第四章 指令系统

4.1 指令系统的发展

4.2 指令格式

4.3 数据表示

4.4 寻址方式（编址方式）

4.5 指令系统的实现

4.6 指令系统举例

4.7 精简指令系统计算机（RISC）和复杂指令系统计算机（CISC）

主要知识点

- 理解指令中应包含的信息。
- 掌握常用的指令及指令格式。
- 深入理解常用的寻址方法及其用途。
- 了解常见指令的种类和功能。

4.1 概述

4.1.1.指令系统的发展

在20世纪50年代和60年代早期，由于计算机采用分立元件(电子管或晶体管)，其体积庞大，价格昂贵，因此，大多数计算机的硬件结构比较简单。所支持的指令系统一般只有定点加减、逻辑运算、数据传送和转移等十几至几十条最基本的指令，而且寻址方式简单。

4.1 概述

到60年代中、后期，随着集成电路的出现，计算机的价格不断下降，硬件功能不断增强，指令系统也越来越丰富。除了具有以上最基本的指令以外，还设置了乘除法运算指令、浮点运算指令、十进制运算指令以及字符串处理指令等，指令数多达一、二百条，寻址方式也趋于多样化。

4.1.2.基本概念

- 指令是指挥机器完成某种操作的命令。
- 指令系统是某计算机能直接识别、正确执行的所有指令的集合，是计算机软件与硬件的界面。
- 程序是若干条指令的有序集合。
- 从计算机组成的层次结构来说，计算机的指令有微指令、机器指令和宏指令之分。

4.1 概述

- 微指令是微程序级的命令，用于产生控制信号，它属于最低层；
- 宏指令是由若干条机器指令组成的软件指令；
- 机器指令则介于微指令与宏指令之间，每一条指令可完成一个独立的操作。

本章所讨论的指令是机器指令。

4.1 概述

- 计算机语言有高级语言和低级语言之分。
- 高级语言语句和用法与具体机器的指令系统无关，面向算法的语言。
- 低级语言分机器语言（二进制语言）和汇编语言（符号语言），这两种语言都是面向机器的语言，它们和具体机器的指令系统密切相关。

4.2 指令格式

1.指令格式

指令有各种各样,作用各不相同,但最主要有两类:

- 指令
  - 动作类
    - 传送类
    - 运算类
    - I/O指令等
  - 控制类
    - 循环指令
    - 移位指令
    - 停机指令等

4.2 指令格式

表示一条指令的机器字，就称为指令字，通常简称指令。

指令格式是指令字用二进制代码表示的结构形式。

一条指令的结构可用如下形式来表示：

操作码字段
地址码字段

4.2 指令格式

- 指令的操作码表示该指令应进行什么性质的操作。
- 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。
- n位操作码能描述的操作最多有2^n。

固定长度操作码：便于译码，扩展性差
可变长度操作码：能缩短指令平均长度

4.2 指令格式

计算机是通过执行指令来处理各种数据的。为了指出数据的来源、操作结果的去向及所执行的操作，一条指令必须包含下列信息：

(1) 操作码，具体说明了操作的性质及功能。一台计算机可能有几十条至几百条指令，每一条指令都有一个相应的操作码，计算机通过识别该操作码来完成不同操作。

(2) 操作数的地址，CPU通过该地址就可以取得所需的操作数。

4.2 指令格式

(3) 操作结果的存储地址，把对操作数的处理所产生的结果保存在该地址中，以便再次使用。

(4) 下一条指令地址，指出当前指令执行完，下一条指令在哪里。

2.零地址指令

格式：

OPCODE—操作码

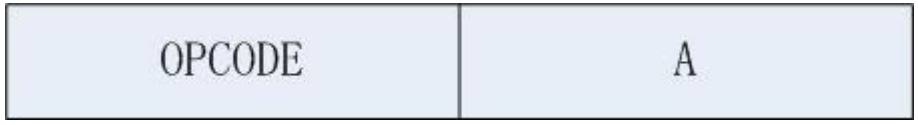
指令中只有操作码，而没有操作数或没有操作数地址。这种指令有两种可能：

(1) 无需任何操作数。如空操作指令，停机指令等。

(2) 所需的操作数是默认的。零地址的运算类指令仅用在堆栈计算机中。通常参与运算的两个操作数隐含地从栈顶和次栈顶弹出，送到运算器进行运算，运算结果再隐含地压入堆栈。

3.一地址指令

格式：



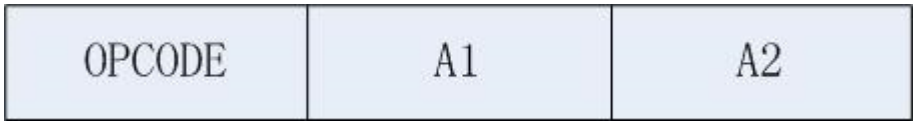
OPCODE—操作码
A—操作数的存储器地址或寄存器名
指令中只给出一个地址，该地址既是操作数的地址，又是操作结果的存储地址。

4.2 指令格式

在某些字长较短的微型机中，大多数算术逻辑运算指令也采用这种格式，第一个源操作数由地址码A给出，第二个源操作数在一个默认的寄存器中，运算结果仍送回到这个寄存器中，替换了原寄存器内容，通常把这个寄存器称为累加器。

4.二地址指令

格式：



OPCODE—操作码

A_1—第一个源操作数的存储器地址或寄存器地址。

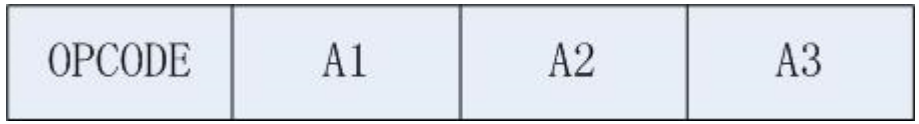
A_2—第二个源操作数和存放操作结果的存储器地址或寄存器地址。

4.2 指令格式

这是最常见的指令格式，两个地址指出两个源操作数地址，其中一个还是存放结果的目的地址。对两个源操作数进行操作码所规定的操作后，将结果存入目的地址 。

5.三地址指令

格式：



OPCODE—操作码

A_1—第一个源操作数的存储器地址或寄存器地址

A_2—第二个源操作数的存储器地址或寄存器地址

A_3—操作结果的存储器地址或寄存器地址，其操作是对A_1，A_2指出的两个源操作数进行操作码(OPCODE)所指定的操作，结果存入A_3中。

6.多地址指令

格式：

OPCODE—操作码
A_1—第一个源操作数的存储器地址或寄存器地址
A_2—第二个源操作数的存储器地址或寄存器地址
A_n—第n个源操作数的存储器地址或寄存器地址

4.2 指令格式

在某些性能较好的大、中型机甚至高档小型机中，往往设置一些功能很强的，用于处理成批数据的指令，如字符串处理指令，向量、矩阵运算指令等。为了描述一批数据，指令中需要多个地址来指出数据存放的首地址、长度和下标等信息。

以上所述的几种指令格式只是一般情况，并非所有的计算机都具有。

7.按操作数类型定义的指令格式

从操作数的物理位置来说，又可归结为三种类型：

（1）访问内存的指令格式，我们称这类指令为存储器 - 存储器（SS）型指令；

（2）访问寄存器的指令格式，我们称这类指令为寄存器 - 寄存器（RR）型指令；

（3）第三种类型为寄存器 - 存储器（RS）型指令。

8.指令字长度

机器字长是指计算机能直接处理的二进制数据的位数，它决定了计算机的运算精度。

一个指令字中包含二进制代码的位数，称为指令字长度。

4.2 指令格式

如果在一个指令系统中，各种指令字长度是相等的，称为等长指令字结构，这种指令字结构简单，且指令字长度是不变的。

如果各种指令字长度随指令功能而异，就称为变长指令字结构。这种指令字结构灵活，能充分利用指令长度，但指令的控制较复杂。

等长指令字结构：取指快、译码简单。
变长指令字结构：可提高编码效率。

4.2 指令格式

指令字的长度主要取决于操作码的长度、操作数地址的长度和操作数地址的个数。

指令长度可变的 ， 其指令长度通常是字节的整数倍。

例如：8086的指令长度为：8、16、24、32、40和48位六种。

9.指令格式的选取

指令格式选取考虑的因素：

（1）指令长度应尽可能的短，短指令比长指令好。

（2）指令长度与机器字长应是字节长度的整倍数。

（3）指令中操作码字段应有足够的位数，使之能表示指令系统中全部操作。

（4）指令中地址段的位数要足够长。

10.指令格式举例(MIPS32)

MIPS32™的指令格式只有3种:

R（register）类型

R指令从寄存器堆中读取两个源操作数，计算结果写回寄存器堆。

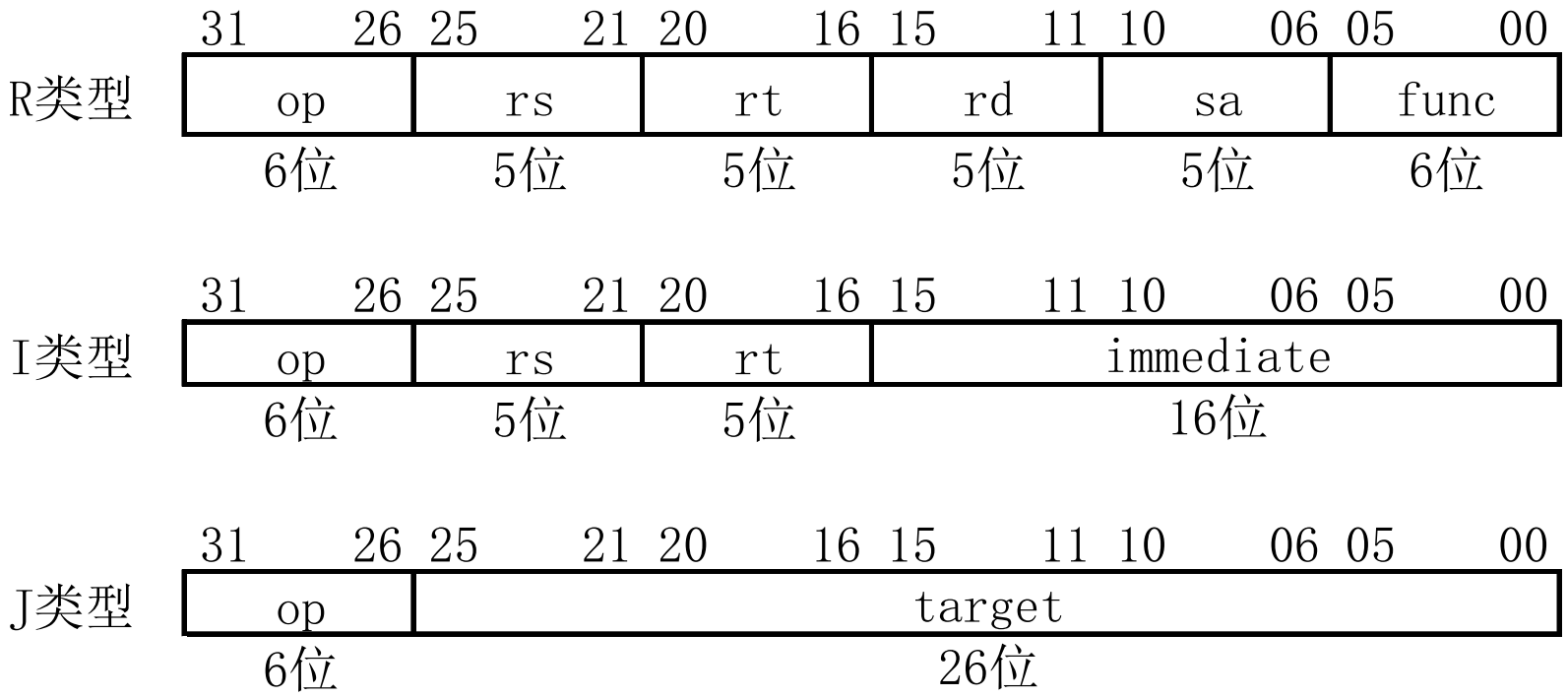
I（immediate）类型

I指令使用一个16位的立即数作为源操作数。

J（jump）类型

J的指令使用一个26位立即数作为跳转的目标地址(target address)。

4.2 指令格式


31 26 25 21 20 16 15 11 10 06 05 00
R类型
op
rs
rt
rd
sa
func
6位 5位 5位 5位 5位 6位
31 26 25 21 20 16 15 11 10 06 05 00
I类型
op
rs
rt
immediate
6位 5位 5位 16位
31 26 25 21 20 16 15 11 10 06 05 00
J类型
op
target
6位 26位

