指 令 系 统

- 概述
- 指令格式
- 指令和数据的寻址方式
- 堆栈寻址方式
- 典型指令

概述

计算机的程序是由一系列的指令组成的，指令就是要计算机执行某种操作的命令。一台计算机中所有机器指令的集合，称为指令系统。它是软件和硬件的主要界面。

指令系统的发展过程：

50年代：指令系统只有定点加减、逻辑运算、数据传送、转移等十几至几十条指令。

60年代后期：增加了乘除运算、浮点运算、十进制运算、字符串处理等指令，指令数目多达一二百条，寻址方式也趋多样化。

60年代后期开始出现系列计算机 (基本指令系统相同、基本体系结构相同)。同一系列的各机种有共同的指令集而且新推出的机种指令系统一定包含所有旧机种的全部指令。解决了软件兼容性问题。

70年代末期：大多数计算机的指令系统多达几百条。我们称这些计算机为复杂指令系统计算机 (CISC)。为降低复杂性，又提出了精简指令系统计算机（RISC）。

指令系统的基本要求

完备性：在有限的存储空间，对于任何可解的问题，编制软件时，指令系统所提供的指令足够使用。一个完备的指令系统应包括数据传送和交换指令、算术及逻辑运算指令、输入输出指令、程序控制指令、CPU控制指令。

有效性：用该指令系统编制的程序能高效率的运行。高效率表现在执行速度快，占有存储空间小。

规整性：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寄存器和存储单元都可同等对待，所有指令可使用相同的寻址方式，操作数和运算结果可不受约束的存入任一单元。例如加法指令，既有A←A＋B，也有B←A＋B。目前很多机器还不能实现这一对称性。匀齐性是指一种性质的操作可以适用于各种数据类型。指令格式和数据格式一致性是指指令字长和数据字长有规整的关系，通常都是字节长度的整数倍。例如机器字长为32位，长指令选32位，短指令选16位。

兼容性：系列计算机应做到“向上”兼容。

指令格式

从便于程序设计、增加操作并行性、提高指令功能看，指令中应包含多种信息。但指令太长会占用更多的存储空间，增加访存次数。一条指令应包含下列信息：

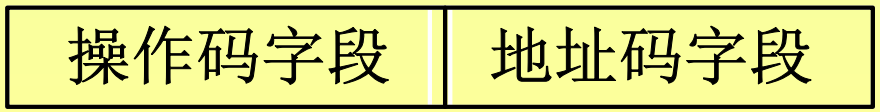
操作码：表明操作的性质和功能。每条指令都有相应的操作码。

操作数地址。

操作结果的存放地址。

下一条指令的地址。为了压缩指令长度，可以用一个程序计数器(Program Counter, PC) 存放指令地址。每执行一条指令，PC自动增加，指向下一条指令。由于使用了PC，指令中可以不包含下一条指令的地址。

一条指令实际上包含两种信息：操作码 (Operation Code, OP) 和地址码。其基本格式为：



指令字长

指令中包含二进制代码的位数，称为指令字长。指令字长等于机器字长的指令，称为单字长指令；指令字长等于半个机器字长的指令，称为半字长指令；指令字长等于两个机器字长的指令，称为双字长指令。

以IBM 370为例，其字长为32位。指令格式有16位（半字）的，有32位（单字）的，也有48位（一个半字）的。

使用多字长指令，目的在于提供足够的地址位来解决访问内存任何单元的寻址问题。其主要缺点是必须两次或多次访问内存以取出一整条指令，降低了CPU的运算速度，又占用了更多的存储空间。

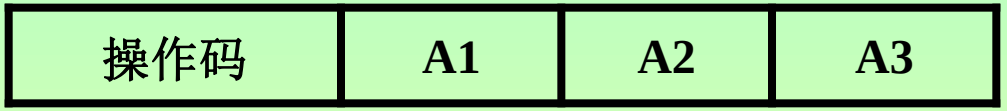
在一个指令系统中，如果各种指令字长度相等，称为等长指令字结构。这是RISC的特点。如果指令字长度随指令的功能而异，称为变长指令字结构。这是CISC的特点。这种指令字结构灵活，能充分利用指令长度，但指令的控制较复杂。PC机和传统的大、中、小型机都属于CISC，采用变长指令字。

地址码

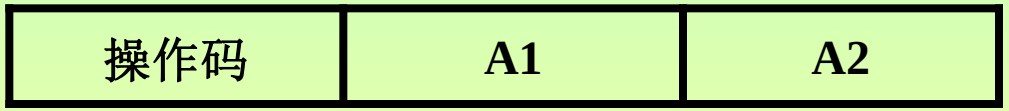
根据一条指令中有几个操作数地址，可将该指令称为几操作数指令或几地址指令。

一般的操作数有被操作数、操作数及操作结果这三种数，因而就形成了三地址指令格式。此外，还有二地址格式、一地址格式和零地址格式。

三地址指令：



二地址指令：



一地址指令：

零地址指令：

(1) 零地址指令的指令字中只有操作码，而没有地址码。如停机、空操作指令。

(2) 一地址指令常称为单操作数指令。通常这种指令以运算器中累加寄存器AC中的数据为被操作数，指令字的地址码字段所指明的数为操作数，操作结果又放回累加寄存器AC中。

$$(AC)\quad OP\quad (A)\quad \rightarrow\quad AC$$

OP表示操作性质；(AC)表示累加寄存器AC中的数；(A)表示内存中地址为A的存储单元中的数或运算器中地址为A的通用寄存器中的数； →表示把操作（运算）结果传送到指定的地方。

(3) 二地址指令常称为双操作数指令，它的两个地址码字段分别指明参与操作的两个数在内存中或运算器中通用寄存器的地址，A1作存放操作结果的地址。A1中原来的内容被覆盖。

$$(A1)\quad OP\quad (A2)\quad \rightarrow\quad A1$$

(4)三地址指令字中有三个操作数地址。

(A1) OP (A2) → A3

A1为被操作数地址，也称源操作数地址； A2为操作数地址，也称终点操作数地址； A3为存放结果的地址。 A1、A2、A3可以是内存中的单元地址，也可以是运算器中通用寄存器的地址。

二地址指令格式中，从操作数的物理位置来说，又可归结为三种类型。

存储器-存储器（SS）型指令：操作时都是涉及内存单元，参与操作的数都放在内存里，从内存某单元中取操作数，操作结果存放至内存另一单元中。

寄存器-寄存器（RR）型指令：需要多个通用寄存器或个别专用寄存器，从寄存器中取操作数，把操作结果放到另一寄存器。机器执行寄存器-寄存器型指令的速度很快，因为执行这类指令，不需要访问内存。

