

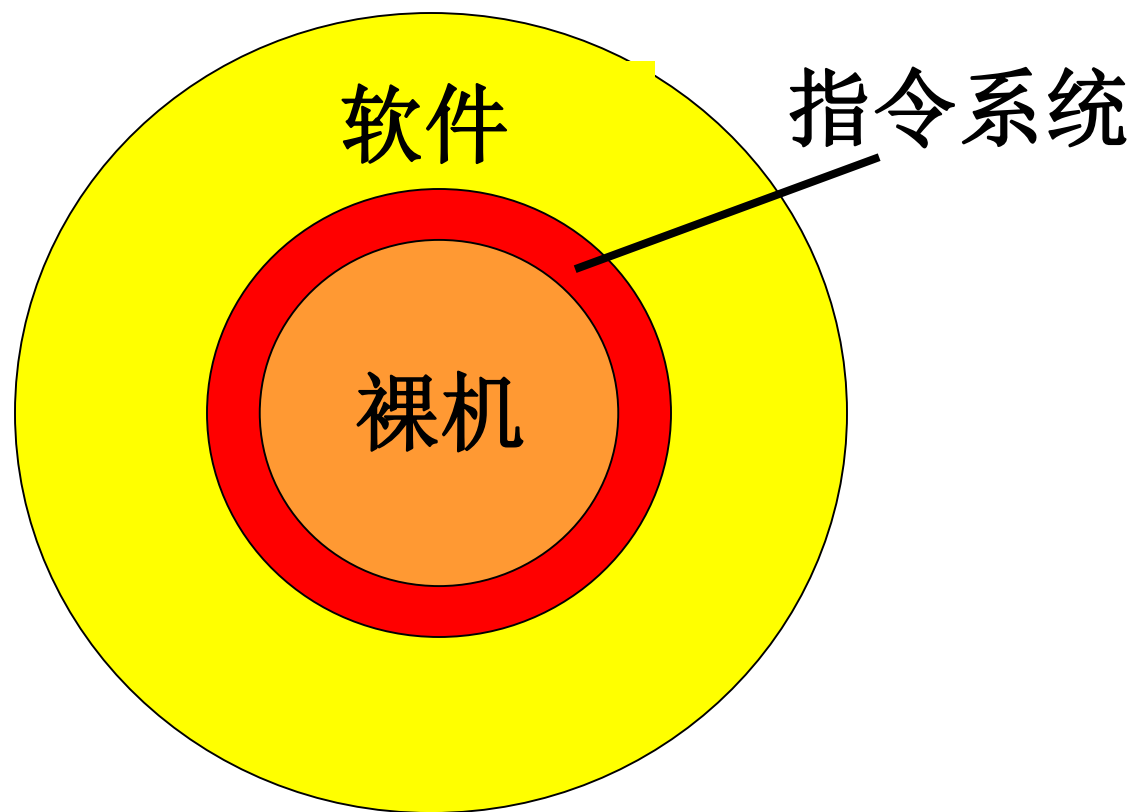


河北师范大学软件学院
Software College of Hebei Normal University

计算机组成原理

第三章 指令系统

指令和指令系统是计算机中最基本的概念。指令是指示计算机执行某些操作的命令，一台计算机的所有指令的集合构成该机的指令系统，也称指令集。指令系统是计算机的主要属性，位于硬件和软件的交界面上。本章将讨论一般计算机的指令系统所涉及的基本问题。





3.1 指令格式

3.2 寻址技术

3.3 堆栈与堆栈操作

3.4 指令类型

3.5 指令系统的发展





3.1 指令格式

3.2 寻址技术

3.3 堆栈与堆栈操作

3.4 指令类型

3.5 指令系统的发展



3.1 指令格式

3.1.1 机器指令的基本格式

一条指令就是机器语言的一个语句，它是一组有意义的二进制代码。指令的基本格式如下：



操作码：指明操作的性质及功能。

地址码：指明操作数的地址，特殊情况下也可能直接给出操作数本身。

注意

3.1 指令格式

指令的长度是指一条指令中所包含的二进制代码的位数，它取决于操作码字段的长度、操作数地址的个数及长度。指令长度应：

- ① 尽可能短
- ② 等于字节的整数倍

指令长度可以等于机器字长，也可以大于或小于机器字长。

在一个指令系统中，若所有指令的长度都是相等的，称为定长指令字结构；若各种指令的长度随指令功能而异，称为变长指令字结构。

3.1 指令格式

3.1.2 地址码结构

一条**双操作数指令**的除操作码之外，还应包含以下信息：

第一操作数地址，用 A_1 表示；

第二操作数地址，用 A_2 表示；

操作结果存放地址，用 A_3 表示；

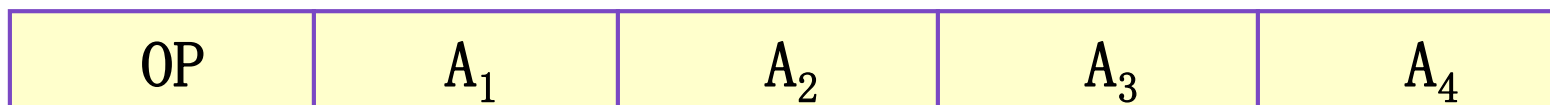
下条将要执行指令的地址，用 A_4 表示。

这些信息可以在指令中明显的给出，称为**显地址**；也可以依照某种事先的约定，用隐含的方式给出，称为**隐地址**。

下面介绍几种指令格式。

3.1 指令格式

1. 四地址指令

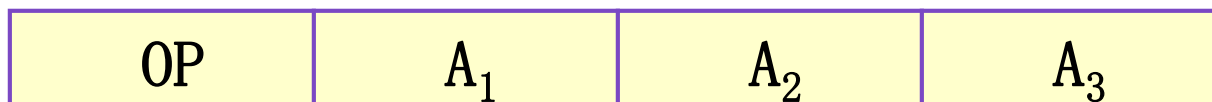


$(A_1)OP(A_2) \rightarrow A_3$

A_4 = 下条将要执行指令的地址

3.1 指令格式

2.三地址指令



程序计数器：
存放当前指令地址

$(A_1) \rightarrow P(A_2) \rightarrow A_3$

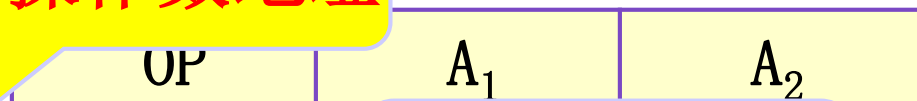
$(PC)+1$ =下条将要执行指令的地址

执行一条三地址指令需4次访问主存。

3.1 指令格式

3 二地址指令

目的操作数地址



$(A_1)OP(A_2) \rightarrow A_2$

源操作数地址

$(PC)+1$ =下条将要执行指令的地址

A_1 中原存内容在指令执行后被破坏。

执行一条二地址指令需4次访问主存。

指令格式

累加寄存器

一地址指令



$$(A_{cc})OP(A_1) \rightarrow A_{cc}$$

$(PC)+1$ =下条将要执行指令的地址

执行一条一地址指令需2次访问主存。

3.1 指令格式

5. 零地址指令

0P

操作数地址是隐含的。参加运算的操作数放在堆栈中，运算结果也放在堆栈中。有关堆栈的概念将在稍后讨论。

3.1 指令格式

指令中地址个数的选取要考虑诸多的因素。从缩短程序长度，用户使用方便，增加操作并行度等方面来看，选用三地址指令格式较好；从缩短指令长度，减少访存次数，简化硬件设计等方面来看，一地址指令格式较好。对于同一个问题，用三地址指令编写的程序最短，但指令长度最长，而用二、一、零地址指令来编写程序，程序的长度一个比一个长，但指令的长度一个比一个短。

3.1 指令格式

3.1.3 指令的操作码

指令系统中的每一条指令都有一个唯一确定的操作码，指令不同，其操作码的编码也不同。为了能表示整个指令系统中的全部指令，指令的操作码字段应当具有足够的位数。

指令操作码的编码可以分为**规整型**和**非规整型**两类：

规整型（定长编码）

非规整型（变长编码）

3.1 指令格式

1. 规整型

操作码字段的位数和位置是固定的。

假定：指令系统共有 m 条指令，指令中操作码字段的位数为 N 位，则有如下关系式：

$$N \geq \log_2 m$$

IBM 370机（字长32位）的指令可分为三种不同的长度形式：半字长指令、单字长指令和一个半字长指令。不论指令的长度为多少位，其中操作码字段**一律都是8位**，8位操作码字段允许容纳256条指令，实际上在IBM 370机中仅有183条指令。

定长编码对于简化硬件设计，减少指令译码的时间是非常有利的，但存在着信息冗余。

3.1 指令格式

2.非规整型

操作码字段的位数不固定，且分散地放在指令字的不同位置上。

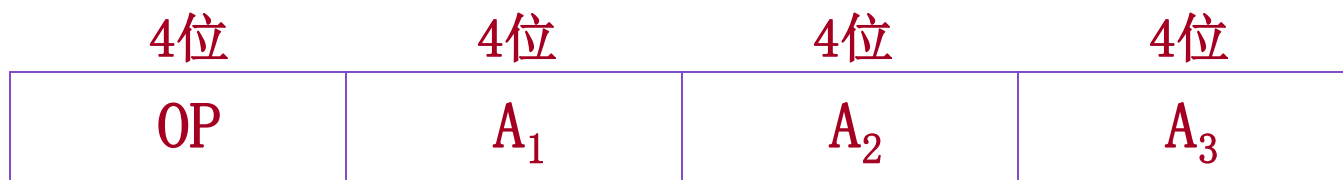
PDP-11机（字长**16位**）的指令分为单字长、两字长、三字长三种，操作码字段占**4~16位**不等，可遍及整个指令长度。