4.2 指令格式

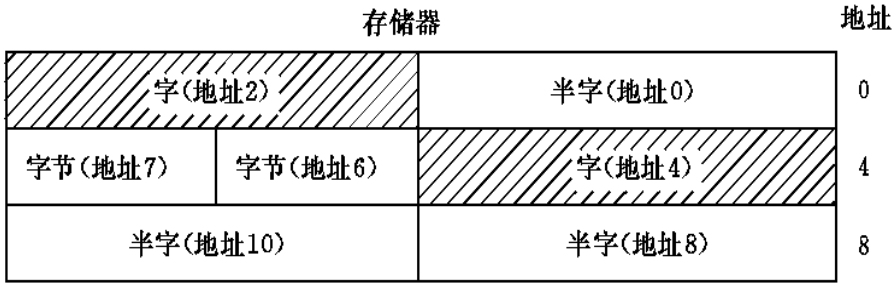
在计算机中，指令和数据一样都是以二进制码的形式存储的，从表面来看，两者没有什么差别。但是，指令的地址是由程序计数器(PC)规定的，而数据的地址是由指令规定的，在CPU控制下访存绝对不会将指令和数据混淆。

4.3 数据表示

一般要求多字节数据对准边界

(a)

未对准边界



(b)

4.3 数据表示

在数据对准边界的计算机中，有两种存放次序。称为小端序和大端序。

如数据：00000000 00000001 00010000 00000011

大端序和小端序的存储方式

32位字存储单元的最低2位地址	小端序	大端序
第00单元	00000011	00000000
第01单元	00010000	00000001
第10单元	00000001	00010000
第11单元	00000000	00000011

4.4 寻址方式

形成有效的指令地址或操作数地址的方式，称为寻址方式（编址方式）。

指令地址段给出的往往是形式地址（Format Address），而有效地址（Effective Address）是通过一定的计算得到。

寻址方式分成两大类：指令寻址和操作数寻址。

4.4.1 指令寻址

所谓指令寻址方式，就是确定下一条将要执行的指令地址的方法。确定指令地址的基本方式有两种：顺序寻址方式和跳跃寻址方式。

4.4 寻址方式

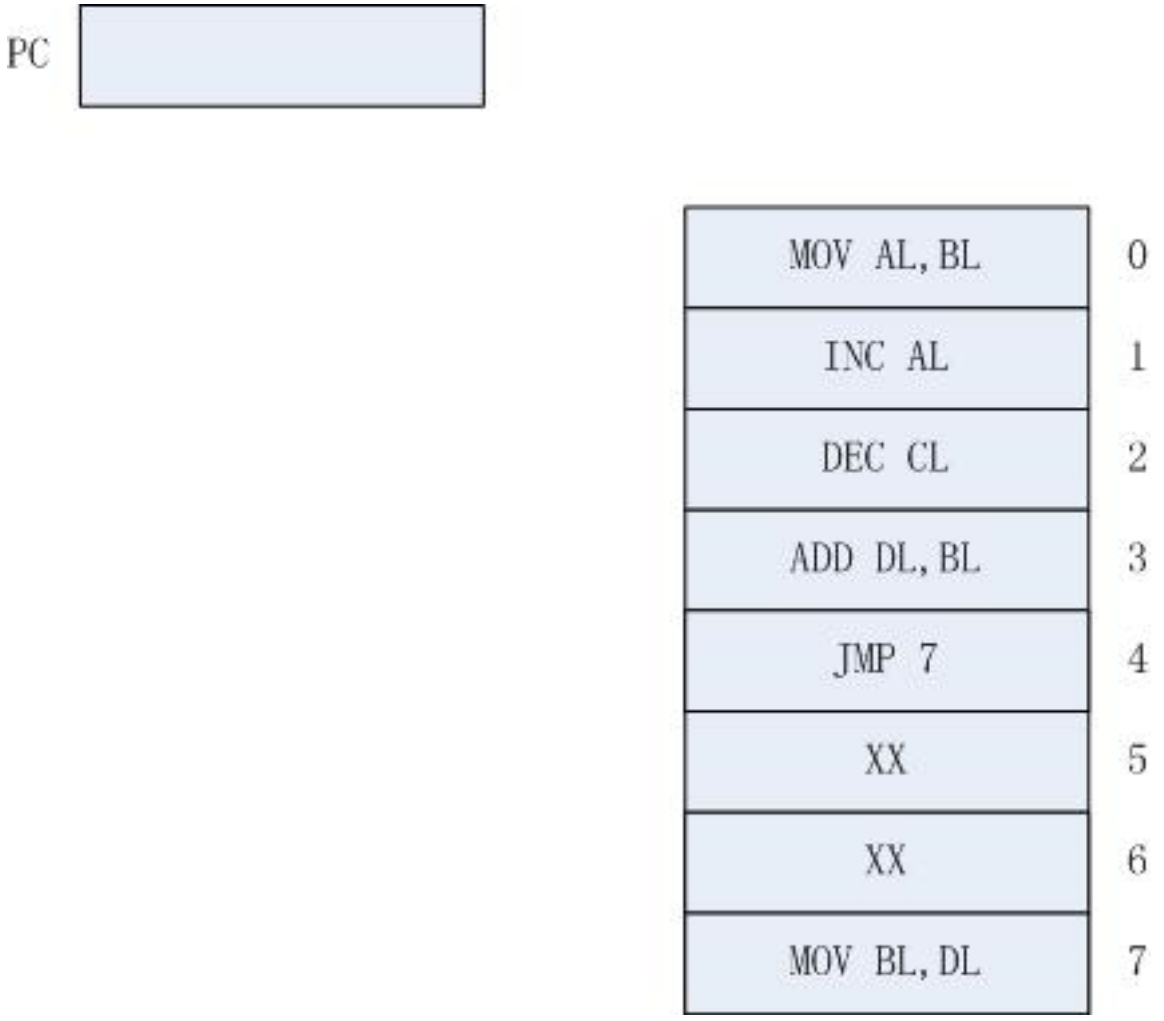
1.顺序寻址方式

指令在内存中是按地址顺序安排的，执行程序时，通常是一条指令接一条指令顺序进行。即从存储器取出第一条指令，然后执行第一条指令；接着从存储器取出第二条指令，再执行第二条指令；接着再取出第三条指令，执行第三条指令……这种程序按指令地址顺序执行的过程，称为指令的顺序寻址方式。

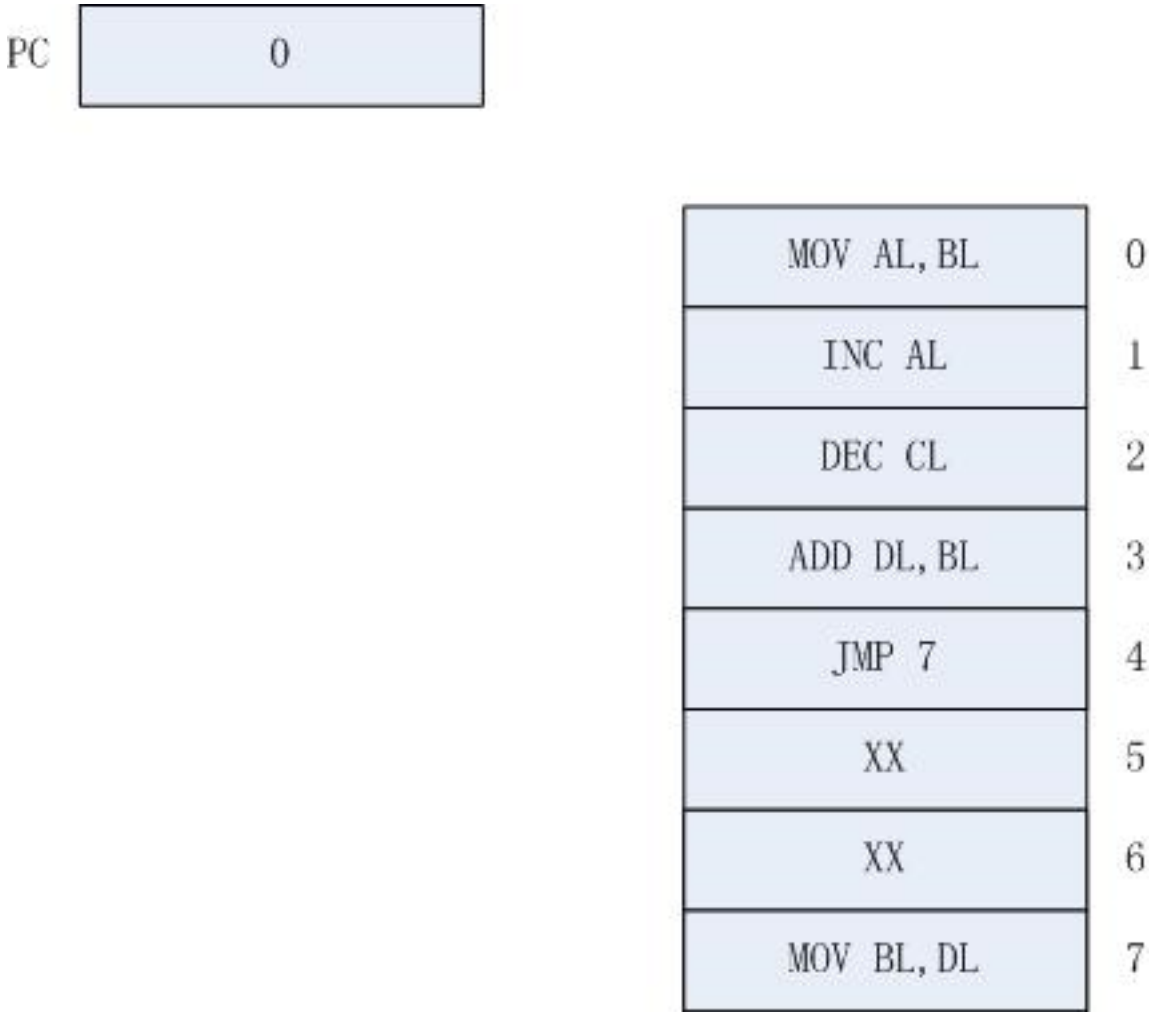
4.4 寻址方式

因此，必须在CPU中设置专用电路来控制指令按照指令在内存中的地址顺序依次逐条执行，该专用控制部件就是程序计数器(又称指令计数器PC)，计算机中就是由PC来计数指令的顺序号，控制指令顺序执行。

