計算机組成原理讲授提綱

（下　册）

福州大学计算机系电子计算机教研室

一九八四年九月

第六章 指令系统

计算机是执行算法的机器，而一个算法总是由若干运算操作所组成。因此计算机必须执行若干种运算操作。我们把计算机所能执行的运算操作叫做计算机的指令。一个计算机所能执行的指令的集合就叫做这个计算机的指令系统。

显然，一个特定的计算机只能执行自己的特定的指令系统中的指令。也就是说它只"认识"并解释它自己的指令。因此，指令系统与高级语言不同，它只是一个特定的计算机的机器语言。我们说的机器语言，就是一个特定的计算机所能"认识"，"理解"的语言，包括特定的指令系统（指令格式）和特定的数据格式。一台没有任何系统软件的计算机是一个计算机系统的"硬核"。它只能运行机器语言。正因如此，任何高层次的计算机系统，尽管从用户角度看来，它可以运行某种通用的高级语言，但根归到底，高级语言还是通过解释程序或编译程序变为机器语言（指令和数据）而在硬核内实现运算。所以说到底，人们总还必须和机器语言打交道，特别是开始建立计算机系统软件之时，尤其必须直接和它打交道。因此我们必须了解一台计算机的指令系统。

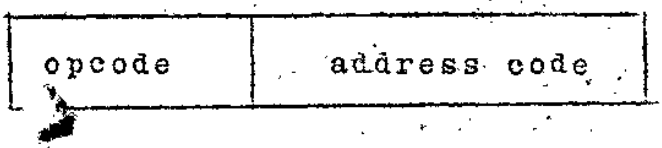
§6·1 指令格式

一个指令首先就是用以说明进行什么运算操作的语句。例如对一个数据执行相加等等。这样一个指令就需要说明两个方面的事情：

（1）执行什么操作。

（2）对什么（数据）进行操作。

现代计算机的一个极卓越的设计思想就是把指令“代码化”，也就是用一个数据表示一个指令。因此一个指令就可以用两个数码表示，第一个数码表示指令执行什么操作，叫做操作码，opcode或简称为OP。第二个数码用以表示要进行运算操作的数据，叫做地址码——address code，或简称ad。由op和ad这两个数码组成的数据就是一个代码化的指令，或指令码——Instruction code，这样指令码的一般格式就是



当然，这两个数码颠倒左、右并没有原则意义，只是习惯多把opcode放在最左，而把address code放在右边。

这样，一般情况是指令码与数据并没有什么区别。当计算机在取指令周期时，所取的数据就把它作为指令给予识别，而当存取数据周期时，所处理的就是数据。由此可见，我们完全可以把指令作数据进行必要的运算，也就是说，我们完全可以设计这样一个程序，它自己“改造”自己。而这将会使人们得到一些意想不到的结果。

下面继续讨论指令格式问题，一个很明显的原则是指令码的长度应和计算机（ALU）所处理的数据的字长相当，它们可以相等，也可以是某个整倍数。但不应该不可约，例如数据字长是32bit，而指令码长度为22bit，这就很不方便了，最方便是两者取一致，但也常取二倍或四倍差。以字节为最小单位，例如，指令码可取8 bit，16 bit，24 bit 或 32 bit，而数据长度也可以按上述的位数选定，例如我们可以选指令码为24 bit，字长为

48 bit则很方便，若选字长为32 bit则又很不方便了。所以指令格式的选择在一定程度上受到了计算机ALU的处理字长所制约，虽然有时并不是决定性的。

opcode 代表运算操作的性质，例如浮点加法，逻辑乘法等等。显然opcode的位数决定了这个指令系统指令类型的最大数目。例如opcode有8 bit，那么这个指令系统最多只能容纳256种操作。如果opcode有6 bit，那么这个指令系统就仅能容纳64种不同的操作了。关于合理选择opeode长度问题留到§5·3再具体讨论。现在讨论address eode的问题。

address code代表参与运算的数据是什么？或者更精确地说，来自何处的数据。这是因为参与运算的数据绝大部分都存贮在存贮器之中，而最方便地指定存贮在存贮器中的数据的方式，就是给出存贮这个数据的地址。因此就把这个代码叫做地址码。但是在指令码中的地址码对某些指令说来，并不就是地址，而可以有其它的含义。例如，有时可以就是参与运算的数据本身，但是我们还是把它叫做指令中的地址码部分。只是，此时的地址码不是代表地址，而是代表一个参与运算的数据罢了。