寄存器-存储器（RS）型指令：执行此类指令时，既要访问内存单元，又要访问寄存器。

操作码

指令系统中每条指令有唯一确定的操作码，表明指令的性质和功能。操作码可以采用固定长度，也可以采用可变长度。固定长度操作码对于简化硬件设计、减少指令译码时间有利，在字长较大的大、中型机和RISC中广泛采用。可变长度操作码可以有效压缩操作码的平均长度，在字长较短的小型机和微型机中采用。

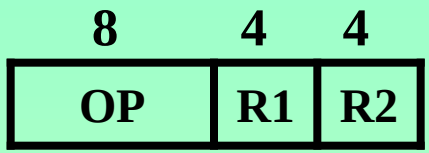
定长编码：操作码的长度决定了指令系统中指令的数目。若操作码长度为k位，最多只能有2^k条指令。

以IBM 370为例，其字长为32位。指令格式有16位（半字）的，有32位（单字）的，也有48位（一个半字）的。半字长指令不包含主存地址，单字长指令包含一个主存地址，一个半字长指令包含两个主存地址。

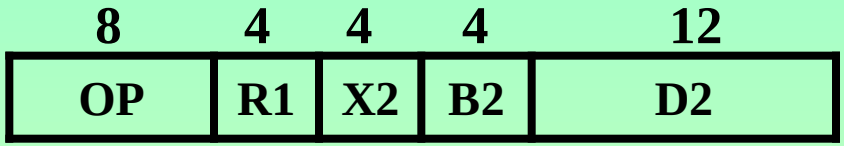
IMB 370机中，不论指令长度为多少，操作码字段都为8位。8位操作码允许指定256条指令，而IBM 370 只有183条指令，存在极大的信息冗余。