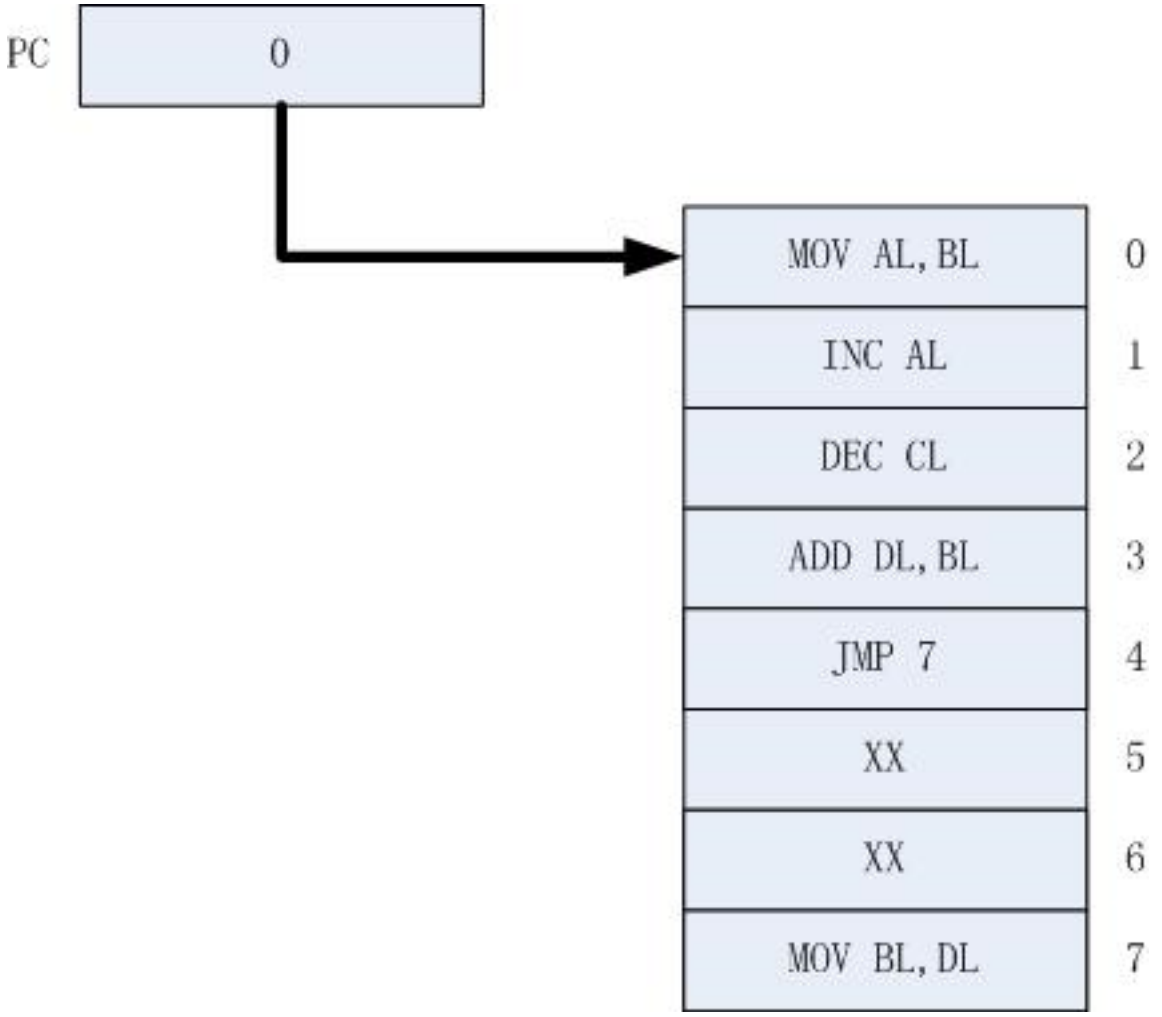
4.4 寻址方式



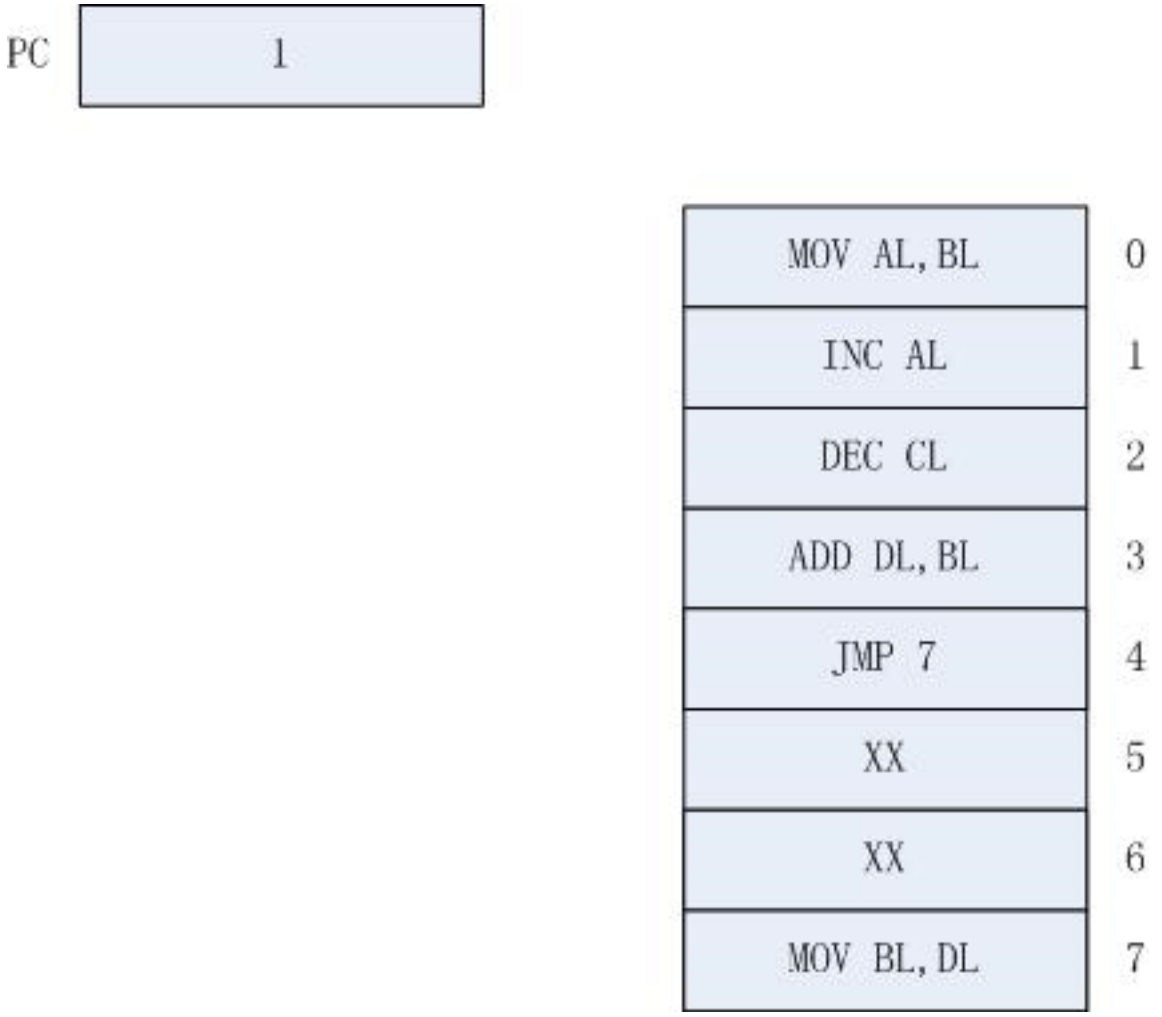
4.4 寻址方式


PC
0
MOV AL, BL
0
INC AL
1
DEC CL
2
ADD DL, BL
3
JMP 7
4
XX
5
XX
6
MOV BL, DL
7

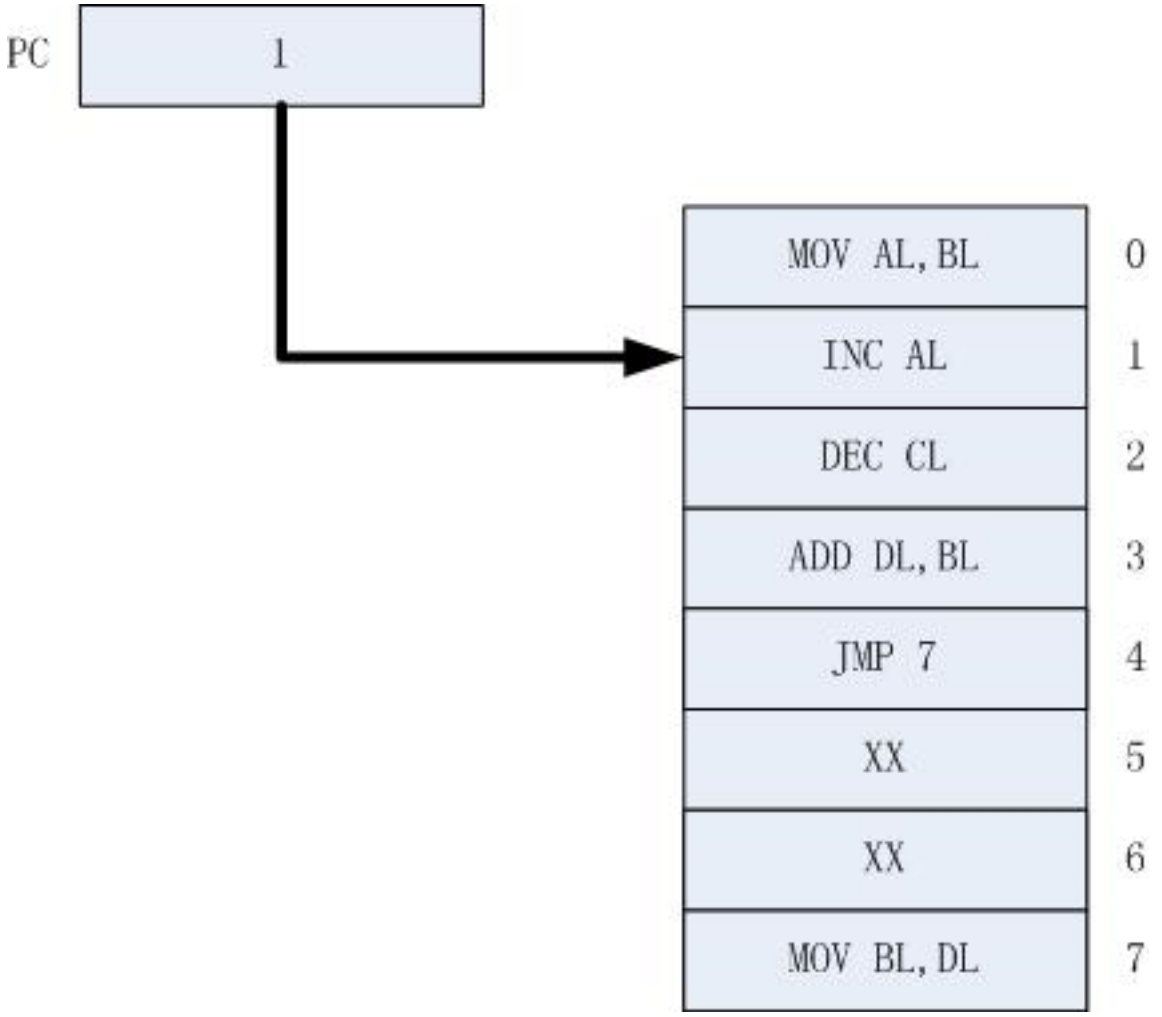
4.4 寻址方式


PC
0
MOV AL, BL
INC AL
DEC CL
ADD DL, BL
JMP 7
XX
XX
MOV BL, DL
0
1
2
3
4
5
6
7

4.4 寻址方式


PC
1
MOV AL, BL
0
INC AL
1
DEC CL
2
ADD DL, BL
3
JMP 7
4
XX
5
XX
6
MOV BL, DL
7

4.4 寻址方式


PC
1
MOV AL, BL
INC AL
DEC CL
ADD DL, BL
JMP 7
XX
XX
MOV BL, DL
0
1
2
3
4
5
6
7

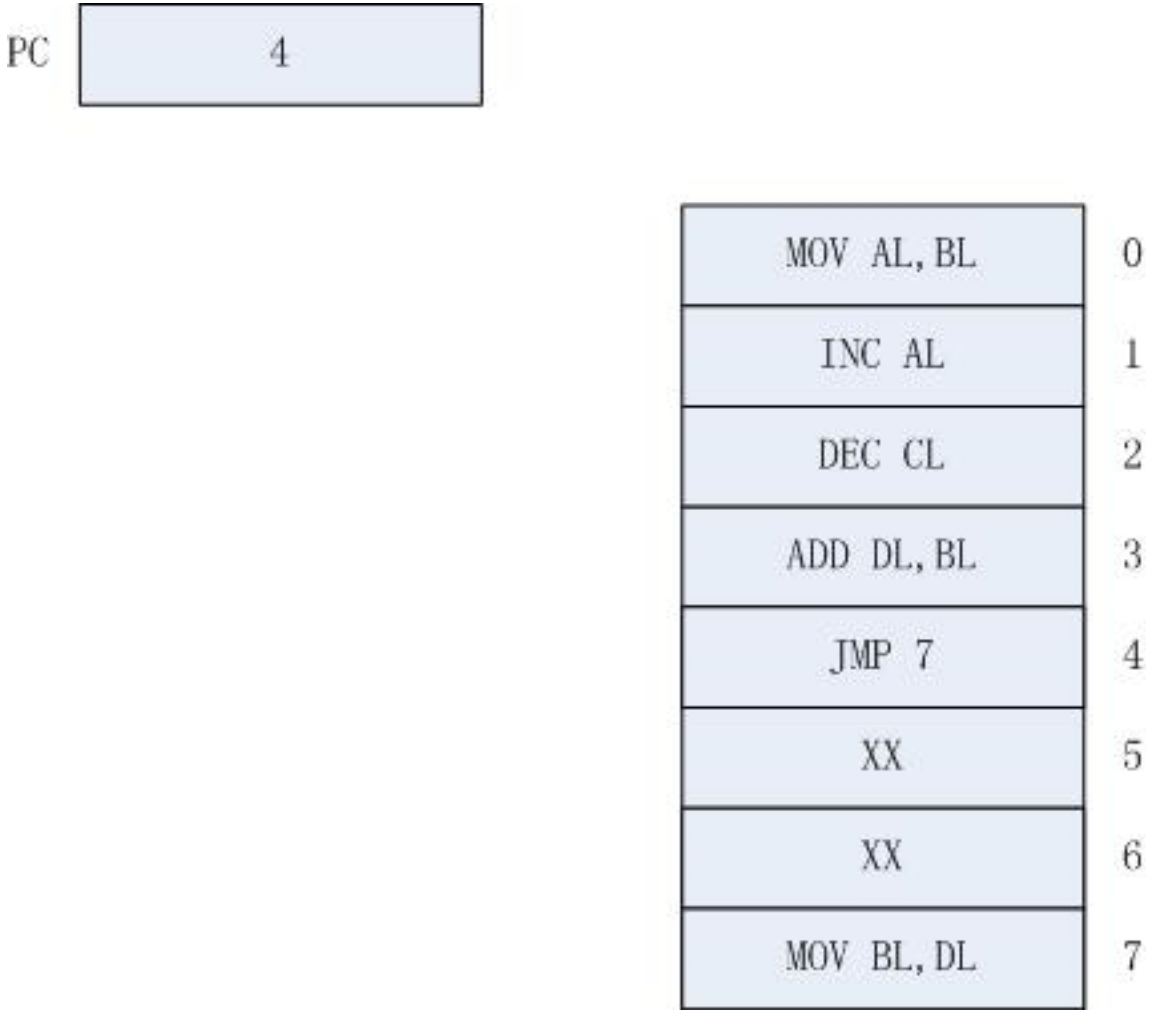
2.跳跃寻址

当程序要改变执行的顺序时，指令的寻址就采取跳跃寻址方式．所谓跳跃，又称跳转，是指下一条指令的地址由本条指令修改PC后给出。注意，程序跳跃后，按新的指令地址开始顺序执行。