操作码字段的位数和位置不固定将增加指令译码和分析的难度，使控制器的设计复杂化。

最常用的非规整型编码方式是扩展操作码法
让操作数地址个数多的指令（如：三地址指令）的操作码字段短些，操作数地址个数少的指令（如一或零地址指令）的操作码字段长些。

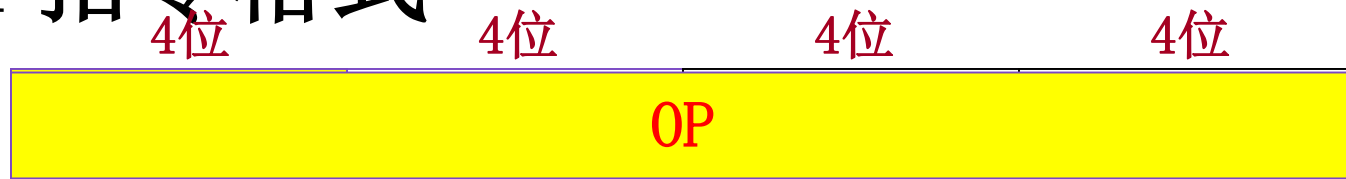
3.1 指令格式

例如：设某机的指令长度为16位，操作码字段为4位，有三个4位的地址码字段，其格式为：



如果按照定长编码的方法，4位操作码字段最多只能表示16条不同的三地址指令。

3.1 指令格式



0000 XXXX YYYY ZZZZ
⋮
110 XXXX YYYY ZZZZ

15条三地址指令

扩展窗口

111 0000 XXXX YYYY
⋮
111 1110 XXXX YYYY

15条二地址指令

扩展窗口

1111 1111 0000 XXXX
⋮
1111 1111 1110 XXXX

15条一地址指令

扩展窗口

1111 1111 1111 0000
⋮
1111 1111 1111 1111

16条零地址指令

练习:

- ▶ 1.某机器字长32位，指令单字长，指令系统中具有二地址指令、一地址指令和零地址指令各若干条，已知每个地址长12位，采用扩展操作码方式，问该指令系统中的二地址指令、一地址指令、零地址指令各最多能有多少条？
- ▶ 2.指令字长为12位，每个地址码为3位，采用扩展操作码的方式，设计4条三地址指令、16条二地址指令、64条一地址指令和16条零地址指令。
 - ▶ (1) 给出一种操作码的扩展方案
 - ▶ (2) 画出指令译码逻辑
 - ▶ (3) 计算操作码的平均长度

若某机为定长指令字结构，要求：三地址指令4条，二地址指令8条，单地址指令180条。设指令字长为12位.每个地址码长为3位。问能否以扩展操作码为其编码?如果其中单地址指令为254条呢?说明其理由。

答案：∵指令字长 12 位，每个地址码占 3 位;↵

∴三地址指令最多是 $2^{(12-3-3-3)}=8$ 条， 现三地址指令需 4 条，↵

∴可有 4 条编码作为扩展码，↵

∴二地址指令最多为 $4 \times 2^3=32$ 条，↵

现要求二地址指令 8 条， ∴可有 24 个编码作扩展码↵

∴单地址指令最多为 $24 \times 2^3=192$ 条↵



3.1 指令格式

3.2 寻址技术

3.3 堆栈与堆栈操作

3.4 指令类型

3.5 指令系统的发展



3.2 寻址技术

寻址，指的是寻找操作数的地址或下一条将要执行的指令地址。寻址技术包括编址方式和寻址方式。

3.2 寻址技术

3.2.1 编址方式

1. 编址

通常，指令中的地址码字段将指出操作数的来源和去向，而操作数则存放在相应的存储设备中。在计算机中需要编址的设备主要有CPU中的通用寄存器、主存储器和输入输出设备等3种。

3.2 寻址技术

2. 编址单位

(1) 字编址

编址单位=访问单位

每个编址单位所包含的信息量（二进制位数）与读或写一次寄存器、主存所获得的信息量是相同的。早期的大多数机器都采用这种编址方式。

3.2 寻址技术

(2) 字节编址

字节编址是为了适应非数值计算的需要。字节编址方式使编址单位与信息的基本单位（一个字节）相一致，这是它的最大优点。然而，如果主存的访问单位也是一个字节的话，那么主存的频带就太窄了。

编址单位 < 访问单位

通常主存的访问单位是编址单位的若干倍。

(3) 位编址

也有部分计算机系统采用位编址方式。

3.2 寻址技术

3. 指令中地址码的位数

指令格式中每个地址码的位数是与**主存容量**和**最小寻址单位**（即编址单位）有关联的。主存容量越大，所需的地址码位数就越长。对于相同容量来说，如果以字节为最小寻址单位，地址码的位数就需要长些；如果以字为最小寻址单位（假定字长为16位或更长），地址码的位数可以减少。设某机主存容量为 2^{20} 个字节，机器字长32位。若最小寻址单位为字节（按字节编址），其地址码应为**20**位；若最小寻址单位为字（按字编址），其地址码只需**18**位。

3.2 寻址技术

3.2.2 数据寻址和指令寻址

寻址可以分为数据寻址和指令寻址。寻找操作数的地址称为数据寻址，**数据寻址方式较多，其最终目的都是寻找所需要的操作数**。寻找下一条将要执行的指令地址称为指令寻址，指令寻址比较简单，它又可以细分为顺序寻址和跳跃寻址。



注意

3.2 寻址技术

顺序寻址可通过程序计数器PC加1，自动形成下一条指令的地址；跳跃寻址则需要通过程序转移类指令实现。

跳跃寻址的转移地址形成方式有三种：直接（绝对）、相对和间接寻址，它与下面介绍的数据寻址方式中的直接、相对和间接寻址是相同的，只不过寻找到的不是操作数的有效地址而是转移的有效地址而已。

3.2 寻址技术

3.2.3 基本的数据寻址方式

寻址方式是根据指令中给出的地址码字段寻找真实操作数地址的方式。

指令中的形式地址A ^{寻址方式} → 有效地址EA

1. 立即寻址



在取指令时，操作码和操作数被同时取出，不必再次访问存储器，从而提高了指令的执行速度。但是，因为操作数是指令的一部分，不能被修改，且立即数的大小将受到指令长度的限制。

