第3章

chapter 3

指 令 系 统

【总体要求】

- 掌握指令格式。
- 理解指令字长的设计方法。
- 掌握指令地址的简化方法。
- 掌握操作码的扩展方法。
- 掌握指令和操作数的寻址方式。
- 理解指令系统的设计原则。
- 了解指令的功能及类型。
- 了解两个典型的指令系统。

【学习重点】

- 指令字长的设计、指令地址的简化、操作码的扩展。
- 指令和操作数的寻址方式。

通过程序，计算机可以完成各种工作，程序是由一系列的指令构成的，指令是程序可执行形态的基本单元，是执行加、减、移位等基本操作的命令，由一组二进制代码表示。所谓指令系统就是一台计算机所能执行的各种不同类型指令的总和，即一台计算机所能执行的全部操作。不同的计算机指令系统所包含的指令种类和数目不同，但是一般都会包含算术运算指令、逻辑运算指令、传送类指令、程序控制类指令等。指令系统是体现一台计算机性能的重要因素，它的格式与功能不仅直接影响到机器的硬件结构，而且也直接影响到系统软件，影响到机器的适用范围，所以说指令系统是软件和硬件的主要界面。本章主要介绍指令系统的有关知识。

3.1 指令格式

3.1.1 机器指令格式

指令就是要计算机执行某种操作的命令，由操作码和地址码两个部分构成，指令的基本格式如图 3-1 所示。

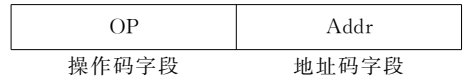


图 3-1 指令格式示意图

其中，操作码说明操作的性质及功能，地址码描述该指令的操作对象，由地址码可以给出操作

数或操作数的地址及操作结果的存放地址。

1. 操作码

指令系统的每一条指令都有一个操作码，用来表示该指令应进行什么性质的操作，如加、减、移位、传送等。操作码字段不同的编码表示不同的指令，每一种编码代表一种指令。组成操作码字段的位数一般取决于计算机指令系统的规模。操作码的位数越多，所能表示的操作种类就越多。例如，若操作码有 3 位，则指令系统只有 8 条指令；若操作码有 5 位，则该指令系统中有 32 条指令。

2. 操作数或操作数地址

操作数即参与运算的数据。少数情况下，在指令中会直接给出操作数，但是大部分情况下，指令中只给出操作数的存放地址，如寄存器号或主存单元的地址码。一般地，将内容不随指令执行而变化的操作数称为源操作数，内容随执行指令而改变的操作数称为目的操作数。

3. 结果存放地址

结果存放地址是最后操作结束时，用来存放运算结果的地址，如存放在某个寄存器中或主存中的某个单元中。

4. 后续指令地址

程序是由一系列的指令构成，当其中的一条指令（现行指令）执行后，为了程序能够连续运行，指令中需要给出下一条指令（后续指令）存放的地址。将存放后续指令的主存储器单元的地址码称作后续指令地址。

大多数情况下程序是顺序执行的，所以可以在硬件上设置一个专门存放现行指令地址的程序计数器 PC，每取出一条指令时，PC 自动增值指向后续指令的地址。如现行指令占 1 个字节的存储单元，则取出现行指令后，PC 的内容加“1”即指向后续指令的地址；若现行指令占 n 个字节的存储单元，则取出现行指令后，PC 的内容加“n”便可以使 PC 指向后续指令的地址。

后续指令地址是一种隐含地址，隐含约定是由 PC 提供的，在指令代码中不会出现，因此可以有效地缩短指令的长度，而且可以根据结果灵活转移。将这种以隐含方式约定、在指令中不出现的地址称为隐地址，指令代码中明显给出的地址称为显地址。使用隐含地址可以减少指令中显地址的数目，缩短指令长度。