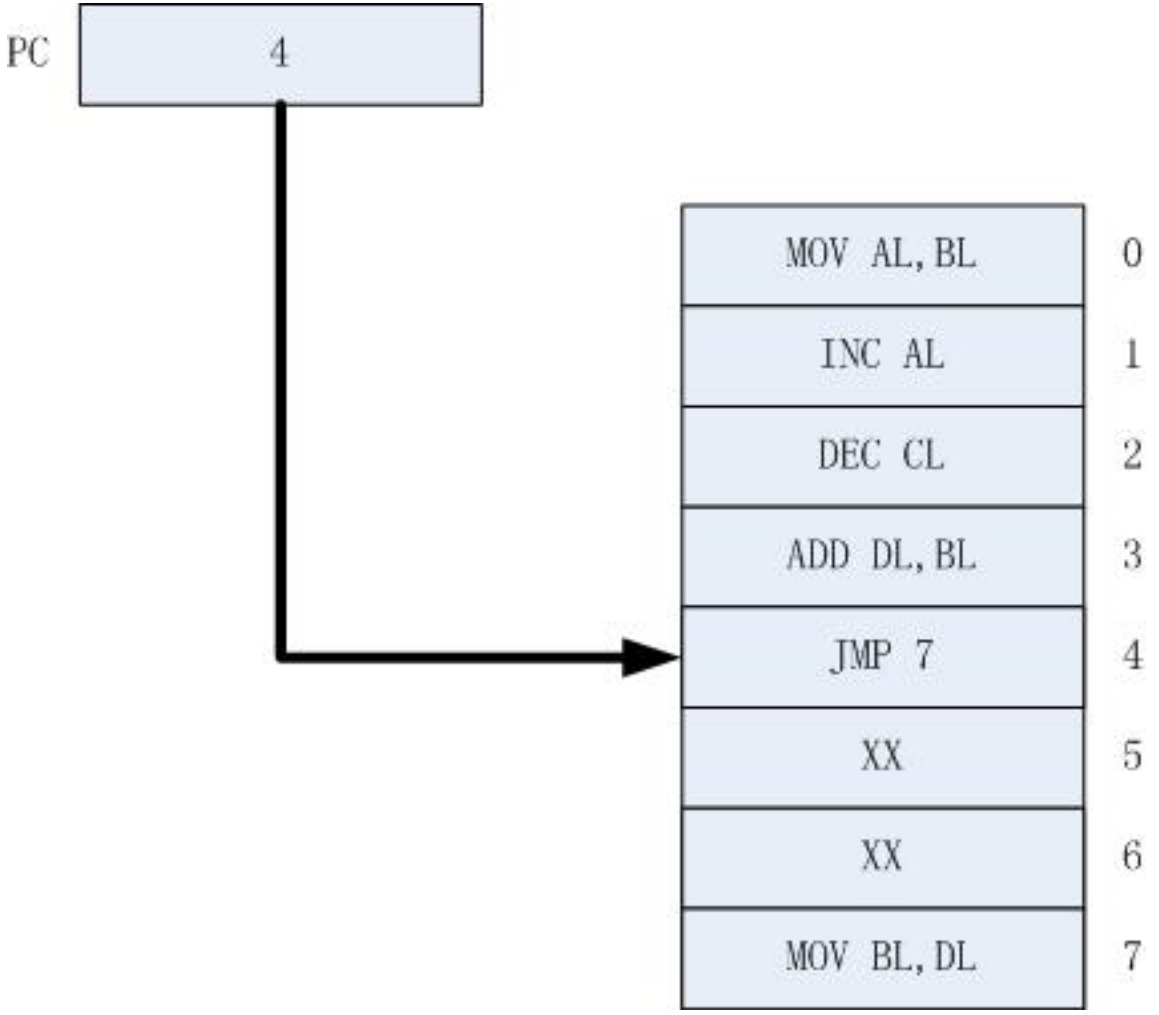
4.4 寻址方式

采用指令跳跃寻址式，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。

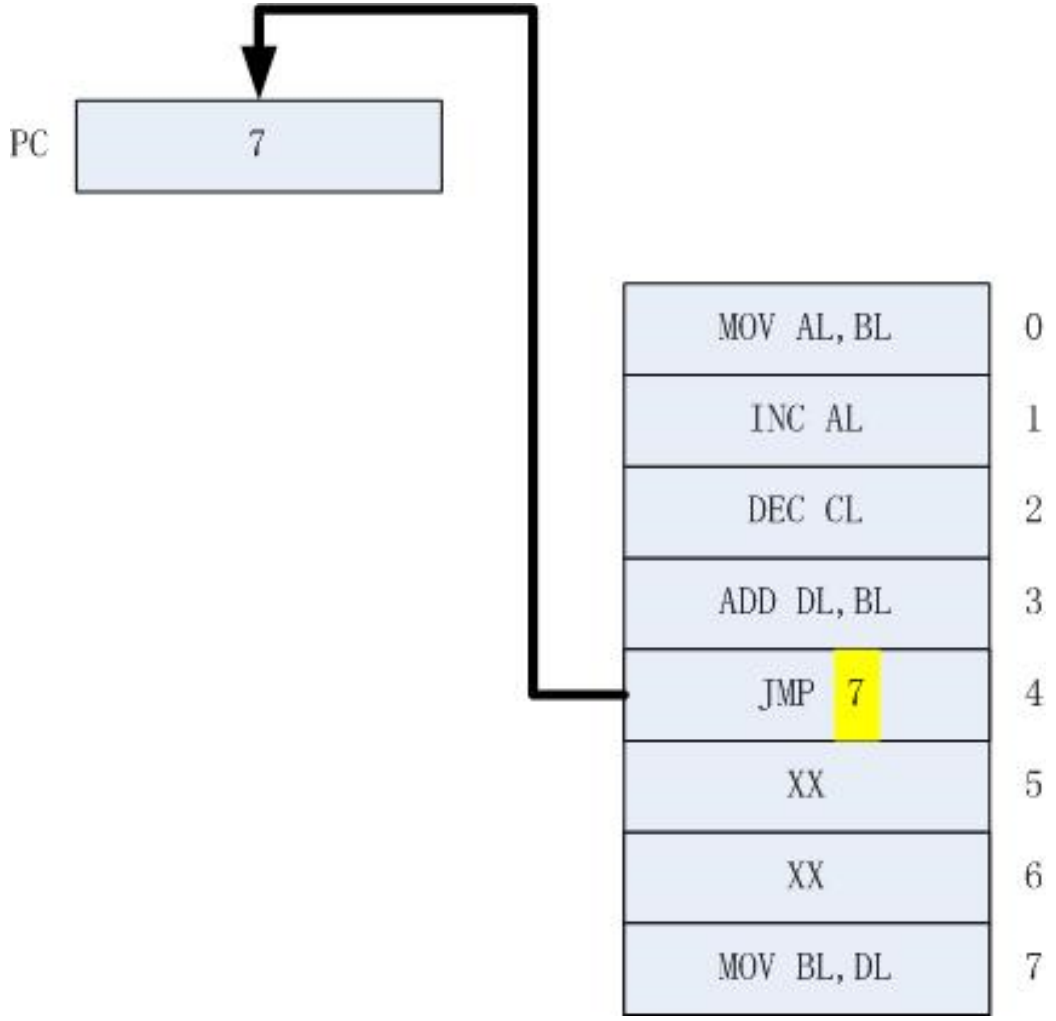
4.4 寻址方式


PC
4
MOV AL, BL
0
INC AL
1
DEC CL
2
ADD DL, BL
3
JMP 7
4
XX
5
XX
6
MOV BL, DL
7

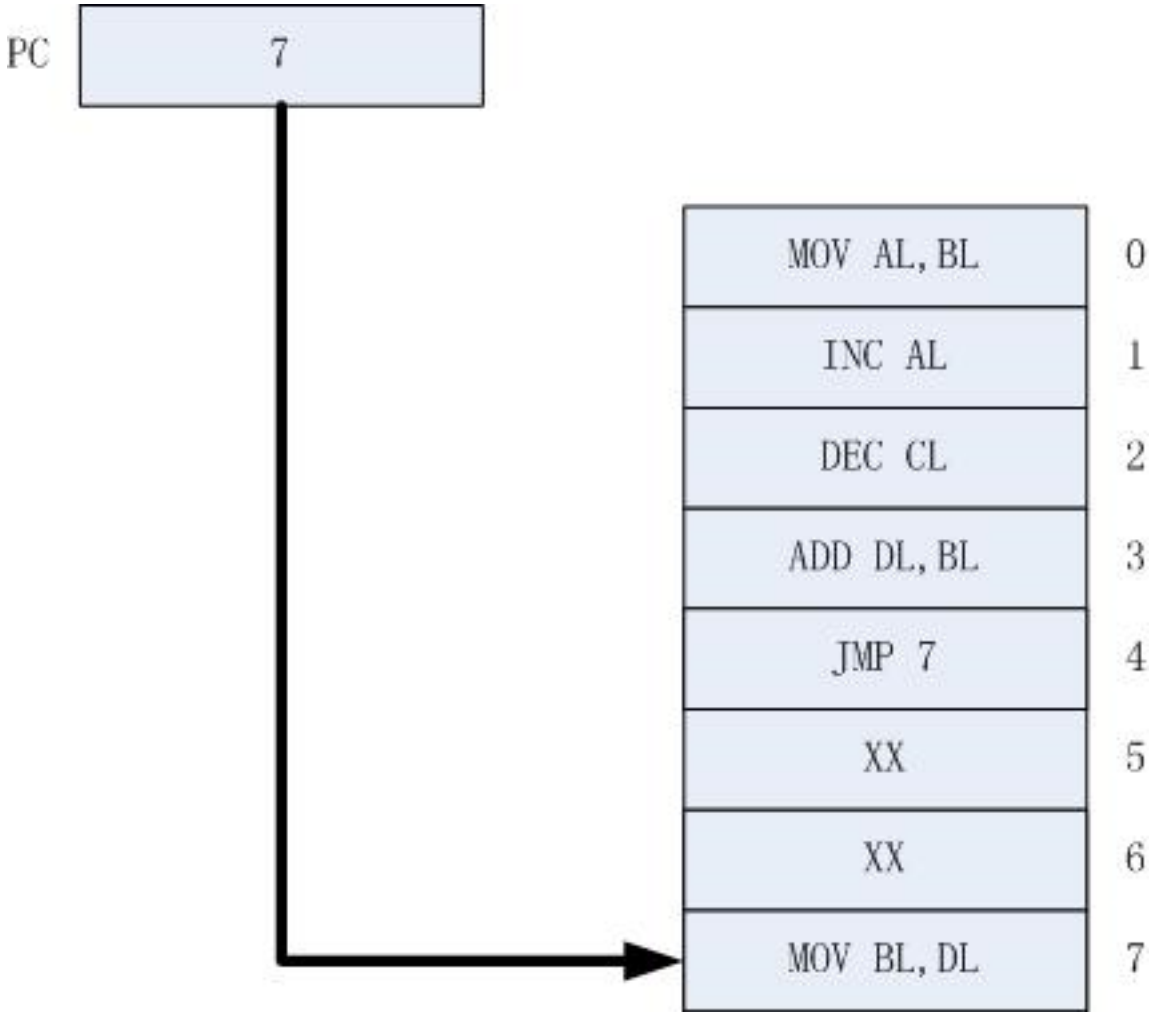
4.4 寻址方式


PC
4
MOV AL, BL
INC AL
DEC CL
ADD DL, BL
JMP 7
XX
XX
MOV BL, DL
0
1
2
3
4
5
6
7

4.4 寻址方式


PC
7
MOV AL,BL
INC AL
DEC CL
ADD DL,BL
JMP 7
XX
XX
MOV BL,DL
0
1
2
3
4
5
6
7

4.4 寻址方式


PC
7
MOV AL, BL
INC AL
DEC CL
ADD DL, BL
JMP 7