一般地说，一个算数运算（或是逻辑运算）总是涉及三个数据，其形式为

$$Z \leftarrow (x)\ \vartheta\ (Y)$$

这里ϑ是一个算符，X，Y是两个参与运算数据的地址，而结果存入地址Z中去。于是一般地说，一个指令码的地址码应有三个地址码，即第一地址（ad_1），第二地址（ad_2）和第三地址（ad_3），指令的形式是

$$ad_3 \leftarrow (ad_1) \vartheta (ad_2)$$

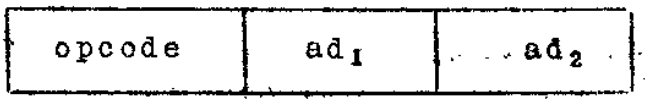
具有三个地址码的指令格式叫做三地址的指令。显然，并不是所有操作都需要三个地址。例如对某个数据作逻辑反，则最多只需要二地址

$$ad_2 \leftarrow (\overline{ad_1})$$

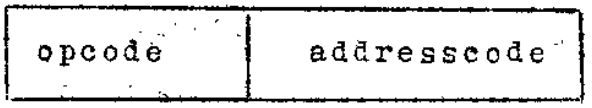
因此也可以组织二地址的指令格式。其运算形式为：

$$ad_2 \leftarrow (ad_1)\ \vartheta\ (ad_2)$$

这时的指令码中只需要有两个地址码，即



甚至可以组织一地址的指令格式，或叫做单地址指令格式，



其运算形式为

$$AR \leftarrow (AR)\vartheta(addresscodo)$$

这里AR是ALU中的累加寄存器，因为它唯一被指定了，所以在指令格式中并不须要明显的指出。

最后，对于算数和逻辑运算甚至可以有“无地址”或“零地址”的指令格式。这就是在第四章中所描叙过的堆栈型的ALU。这时运算始终是在ALU中的堆栈的栈顶前两个数之间进行，结果仍在栈顶中。显然，为了执行把存贮在存贮器中数据送入堆栈，或把运算结果移入存贮器，这样一些操作的指令仍需有地址（单地址）。所以

虽然人们把堆栈型运算的指令系统叫做“无地址指令系统”，但并且不是说，在这个指令系统中任何种指令都无地址，而仅是涉及堆栈的运算才是“无地址”的。

在早期的计算机设计工作中，采用怎样的地址格式的指令系统曾是一个有重大的争论问题，这是因为当时存贮容量很小，节约存贮容量是很重要的。从这个观点说，单地址指令的长度比较短，所以常常被认为是最优的。另一方面早期的计算机、硬件费用高，通常一台计算机的指令系统只采用一种地址格式，以求得结构简单、节约。这时单地址指令格式确有优越之处。所以早期的计算机的指令系统以单地址较多。时至今日，单地址的指令系统仍占有优势。但是现在已不拘于一个指令系统中只用一种地址格式了，而常是混合的，即在一个指令系统中即有单地址指令也有多地址指令，甚至无地址指令。这样做法自然使结构复杂一些，但却增加了灵活性。这样作法是基于。

（1）指令的代码长度不固定，一般按字节计算，例如美国IBM公司（International Business Macnine）生产的有名的IBM 370系列的计算机，其指令系统就有二字节（16 bit），四字节（32 bit）和六字节（48 bit）这样三种长度的指令。这时指令长度与数据的字长不一致，而地址就不能按数据的字长编号，而以字节为存贮器地址编址的基本单位。

（2）在ALU中设置了相当数量通用寄存器，这样不仅可以用存贮器中的数据进行运算，并且（更多）利用寄存在通用寄存器中的数据进行运算，此时，为了指定通用寄存器的地址（编号）只须很少的代码即可，例如为了指定16个通用寄存器只需四位码，三个地址码也只要8 bit。这时的指令长度并不大。