3.1.2 指令字长

指令字长指的是一个指令字中所包含的二进制代码的位数。由于指令长度＝操作码的长度＋地址码的长度，所以各指令字长会因为操作码的长度、操作数地址的长度及地址数目的不同而不同。

指令字的位数越多，所能表示的操作信息及地址信息就越多，指令功能越丰富。但是指令位数增多时，存放指令所需的存储空间就越多，读取指令时所花费的时间也会越长，此外，指令越复杂，相应的执行时间也就越长。若指令字长固定不变，则格式简单，读取执行时所需的时间会较短。因此，对指令字长有两种不同的设计方法：变字长指令和定字长指令。

1. 变字长指令

变字长指令结构中，各种指令字长度随指令功能而异，“需长则长，能短则短”，结构灵

活，能充分利用指令长度，但指令的控制较为复杂。

由于主存储器一般是按字节编址（以字节为基本单位）的，所以指令字长通常设计为字节的整数倍，例如 PC 的指令系统中，指令长度有单字节、双字节、三字节、四字节等。若采用短指令，可以节省存储空间、提高取指令的速度，但有很大的局限性；若采用长指令，可以扩大寻址范围或者带几个操作数，但是存在占用地址多、取指令时间相对较长的问题。若考虑将二者在同一机器中混合使用，则可以取其长处，给指令系统带来很大的灵活性。

为了便于处理，一般将操作码放在指令字的第一个字节，当读出操作码后马上就可以判定该指令是双操作数指令，还是单操作数指令，或是零地址指令，从而确定该指令还有几个字节需要读取。在主流 PC 和传统的大、中、小型计算机中仍然广泛采用的是复杂指令系统（CISC），相应地采用的指令格式为变字长指令。

2. 定字长指令

定字长指令结构中的各种指令字长度均相同，且指令字长度不变。采用定字长格式的指令执行速度快，结构简单，便于控制。

为了获得更快的执行速度，出现了一个非常重要的发展趋势，即采取精简指令系统 RISC，相应地采用固定字长指令。逐渐成熟的精简指令系统技术被广泛地应用于工作站一类的高档计算机，或者采用众多的 RISC 处理器构成大规模并行处理阵列，而且 RISC 技术的发展对 PC 的发展也产生了重大影响。

3.1.3 指令的地址码

指令中的地址码字段包括操作数的地址和操作结果的地址，在大多数指令中，地址信息所占的位数最多，所以地址结构是指令格式中的一个重要问题。由指令格式可知，对于常规的双操作数运算来说，指令中应该包括 4 个地址：两个操作数的地址、存放结果的地址及后续指令地址。明显可看出，这种四地址结构的指令所需的位数太多，所以采用隐含地址以减少指令中显地址的数目即简化地址结构。按照指令中的显地址的数目，可以将指令分为三地址指令、二地址指令、一地址指令及零地址指令。指令中给出的各地址 Ai 可能是寄存器号，也可能是主存储器单元的地址码，(Ai)表示 Ai 中的内容，(PC)表示 PC 中的内容。

1. 三地址指令格式

指令格式如下：

OP	A1	A2	A3
操作码	操作数 1 地址	操作数 2 地址	结果存放地址

指令功能：(A1) OP (A2)→A3

(PC) + n→PC

功能描述：3 个地址均由指令给出，指令要求分别按 A1 和 A2 地址读取操作数，按照操作码 OP 进行有关的运算操作，然后将运算结果存入 A3 地址所指定的寄存器或主存单元中；现行指令读取后，PC 的内容加 n，使 PC 指向后续指令地址。

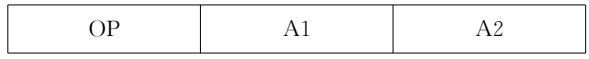
例如，要完成“加”操作(X) + (Y)→Z，使用三地址指令时，可使用下面指令：

```
ADD X,Y,Z;
```

