}）来发送存储器及外部设备的地址。从 8080 送出的有 6 个定时及控制输出信号（SYNC、DBIN、WAIT、$\overline{WR}$、HLDA 及 INTE）*，而为 8080 接纳的有 4 个控制输入信号（READY、HOLD、INT 及 RESET）、4 个电源端（+12 V、+5 V、−5 V 及地）和两个时钟（ϕ_1 及 ϕ_2）输入信号。

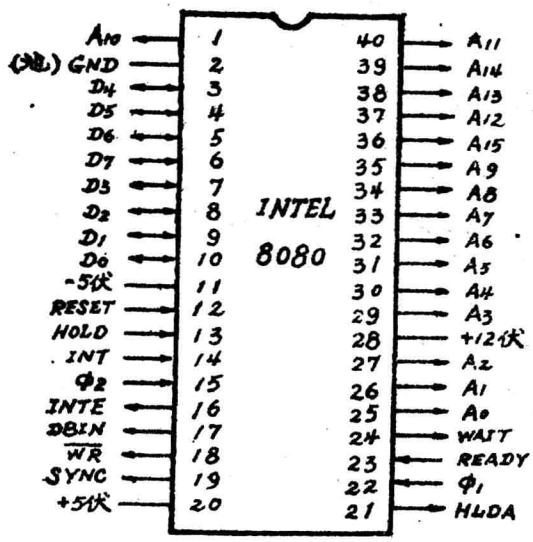


图 2-1 8080 引出端排列图

§ 2. 8080 CPU 的总体结构

8080 CPU 包括下列功能单元：

- 寄存器阵列及地址逻辑；
- 算术及逻辑单元（ALU）；
- 指令寄存器及控制部分；
- 双向 3 状态的数据总线缓冲器。

图 2-2 所示为 8080 CPU 内部的功能方框图。

1. 寄存器

寄存器阵列包含一个静态的读写存储器*（RAM）的阵列，它由 6 个 16 位寄存器构成：

- 程序计数器（PC）；
- 堆栈指示器（SP）；
- 6 个 8 位通用寄存器成对排列，称作 B, C D, E；及 H, L；
- 1 个称作 W, Z 的成对暂存寄存器。

程序计数器保存现行程序指令的存储器地址，每当指令取出后它就自动加 1。堆栈指示器保存下一个可供使用的存储器中堆栈单元的地址。堆栈指示器能够设置初值，因而可使用读写存储器的任何部分作为堆栈。堆栈指示器在数据被“压入”堆栈时减 1；而当数据“弹出”堆栈时加 1（即堆栈是“向下”增长的）。

6 个通用寄存器既可以作为单个的寄存器（8 位）使用，也可以作为寄存器对（16 位）使用。暂存寄存器对 W，Z 不能用程序来选址，只在内部执行指令时才用到它。

8 位的数据字节通过寄存器选择多路转换开关能够在内部的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 器、减 1 器线路之间进行传送。地址锁存器从 3 个寄存器对中的任何一个接收数据并送至 16 位地址输出缓冲器（A_0～A_{15}）或加 1 器