由于以上的情况，现代的计算机的指令系统很少有单一的地址格式了。不能象早期的计算机那样明确地说这台计算机是单地址，或多地址的计算机了。当然这时在整个指令码中要有说明这条指令的长度的标志。例如在IBM 370中的op有八位，其中最左两位就是说明指令的格式和长度的，它一共分五种格式：（RS和SI表示相同）

op 左两位	指令格式	指令长度
00	RR	16
01	RX	32
10	RS(SI)	32
11	SS	48

因为op共有八位，去掉最左标志格式的两位，还余六位，这就是说对每一种指令格式可以组织64种操作指令。（一共有256种操作指令。事实上IBM 370的指令系统并没有用完这个可能。而只有183种指令，这样作还有一个好处，即为今后改进扩充指令系统提供可能）。

将在§5·4地址变换中进一步讨论指令格式问题。

§6·2 指令系统

由于计算机是一个面向算法的机器，因此它的指令系统直接影响它的计算能力。早期的计算机，由于硬件价格很贵，所以一般说来，它的指令系统比较简单。但是随着人们对计算机的能力[illegible]

愈来愈高，而硬件价格迅速降低，所以指令系统逐渐趋于复杂。当然，还有另一种趋势，就是微型计算机又把指令系统简化了，以适应LSI的能力。但是不管怎样，一个计算机的指令系统还是都应满足一些基本要求的。这些要求是：

（一）指定系统应是完备的，即完备性。所谓完备性是说，在一个可用的存贮空间内，对于任何一个能行的算法说来，所提供的指令系统都是足够使用的。为什么要提出在一个"可用的"存贮空间内"（即现实的存贮空间）这个条件呢？因为如果对存贮空间不作限制，那么极小的指令系统（也就是只有很少的可用的指令）也是完备的。一般说来，至少要有以下五类操作的指令：

（1）数据传达指令：把这类指令叫做"传送"其实并不恰当，因为一个数据由某一存贮单元或一个寄存器"传送"到另一个存贮单元或另一个寄存器去，只是把其内容（的"复本"）赋值与另一处，而原来的内容并未消灭。凡属指令中出现的可传送的地址（存贮器或寄存器）都是面向用户的。还有一些地址（存贮器或寄存器）虽然在硬件中，但在指令中都不出现，那么用户就不在直接使用它们。它们对用户似乎并不存在。我们把传传送的地址叫做"源地址"而把接受数据的地址叫做"目的地址"。很明显，对于多地址格式的指令系统说来，显示源地址和目的地址没有什么困难，而对单地址指令说来，实际上是以累加寄存器（AR）作为隐含地址，任何数据的传达，都要经结AR，所以这时须要两条指令来完成传送，即

AR←（源地址）

（目的地址）←AR

（2）算数运算指令和逻辑运算指令。这些指令直接对数据进

行计算。虽然从完备性角度说，只要有定点数的加法和求补（补码系统）就可以完成一切浮点和定点的四则运算，但是为了方便通常还要有定点数运算指令和浮点数运算指令，而逻辑运算当然是对布尔量的运算了。另一方面，所有算数运算和逻辑运算指令除了给出直接的计算结果而外，还要给出与这个结果有关的信息，诸如，结果为正、负、全零、是否溢出等等。所有这些在第四章中已作了讨论。通常这类指令还包括各种移位操作的指令。

（3）程序控制指令，这是一组可改变计算程序的顺序的指令，有时叫做转移指令，分支指令或跳跃指令，属于这类指令的还有"调用"指令。我们将在下节中仔细讨论这类指令，这里只是指出这类指令是一组十分重要的指令。

（4）输入／输出指令（I／O）显然这组指令是绝对必须的，在现代的计算机系统中I／O指令常常并不直接由主机实现，而是由主机把数据送入或接受）通道；然后由通道再组织数据的输入或输出，所以从主机的指令系统看来，I／O的指令很简单，可是事实上，通道中却有很复杂的I／O指令系统。