IBM 370的五种指令格式：

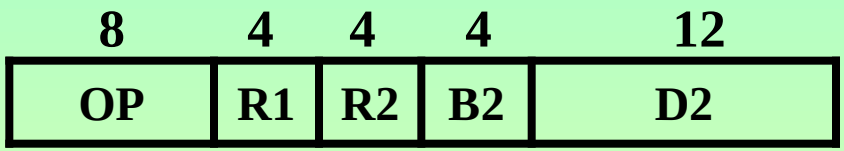
RR型：



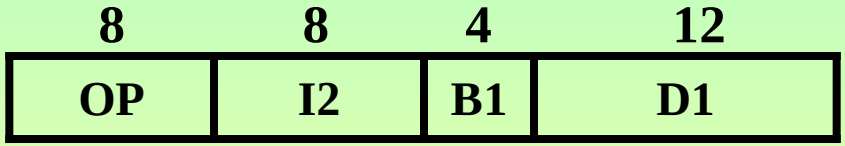
RX型：



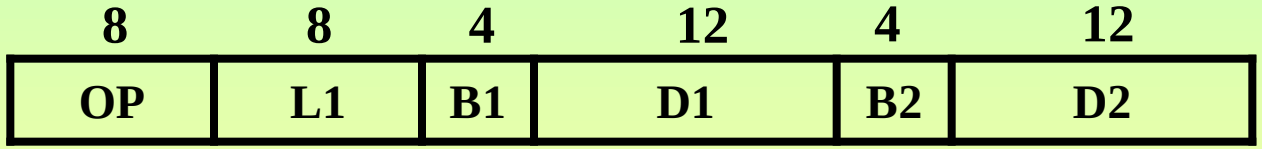
RS型：



SI型：



SS型：

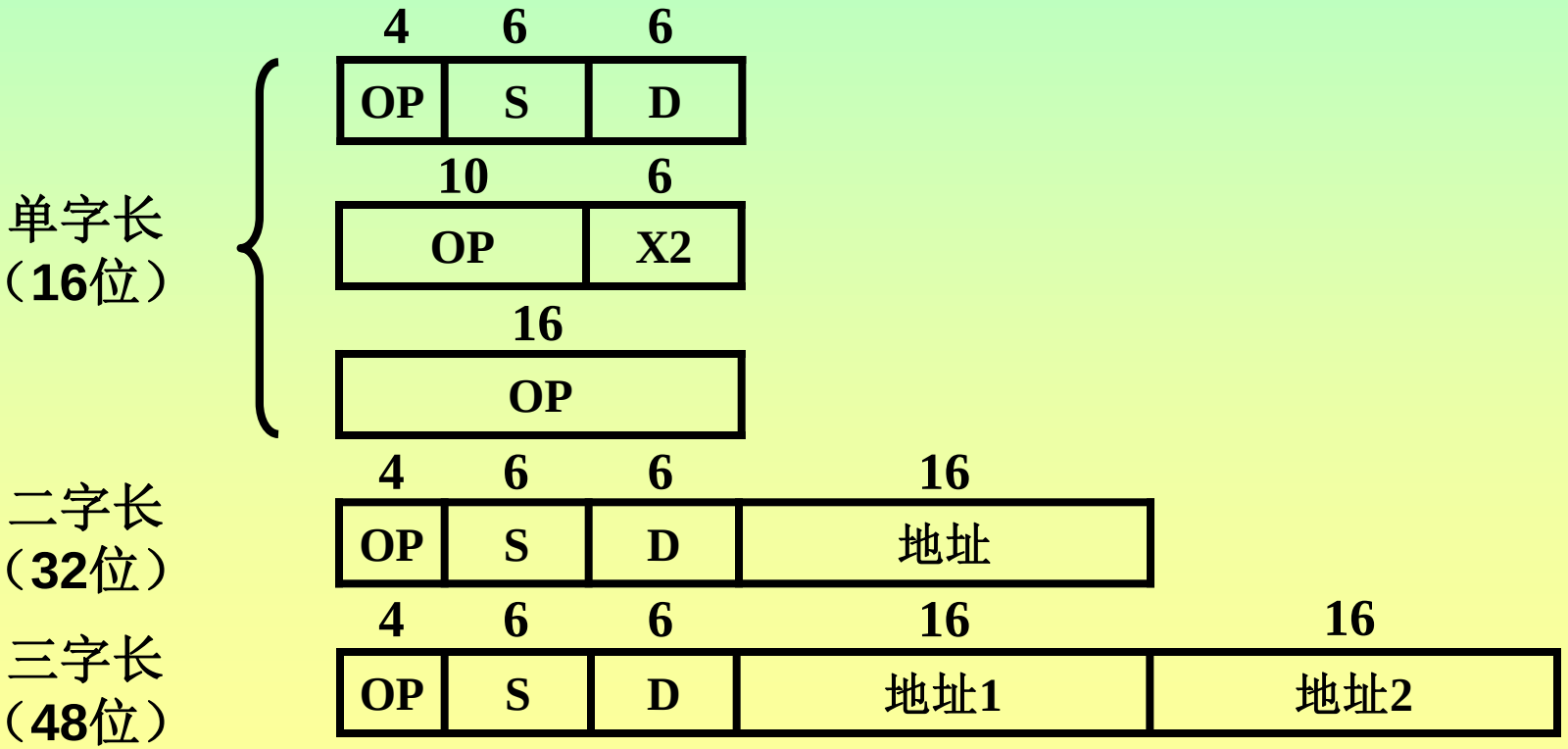


OP：操作码；R：寄存器；X：基址寄存器；L：指定操作数的长度；D：位移量；I：直接操作数；B：变址寄存器。

RR型：寄存器－寄存器型；RX型和RS型：寄存器－存储器型；SI型：直接带操作数；SS型：存储器－存储器型。

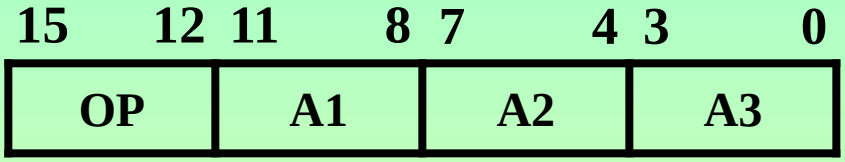
变长编码：在指令字长有限的情况下，变长编码可增加指令种类。当指令中地址部分位数较多时，让操作码的位数少些；地址部分位数较少时，让操作码的位数增多。

PDP－11的字长为16位。指令分为单字长、二字长、三字长三种。操作码占4～16位不等。下面是PDP－11的部分指令格式：



操作码长度不固定会增加指令译码的难度，使控制器设计复杂化。通常用指令中固定长度的字段表示基本操作码，对于不需要某个地址字段的指令，将操作码扩展到该地址字段。

设某机器指令字长16位，包括4位基本操作码和三个4位地址字段：



可按以下方式设置操作码：

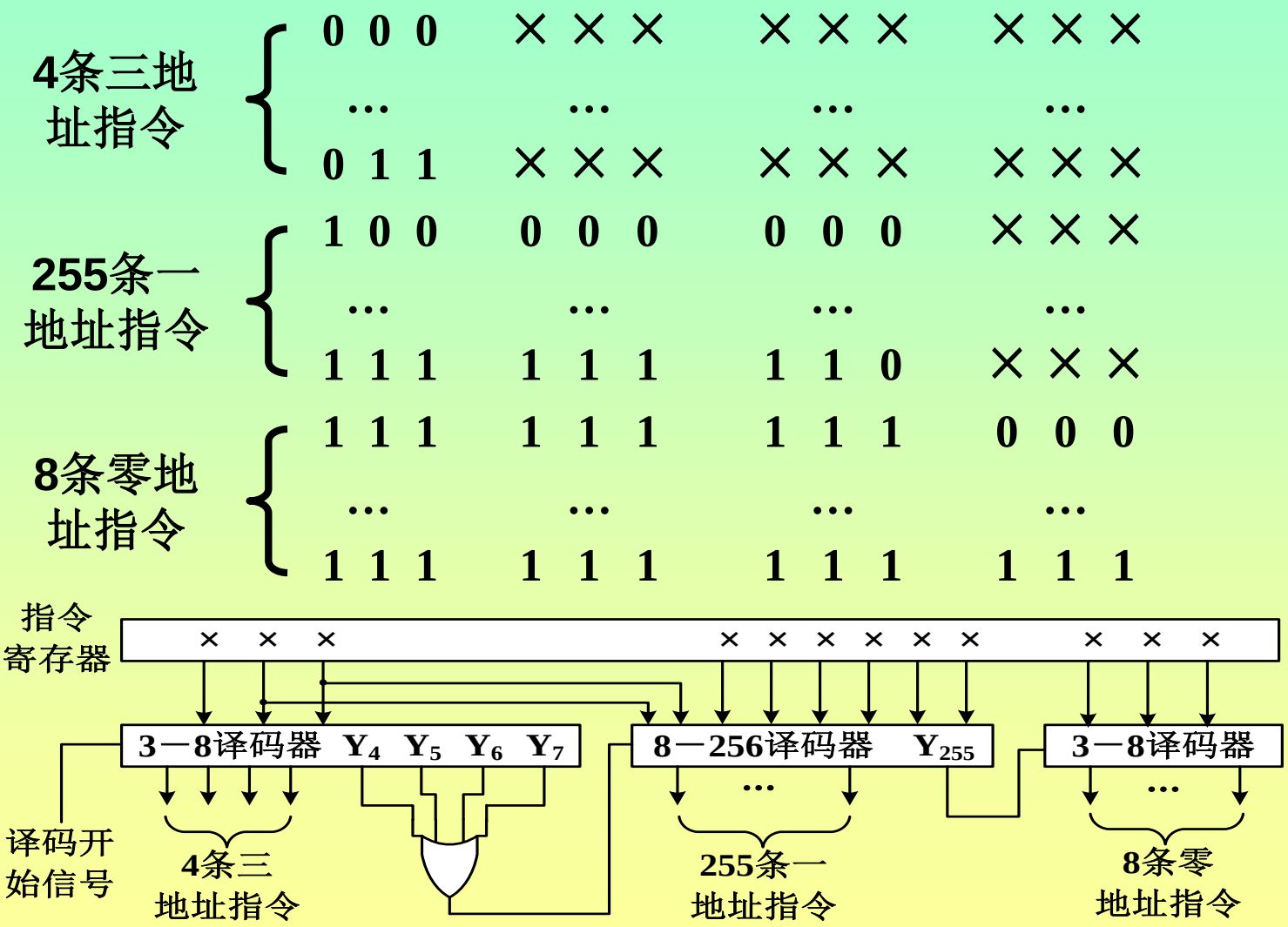
（1）15条三地址指令由基本操作码从0000～1110给出，剩下的1111用于将操作码扩展到A1字段；