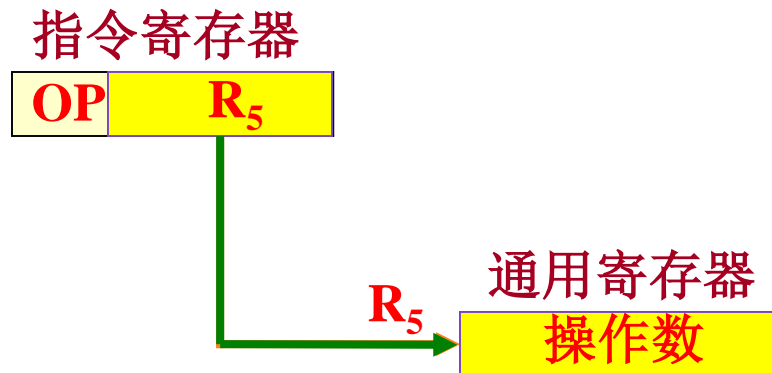
3.2 寻址技术

2. 寄存器寻址

指令中地址码部分给出某一通用寄存器的编号，所指定的寄存器中存放着操作数。

两个明显的优点：

- 从寄存器存取数据比主存快得多；
- 由于寄存器的数量较少，其地址码字段比主存单元地址字段短得多。



$EA = R_i$
操作数 $S = (R_i)$

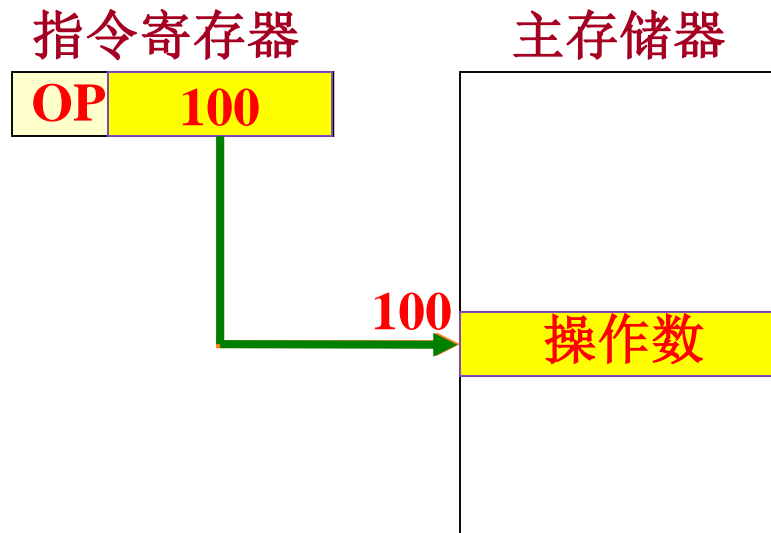
3.2 寻址技术

3.直接寻址

指令中地址码字段给出的地址A就是操作数的有效地址：

$$EA=A$$

由于操作数地址是不能修改的，与程序本身所在的位置无关，所以又叫做**绝对寻址**方式。



$$\text{操作数} S = (A)$$

3.2 寻址技术

4. 间接寻址

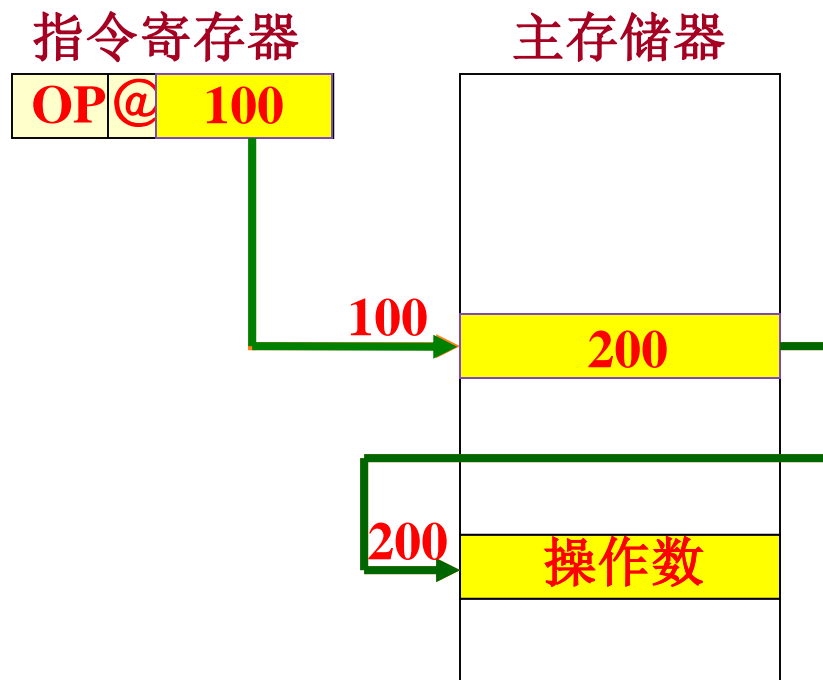
指令中给出的地址A不是操作数的地址，而是存放操作数地址的地址。

$EA = (A)$

通常在指令格式中划出一位@作为标志位。

@=0 直接寻址

@=1 间接寻址



操作数 $S = ((A))$

3.2 寻址技术

间接寻址要比直接寻址灵活得多，它的主要优点为：

- 扩大了寻址范围，可用指令的短地址访问大的主存空间。
- 可将主存单元作为程序的地址指针，用以指示操作数在主存中的位置。当操作数的地址需要改变时，不必修改指令，只需修改存放有效地址的那个主存单元（间接地址单元）的内容就可以了。

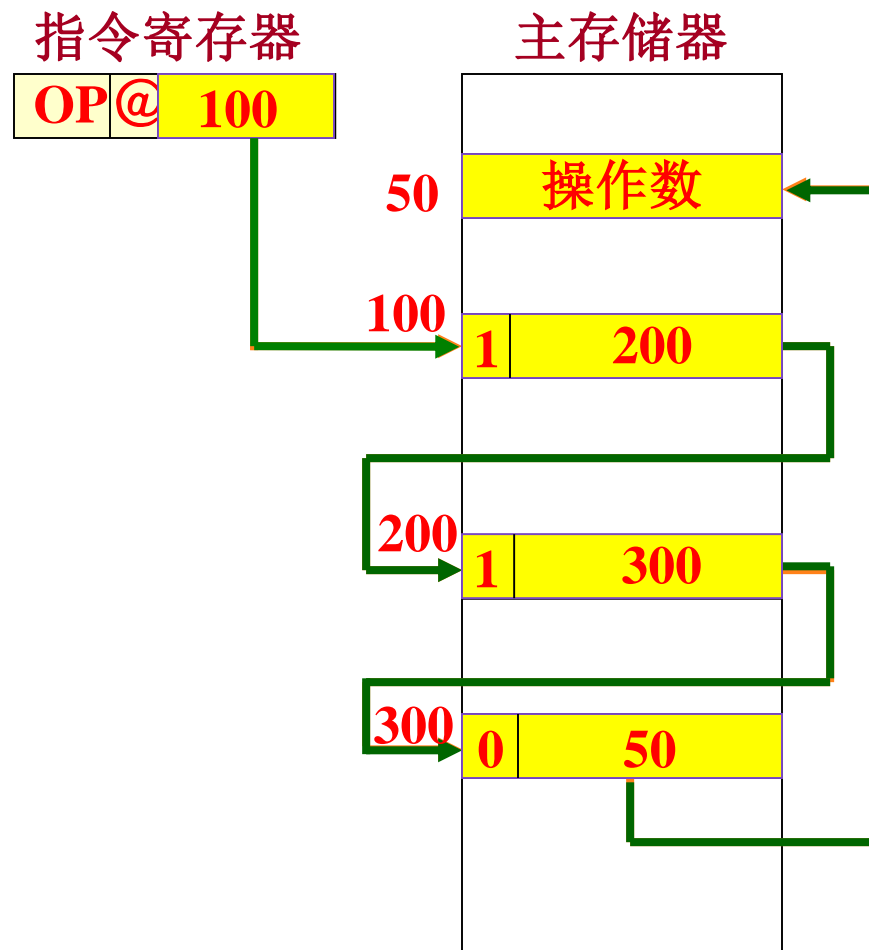
3.2 寻址技术

除去一级间接寻址外，还有多级间接寻址。多级间接寻址为取得操作数需要多次访问主存，即使在找到操作数有效地址后，还需再访问一次主存才可得到真正的操作数。

多级间接标志：

0：找到有效地址

1：继续间接寻址

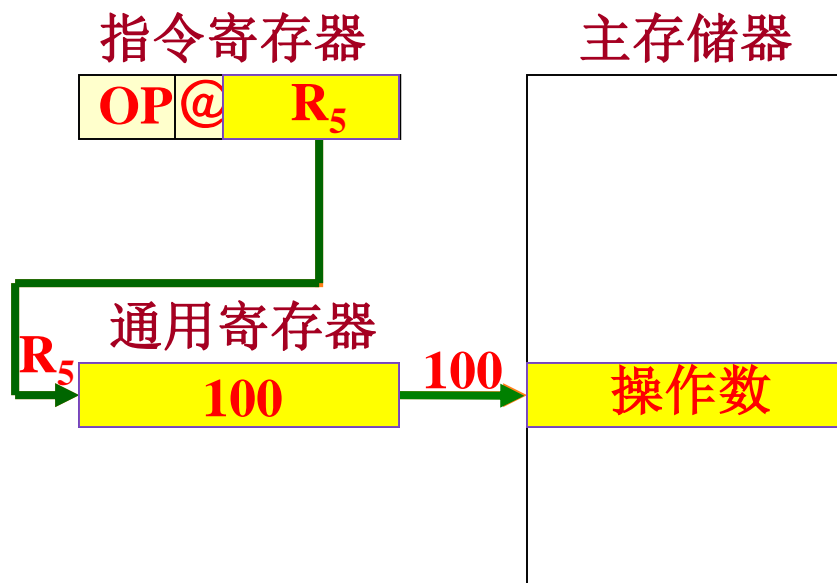


3.2 寻址技术

5. 寄存器间接寻址

指令中的地址码给出某一通用寄存器的编号，被指定的寄存器中存放操作数的有效地址，而操作数则存放在主存单元中。

这种寻址方式的指令较短，并且在取指后只需一次访存便可得到操作数。



$$EA = (R_i)$$

$$\text{操作数} S = ((R_i))$$

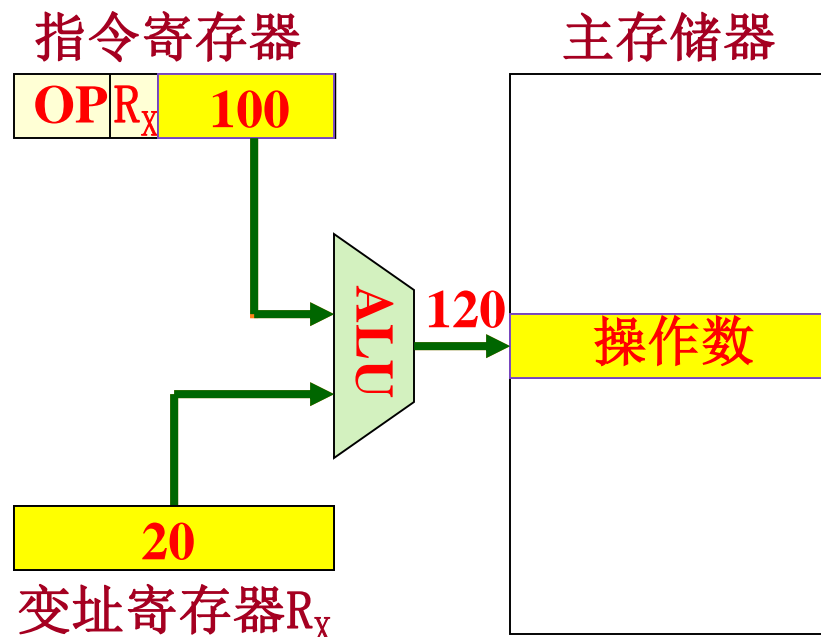
3.2 寻址技术

6.变址寻址

把指令给出的形式地址A与变址寄存器R_x的内容相加，形成操作数有效地址：

$$EA = A + (R_x)$$

R_x 的内容为变址值。



$$\text{操作数} S = (A + (R_x))$$

3.2 寻址技术

变址寻址是一种广泛采用的寻址方式，通常指令中的形式地址作为基准地址，而 R_x 的内容作为修改量。在遇到需要频繁修改地址时，无须修改指令，只要修改变址值就可以了。