XX
XX
MOV BL, DL
0
1
2
3
4
5
6
7

4.4.2 操作数寻址

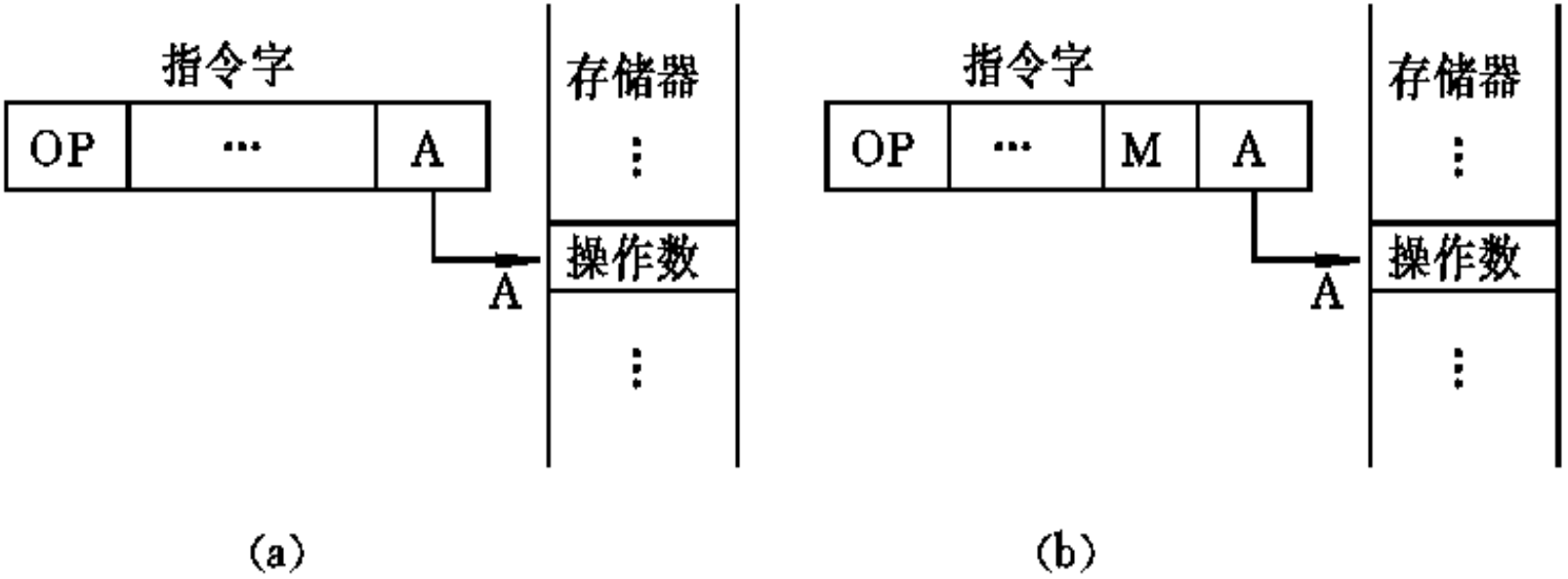
在程序执行过程中，操作数可能在运算部件的某个寄存器中或存储器中，也可能就在指令中。组成程序的指令代码，一般是在存储器中的。所谓操作数寻址方式(或编址方式)指的是确定数据地址的方法，它与计算机硬件结构紧密相关，而且对指令格式和功能有很大影响。不同的计算机有不同的寻址方式，但其基本原理是相同的。在这里我们仅介绍几种被广泛采用的基本寻址方式。

4.4 寻址方式

1.直接寻址

指令的地址码部分给出操作数在存储器中的地址，图a仅给出一个操作数地址；当有多个地址时，情况类似，不再重复，该指令的寻址方式由操作码表示。图b增加了一个寻址方式字段M。

4.4 寻址方式



4.4 寻址方式

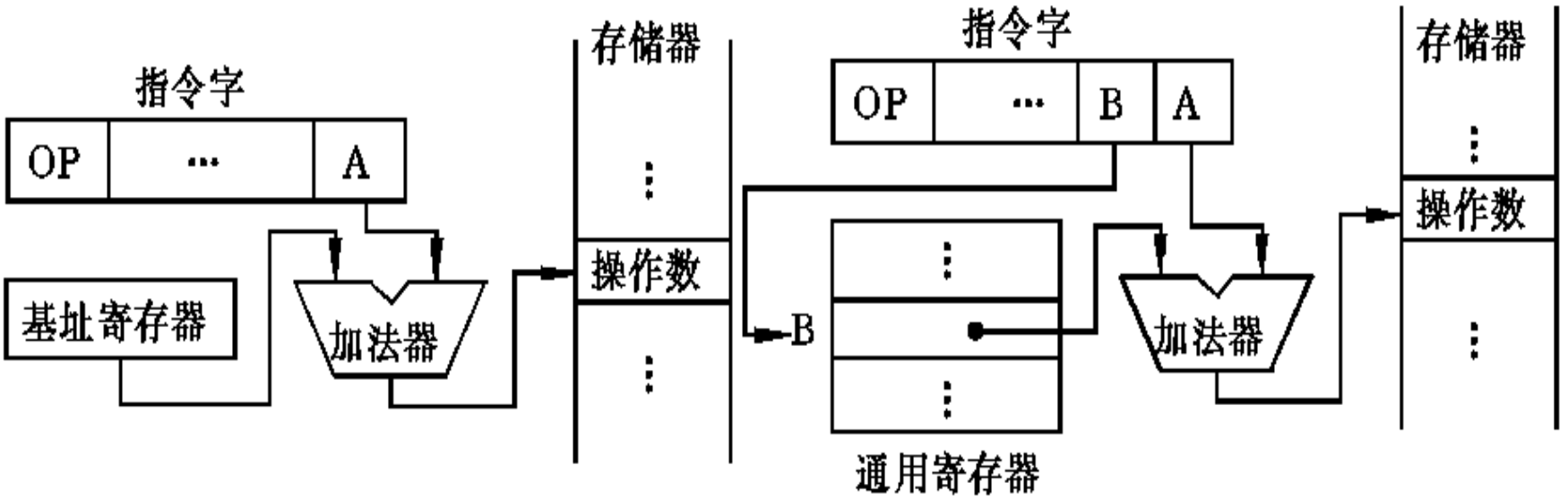
2. 寄存器寻址

计算机的中央处理器一般设置有一定数量的通用寄存器，用以存放操作数、操作数的地址或中间结果。假如指令地址码部分给出某一通用寄存器地址，而且所需的操作数就在这一寄存器中，则称为寄存器寻址。