（2）15条二地址指令由8位操作码从11110000～11111110给出，剩下的11111111用于将操作码扩展到A2字段；

（3）15条一地址指令由12位操作码从111111110000～111111111110给出，剩下的111111111111用于将操作码扩展到A3字段；

（4）16条零地址指令由16位操作码从1111111111110000～1111111111111111给出。

例：指令字长12位，基本操作码3位，每个地址码3位，设计4条三地址指令、255条一地址指令和8条零地址指令。



某机器指令字长16位，具有双操作数、单操作数和无操作数三类指令，每个操作数地址用6位表示。

1、若采用定长操作码方式设计指令，现已有4条双操作数，8条无操作数指令，此时最多还可以设计多少条单操作数指令？

2、若采用扩展操作码的方式来设计指令，现已有12条双操作数，66条无操作数指令，此时最多还可以设计多少条单操作数指令？

1、操作码为：16－6－6＝4

2^4－4－8＝4

2、

2^4－12＝4

66条无操作数指令占用2条单操作数指令

4×2^6－2 ＝254

操作码的优化

为缩短操作码的平均长度，应该给使用频度高的指令分配短的操作码，使用频度低的指令分配长的操作码。Huffman编码就是按照这一原则进行编码的。假如某计算机有7条指令，使用频度用P_i表示，Huffman编码如下：

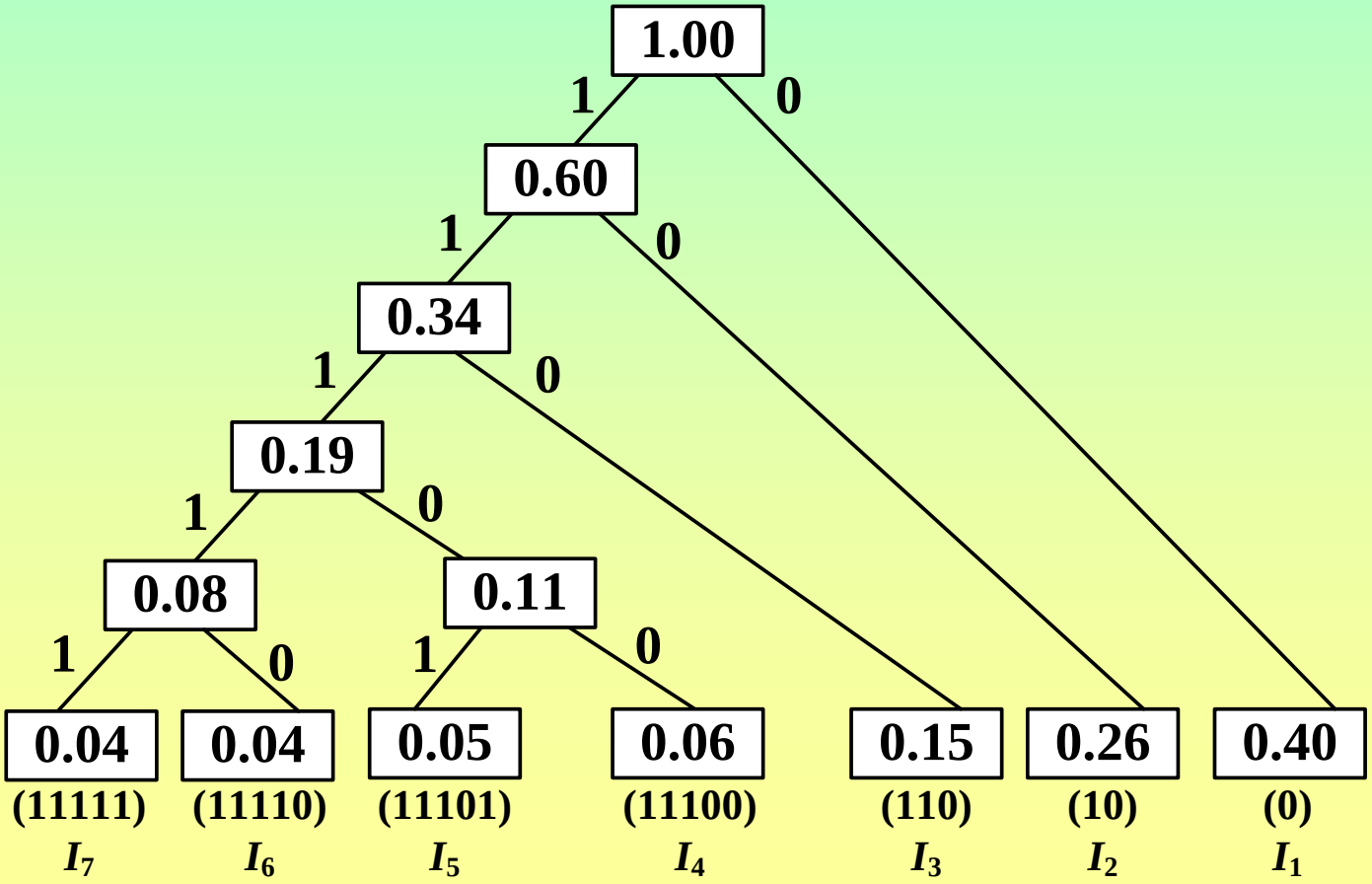
指令I_i	使用频度P_i	Huffman编码	操作码长度L_i
I_1	0.40	0	1
I_2	0.26	1 0	2
I_3	0.15	1 1 0	3
I_4	0.06	1 1 1 0 0	5
I_5	0.05	1 1 1 0 1	5
I_6	0.04	1 1 1 1 0	5
I_7	0.04	1 1 1 1 1	5

操作码的平均编码长度为：

$$L = 0.40\times1+0.26\times2+0.15\times3+0.19\times5 = 2.32\text{位}$$

Huffman编码

将7条指令按使用频度从小到大的顺序排序，每次将使用频度最小的两个结点合并为一个新的结点并重新排序，如此反复。最后将每个结点的两个分支分别用“1”和“0”标注。



Huffman编码可以使操作码平均长度最小，但形成的操作码不规整，不利于译码。一种实际可行的优化编码是**Huffman**扩展编码，它是介于定长编码和**Huffman**编码之间的编码方式。

指令I_i	使用频度P_i	Huffman编码	Huffman扩展编码	定长编码
I_1	0.40	0	0 0	0 0 0
I_2	0.26	1 0	0 1	0 0 1
I_3	0.15	1 1 0	1 0	0 1 0
I_4	0.06	1 1 1 0 0	1 1 0 0	0 1 1
I_5	0.05	1 1 1 0 1	1 1 0 1	1 0 0
I_6	0.04	1 1 1 1 0	1 1 1 0	1 0 1
I_7	0.04	1 1 1 1 1	1 1 1 1	1 1 0
$\Sigma L_i \times P_i$		2.32	2.38	3