* 输入和输出信号的符号名称，后面均有说明——译注。

* RAM 在前面译作随机存取存储器，这里按功能译为读写存储器——译注

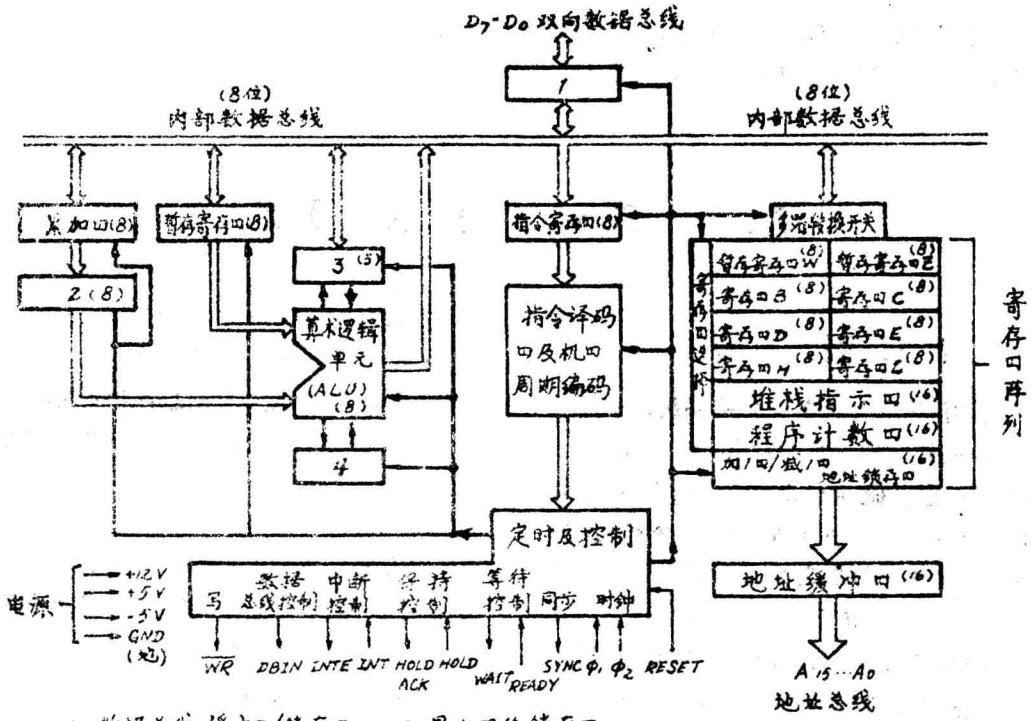


图 2-2 8080 CPU 的功能方框图

及减 1 器。加 1 器减 1 器线路从地址锁存器接收数据，然后把它送到寄存器阵列，因此 16 位数据能够被加 1 或减 1 或者仅在寄存器之间传送。

2．算术及逻辑单元(ALU)

ALU 包括下面这些寄存器

- 一个 8 位累加器；
- 一个 8 位暂存累加器(ACT)；
- 一个 5 位条件标志寄存器：零位、进位位、符号位、奇偶位及辅助进位位；
- 一个 8 位暂存寄存器(TMP)。

算术、逻辑及循环移位等操作是在 ALU 中履行的。ALU 的数据由暂存寄存器(TMP)、暂存累加器(ACT)以及进位触发器来供给，操作结果能传送到内部总线或累加器；ALU 也能传送信息给标志寄存器。

暂存寄存器(TMP)从内部总线接收信息并能把它全部或部分信息送到 ALU，标志寄存器和内部总线。

累加器能自 ALU 及内部总线取数并能把数据传送到暂存累加器(ACT)及内部总线。在执行 DAA 指令(见第 4 章)期间，为了进行十进位调整，累加器(ACC)及辅助进位触发器的内容可以检测。

3．指令寄存器及其控制

在取指令周期，一条指令的第一个字节内有操作码，从内部总线传送到 8 位的指令寄存器。指令寄存器的内容又送给指令译码器。指令译码器的输出结合各种定时信号，能为寄存器阵列、ALU 及数据缓冲器部件提供控制信号。此外指令译码器的输出和从外部来的控制信号送到定时及状态控制部分，产生状态及周期的定时信号。

4．数据总线缓冲器

8 位双向的 3 状态缓冲器用于隔离 CPU 的内部总线和外部数据总线(D_0～D_7)。在输出方式时，内部总线的内容被送进一个 8 位的锁存器，再送至数据总线的输出缓冲器。输出缓

冲器在输入或者没有传送操作时被关闭。在输入方式时，数据从外部数据总线传送到内部总线。除了传送状态(即T3，在这一章后面要叙述到。)内部总线在每个状态起始时，都被予先放上数 。