3. 基址寻址

在计算机中设置一个专用的基址寄存器，或由指令指定一个通用寄存器为基址寄存器。操作数的地址由基址寄存器的内容和指令的地址码A相加得到，如图所示。

4.4 寻址方式

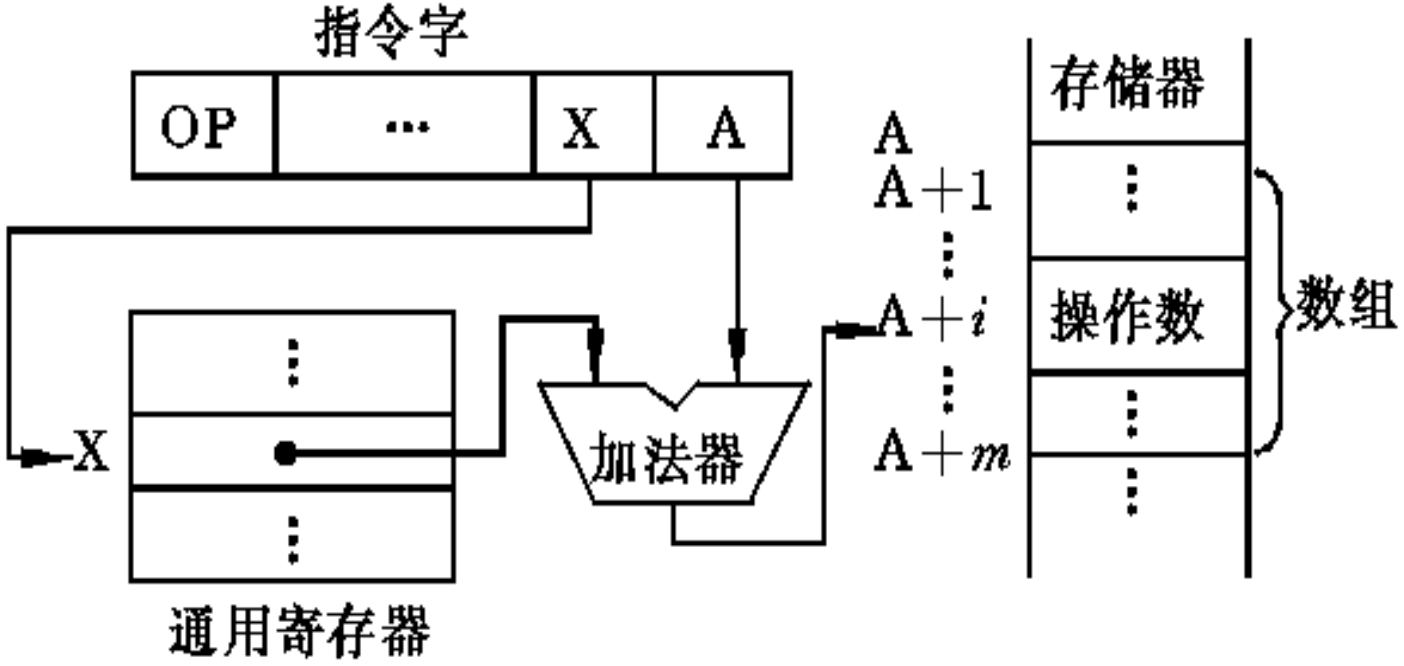


(a) 专用基址寄存器　　(b) 通用寄存器作为基址寄存器

4.4 寻址方式

4. 变址寻址

变址寻址的过程如图所示。

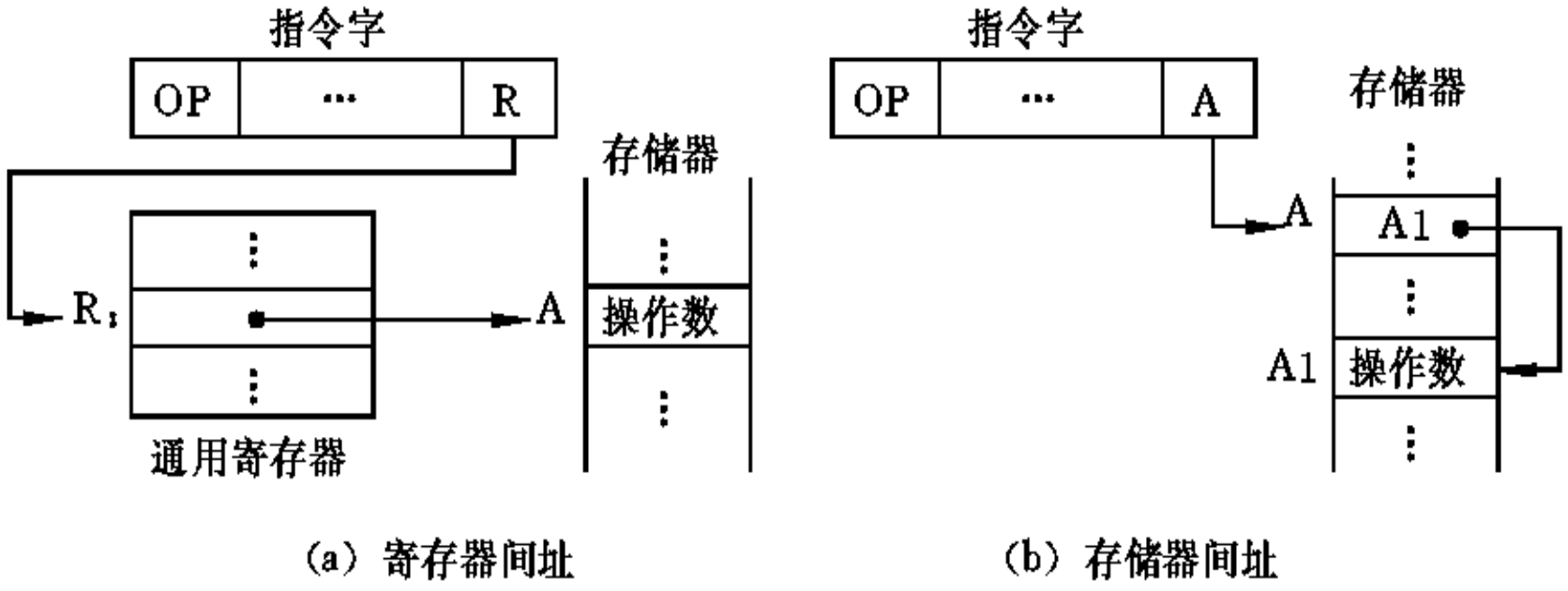


5. 间接寻址

在寻址时，有时根据指令的地址码所取出的内容不是操作数，而是操作数的地址，这种方式称为间接寻址或间址。根据地址码指的是寄存器地址还是存储器地址，间接寻址又可分为寄存器间接寻址和存储器间接寻址两种方式。间接寻址有一次间址和多次间址两种情况，大多数计算机只允许一次间址。对于存储器一次间址情况，需访问两次存储器才能取得数据，第一次从存储器读出操作数地址，第二次读出操作数。

4.4 寻址方式

图a和图b分别为寄存器间址与存储器间址的操作数寻址过程。



(a) 寄存器间址　　(b) 存储器间址

6.立即数寻址

所需的操作数由指令的地址码部分直接给出，就称为立即数(或直接数)寻址方式。这种方式的特点是取指时，操作码和一个操作数同时被取出，不必再次访问存储器，提高了指令的执行速度。通常用于给某一寄存器或存储器单元赋初值或提供一个常数等。

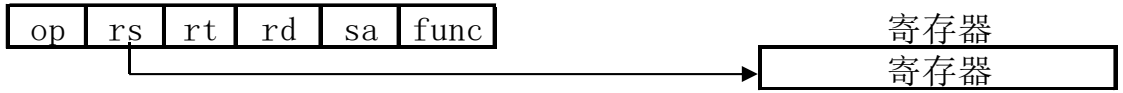
4.4.3寻址方式举例（MIPS32）

MIPS的寻址方式有以下几种：

(1) 寄存器寻址—操作数在寄存器堆中。
(2) 立即数寻址—操作数是一个常数，包含在指令中。
(3) 基址偏移量寻址—操作数在存储器中，存储器地址由一个寄存器的内容与指令中的常数相加得到。
(4) PC相对寻址—转移指令计算转移地址时使用。PC的相对值是指令中的一个常数。
(5) 伪直接寻址—跳转指令形成转移地址时使用。指令中的26位目标地址值与PC的高4位拼接，形成30位的存储器“字地址”。