例如：要把一组连续存放在主存单元中的数据（首地址是A）依次传送到另一存储区（首地址为B）中去，则只需在指令中指明两个存储区的首地址A和B（形式地址），用同一变址寄存器提供修改量K，即可实现 $(A+K) \rightarrow B+K$ 。变址寄存器的内容在每次传送之后自动地修改。

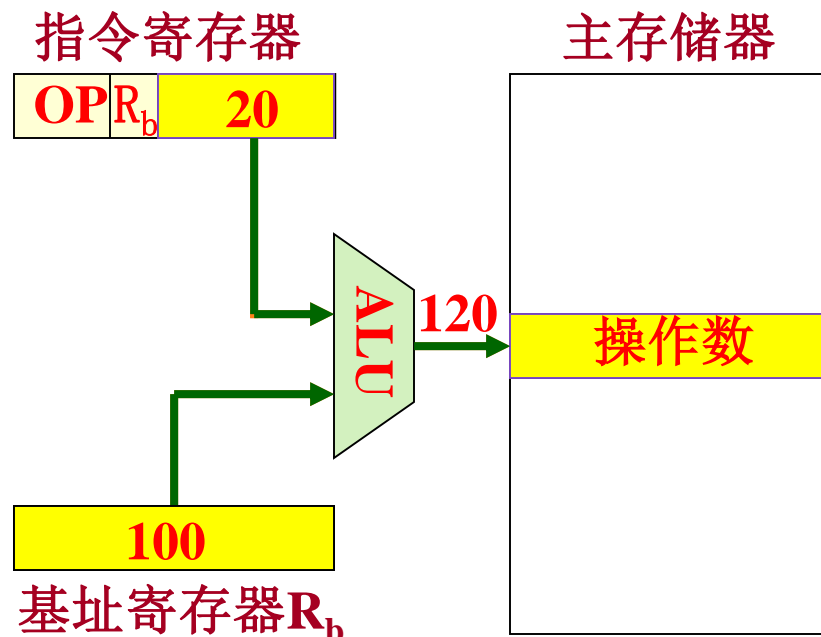
3.2 寻址技术

7. 基址寻址

将基址寄存器 R_b 的内容与位移量 D 相加，形成操作数有效地址：

$$EA = (R_b) + D$$

基址寄存器的内容称为基址值，指令的地址码字段是一个位移量，**位移量可正可负。**



$$\text{操作数} S = ((R_b) + D)$$

3.2 寻址技术

基址寻址和变址寻址在形成有效地址时所用的算法是相同的，而且在一些计算机中，这两种寻址方式都是由同样的硬件来实现的。

但这两种寻址方式应用的场合不同，变址寻址是面向用户的，用于访问字符串、向量和数组等成批数据；而基址寻址面向系统，主要用于逻辑地址和物理地址的变换，用以解决程序在主存中的再定位和扩大寻址空间等问题。在某些大型机中，基址寄存器只能由特权指令来管理，用户指令无权操作和修改。

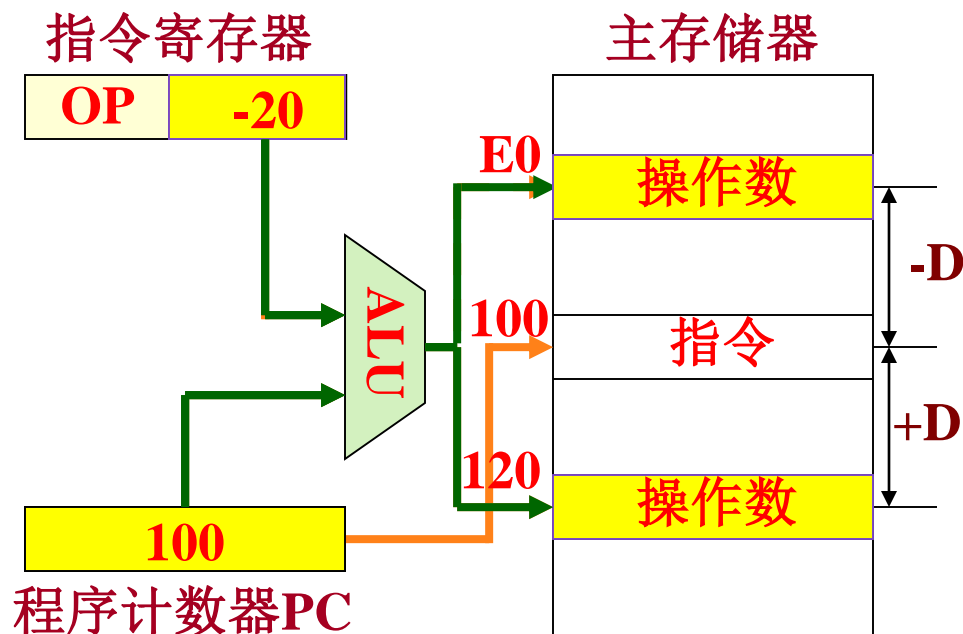
3.2 寻址技术

8. 相对寻址

相对寻址是基址寻址的一种变通，由程序计数器PC提供基准地址，即：

$$EA = (PC) + D$$

位移量指出的是操作数和现行指令之间的相对位置。



$$\text{操作数} S = ((PC) + D)$$

3.2 寻址技术

相对寻址方式的特点：

- 操作数的地址不是固定的，它随着PC值的变化而变化，并且与指令地址之间总是相差一个固定值 $\pm D$ 。当指令地址改变时，由于其位移量不变，使得操作数与指令在可用的存储区内一起移动，所以仍能保证程序的正确执行。采用PC相对寻址方式编写的程序可在主存中任意浮动，它放在主存的任何地方，所执行的效果都是一样的。
- 由于指令中给出的位移量可正、可负，所以对于指令地址而言，操作数地址可能在指令地址之前或之后。

3.2 寻址技术

9. 页面寻址

页面寻址相当于将整个主存空间分成若干个大小的相同的区，每个区称为一页，每页有若干个主存单元。每页都有自己的编号，称为页面地址；页面内的每个主存单元也有自己的编号，称为页内地址。这样，操作数的有效地址就被分为两部分：前部为页面地址，后部为页内地址。

页面地址	页内地址
------	------

3.2 寻址技术

根据页面地址的来源不同，页面寻址又可以分成3种不同的方式：

① 基页寻址，又称零页寻址。由于页面地址全等于0，所以有效地址 $EA=0 // A$ （//在这里表示简单拼接），操作数S在零页面中，基页寻址实际上就是直接寻址。

② 当前页寻址。页面地址就等于程序计数器PC的高位部分的内容，所以有效地址 $EA=(PC)_H // A$ ，操作数S与指令本身处于同一页面中。

③ 页寄存器寻址。页面地址取自页寄存器，与形式地址相拼接形成有效地址。

3.2 寻址技术

有些计算机在指令格式中设置了一个页面标志位（ Z/C ）。当 $Z/C=0$ ，表示零页寻址，当 $Z/C=1$ ，表示当前页寻址。

3.2 寻址技术

- ▶ 各种数据寻址方式获得数据的速度（由快到慢）
 - 立即寻址
 - 寄存器寻址
 - 直接寻址
 - 寄存器间接寻址
 - 页面寻址
 - 变址寻址（基址寻址、相对寻址）
 - 一级间接寻址
 - 多级间接寻址



注意

3.2 寻址技术

为了能区分出各种不同寻址方式，必须在指令中给出标识。标识的方式通常有两种：**显式和隐式**

。

显式的方法就是在指令中设置专门的寻址方式（**MOD**）字段，用二进制代码来表明寻址方式类型

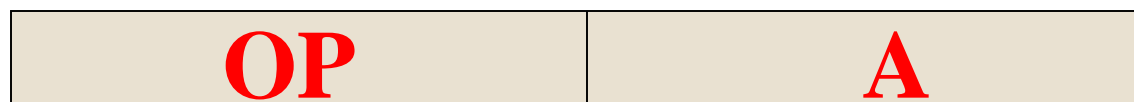
。

隐式的方式是由指令的操作码字段说明指令格式并隐含约定寻址方式。

显式



隐式



3.2 寻址技术

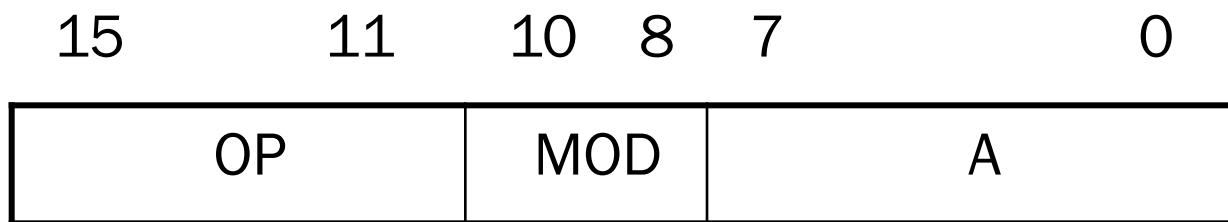
注意，一条指令若有两个或两个以上的地址码时，**各地址码可采用不同的寻址方式**。例如，源地址采用一种寻址方式，而目的地址采用另一种寻址方式。