注意：当从寄存器或是存储单元读取指令或数据后，原来存放的内容并没有丢失，除非有新的内容写入寄存器或是存储单元。所以在三地址指令执行之后，存放在 A1 和 A2 中的原操作数还可以被再次使用，该指令也可以被再次调用。

2. 二地址指令格式

指令格式如下：



指令功能：(A1) OP (A2)→A1

(PC) + n→PC

功能描述：指令要求分别按 A1 和 A2 地址读取操作数，按照操作码 OP 进行有关的运算操作，然后将运算结果存入 A1 中替代原来的操作数；现行指令读取后，PC 的内容加 n，使 PC 指向后续指令地址。

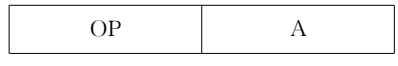
运算后，由 A2 提供的操作数仍然保留在原处，称 A2 为源操作数地址；由 A1 提供的操作数被运算结果替代，即 A1 成为存放运算结果的地址，被称为目的操作数地址。采用隐含约定，三地址指令中存放结果的地址被简化，减少了指令中显地址的数目。

例如，要完成“加”操作(X) + (Y)→Z，使用二地址指令时，可使用下面指令：

```
ADD X,Y;
MOV Z,X;
```

3. 一地址指令格式

指令格式如下：



一地址指令中只给出了一个操作数地址 A，所以需要根据操作码的含义确定其具体形态。一地址指令有两种常见的形态：只有目的操作数的单操作数指令和隐含约定目的地址的双操作数指令。

(1) 只有目的操作数的单操作数指令

所谓单操作数指令是指指令中只需要一个操作数，如加 1、减 1、求反、求补等操作。对于单操作数指令，按地址 A 读取操作数，进行操作码 OP 指定的操作，将运算结果存回原地址。

指令功能：OP (A)→A

(PC) + n→PC

(2) 隐含约定目的地址的双操作数指令

因为指令中只给出了一个操作数地址 A，对于双操作数指令来说，另一个操作数则采用“隐含”方式给出。若操作码含义为加、减、乘、除之类，则说明该指令是双操作数，按指令给出的源操作数地址读取源操作数，目的操作数隐含在累加寄存器 AC 中，运算后的结果存放在 AC 中，替代 AC 中原来的内容。累加寄存器 AC 通常简称为累加器，其功能是当运算器的算术逻辑单元 ALU 执行算术或逻辑运算时，为 ALU 提供一个工作区，暂时存放 ALU 运

算的结果信息。

指令功能：(A) OP (AC)→AC

(PC) + n→PC

例如，要完成“加”操作(X) + (Y)→Z，使用一地址指令时，可使用下面指令：

```
LDA X;
ADD Y;
STA Z;
```

4. 零地址指令格式

指令格式如下：

操作码

零地址指令中，只有操作码而没有显地址。使用零地址指令的情况有以下 3 种：

(1) 不需要操作数的指令

不需要操作数的指令如停机指令和空操作指令。执行空操作指令的目的是消耗时间达到延时的目的，本身并没有实质性的运算操作，所以不需要操作数。

(2) 单操作数指令

对于单操作数指令，采用零地址指令格式时，可以隐含约定操作数在累加器 AC 中，即对累加器 AC 的内容进行操作。

指令功能：OP (AC)→AC

(3) 双操作数指令

对于双操作数指令，可将操作数事先存放在堆栈中，由堆栈指针 SP 隐含指出。由于堆栈是一种按照“先进后出”的顺序进行存取的存储组织，所以每次存取的对象都是栈顶单元的数据，因此这种指令仅对栈顶单元中的数据进行操作，运算结果仍然存回堆栈中。