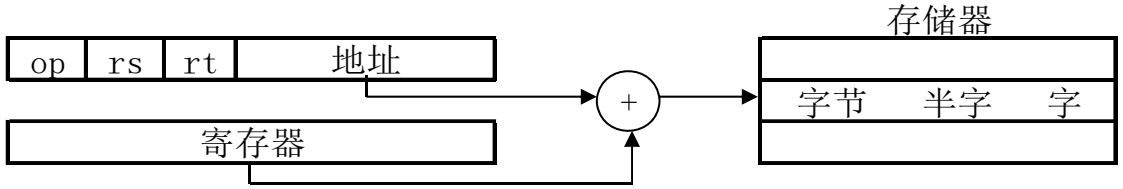
4.4 寻址方式

1. 寄存器寻址

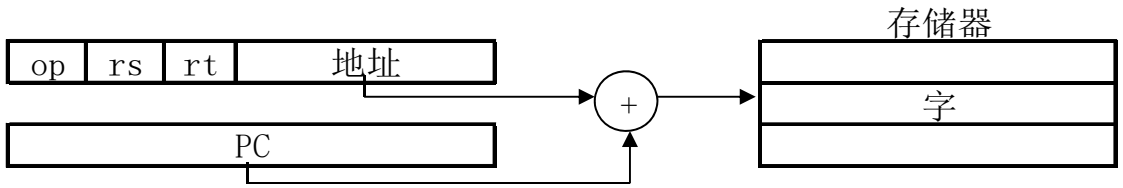


2. 立即数寻址

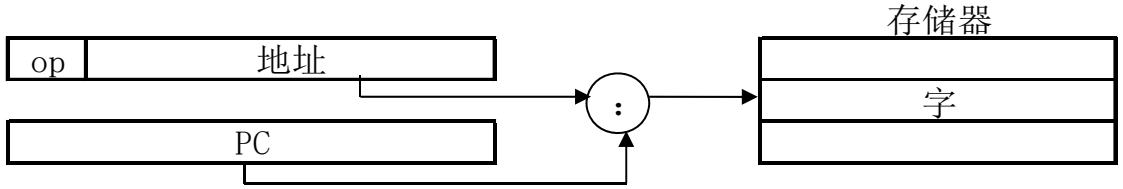
3. 基址偏移量寻址



4. PC相对寻址



5. 伪直接寻址



4.5 指令系统的实现

一台计算机中所有机器指令的集合，称为这台计算机的指令系统。

指令系统是表征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。

庞大的指令系统能使编写程序比较方便，然而，要实现一个庞大的指令系统会使控制电路复杂化，因此，通常倾向于在大型机采用大的指令系统。

4.5 指令系统的实现

一个完善的指令系统应满足如下四方面的要求：

1. 指令系统的完备性

完备性是指用汇编语言编写各种程序时，指令系统直接提供的指令足够使用，而不必用软件来实现。完备性要求指令系统丰富、功能齐全、使用方便。

这是一个原则性要求，一般说，一个完备的指令系统应包括以下一些类型指令：

4.5 指令系统的实现

(1) 数据传送指令

(2) 算术、逻辑运算指令

(3) 程序控制指令

(4) 输入/输出指令

2. 指令系统的有效性

有效性是指利用该指令系统所编写的程序能够高效率地运行。高效率主要表现在程序占据存储空间小、执行速度快。

3. 指令系统的规整性

规整性包括指令系统的对称性、匀齐性、指令格式和数据格式的一致性。

对称性是指在指令系统中所有的寄存器和存储器单元都可同等对待，所有的指令都可使用各种寻址方式；

匀齐性是指一种操作性质的指令可以支持各种数据类型；

4.5 指令系统的实现

指令格式和数据格式的一致性是指：指令长度和数据长度有一定的关系，以方便处理和存取。

4.指令系统的兼容性

- 要能做到“向上兼容”，即低档机上运行的软件可以在高档机上运行。
- 要能做到“向后兼容”，即先生产机器上运行的软件可以在后生产的机器上运行。

4.6 指令系统的优化设计

4.6.1 操作码的优化设计

1．Huffman编码法

根据Huffman编码法的原理，采用最优Huffman编码法表示的操作码的最短平均长度可以通过如下公式计算得到：

$$H = -\sum_{i=1}^{n} p_i * \log_2 p_i$$

其中：p_i表示第i种指令在程序中出现的概率

一共有n种操作码

4.6 指令系统的优化设计

相对于最优**Huffman**编码法，固定长度操作码编码法的信息冗余量可以表示为：

$$R = 1 - \frac{-\sum_{i=1}^{n} p_i \cdot \log_2 p_i}{\lceil \log_2 n \rceil}$$

4.6 指令系统的优化设计

Huffman编码的基本思想：

当各种事件发生的概率不均时，采用优化技术对发生概率最高的事件用最短的位数（时间）来表示（处理），而对出现概率较低的，用较长的位数（时间）来表示（处理），就会导致表示（处理）的平均位数（时间）的缩短。

4.6 指令系统的优化设计

用Huffman编码概念进行编码的步骤：

（1）把所有指令按照操作码出现的概率从低到高的次序自左向右排列；

（2）选取两个出现概率最小的节点合并成一个概率值是二者之和的新节点，并把这个新节点与其他还没有合并的节点一起形成新节点集合，并按合并后的概率重新排序。

（3）在新节点集合中选取两个概率最小的节点进行合并，继续（2）过程，直至全部节点合并完。

4.6 指令系统的优化设计

（4）最后得到的根节点的概率值为1；

（5）每个节点都有两个分支，分别用一位代码“0”和“1”表示；

（6）从根节点开始，逐级下移直至每个节点，将每个节点的代码组合起来就是该指令的操作码编码。

4.6 指令系统的优化设计

采用Huffman编码法所得到的是操作码的平均长度，计算方法为：

$$H = \sum_{i=1}^{n} p_i \cdot l_i$$

其中：*pi*代表第*i*种操作码在程序中出现的概率

*li*表示第*i*种操作码的二进制编码位数

*n*表示操作码的数量

4.6 指令系统的优化设计

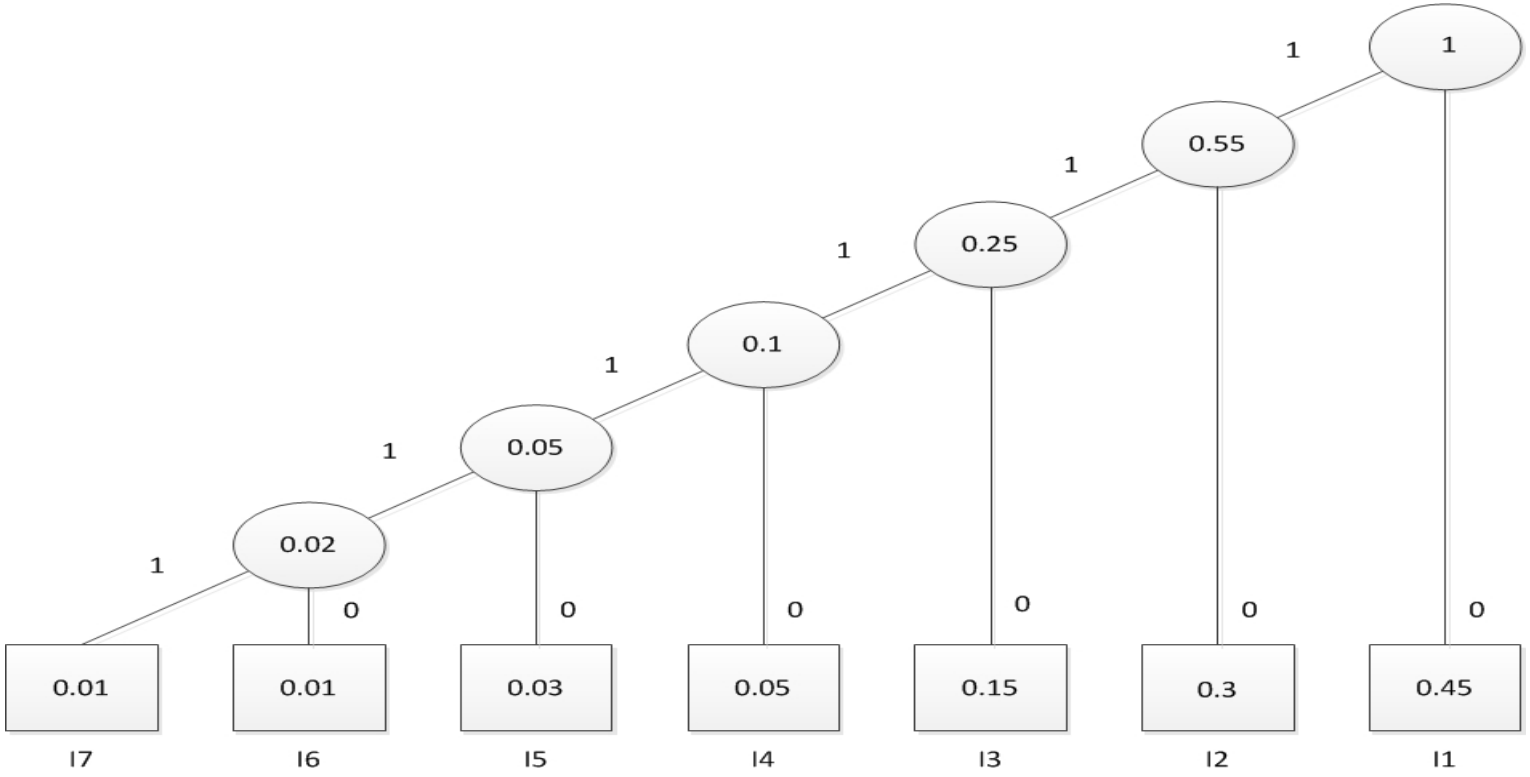
例：某计算机共有7种不同的指令，如采用定长操作码则操作码字段需要3位。已知各种指令在程序中出现的概率如表所示，计算采用Huffman编码法的操作码平均长度。

指令	I1	I2	I3	I4	I5	I6	I7
使用频度	0.45	0.30	0.15	0.05	0.03	0.01	0.01

解：

4.6 指令系统的优化设计

指令	I1	I2	I3	I4	I5	I6	I7
使用频度	0.45	0.30	0.15	0.05	0.03	0.01	0.01



4.6 指令系统的优化设计

指令	使用频度（Pi）	Huffman	操作码长度
I1	0.45	0	1位
I2	0.30	10	2位
I3	0.15	110	3位
I4	0.05	1110	4位
I5	0.03	11110	5位
I6	0.01	111110	6位
I7	0.01	111111	6位

4.6 指令系统的优化设计

Huffman编码平均长度：

$$H = \sum_{i=1}^{n} p_i \cdot l_i$$

H=0.45×1+0.30×2+0.15×3+0.05×4+0.03×5+0.01×6+0.01×6=1.97位

Huffman编码最短平均长度：

$$H = -\sum_{i=1}^{n} p_i * \log_2 p_i$$

H=0.45×1.152+0.30×1.737+0.15×2.737+0.05×4.322+
0.03×5.059+0.01×6.644+0.01×6.644=1.95位

2．扩展编码法

最终目的是使操作码在冗余量尽量小的基础上，努力使操作码规整一些。采用扩展编码法，可以有多种结果。

在扩展编码时要先考虑地址多的指令的设计，然后再考虑地址次多的指令，最后考虑零地址指令。

4.6.2 地址码的优化设计

1．地址码个数的选择

系统设计时如何选择地址码的个数，其主要标准有两个：

（1）对于程序存储量的计算

（2）对于程序执行速度的计算

2．缩短单个地址码长度的方法

(1) 用间接寻址方式缩短地址码长度。

(2) 用变址寻址方式缩短地址码长度。

(3) 用寄存器间址寻址方式缩短地址码长度。

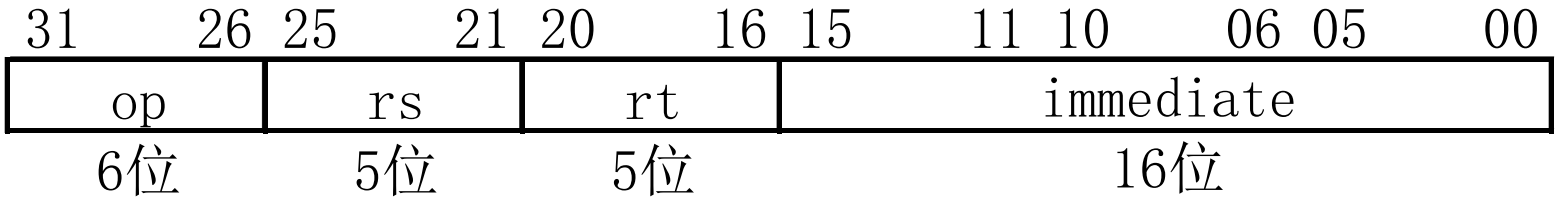
1.指令系统举例(MIPS32)