Pentium指令格式

Pentium机指令字长从1字节到12字节，还可以带有前缀。

指令前缀：包括LOCK（锁定）前缀和重复前缀。LOCK前缀用于多CPU环境中对共享存储器的排他性访问。重复前缀用于字符串的重复操作。

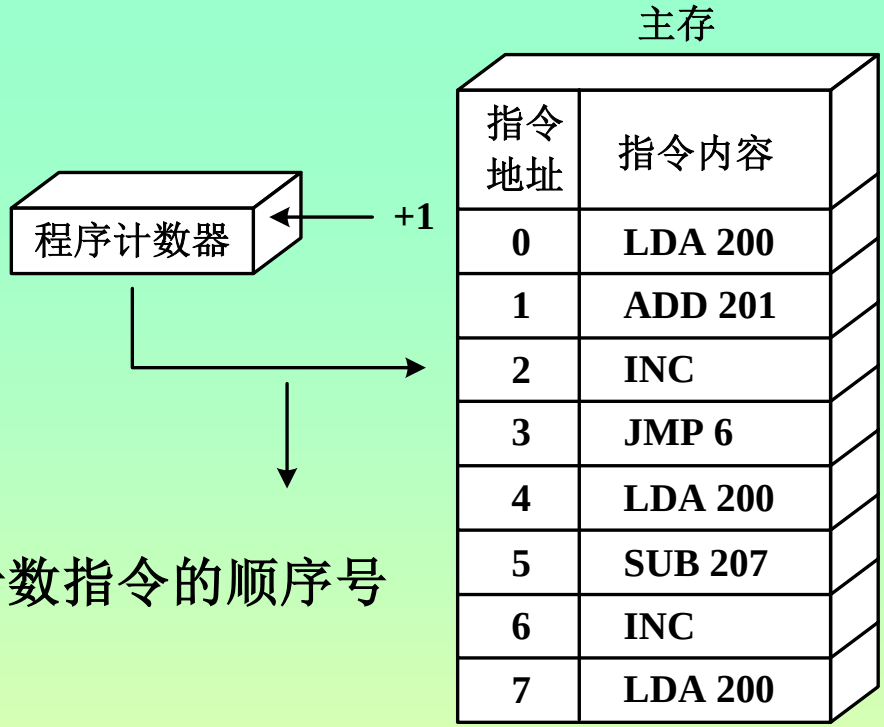
段取代前缀：一条指令不按缺省规则使用某个段寄存器时，必须以段取代前缀明确指明此段寄存器。

操作数长度取代前缀和地址长度取代前缀：一条指令不采用默认的操作数或地址长度时，用这两类前缀予以显式说明。

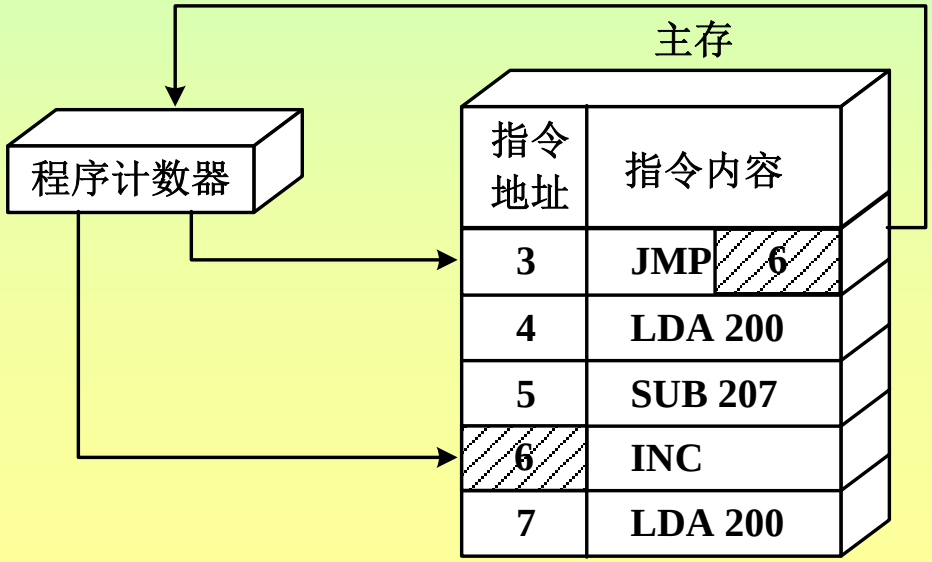
指令本身由操作码字段、Mod-R/M字段、SIB字段、位移量字段、立即数字段组成。除操作码字段外，其他四个字段都是可选字段。

指令寻址方式

顺序寻址方式：指令地址在主存中按顺序安排，当执行一段程序时，通常是一条指令接一条指令的顺序执行。



为此，必须使用程序计数器PC来计数指令的顺序号，该顺序号就是指令在主存中的地址。

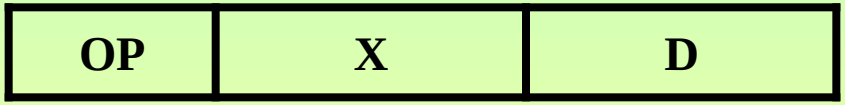


跳跃寻址方式：下条指令的地址码不是由程序计数器给出，而是由本条指令给出。程序跳跃后，按新的指令地址开始顺序执行。程序计数器的内容也必须相应改变。采用指令跳跃寻址方式，可以实现程序转移或构成循环程序。

操作数寻址方式

操作数在主存中的存放并无一定规律，给操作数寻址带来困难。此外，受指令长度的限制，指令中的操作码不会很长，而主存容量却越来越大，因此仅能直接访问主存的一小部分，无法直接访问整个主存。

把指令中地址码字段给出的地址称为形式地址，这个地址可能不能直接访问主存。能够直接访问主存的地址称为有效地址。寻址就是把操作数的形式地址变换为有效地址。



上图是一种单地址指令。OP为操作码，X为寻址特征码，D为形式地址也称偏移量，有效地址E由X和D的组合给出。

隐含寻址：在指令中不明显地给出而是隐含着操作数的地址。例如，单地址的指令格式，没有在地址字段中指明第二操作数地址，而是规定累加寄存器AC作为第二操作数地址，AC对单地址指令格式来说是隐含地址。