MOV AX,(BX)

寄存器间接寻址

寄存器直接寻址

- ▶ 某计算机的字长为16位，存储器按字编址，访存指令格式如图所示：



- ▶ 其中，OP是操作码，MOD定义寻址方式，A为形式地址。设PC和 R_x 分别为程序计数器和变址寄存器，字长为16位，问：
 - ▶ 1.该格式能定义多少种指令？
 - ▶ 2.各种寻址方式的寻址范围是多少字？
 - ▶ 3.写出各种寻址方式的有效地址EA的计算公式。



3.1 指令格式

3.2 寻址技术

3.3 堆栈与堆栈操作

3.4 指令类型

3.5 指令系统的发展



3.3 堆栈与堆栈操作

堆栈是一种按特定顺序进行存取的存储区，这种特定顺序可归结为“**后进先出**”（LIFO）或“**先进后出**”（FILO）。

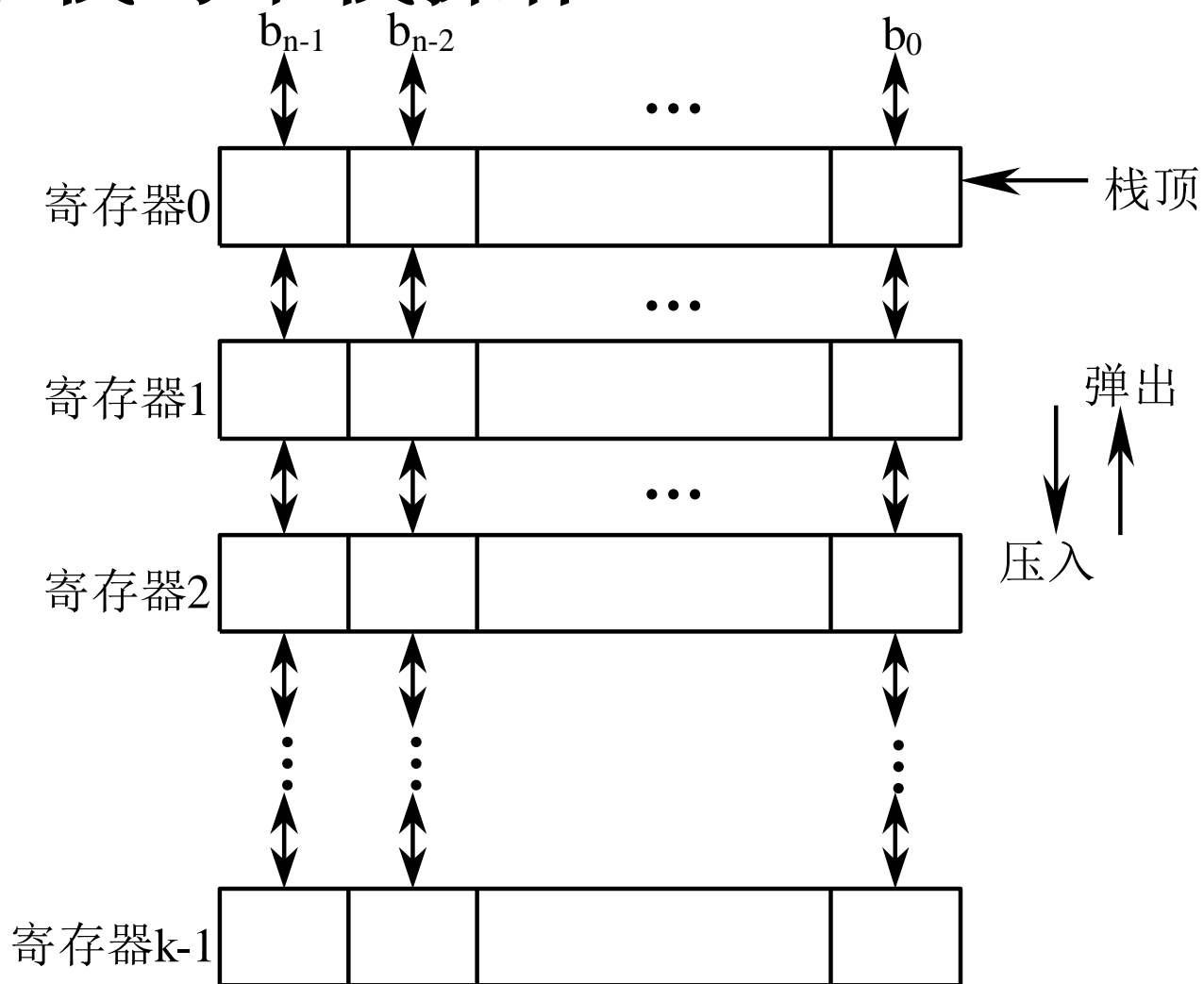
3.3 堆栈与堆栈操作

3.3.1 堆栈结构

1. 寄存器堆栈

用一组专门的寄存器构成寄存器堆栈，又称为硬堆栈。这种堆栈的栈顶是固定的，寄存器组中各寄存器是相互连接的，它们之间具有对应位自动推移的功能，即可将一个寄存器的内容推移到相邻的另一个寄存器中去。

3.3 堆栈与堆栈操作



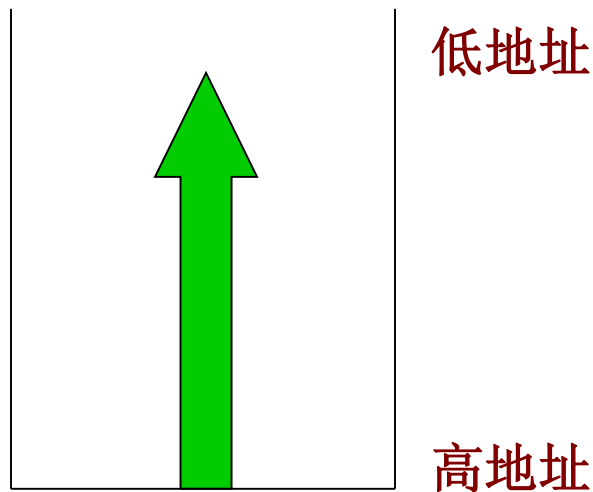
3.3 堆栈与堆栈操作

2. 存储器堆栈

从主存中划出一段区域来作堆栈，这种堆栈又称为软堆栈，堆栈的大小可变，栈底固定，栈顶浮动，故需要一个专门的硬件寄存器作为堆栈栈顶指针**SP**，简称栈指针。栈指针所指定的主存单元，就是堆栈的栈顶。