（1）计算类指令(Computational)

计算类指令用于执行算术操作，乘/除，逻辑操作和对寄存器进行移位操作。这些指令有两种类型：寄存器类型和立即数类型。寄存器类型的指令使用两个源寄存器的值作为源操作数，立即数类型使用一个寄存器和立即数作为源操作数。根据操作的不同，这些指令分为下面4种：

4.7 指令系统举例

1）ALU立即数指令



ADDI rt, rs, immediate ; rt←rs+immediate; 有符号加

- ADDI $0,$1,0001H ; $0←$1+0001H

001000 00001 00000 0000000000000001

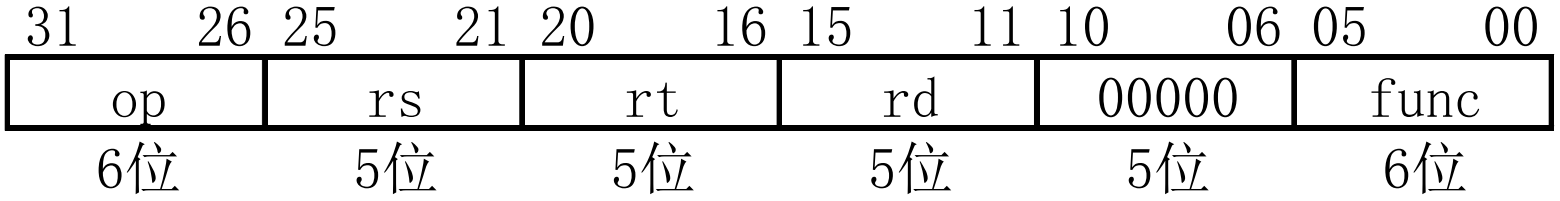
OP rs, rt,mmediate

- ADDI $2,$5,0010H ; $2←$5+0010H

001000 00101 00010 0000000000010000

4.7 指令系统举例

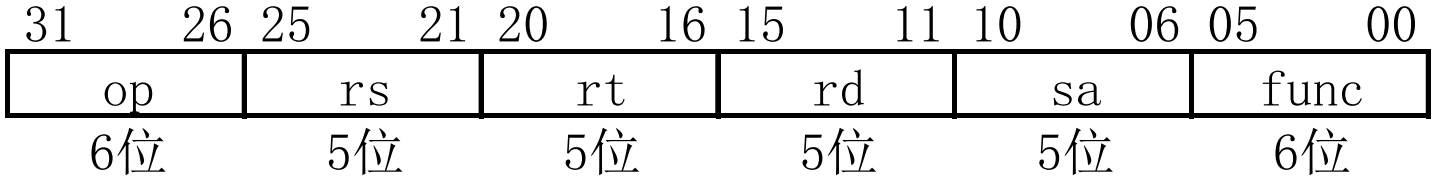
2）3操作数指令




```
ADD rd, rs, rt ; rd←rs+rt; 有符号加
ADD $0, $1, $2 ; $0←$1+$2
000000 00001 00010 00000 00000 100000
```

4.7 指令系统举例

3）移位指令



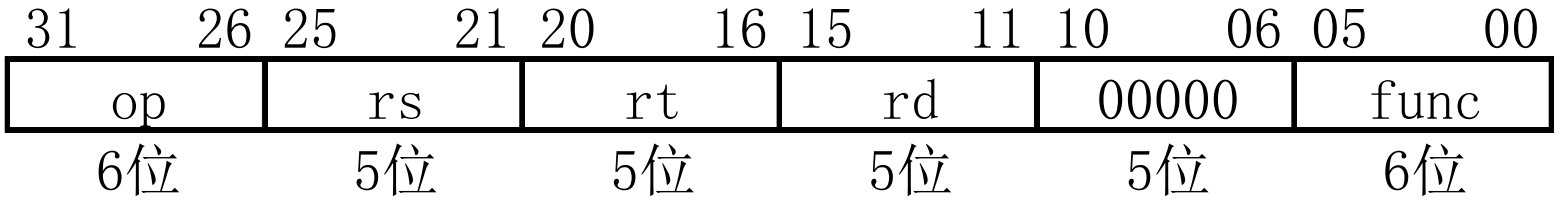
SLL rd, rt, sa ; rd←rt << sa; 逻辑左移sa位

SLL $4, $5, 6 ; $4←$5 << 6; 逻辑左移6位

000000 00000 00101 00100 00110 000000

4.7 指令系统举例

4）乘/除法指令



MUL rd, rs, rt ； rd←rs×rt（只存储结果低位）

MUL $1, $2, $3 ； $1←$2×$3

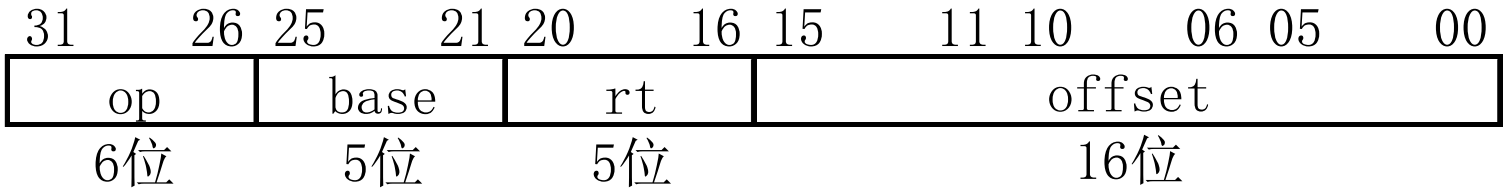
011100 00010 00011 00001 00000 000010

4.7 指令系统举例

(2) Load/Store指令

Load和Store指令都为立即数（I-type）类型，用来在存储器和通用寄存器之间的储存和装载数据。值得一提的是MIPS指令集只有该类指令访问内存，而其他指令都在寄存器之间进行，所以指令的执行速度较高。该类指令只有基址寄存器的值加上扩展的16位有符号立即数一种寻址模式，数据的存取方式可以是字节（byte）、字（word）和双字（Double word）。

4.7 指令系统举例



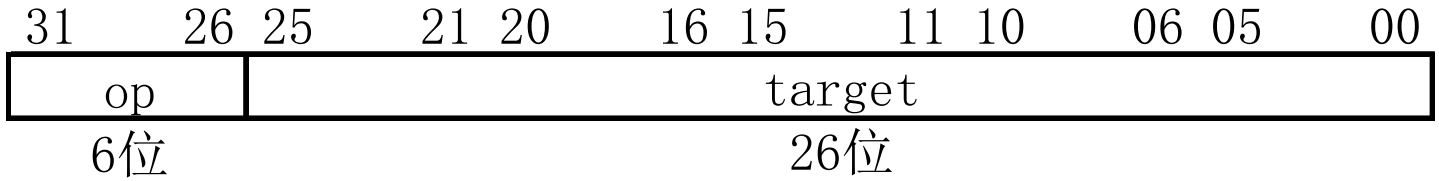

```
LW rt, offset(base) ; rt←memory[base+offset]; Load全字
Lw $5, 0x100($7)      ; $5←[$7+0x100]
100011 00111 00101 0000000100000000

SW rt, offset(base) ; memory[base+offset]←rt
```

4.7 指令系统举例

（3）跳转jump

所有的跳转目的的第一条指令从存储器中取出并执行。

Jump指令格式：



J target ；在当前指令附近256MB的范围内跳转

4.7 指令系统举例

(4) 寄存器传送指令

寄存器传送指令用来在系统的通用寄存器（GPR）、乘除法专用寄存器（HI、LO）之间传送数据，这些指令分为有条件传送和无条件传送2种类型。

(5) 专用指令

专用指令用来产生软件中断，当执行这类指令的时候，CPU产生异常并转入中断处理程序。这些指令有系统调用（Syscall），暂停（Break）和Trap指令等，主要用于软件的异常处理。

4.7 指令系统举例

（6）协处理器指令

协处理器指令对协处理器进行操作。协处理器的Load和Store指令是立即数类型，每个协处理器指令的格式依协处理器不同而不同。

（7）系统控制协处理器（CP0）指令

系统控制协处理器（CP0）指令执行对CP0寄存器的操作来控制处理器的存储器并执行异常处理。

4.8.1 复杂指令系统计算机

随着VLSI技术的发展，计算机的硬件成本不断下降，软件成本不断提高，使得人们热衷于在指令系统中增加更多的指令和复杂的指令，来提高操作系统的效率，并尽量缩短指令系统与高级语言的语义差别，以便于高级语言的编译和降低软件成本，某些计算机的指令多达几百条。这种计算机被称为复杂指令系统计算机CISC(Complex Instruction Set Computer)。

4.8 精简指令系统计算机（RISC）和复杂指令统计算机（CISC）

但是日趋庞大的指令系统不但使计算机的研制周期变长，而且增加了调试和维护的难度，还可能降低系统的性能。

如：Intel X86、VAX-11/780、M68000等。

CISC主要特点：

- 指令系统庞大，指令功能复杂，指令格式、寻址方式多；
- 绝大多数指令需多个机器周期完成；
- 各种指令都可访问存储器；

4.8 精简指令系统计算机（RISC）和复杂指令统计算机（CISC）

- 采用微程序控制；
- 有少量专用寄存器；
- 难以用优化编译技术生成高效的目标代码程序；

主张CISC的理由：

① 简化编译器

② 改善性能

统计11个C程序：

系统	占用存储器
RISC I	1.0
VAX-11/780	0.8
M68000	0.9
Z8002	1.2
PDP-11/70	0.9

4.8.2 精简指令系统计算机

研究表明，实际程序执行过程中80%到90%时间是花在指令系统10%到20%的常用指令上，其次，复杂的指令系统需要有复杂的控制器，这将占用60%的芯片面积。

1. RISC的产生

1975年IBM公司开始研究指令系统的合理性问题，IBM的John cocke提出精简指令系统的想法。

4.8 精简指令系统计算机（RISC）和复杂指令统计算机（CISC）

后来美国加州伯克莱大学的RISC Ⅰ和RISC Ⅱ机、斯坦福大学的MIPS机的研究成功，为精简指令系统计算机(reduced instruction set computer，简称RISC)的诞生与发展起了很大作用。

4.8 精简指令系统计算机（RISC）和复杂指令统计算机（CISC）

RISC主要特点：

- 指令系统小，指令格式少、寻址方式少；
- 绝大多数指令1个机器周期完成；
- 通用寄存器数量多；
- 只有Load/Store指令可访问存储器，其余指令只能访问寄存器；
- 以硬布线控制器为主。

CISC与RISC的主要特征对比

功　　能	CISC	RISC
指令系统	复杂、庞大	简单、精炼
指令条数	>200	<100
可访存指令	不加限制	只有LOAD/STORE
各种指令使用频率	相差太大	相差不大
各种指令执行时间	相差太大	绝大多数一周期完成
优化编译系统	很难	较容易
程序源代码长度	短	长
控制逻辑实现方式	绝大多数微程序控制	绝大多数为硬连线控制

习 题

P112

习题：1,2,5,9,11

请分析MIPS指令系统的指令格式和指令的二进制编码，从MIPS指令的格式、编码和功能上分析MIPS指令的特点。