（5）停机指令　这当然是必须的，但是就是停机指令也常很复杂，因为除了绝对停机而外，还应有跟踪停机之类的指令，这类指令常为调试程序而设，即在调试过程中，设有若干跟踪停机点而在程序正式运行时，它不再执行停车，而作为"空指令"（即无操作）而跳过。同样很多指令系统中还设有空指令，等待指令等等。（一个等待指令使计算程序停下来，而处于等待状态直到由外界，可以是人或联结的某个设备来信号后，再重新启动）。

事实是，以上分的五种指令类型并不全是相互独立的。它们在一定条件下可以相互代用和取舍，最明显的是传送指令就可以由算

数指令（或逻辑指令）所代替。所以我们说指令系统的完备性，必须和下述的有效性相联系。

在讨论指令系统的完备性之时，还要指出，还有两类指令，这两类指令是用户看不到的，一是所谓"特权指令"这类指令在用户指令表上是没有的，它是一组（一般很少几条）执行特殊操作的指令，只为系统软件来使用。另一是所谓"隐指令"，这类指令是系统软件也看不到的，是硬件构成的，典型的隐指令是对中断处理的操作，关于这个问题在第六章再作讨论。

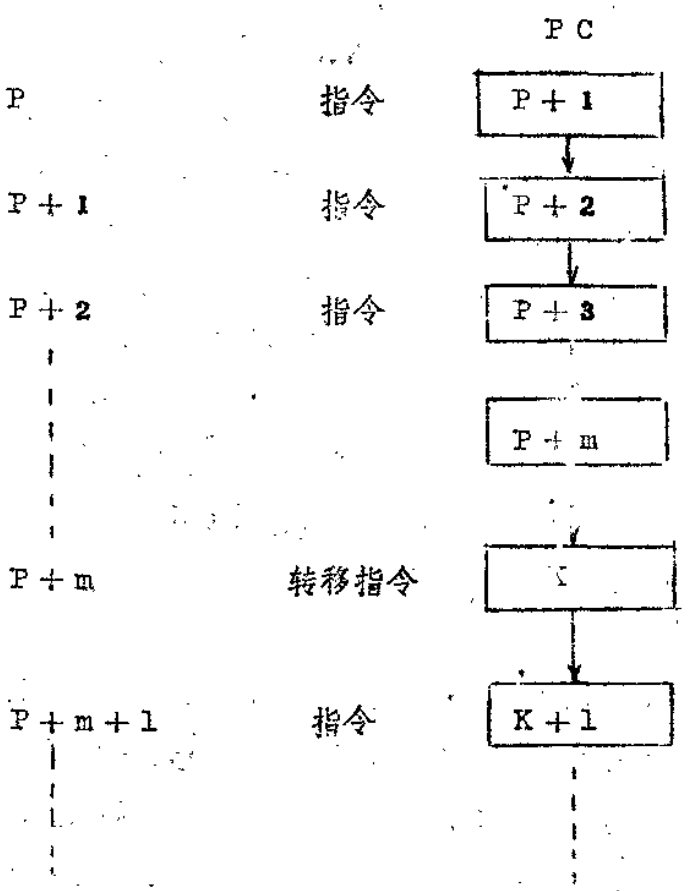
（二）指令系统的有效性，所谓有效性就是一台计算机的指令系统应该高效率地进行运算，这个高效率归根到底可归结为解题的速度高（用的时间少）。当然这是一个十分复杂的问题。我们绝不能简单地以为ALU的速度高，就是有效性好，也不是执行一条指令快就是有效性好，所以在指令系统中指出这个问题，这是因为它涉及到指令系统的全部，当然一个更完备的指令系统就有更好的有效性。也就是说指令系统中应包括一些强有力的指令。举例说，一个指令系统如果包括多种移位指令和字符处理指令，那么对数据处理就有较高的有效性。特别是对程序分枝的操作的指令对有效性有很大的影响。还要指出，特权指令组固然数目不大，但它是系统软件所用，所以强有力的特权指令可以降低系统开销而使指令系统的有效性提高。