例如，要完成“加”操作(X) + (Y)→Z，使用零地址指令时，可使用下面指令：

```
PUSH X;
PUSH Y;
ADD ;
POP Z;
```

通过以上各指令的分析可以看出，采用隐含地址可以减少显地址的个数，简化指令的地址结构。一般来说，指令中的显地址数目较多，则指令的字长较长，所需存储空间较大，读取时间较长，但是使用较为灵活；反之，若指令中的显地址数目较少，采用隐含地址，则指令的字长较短，所需存储空间较小，读取时间较短，但是使用隐含地址的方式对地址选择有一定的限制。所以说二者各有利弊，设计者往往采用折中的办法。

3.1.4 指令的操作码

机器执行什么样的操作由操作码来指示。目前在指令操作码设计上主要有定长操作码、变长操作码、单功能型操作码或复合型操作码。

1. 定长操作码、变长指令码

这种设计操作码的长度及位置固定，集中放在指令字的第一个字段中，指令的其余字段均为地址码。该格式常用于指令字较长，或是采用可变长指令格式的情况，如 PC 系列机中。一般 n 位操作码的指令系统最多可以表示 2^n 条指令，如操作码的长度为 8 位，则可以表示 256 种不同的操作。

由于操作码的位数及位置固定，对于指令的读取和识别较为方便。因为所读取的指令代码的第一个字段即操作码，所以可以判断出该指令的类型及相应的地址信息组织方法。又由于不同的操作码涉及的地址码的个数不同，采用这种格式的指令，可以使指令的长度随着操作码的不同而变化。如加、减指令可以有三个地址码（两个操作数地址和结果存放地址），传送指令有两个地址码（源地址和目的地址），加 1 指令只需一个地址（操作数的地址），返回指令不涉及操作数，所以没有地址码。

采用定长操作码、变长指令码方式的指令操作码字段规整，有利于简化操作码译码器的设计。因为字长较长的机器不是十分在意每位二进制的编码效率，所以被广泛用于指令字长较长的大、中、超小型机中。精简指令系统计算机 RISC 中的指令较少，相应地所需的操作码也较少，因而也常用定长操作码的指令。

2. 变长操作码、定长指令码

变长操作码、定长指令码是一种操作码长度不定，但指令字长固定的设计方法，这种方式可以在指令字长有限的前提下仍然保持较丰富的指令种类。由于不同的指令需要的操作码位数不同，所以为了有效利用指令中的每一位二进制位，可采用扩展操作码的方法，即操作码和地址码的位数不固定，操作码的位数随着地址码位数的减少而增加。采用该方法时，在指令字长一定时，对于地址数少的指令可以允许操作码长些，对于地址数多的指令可以允许操作码短些。下面通过例子来具体说明这种方法。

【例 3-1】 设某机器指令长度为 16 位，包括 1 个操作码字段和 3 个地址码字段，每个字段长度均为 4 位，格式如图 3-2 所示。现在要求扩展为 15 条三地址指令、15 条二地址指令、15 条一地址指令及 16 条零地址指令。试给出扩展操作码的方案。

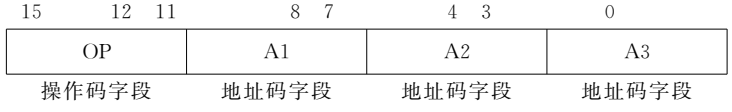


图 3-2 例 3-1 指令格式示意图

解： 4 位操作码有 $2^4=16$ 种组合（0000～1111），如果全部用来表示三地址的指令，则只能表示 16 条不同的指令。若只取其中的 15 条指令（操作码为 0000～1110）作为三地址指令，则可以将剩下的一组编码（1111）作为扩展标志，把操作码扩展到 A1，即操作码从 4 位扩展为 8 位（11110000～11111111），可表示 16 条二地址指令。同理，若只取其中的 15 条指令（操作码为 11110000～11111110）作为二地址指令，则可以将剩下的一组编码（11111111）作为扩展标志，把操作码扩展到 A2，即操作码扩展为 12 位，又可表示 16 条一地址指令。采用同样方法继续向下扩展，即可得到 16 条零地址指令。该扩展方案的示意图如图 3-3 所示。