立即寻址：指令中地址字段给出的不是操作数的地址，而是操作数本身。

OP (移位)	F	D

上图为单地址移位指令。D不表示地址，而是一个操作数。F为标志位，F＝1操作数右移，F＝0操作数左移。

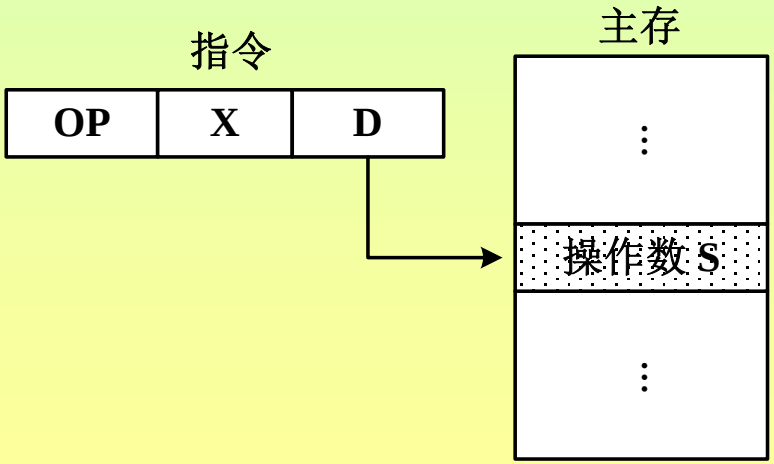
取指时同时取出操作码和操作数，不需要访问存储器，提高了指令执行的速度。但操作数是指令的一部分，不能修改，而且立即数的大小受指令长度的限制，灵活性差。

直接寻址：指令中地址字段直接给出操作数在主存中的地址。

指令中的形式地址D就是有效地址E，即E＝D。如果操作数用S表示，则

S＝（E）＝（D）

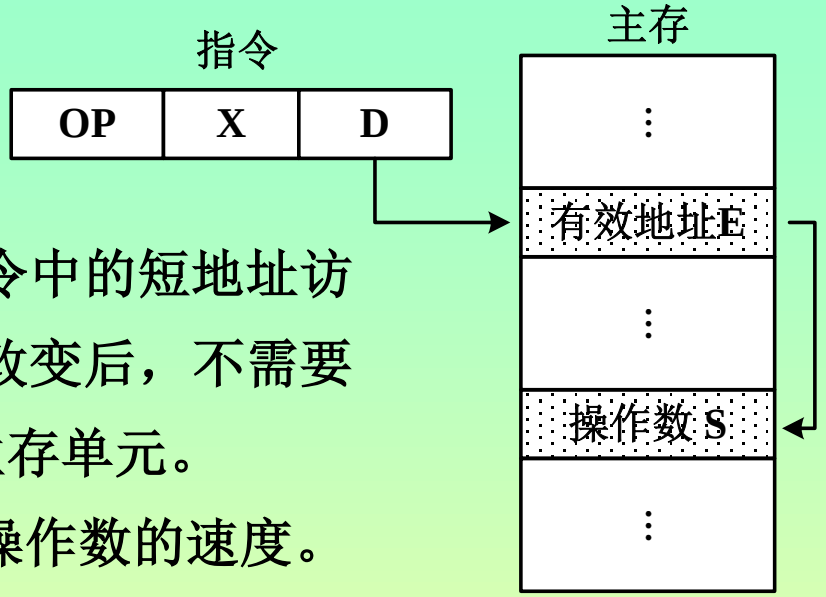
随着主存容量不断扩大，所需要的地址码越来越长，而指令中地址码字段长度有限，限制了访问主存的范围。