（三）指令的格式，应与数据格式相应，这样作法将不但使存贮器的格式易于处理，并且也便于对程序（由指令组成）进行加工。因此在选择指令格式之时，必然受到数据格式的影响。例如，一台计算机，其基本字长是三十二位二进制数，那么指令格式可选长指令为三十二位，短指令为十六位二进制。这时就只能在这个字长条

件下选择地址格式了。

§6·3 程序控制指令组

除了程序控制指令而外，上节所讨论的其它类型指令，其意义都很明显，特别是算数——逻辑操作指令在第四章中已作了相当详尽的讨论。所以我们只将着重讨论程序控制指令的意义和功能。为此我们考虑一个运行中的程序P。这个程序开头存贮在地址P，顺序存下去。到P＋m地址，存入了一种程序控制指令：



无条件转移指令。这样在这个程序运行时程序计数器（指令计数器）的内容，依次是p＋1，p＋2，p＋3，……在执行了p＋m－1号地址的指令之后，PC中的内容就是p＋m，这就是说下一条要执行的指令存贮在p＋m单元中，现在从p＋m存贮单元中取出指令，这是一条无条件转移指令。它所执行的操作是把PC的内容修改为K。这个k值是由转移指令给出的，转移指令的基本格式也就是一个操作码opoode表示转移操作。而地址部分就是对PC赋值的那个地址值（K）。

转移操作码　　转移地址

一般说来这是一个适合单地址指令系统的指令。由于这时PC内容已是K，所以下一条所执行的指令并不是p＋m＋1地址内存贮的指令，而是要执行在k地址中存贮的指令，接下去的是执行存贮在（k＋1）地址的指令，可以看出，所谓转移指令就是改变了程序执行的顺序。静态地看一个程序，也就是直接读出一个程序，那么它是顺序写下去的，可是动态地看，即程序在运行时，程序遇到转移指令之时，其顺序都改变了，在转移指令之后的程序段（即p＋m＋1地址后的程序段）并不运行，而却转为运行存贮在k地址以及其后的程序段。所以一程序的静态与其动态并不一致。用箭头作为程序在运行时（动态）的顺序，则有转移指令之时改变其运行顺序如图5·1所示。

有时把转移指令叫做GOTO指令。一个程序中GOTO指令使它变的静态与动态不一致，如果一个程序中有大量GOTO指令，那么

将使程序很难清晰地读通。所以现在的程序设计中尽量少用GOTO指令。

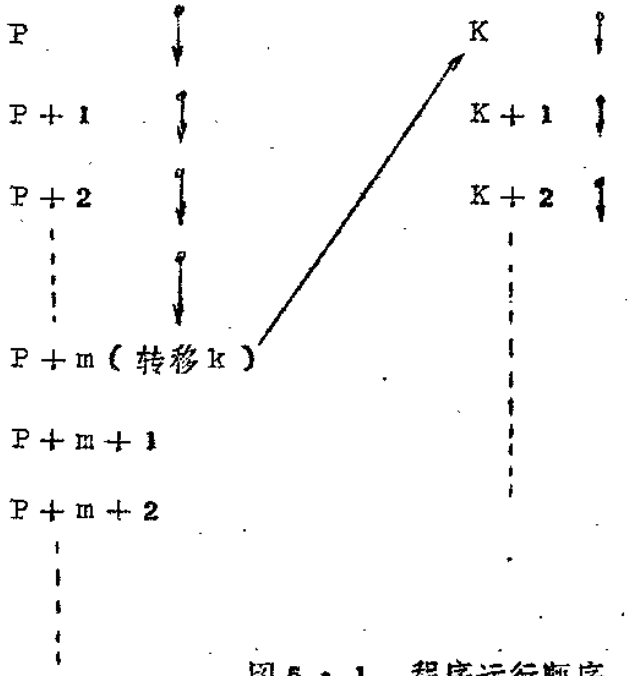


图 6 · 1　程序运行顺序

§6 · 3 · 1　条件转移指令

原则上说一个完备的指令系统中可以不包括上述的（无条件）转移指令。这并不是因为GOTO指令会使程序难读而不用。而是因为如果有条件转移指令，那么就可以用它（极简单地）代替转移指令。所谓条件转移指令，其实就是一种转移指令，只是它仅当满足一定条件时才执行转移操作，否则只相当于一个空指令。所谓条件，一般是指上次运算的结果，即根据运算状态字作为条件，通常有四种可以表征转移的条件，即有四种条件转移指令：