15　12	11　8	7　4	3　0	
OP	A1	A2	A3	
0 0 0 0 …… 1 1 1 0	A1 …… A1	A2 …… A2	A3 …… A3	15 条三地址指令
1 1 1 1 …… 1 1 1 1	0 0 0 0 …… 1 1 1 0	A2 …… A2	A3 …… A3	15 条二地址指令
1 1 1 1 …… 1 1 1 1	1 1 1 1 …… 1 1 1 1	0 0 0 0 …… 1 1 1 0	A3 …… A3	15 条一地址指令
1 1 1 1 …… 1 1 1 1	1 1 1 1 …… 1 1 1 1	1 1 1 1 …… 1 1 1 1	0 0 0 0 …… 1 1 1 1	16 条零地址指令

图 3-3　扩展方案的示意图

除了以上的扩展方法之外，还有很多的扩展方法，如形成 14 条三地址指令、30 条二地址指令、31 条一地址指令及 16 条零地址指令等。实际设计指令系统的时候，应该根据各类指令的条数采用更为灵活的扩展方式。

使用操作码扩展技术的另一种考虑是霍夫曼原理，根据在程序中出现的概率大小来分配操作码，即出现概率大的指令（也就是使用频率高的指令）分配较短的操作码，而出现概率小的指令（也就是使用频率低的指令）分配较长的操作码，以此来减少操作码在程序中的总位数。所以说，操作码扩展技术是一种重要的指令优化技术，可以缩短指令的平均长度，且增加指令字表示的操作信息，广泛应用于指令字长较短的微、小型机中。

3. 单功能型或复合型操作码

多数指令常采用单功能型操作码，即操作码只表示一种操作含义，以便能够快速地识别操作码并执行操作。有的计算机指令字长有限、指令的数量也有限，为了使一条指令能够表示更多的操作信息，常采用复合型的操作码，也就是说将操作码分为几个部分，表示多种操作含义，使操作的含义比较丰富。

3.2　指令和数据的寻址方式

寻址方式是指令系统设计的重要内容。从计算机硬件设计者的角度来看，寻址方式与计算机硬件结构密切相关；从程序员角度来看，寻址方式不但与汇编语言程序设计有关，而且与高级语言的编译程序也有密切联系。所谓寻址方式即寻找指令或是操作数的有效地址的方式。因为存储器可以用来存放数据，也可以用来存放指令，所以寻址包括对指令的寻址和对操作数的寻址。相比较而言，对操作数的寻址方式较指令的寻址方式更为复杂。

3.2.1　指令的寻址方式

指令寻址是指找出下一条将要执行的指令在存储器中的地址。一般来说，指令寻址的方式有两种：一种是顺序寻址方式；另一种是跳跃寻址方式。

1. 顺序寻址方式

计算机的工作过程是“先取指令,再执行指令”。指令在存储单元中被顺序存储,所以当执行一段程序时,通常是一条指令接一条指令按顺序进行。也就是说,从存储器中取出第一条指令,然后执行该指令;接着从存储器中取出第二条指令,再执行第二条指令;然后再取第三条指令……直至该段程序的指令都读取执行结束,这种程序顺序执行的过程即为顺序寻址方式。在该过程中,可用程序计数器 PC 来指示指令在存储器中的地址。

程序计数器 PC 是指令寻址的焦点,存储指令寻址的结果。PC 具有自动修改(+1)功能,可用于执行非转移类指令;还有接收内部总线数据的功能,可用于执行转移类指令或中断处理时的转移类操作,所以改变 PC 的内容就会改变程序执行的顺序,多种寻址方式的实质是改变 PC 的内容。