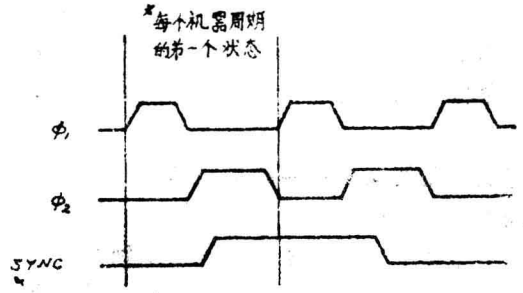
§ 3. 处理机周期

一个指令周期长短是按取和执行指令来确定的。在取指令期间，一条被选的指令（一、二或三个字节)从存储器提出并放入CPU的指令寄存器去。在执行阶段指令被译码，译成指定的处理动作。

每个指令周期包含一、二、三、四或五个机器周期。每当CPU要自存储器或输入输出口存取数，就得有一个机器周期。一条指令的提取阶段是每取一个字节需要一个机器周期。一条指令执行阶段的时间，则依已被取出的指令属什么类型来定。有些指令除了为取指令而必需的机器周期外，不需要其他的机器周期；然而另一些指令却要用附加的机器周期来从存储器或I/O设备读出或写入数据。DAD这条指令是个例外，它要求两个附加的机器周期以完成内部寄存器对的相加（见第4章)。

每个机器周期包括3、4或5个状态。每个状态是处理动作的最小单位，其宽度是按两个相邻的ϕ_1驱动时钟脉冲前沿的间隔来确定。8080是由一个两相的时钟振荡器来驱动的，所有的处理动作都以这个时钟周期为基准。两个不重迭的时钟脉冲用ϕ_1及ϕ_2表示，都由外部电路提供。划分每个机器周期为若干状态的只是ϕ_1时钟脉冲。在8080的定时逻辑中使用时钟输入信号产生SYNC脉冲，这个脉冲示明每个机器周期开始。SYNC脉冲是由ϕ_2从低到高的跳变来触发的如图2-3所示。

确定一个状态时间的长度有3种例外情况，那就是等待(WAIT)状态、保持(HLDA)状态及暂停(HLTA)状态，这些状态将在本章后面叙述。其所以例外是因为这些状态本身长度不确定的缘故，即要由外部情况的变化来决定。但即使就是这些例外的状态，也必须由驱动的时钟脉冲来同步，因而所有状态的时间长度就必然是时钟周期的整数倍。



*对一条DAD指令来说，在第二、三个机回不产生这个SYNC信号，因为那些周期是用于内部寄存回对做加法的。

图 2-3 ϕ_1,ϕ_2及SYNC的定时

扼要的说，每个时钟周期表明一个状态，3至5个状态构成1个机器周期；而1至5个机器周期组成一个指令周期。完成整个指令周期需要有4至18个状态，根据指令的不同种类，它所拥有的状态数也不相同。

1．机器周期的划分

除了DAD这条指令之外，决定任何一条指令需要用多少个机器周期，只有一个考虑，即取决于处理器为了取和执行指令需访问存储器的次数或可寻址的外部设备的次数。象许多处理器那样，8080也是属于这样的结构：每个机器周期仅发送一次地址。这样，如果取和执行一条指令需要访问存储器两次，那么这条指令的指令周期就包括两个机器周期；如果要求访问五次，则指令周期包含五个机器周期。

每个指令周期至少访问存储器一次，在这期间取出指令；即使指令的执行不要求再访问存储器，一个指令周期也总得有一次取指令。每条指令的第一个机器周期当然就是一个取周期(FETCH)。此外就没有更多的规则，只决定于所取指令的类型。

这里分析某些例子。加寄存器(ADD r)指令是仅需要单个机器周期(即FETCH周期)就能完成的指令，在这条一字节指令中，CPU 6个通用寄存器有一个它的内容加到累加器现存内容去。因为执行命令需要的所有信息都包含在8位指令码中，有一次存储器访问也就够了。

从存储器提取指令，用3个状态，另一个附加的状态用来完成拟进行的加法，整个指令周期只用一个机器周期包括4个状态或4个外部时钟周期。

然而若要使某个存储器单元的内容加到累加器的现存内容中去(ADD M 指令)。虽然原则上颇象刚才引用的例子，但还要用到另外的几个步骤，为了选取那个所指的存储单元就要有一个额外的机器周期。