自底向上生成
方式的堆栈

堆
栈
区

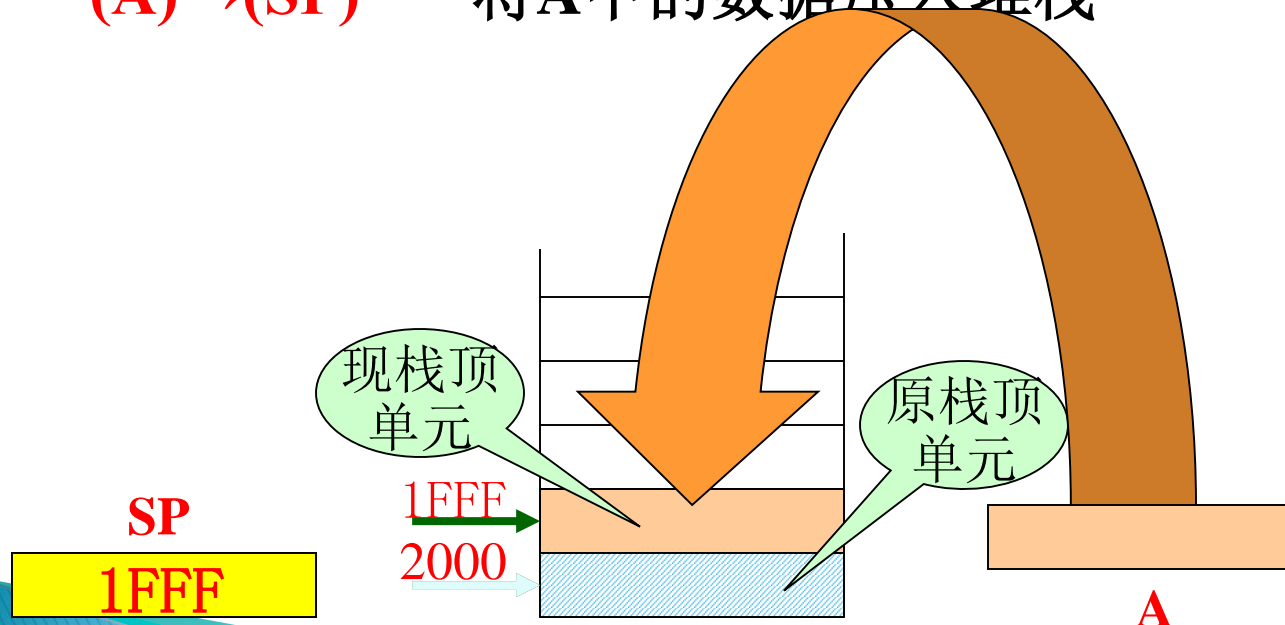


3.3 堆栈与堆栈操作

堆栈的栈底地址大于栈顶地址，通常栈指针始终指向**栈顶的满单元**。进栈时，SP的内容需要先自动减1，然后再将数据压入堆栈。

$(SP)-1 \rightarrow SP$ 修改栈指针

$(A) \rightarrow (SP)$ 将A中的数据压入堆栈



注意

3.3 堆栈与堆栈操作

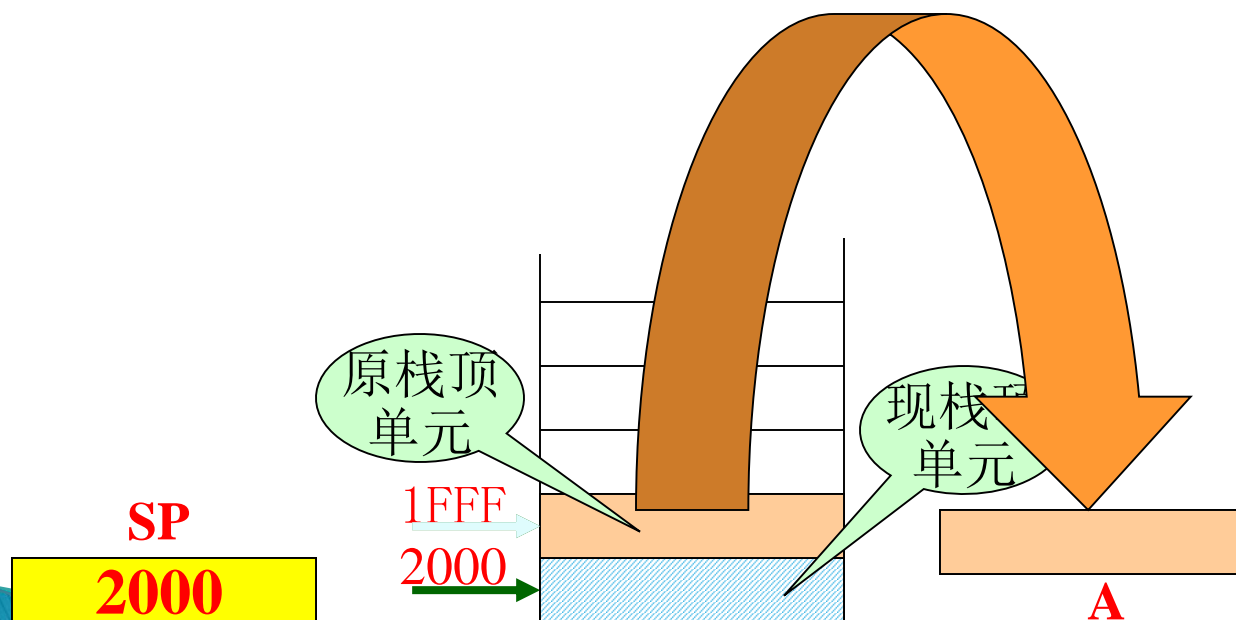
出栈时，需要先将堆栈中的数据弹出，然后SP的内容再自动加1。

$((SP)) \rightarrow A$

将栈顶内容弹出，送入A中

$(SP)+1 \rightarrow SP$

修改栈指针



注意

3.3 堆栈与堆栈操作

3.3.2 堆栈操作

在一般计算机中，堆栈主要用来暂存中断断点、子程序调用时的返回地址、状态标志及现场信息等，也可用于子程序调用时参数的传递，所以用于访问堆栈的指令只有进栈（压入）和出栈（弹出）两种。

在堆栈计算机（如HP-3000、B5000等）中，没有一般计算机中必备的通用寄存器，因此堆栈就成为提供操作数和保存运算结果的唯一场所。



3.1 指令格式

3.2 寻址技术

3.3 堆栈与堆栈操作

3.4 指令类型

3.5 指令系统的发展



3.4 指令类型

3.4.1 数据传送类指令

数据传送类指令主要用于实现寄存器与寄存器之间，寄存器与主存单元之间以及两个主存单元之间的数据传送。数据传送类指令又可以细分为：

1. 一般传送指令

一般传送指令（**MOV**）具有数据复制的性质，即数据从源地址传送到目的地址，而源地址中的内容保持不变。传送通常以字节、字、双字或数组为单位，特殊情况下也能按位为单位进行传送。

3.4 指令类型

(1)主存单元之间的传送

MOV mem₂,mem₁, 其含义为 $(\text{mem}_1) \rightarrow \text{mem}_2$

(2)从主存单元传送到寄存器

MOV reg,mem, 其含义为 $(\text{mem}) \rightarrow \text{reg}$

在有些计算机中, 该指令用助记符**LOAD**表示, 又称为取数指令。

3.4 指令类型

(3)从寄存器传送到主存单元

MOV mem,reg, 其含义为 $(\text{reg}) \rightarrow \text{mem}$

在有些计算机里, 该指令用助记符**STORE**表示, 又称为存数指令。

(4)寄存器之间的传送

MOV reg₂,reg₁, 其含义为 $(\text{reg}_1) \rightarrow \text{reg}_2$

3.4 指令类型

2.堆栈操作指令

堆栈指令是一种特殊的数据传送指令，分为进栈（**PUSH**）和出栈（**POP**）两种。

因为堆栈（指软堆栈）是主存的一个特定区域，所以对堆栈的操作也就是对存储器的操作。

3.数据交换指令

前述指令的传送都是单方向的，然而，数据传送也可以是双向的，即将源操作数与目的操作数（一个字节或一个字）相互交换位置。

3.4 指令类型

3.4.2 运算类指令

1. 算术运算指令

算术运算指令主要用于进行定点和浮点运算。这类运算包括加、减、乘、除以及加1、减1、比较等，有些机器还有十进制算术运算指令。

绝大多数算术运算指令都会影响到状态标志位，通常的标志位有进位、溢出、全零、正负和奇偶等。

3.4 指令类型

2. 逻辑运算指令

一般计算机都具有与、或、非、异或等逻辑运算指令。这类指令在没有设置专门的位操作指令的计算机中常用于对数据字（字节）中某些位（一位或多位）进行操作。

(1) 按位测（位检查）

$$\begin{array}{r} \text{XXXXX} \\ \wedge \quad 00010000 \\ \hline 000\text{X}0000 \end{array} \quad \text{AND AL,10H}$$

3.4 指令类型

(2) 按位清（位清除）

$$\begin{array}{r} \text{XXXXXXXXX} \\ \wedge \quad 1111101 \\ \hline \text{XXXXXX0X} \end{array} \quad \text{AND AL,FDH}$$

(3) 按位置（位设置）

$$\begin{array}{r} \text{X~~X~~XXXXXXXX} \\ \vee \quad 01000000 \\ \hline \text{X1XXXXXXXX} \end{array} \quad \text{OR AL,40H}$$

3.4 指令类型

(4)按位修改

利用“异或”指令可以修改目的操作数的某些位，只要源操作数的相应位为“1”，其余位为“0”，异或之后就达到了修改这些位的目的（因为 $\bar{A} \oplus 1 = A$ ， $A \oplus 0 = A$ ）。

$$\begin{array}{r} \text{XXXXX} \text{X} \text{XXXX} \\ \oplus \quad 00001000 \\ \hline \text{XXXXX} \bar{\text{X}} \text{XXXX} \end{array}$$

3.4 指令类型

(5) 判符合

若两数相符合，其异或之后的结果必定为“0”。

(6) 清0

XOR AL,AL

3.4 指令类型

3.移位指令

分为算术移位、逻辑移位和循环移位3类，它们又可分为左移和右移两种。

算术移位的对象是带符号数，算术移位过程中必须保持操作数的符号不变，左移一位，数值 $\times 2$ ，右移一位，数值 $\div 2$ 。

3.4 指令类型

逻辑移位的对象是没有数值含义的二进制代码，因此移位时不必考虑符号问题。

循环移位又按进位位是否一起循环分为两类

:

- 小循环（不带进位循环）
- 大循环（带进位循环）

3.4 指令类型

3.4.3 程序控制类指令

程序控制类指令用于控制程序的执行方向，并使程序具有测试、分析与判断的能力。

1. 转移指令

在程序执行过程中，通常采用转移指令来改变程序的执行方向。转移指令又分无条件转移和条件转移两种。

无条件转移指令（**JMP**）不受任何条件的约束，直接把程序转向新的位置执行。

3.4 指令类型

条件转移指令必须受到条件的约束，若条件满足时才转向新的位置执行，否则程序仍顺序执行。

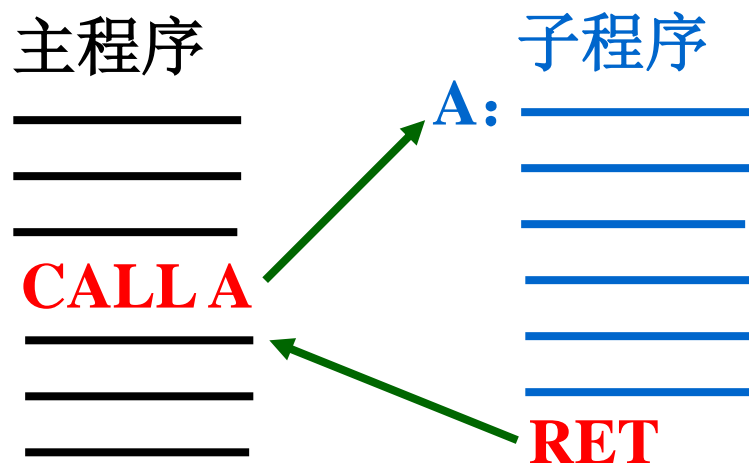
无论是条件转移还是无条件转移都需要给出转移地址。若采用相对寻址方式，转移地址为当前指令地址（即PC的值）和指令中给出的位移量之和，即： $(PC) + \text{位移量} \rightarrow PC$ ；若采用绝对寻址方式，转移地址由指令的地址码直接给出，即： $A \rightarrow PC$ 。

3.4 指令类型

2. 子程序调用指令

子程序是一组可以公用的指令序列，只要知道子程序的入口地址就能调用它。

从主程序转向子程序的指令称为子程序调用指令（**CALL**）；而从子程序转向主程序的指令称为返回指令（**RET**）。



3.4 指令类型

主程序和子程序是相对的概念，调用其它程序的程序是主程序；被其它程序调用的程序是子程序。

转子指令安排在主程序中需要调用子程序的地方，**转子指令是一地址指令**。

3.4 指令类型

子程序调用指令和转移指令都可以改变程序的执行顺序，但两者存在着很大的差别：

- 转移指令转移到指令中给出的转移地址处执行指令，不存在返回要求，没有返回地址问题；而子程序调用指令必须以某种方式保存返回地址，以便返回时能找到原来的位置。

- 转移指令用于实现同一程序内的转移；而子程序调用指令转去执行一段子程序，实现的是程序与程序之间的转移。

3.4 指令类型

保存返回地址的方法除去堆栈以外还有：

(1) 用子程序的第一个字单元存放返回地址。转子指令把返回地址存放在子程序的第一个字单元中，子程序从第二个字单元开始执行。返回时将第一个字单元地址作为间接地址，采用间址方式返回主程序。这种方法可以实现多重转子，但不能实现递归循环，Cyber70采用的就是这种方法。

3.4 指令类型

(2) 用寄存器存放返回地址。转子指令先把返回地址放到某一个寄存器中，再由子程序将寄存器中的内容转移到另一个安全的地方，比如存储器的某个区域。这是一种较为安全的方法，可以实现子程序的递归循环。**IBM370**采用这种方法，这种方法相对增加了子程序的复杂程度。

3.4 指令类型

3.返回指令

从子程序转向主程序的指令称为返回指令，其助记符一般为**RET**，子程序的最后一条指令一定是返回指令。

返回地址存放的位置决定了返回指令的格式，如果返回地址保存在堆栈中，则**返回指令是零地址指令**，如果返回地址保存在某个主存单元中，则返回指令中必须是一地址指令。

3.4 指令类型

3.4.4 输入/输出类指令

输入/输出（I/O）类指令用来实现主机与外部设备之间的信息交换，包括输入/输出数据、主机向外设发控制命令或外设向主机报告工作状态等。从广义的角度看，I/O指令可以归入数据传送类。

各种不同的计算机的I/O指令差别很大，通常有两种方式：

- 独立编址方式

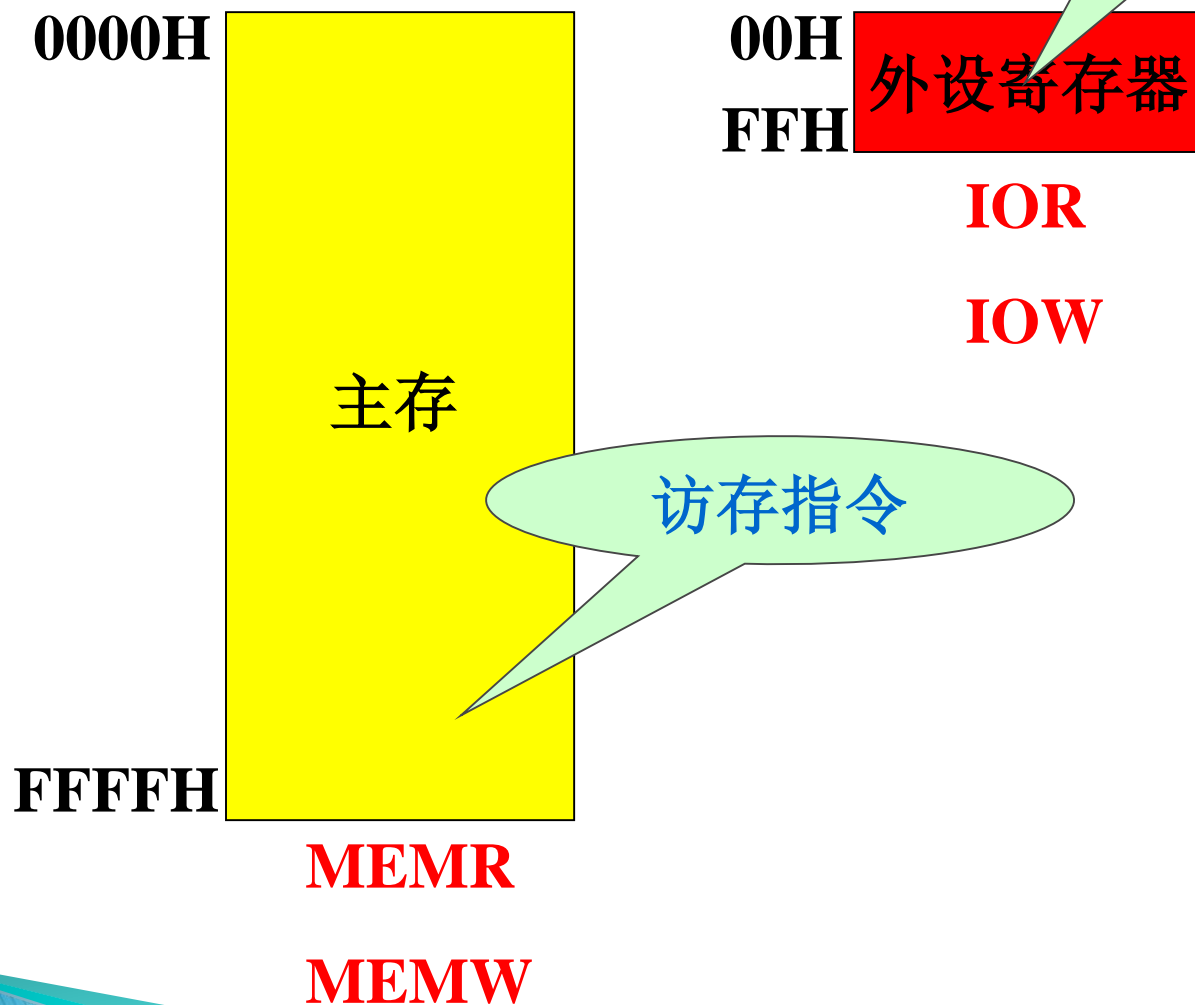
- 统一编址方式

3.4 指令类型

1. 独立编址的I/O

所谓独立编址就是把外设端口和主存单元分别独立编址。指令系统中有专门的**IN/OUT**指令。以主机为基准，信息由外设传送到主机称为输入，反之称为输出。指令中需要给出外设端口地址。这些端口地址与主存地址无关，是另一个独立的地址空间。