在顺序寻址方式中,在执行指令时 PC 会自动修改其内容,为下一条指令的读取做准备,这样周而复始地进行就可以完成顺序执行的程序。示意图如图 3-4 所示。

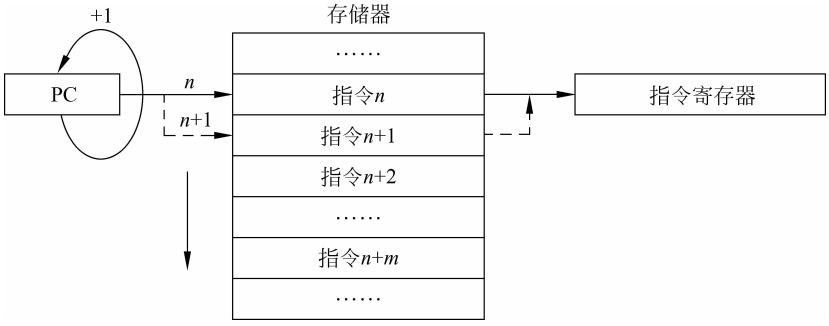


图 3-4 顺序寻址方式示意图

2. 跳跃寻址方式

当程序转移执行的顺序时,如执行了转移类指令或有外部中断发生时,要按照新的指令地址开始执行,所以 PC 的内容必须发生相应的改变,以便及时跟踪新的指令地址。这种情况下,指令的寻址采取跳跃寻址方式。所谓跳跃是指下条指令的地址码不是由 PC 给出,而是由本条指令给出。采用指令跳跃寻址方式,可以实现程序转移或构成循环程序,从而能缩短程序长度,或将某些程序作为公共程序引用。指令系统中的各种条件转移或无条件转移指令,就是为了实现指令的跳跃寻址而设置的。示意图如图 3-5 所示。

3.2.2 操作数的寻址方式

操作数不像指令那样顺序存储在存储单元中,有些公用的操作数会集中存放在某一区域,而大多数操作数的存放没有规律,这就给操作数的寻址带来一定的困难。又由于程序设计技巧的发展,提出了很多操作数的设置方法,所以出现了各种各样的操作数寻址方式。

操作数可能被放在指令中,或是某个寄存器中,或是主存的某个单元中,也可能在堆栈或 I/O 接口中。当操作数存放在主存的某个存储单元时,若指令中的地址码不能直接用来访问主存,则这样的地址码被称为形式地址;对形式地址进行一定计算后得到的存放操作数的主存单元地址,即存放操作数的内存实际地址被称为“有效地址”。操作数寻址方式就是由指令中提供的形式地址演变为有效地址的方法,也就是说,寻址方式是规定如何对地址

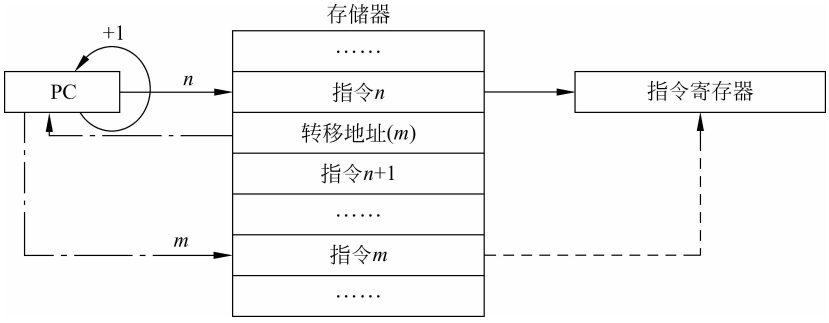
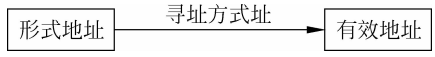


图 3-5　跳跃寻址方式示意图

作出解释以找到所需的操作数的方式。



若在指令中设置寻址方式字段，由寻址方式字段不同的编码来指定操作数的寻址方式，则称之为“显式”寻址方式；若是由操作码决定有关的寻址方式，则称为“隐式”寻址方式。可将众多的寻址方式归纳为以下 4 种基本方式或是它们的变型组合：