（1）结果为正，（对单地址指令系统有AR＞0）则转移，否则不转移。

（2）结果为负（即AR＜0）则转移，否则不转移。

（3）结果为零（即AR＝0）则转移，否则不转移。

（4）比较结果相等（即AR～（M））则转移，否则不转移。

当然，从指令系统的完备性说来，其实一个指令系统中只须要有上述若干种条件转移指令中的一条（早期计算机指令系统较小，通常优选第二种即逢负转移）即可。但是从指令系统的有效性角度看，至少（2），（3），（4）这三种条件转移指令能使程序运行更为有效。所以在现在的计算机指令系统中，一般有三、四种条件转移指令。因此相应的条件控制字也有若干位了。

条件转移是一种非常重要的指令。可以说，如果指令系统中没有条件转移，计算机实际上不可能完成计算任务。为什么呢？只是单纯地有很快的计算速度，比如说每秒执行几百万次、几千万次运行操作。并不能说明它能很快的计算题目，因为每个运行一操作都由相应的指令指挥，每秒几百万、几千万次运行操作就是说计算每秒执行几百万、几千万条指令。为了存贮这个每秒执行的指令就几百万，几千万个存贮地址，何况只是把几百万、几千万的指令输入计算机要用多少时间哩！更不用说要手写出要多少时间哩！校核更是不可能的。幸好，所有的能行算法都是要重复地，反复地执行某些程序，所以一段程序尽管写起来不长，但在运行时可在反复执行几千次几万次，因此尽管完成一个算法的程序可能须要执行千万，万万条指令。可以这个程序的静态，即书写起来，也许只有几十条，几百条指令。于是，不但是可以存贮在存贮器中（因为并不太多），并且也是可写，可读，可以输入到计算中去。因此，反复执行一段

程序是计算机可以成为现实的决定性因素。反复执行一段程序叫做程序的“循环”，或循环程序，在第二章中对它的性质已作了一些描叙。但是如何控制循环呢？这就需要使用条件转移指令了。所以从指令系统的完备性说，条件转移指令是必需的。

循环程序能重复使用一个程序段若干次，这是完全必要的。但又不能无即地重复下去（在程序有错误时，可能出现这种情况，叫做“死循环”遇到这种情形必须强迫停机中断，由人来寻找和排除故障）。条件转移指令，一方面提供程序循环的操作，另一方面又提供了脱离循环（结束循环）的条件。

这种脱离循环的条件叫做循环控制。从循环控制角度看可划分为两大类：第一：在程序中不明显指出循环的次数，而只是指明循环控制的逻辑条件，这种循环控制通常叫做“IF”型的循环控制。第二：在程序中明确指出这段程序运行时的循环次数，通常叫做计数型循环，或叫做“DO”型的循环控制（也有时叫做“LOOP”）

§6·3·2 转移指令的逻辑实现

前面我们只讨论了各种转移指令的功能，面没有讨论它们如何在计算机中实现其功能，本节（以及下节）将讨论它们的逻辑实现问题。

大家都知道，在一个计算机中程序运行的顺序是由在控制器中的一个计数器PC，叫做程序计数器或指令计数器所决定的。PC的内容就是当前执行的指令的地址。指令执行后，PC加1，因此确定了下一条指令地址，根据PC的内容再取指令进行操作。所谓转移指令，就是把转移指令中的地址部分送入PC，从而改变了程

序执行的顺序。所以从原则上说，转移指令是一种操作极简单的指令。就是把指令寄存器ＩＲ（从存贮器取出指令寄存在指令寄存器之中）的地址部分传送到ＰＣ中去。如果是条件转移指令，那么这个传送还要由运算控制字Ｗ所控制。图６·７就是这个逻辑设计。