间接寻址：形式地址不是操作数的有效地址，而是操作数地址的地址。

E＝（D）

S＝（E）＝（（D））

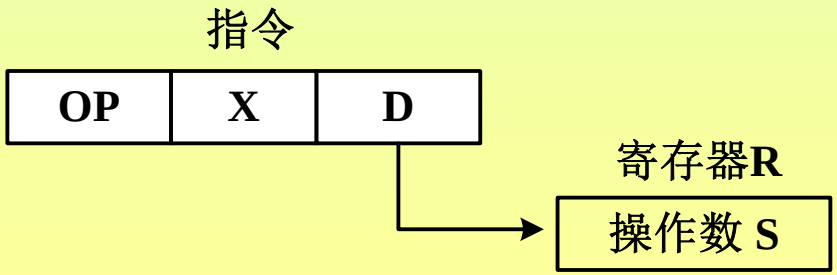


间接寻址扩大了寻址范围，可用指令中的短地址访问大的主存空间。此外，操作数的地址改变后，不需要修改指令，只要修改存放有效地址E的主存单元。

间接寻址需要两次访存，降低了取操作数的速度。

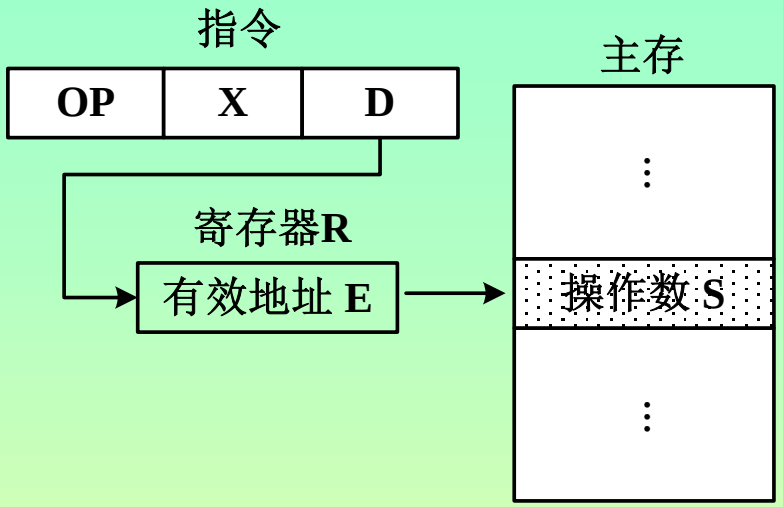
寄存器寻址：指令中的地址码部分给出一个通用寄存器编号，该寄存器中存放着操作数。

从寄存器取数据比从主存取数据快得多。由于寄存器数量较少，寄存器编号的长度比主存地址的长度短得多，可以缩短指令长度。



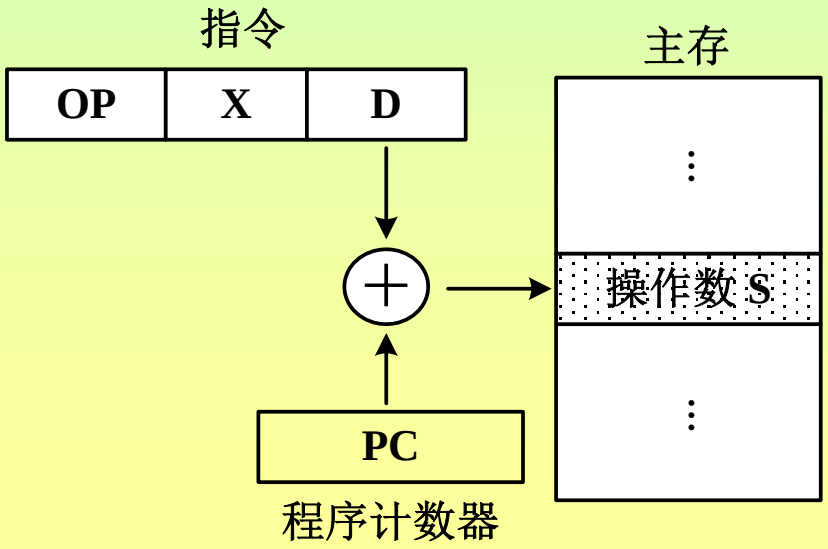
寄存器间接寻址：寄存器中的内容是操作数的有效地址，该地址在主存中。

寄存器间接寻址的指令较短，而能够访问的主存范围很大，并且只需访问主存一次，执行速度比间接寻址快，是广泛采用的寻址方式。寄存器在编程时常用作地址指针。



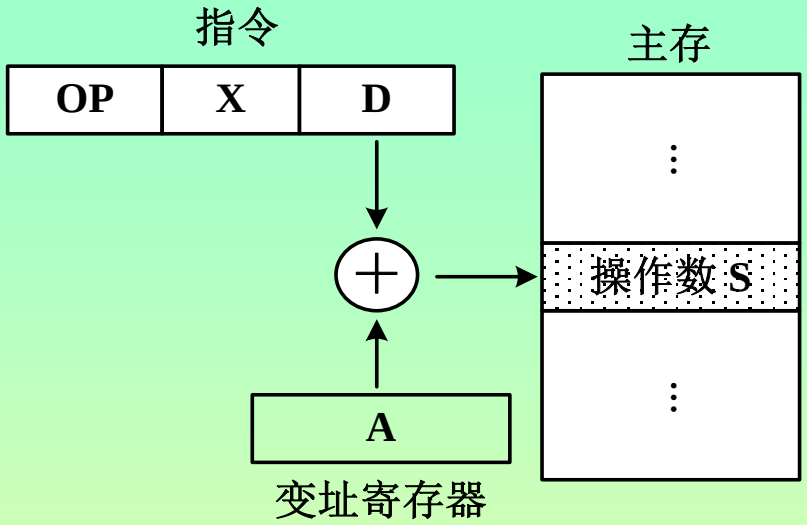
相对寻址：把程序计数器PC中的内容加上指令中的形式地址D，得到操作数的有效地址。

所谓“相对”，是相对于当前指令的地址。形式地址D称为偏移量，可正可负。操作数地址和指令地址相差一个固定值。操作数可以和指令一起移动，放在主存中任何地方，执行的效果都相同。



变址寻址：把CPU中变址寄存器的内容加上指令中的形式地址D，得到操作数的有效地址。

通常把指令中的形式地址作为基准地址，把变址寄存器的内容作为修改量。需要频繁修改地址时不必修改指令，只需修改变址寄存器。这对于数组运算、字符串操作等成批数据处理是很有用的。



基址寻址：把CPU中基址寄存器的内容加上指令中的形式地址D，得到操作数的有效地址。

基址寻址和变址寻址形成操作数有效地址的算法相同，但应用有区别。变址寻址中变址寄存器提供修改量，指令提供基准值；基址寻址中基址寄存器提供基准值，指令提供修改量。变址寻址面向用户，主要用于数组操作；基址寻址面向系统，主要用于逻辑地址和物理地址的转换，解决程序在主存中的再定位和扩大寻址空间的问题。

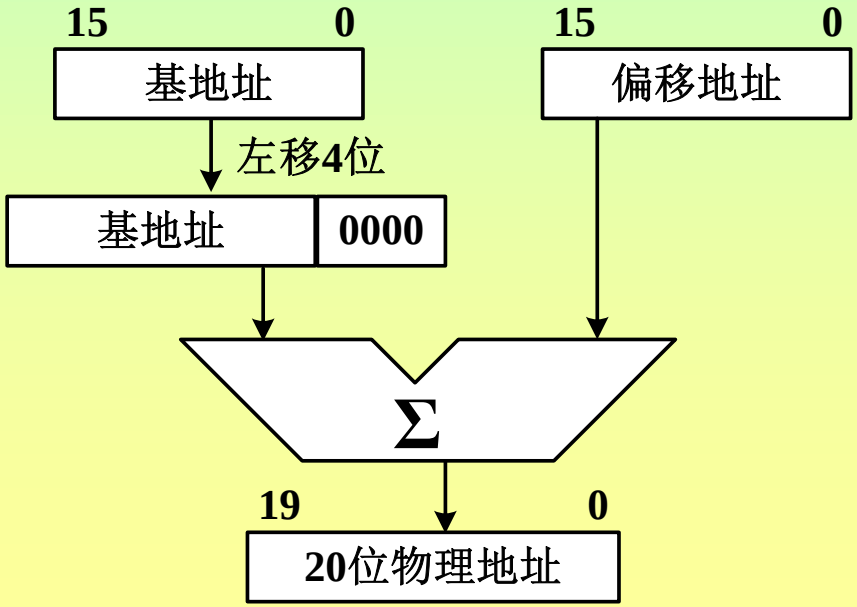
块寻址：块寻址方式可用来实现外存储器或外围设备同主存之间的数据块传送。块寻址方式在主存中还可用于数据块搬家。块寻址时，通常在指令中指出数据块的起始地址和数据块的长度。