（1）立即寻址：指令中直接给出操作数。读取指令时，可直接从指令中获得操作数。

（2）直接寻址：在指令中直接给出存放操作数的主存单元的地址或是寄存器号。通过访问存储器或寄存器即可获得操作数。

（3）间接寻址：指令中所给的主存单元或寄存器中存放的是操作数的地址。先取出操作数的地址，然后按照该地址再访问主存单元获得操作数。

（4）变址：指令中所给的是形式地址，依照寻址方式得到有效地址后，再根据有效地址访问主存单元获得操作数。

下面介绍一些常用的基本寻址方式。

1. 立即寻址

在指令中给出操作数，操作数占据一个地址码部分，在取出指令的同时取出可以立即使用的操作数，所以该方式称为立即寻址，该操作数被称为立即数。立即寻址方式示意图如图 3-6 所示，图中 OP 为操作码字段用以指明操作种类，M 为寻址方式字段用以指明所用的寻址方式。

图 3-6　立即寻址方式示意图

立即寻址方式不需要根据地址寻找操作数，所以指令的执行速度快。但是由于操作数是在指令中给出的，是指令的一部分，不能修改，因此立即寻址只适用于操作数固定的情况，通常用于为主存单元和寄存器提供常数，设定初始值。使用时应注意，立即数只能作为源操作数。其优点是立即数的位置随着指令在存储器中位置的不同而不同。

2. 存储器直接寻址

指令中的地址码字段所给的就是存放操作数的主存单元的实际地址，即有效地址 EA。

按照指令中所给的有效地址直接访问一次主存便可获得操作数，所以称这种寻址方式为存储器直接寻址或直接寻址。其寻址方式示意图如图 3-7 所示，图中有效地址 EA 为主存储器的单元地址。

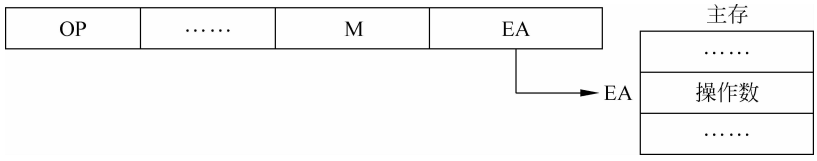


图 3-7 存储器直接寻址方式示意图

直接寻址方式较为简单，无须作任何寻址计算。由于指令中给出了操作数的有效地址，是指令中的一部分，所以不能进行修改，因此只能用于访问固定的存储单元或者外部设备接口中的寄存器。此外，因为存储单元的地址位数较多，所以包含在指令中时，指令字长会较长。如果减少指令中有效地址的位数，则会限制访问主存的范围。

【例 3-2】 指令中所给的地址码 EA 为“2001H”，按照存储器直接寻址方式读取操作数。主存中部分地址与相应单元存储的操作数之间的对应关系如下：

地址	存储内容
2000H	3BA0H
2001H	1200H
2002H	2A01H

解：因为在存储器直接寻址方式中，指令中的有效地址即主存中存储操作数的地址，所以地址为“2001H”的存储单元中的内容“1200H”即操作数。

3. 寄存器直接寻址

一般计算机中都设置有一定数量的通用寄存器，用以存放操作数、操作数地址及运算结果等。指令中地址码部分给出某一通用寄存器的寄存器号，所指定的寄存器中存放着操作数，这种寻址方式称为寄存器直接寻址，也称为寄存器寻址。其寻址方式示意图如图 3-8 所示，图中 Rx 为寄存器号。