3.4 指令类型



3.4 指令类型

2.统一编址的I/O

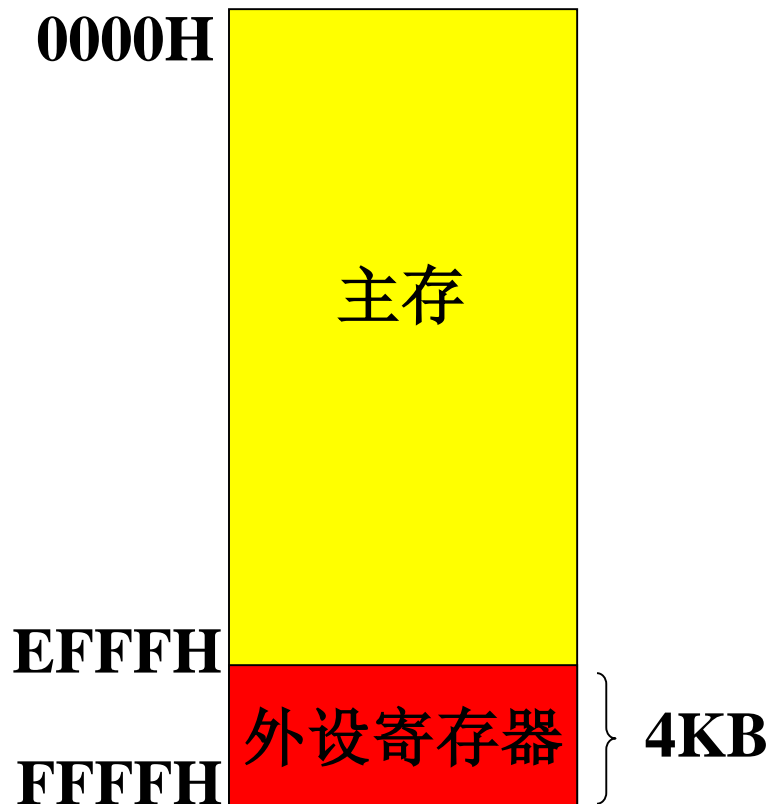
所谓统一编址就是把外设寄存器和主存单元统一编址。指令系统中没有专门的I/O指令，就用一般的数据传送类指令来实现I/O操作。

数据传送指令

输入指令

MOV R₀, (50H)

MOV R₀, (F000H)





3.1 指令格式

3.2 寻址技术

3.3 堆栈与堆栈操作

3.4 指令类型

3.5 指令系统的发展



3.5 指令系统的发展

3.5.1 x86架构的扩展指令集

- ▶ MMX指令集
 - ▶ SSE指令集
 - ▶ 3DNow指令集
 - ▶ SSE2指令集
 - ▶ SSE3指令集
- 

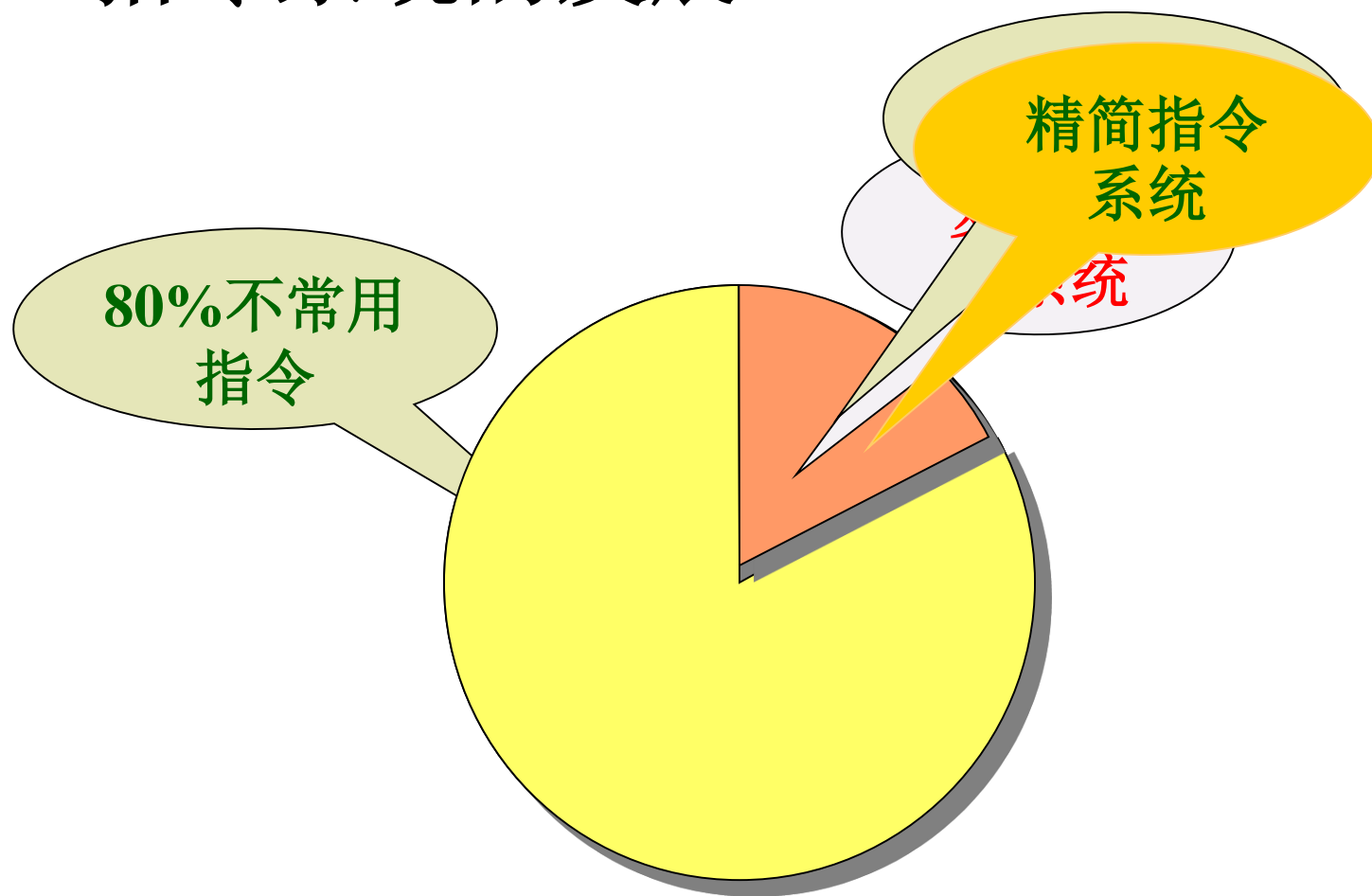
3.5 指令系统的发展

3.5.2 从复杂指令系统到精简指令系统

指令系统中指令丰富、功能强。这些计算机被称为复杂指令系统计算机，简称**CISC**。

大量测试表明，最常使用的是一些比较简单的指令，这类指令仅占指令总数的**20%**，但在各种程序中出现的频度却占**80%**，其余大多数指令是功能复杂的指令，这类指令占指令总数的**80%**，但其使用频度很低，仅占**20%**。因此，人们把这种情况称为“**20%-80%律**”。

3.5 指令系统的发展



3.5 指令系统的发展

从“20%-80%律”出发，人们开始了对指令系统合理性的研究，提出了精简指令系统的想法，出现了精简指令系统计算机，简称**RISC**。

RISC技术将在后继课程中介绍。

3.5 指令系统的发展

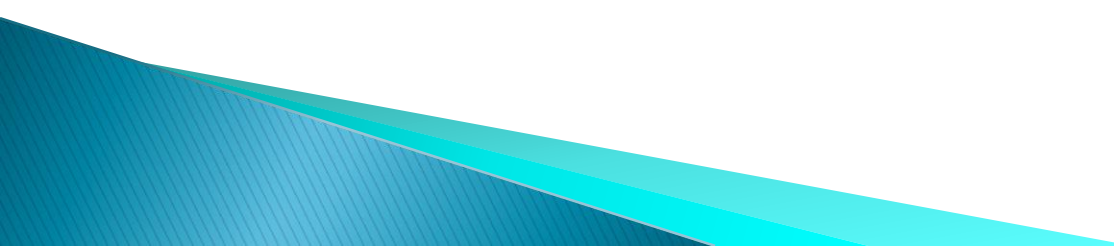
3.5.3 VLIW和EPIC

VLIW是英文“**Very Long Instruction Word**”的缩写，中文含义是“超长指令字”，即一种非常长的指令组合，它把许多条指令连在一起，增加了运算的速度。在这种指令系统中，编译器把许多简单、独立的指令组合到一条指令字中。当这些指令字从主存中取出放到处理器中时，它们被容易地分解成几条简单的指令，这些简单的指令被分派到一些独立的执行单元去执行。

EPIC是英文“**Explicit Parallel Instruction Code**”的缩写，中文含义是“显式并行指令代码”。EPIC是从VLIW中衍生出来的，通过将多条指令放入一个指令字，有效的提高了CPU各个计算功能部件的利用效率，提高了程序的性能。

第3章 小结

3.1 指令格式

- ▶ 指令的基本格式
 - ▶ 指令的地址码结构（3、2、1、0地址指令的区别）
 - ▶ 非规整型指令的操作码（扩展操作码）
- 

第3章 小结

3.2 寻址技术

- ▶ 编址方式

 - 字编址、字节编址

- ▶ 指令中地址码的位数

 - 主存容量、最小寻址单位

- ▶ 数据寻址和指令寻址

第3章 小结

- ▶ 常见数据寻址方式的特点

立即寻址、直接寻址、间接寻址、
相对寻址、变址寻址、页面寻址

- ▶ 各种数据寻址方式的速度区别

- ▶ 有效地址EA的计算

直接寻址、间接寻址、变址寻址、
页面寻址

第3章 小结

3.3 堆栈与堆栈操作

- ▶ 存储器堆栈
- ▶ 存储器堆栈操作

进栈、出栈时栈指针的修改和数据的压入和弹出

3.4 指令类型

- ▶ 程序控制类指令
转移、转子、返回指令的区别
- ▶ 输入/输出类指令
独立编址I/O、统一编址I/O

第3章 小结

3.5 指令系统的发展

- ▶ CISC
 - ▶ RISC
 - ▶ VLIW
 - ▶ EPIC
- 