实际进行的次序是这样的：首先处理器按照它的程序计数器所选地址从存储器提取1字节的指令字，这就占用了3个状态。在FETCH的机器周期时间取来的8位指令字放入CPU指令寄存器中，并用来指定在这个指令周期其余部分期间的动作。其次，处理器又把它的H及L寄存器的内容作为一个地址送出去。在读存储器(MEMORY READ)的机器周期期间取来8位数据字，放入8080 CPU内部一个暂存寄存器。由此又占用了3个时钟周期（3个状态)。在第7个，也是最后的状态，暂存寄存器的内容加到累加器的内容中去。全部是两个机器周期包括7个状态，完成"ADD M"的指令周期。

另一个极端相反的例子，如存入H及L寄存器指令(SHLD)，它要求有5个机器周期。在一个"SHLD"指令周期时间，处理器的H及L寄存器内容放进两个相邻顺序的存储单元中去。这两个单元的位置由操作码字后面的两个地址字节所指明。执行指令的次序如下：

(1) 一个FETCH机器周期包括4个状态；在这个机器周期的前3个状态，处理器按程序计数器所指定的地址取指令。程序计数器随即加1，第4个状态用来进行内部的指令译码。

(2) 一个MEMORY READ机器周期包括3个状态，在这个机器周期时间，按程序计数器所指定的地址从存储器读出并放进处理器的Z寄存器。程序计数器又加1。

(3) 再一个MEMORY READ机器周期包括3个状态；这时从存储器读出程序计数器所指定地址的内容并放到W寄存器。程序计数器加1，为取下条指令予先作准备。

(4) 一个MEMORY WRITE机器周期包括3个状态；这时L寄存器的内容被传送到W和Z寄存器现存内容所指定的存储器单元去。传送过后的状态，用来使W, Z寄存器对加1，以便指定下一个接收数据的存储器单元。

(5) 又一个包括3个状态的MEMORY WRITE机器周期；H寄存器的内容被传送W, Z寄存器对指定的新的存储器单元中去。

简要的说"SHLD"指令周期包含5个机器周期，共用了16个状态来执行这条指令。

绝大多数指令是界于这两种极端不同的指令"ADD r"及"SHLD"之间。例如输入(INP)及输出(OUT)指令，需要3个机器周期；一个FETCH周期取得指令，一个MEMORY READ周期得到外部对象的地址，和一个INPUT(输入）或一个OUTPUT（输出）机器周期来完成传送操作。

不会出现一个指令周期包括多于5个机器周期。下面是一个指令周期内可能出现的10种不同的机器周期类型。

(1) FETCH（取）或 M_1

(2) MEMORY READ（存储器读出）

(3) MEMORY WRITE(存储器写入)

(4) STACK READ（堆栈读出）

(5) STACK WRITE（堆栈写入）

(6) INPUT（输入）

(7) OUTPUT（输出）

(8) INTERRUPT（中断）

(9) HALT（暂停）

(10) HALT·INTERRUPT(暂停·中断)

在一个特定的指令周期内，具体执行哪些机器周期，取决于指令的种类，但规定任何指令周期中的第一个机器周期总是FETCH周期。

处理器执行的机器周期，由每个机器周期第一状态期间发出的8位周期码来区别。在SYNC的时间间隔内要设置的周期信息出现在8080的数据总线上 (D_0～D_7)。这个数据需保存在锁存器中并用来产生外电路的控制信号。表2-1指出周期信息的正真值在处理机数据总

线上的分布。

周期信号主要是为外部电路的控制而提供的。各个周期位的逻辑含义不是为区别机器周期而是为了接口简单。后面将看到某些处理器机器周期只靠单个周期位就可唯一地确定了，而不用其他位。举例来说 M_1 周期位(D_5)明确地确定 FETCH 机器周期。相反 STACKREAD 周期则同时由 STACK 和 READ 信号来确定。机器周期的区分码对研制系统时的检测和排除错误阶段也是有用的。表 2-1 列出对各种型式的机器周期所输出的周期位。