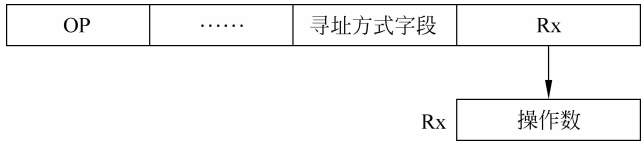


图 3-8 寄存器直接寻址方式示意图

采用寄存器寻址方式具有以下特点。

- 与立即数寻址方式相比，寄存器寻址中的操作数是可变的。
- 由于寄存器的数量较少，地址码的编码位数比主存单元地址位数短很多，所以可以有效缩短指令长度，减少取指令的时间，如指令中只需要 3 位编码就可以表示 8 个寄存器，如“000”表示 R0，“001”表示 R1，…，“111”表示 R7。
- 与直接寻址相比，寄存器存取数据的速度比主存快得多，所以可以加快指令的运行速度。

• 用寄存器存放基址值、变址值可派生出其他更多的寻址方式，使编程更具有灵活性。

【例 3-3】 指令中所给的寄存器号为“001”，按照寄存器寻址方式读取操作数。CPU 中寄存器的内容如下：

R0——2101H，R1——2A01H，R2——3BA0H，R3——1200H，……

解：因为在寄存器直接寻址方式中，所给出的寄存器中所存放的就是所需操作数，编码为“001”的寄存器 R1 中的内容“2A01H”即操作数。

4. 存储器间接寻址

如果指令中地址码 A 给出的不是操作数的直接地址，而是存放操作数地址的主存单元地址（简称为操作数地址的地址），这种寻址方式称为存储器间接寻址或间接寻址。其寻址方式示意图如图 3-9 所示。字段 A 中存放的是操作数的有效地址 EA 在主存中的地址。

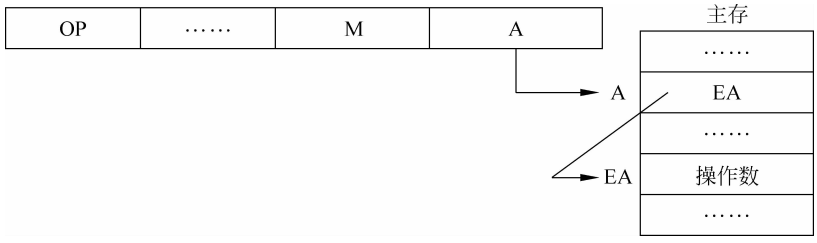


图 3-9　存储器间接寻址方式示意图

通常将主存单元 A 称为间址单元或间址指示器，间接地址 A 与有效地址 EA 的关系是 EA＝(A)，即 EA 为地址 A 所对应存储单元中的内容。采用间址方式可将主存单元 A 作为操作数地址的指针，用以指示操作数的存放位置，只要修改指针的内容就修改了操作数的地址，而无须修改指令，所以该方式较为灵活，便于编程。除此之外，采用间接寻址可以做到用较短的地址码来访问较大的存储空间。指令中给出的形式地址字段 A 虽然较短，只能访问到主存的低地址部分，但是存储在这些单元中的操作数的地址可以访问到整个主存空间。

应注意，间接寻址至少要访问两次主存才能取出操作数，所以指令执行的速度较慢。

【例 3-4】 指令中所给的地址码 A 为“2001H”，按照存储器间接寻址方式读取操作数。主存中部分地址与相应单元存储的操作数之间的对应关系如下：

地址	存储内容
2000H	3BA0H
2001H	2002H
2002H	2A01H

解：因为在存储器间接寻址方式中，指令中的形式地址 A 是操作数地址的地址，所以地址为“2001H”的存储单元中的内容“2002H”即操作数地址的地址，再根据地址“2002H”访问一次主存，可得到操作数“2A01H”。

5. 寄存器间接寻址

为了克服直接寻址中指令过长及间接寻址中访问主存次数多的缺点，可以采用寄存器间接寻址，即指令中给出寄存器号，被指定的寄存器中存放操作数的有效地址，根据该有效地址访问主存获得操作数。其寻址方式示意图如图 3-10 所示，寄存器 Rx 中存放着操作数的有效地址 EA，EA＝(Rx)。