ＰＣ是一个ｎ位的计数器（2^n是存贮器的容量）。ＩＲ的地址码也有ｎ位，ＰＣ还可以通过接收一个时钟脉冲而把ＩＲ的地址码送入。这个ＣＰ是由控制器来的传送脉冲所产生的，从图中很容易看出，这里可以有五种转移，即无条件转移，执行这条指令时，无条件转移的控制线出现１，其余自然是０，那么ＩＲ的Address必然送入ＰＣ，从而实现了转移。其余四种都是条件转移，仅当出现其中之一的指令，并且相应的运算控制字Ｗ的位为“１”，才执行Adrresscode送入ＰＣ。否则ＰＣ不变，因此也就实现了条件转移。

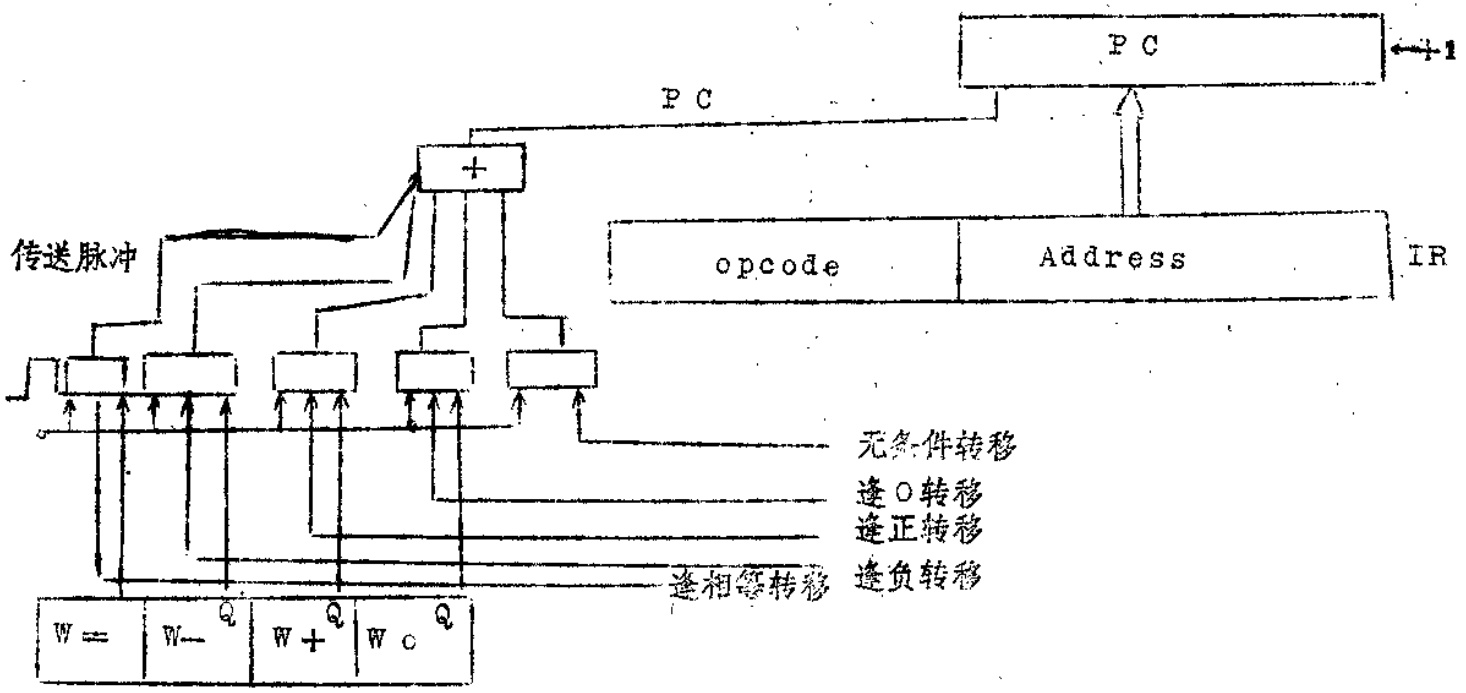


图 6 • 7 转移指令的逻辑图