2．状态转换的时序

在一个指令周期之内每个机器周期包括 3 至 5 个工作状态（称作 T_1，T_2，T_3，T_4，T_5 或 T_W）。实际用多少个状态决定于正被执行的指令和指令周期里面的具体机器周期。在图 2-4 的状态转换图中展示一个机器周期的过程。说明 8080 是怎样从一个状态前进到另一个状态。图上还展示出机器周期内 READY（准备）、HOLD(保持)及 INTERRUPT(中断)线的信号是怎样被采访的，以及在这些线上什么样的条件可能修改基本的变换时序。在现在的讨论中我们只述及基本时序和READY的功能。HOLD 及 INTERRUPT 的功能将在后面讨论。

8080 CPU 不是在每个状态期间发送一个“状态控制”输出信号来直接指示它的内部状态，而是提供直接控制的输出信号（INTE, HLDA, DBIN, $\overline{WR}$ 及 WAIT）供外部电路使用。

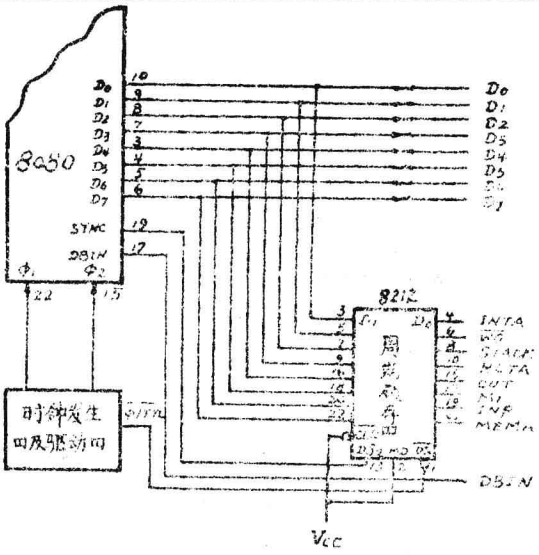
前面已提到在每个机器周期中 8080 至少要经过 3 个状态，由 ϕ_1 时钟相邻的两个低至高的跳变来确定每个状态。图 2-5 展示一个典型的 FETCH 机器周期中的定时关系。每个状

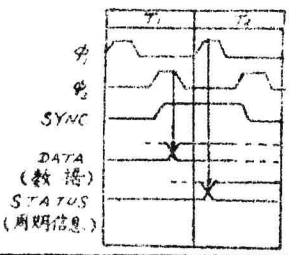
表 2-1 8080 周 期 位 定 义

为完满地执行 8080 的指令，要求有 1 至 5 个机器周期。8080 在每个机器周期开始(在 SYNC 期间)送出 8 位周期信息码到数据总线。下面列表说明信息码含义。

周期信息码定义

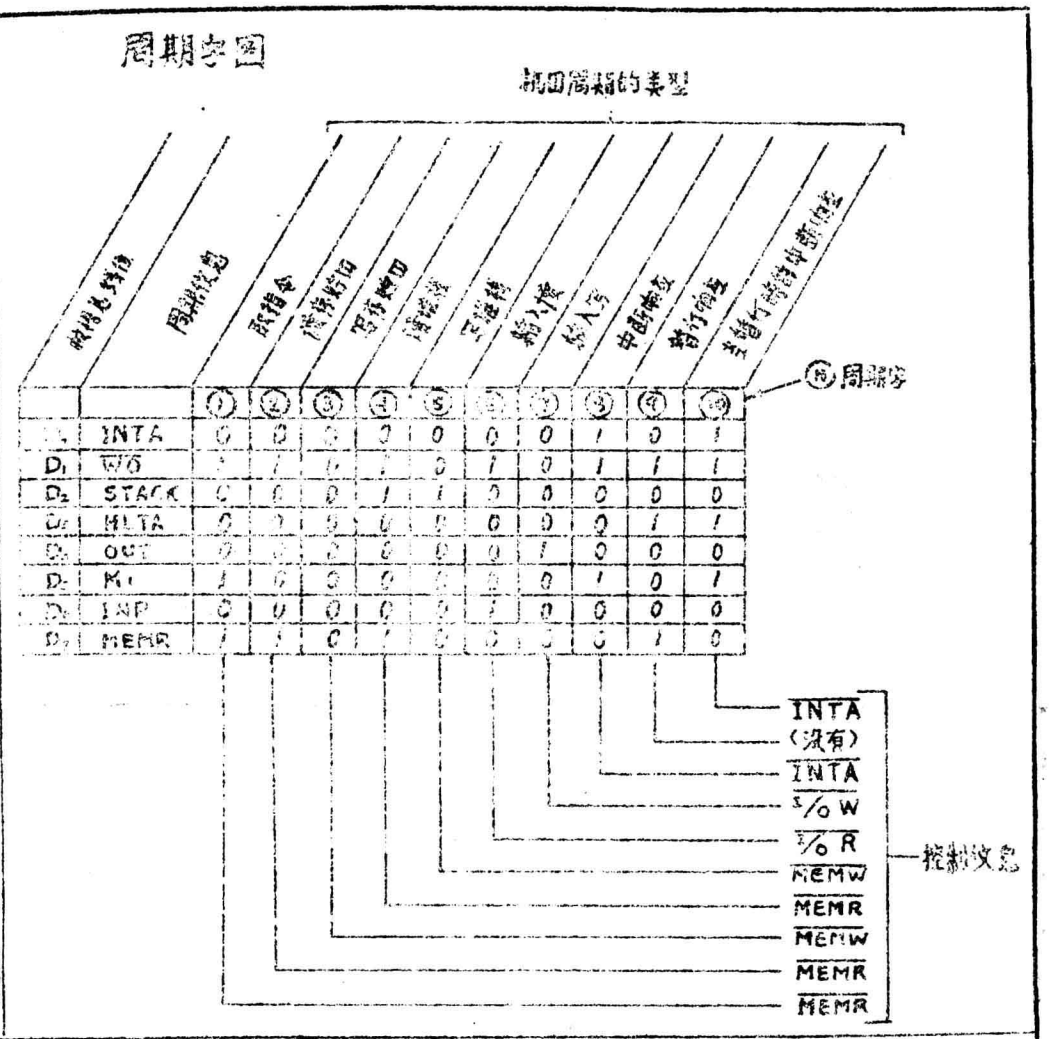
符号	数据线的总位	定义
INTA*	D_0	中断申请的响应信号。该信号在DBIN有效时，数据总线许可输入用来门控RST指令放到数据总线去。
$\overline{WO}$	D_1	指明本机器周期中的操作是存储器写(WRITE)或输出(OUTPUT)的功能。此时($\overline{WO}$=0)，否则将执行存储器读(REVD)或输入(INPUT)操作。





续表 2-1

符号	数据总线的位	定义
STACK	D_2	指明地址总线上拥有从堆栈指示器来的堆栈地址。
HLTA	D_3	暂停(HALT)指令的响应信号。
OUT	D_4	指明当WR(写)有效时，地址总线上拥有输出设备地址，而数据总线上则拥有输出数据。
M_1	D_5	这个信号指明CPU处在一条指令的取第一个字的周期。
INP*	D_6	指明地址总线拥有输入设备的地址，当DBIN有效时，输入数据被放到数据总线上。
MEMR*	D_7	指明数据总线将用于读存储器数据。



* 这3个周期位用来控制数据通向8080的数据总线。

数据总线位	周期信息	机器周期的类型 ① 取指令	② 存储器读	③ 存储器写	④ 堆栈读	⑤ 堆栈写	⑥ 读入输	⑦ 输出写	⑧ 中断响应	⑨ 暂停响应	⑩ 中断响应时
		N周期字									
D_0	INTA	0	0	0	0	0	0	0	1	0	1
D_1	$\overline{WO}$	1	1	0	1	0	1	0	1	1	1
D_2	STACK	0	0	0	1	1	0	0	0	0	0
D_3	HLTA	0	0	0	0	0	0	0	0	1	1
D_4	OUT	0	0	0	0	0	0	1	0	0	0
D_5	M_1	1	0	0	0	0	0	0	1	0	1
D_6	INP	0	0	0	0	0	1	0	0	0	0
D_7	MEMR	1	1	0	1	0	0	0	0	1	0