如果数据块是变长的，可用三种方法指出它的长度：(1) 指令中划出字段指出长度；(2) 指令格式中指出数据块的首地址与末地址；(3) 由块结束字符指出数据块长度。

段寻址：为了扩大寻址范围而采用的方法，在Intel 8088/8086中采用。

8088/8086地址寄存器为16位。将段寄存器中的16位基地址左移4位，再和某个寄存器提供的16位偏移量相加，得到20位物理地址，可寻址1 M的地址空间。

这种寻址方式的实质还是基址寻址。



PDP/11系列机寻址方式

PDP/11寻址方式的特点是采用寄存器进行寻址。CPU中有8个寄存器，R_7作为程序计数器PC，R_6作为堆栈指示器SP，R_0－R_5作为通用寄存器。指令中有3位对应寄存器编号，3位作为寻址特征位，共8种寻址方式。此外还有4种程序计数器型寻址方式。

	寻址名称	寻址特征位	有效地址 E	汇编格式	说　明
直接型	寄存器型	000	E = R	R_n	寄存器R_n的内容是操作数，相当于直接地址
	自增型	010	E = (R); (R) + 2→R	$(R_n)+$	寄存器R_n的内容是操作数地址，然后递增寄存器内容
间接型	寄存器间接	001	E = (R)	@R_n或 (R_n)	寄存器R_n的内容作为操作数地址，相当于间接地址
	自增间接	011	E = ((R)); (R) + 2→R	@$(R_n)+$	寄存器R_n的内容作为操作数地址的地址，然后递增寄存器内容
程序计数器型	立即型	010	(PC) + 2→PC; E = (PC)	#n	指令下一个单元内容是操作数
	绝对值	011	(PC) + 2→PC; E = ((PC))	@#A	指令下一个单元内容是操作数的地址

Pentium寻址方式

在实地址模式下，采用段寻址方式：将段寄存器内容（16位）左移4位，低4位补0，得到20位段基地址，再加上16位段内偏移，得20位物理地址。

在保护模式下，32位段基地址加上段内偏移得到32位线性地址。由存储管理部件将其转换成32位的物理地址。

序号	寻址方式名称	有效地址E算法	说　明
(1)	立即		操作数在指令中
(2)	寄存器		操作数在寄存器中
(3)	直接	E = Disp	Disp为偏移量
(4)	基址	E = (B)	B为基址寄存器
(5)	基址＋偏移量	E = (B) + Disp	
(6)	比例变址＋偏移量	E = (I)*S + Disp	S为比例因子
(7)	基址＋变址＋偏移	E = (B) + (I) + Disp	
(8)	基址＋比例变址＋偏移量	E = (B) + (I)*S + Disp	
(9)	相对	指令地址 = (PC) + Disp	

几点说明：

⑴ 立即数可以是8位，16位，32位。

⑵ 寄存器寻址：通用寄存器可以为8位、16位或32位。对64位浮点数，要使用一对32位寄存器。少数指令以段寄存器来实施寄存器寻址方式。

⑶ 直接寻址：偏移量长度可以是8位，16位，32位。

⑷ 基址寻址：基址寄存器B可以是上述通用寄存器中任何一个。

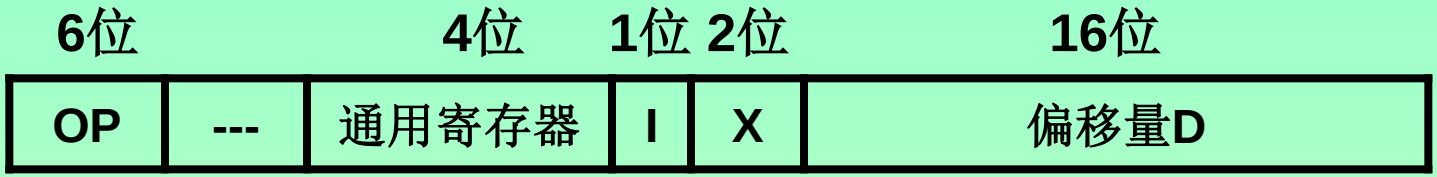
⑸ 基址＋偏移量寻址：基址寄存器B是32位通用寄存器中任何一个。

⑹ 比例变址＋偏移量寻址：变址寄存器I是32位通用寄存器中除ESP外的任何一个。I的内容乘以比例因子1，2，4或8，再加上偏移量得到有效地址。

⑺，⑻ 两种寻址方式是⑷，⑹两种寻址方式的组合，偏移量可有可无。

⑼ 相对寻址：适用于转移控制类指令。用当前指令指针寄存器EIP或IP的内容（下一条指令地址）加上一个有符号偏移量，形成CS段的段内偏移。

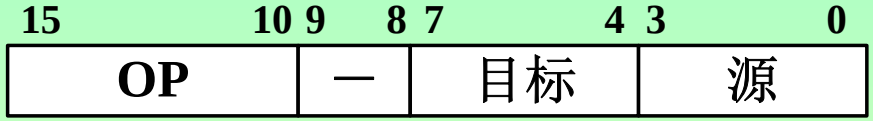
例：一种二地址RS型指令的结构如下所示：



其中I为间接寻址标志位，X为寻址模式字段，D位偏移量字段。通过I，X，D的组合，可构成下表所示的寻址方式。 请写出六种寻址方式的名称。

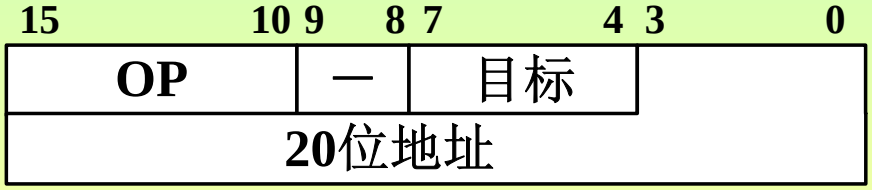
寻址方式	I	X	有效地址E算法	说明
(1)	0	00	E = D	
(2)	0	01	E = (PC) + / - D	PC为程序计数器
(3)	0	10	E = (R2) + / - D	R2为变址寄存器
(4)	1	11	E = (R3)	
(5)	1	00	E = (D)	
(6)	0	11	E = (R1) + / - D	R1为基址寄存器

例：某16位机器有两个20位基址寄存器，四个16位变址寄存器，十六个16位通用寄存器。指令汇编格式中的S，D都是通用寄存器，M是主存中的一个单元。三种指令的操作码分别是MOV (OP)＝$(A)_H$，STA (OP)＝$(1B)_H$，LDA (OP)＝$(3C)_H$。MOV是传送指令，STA为写数指令，LDA为读数指令。



MOV指令

STA指令



LDA指令

⑴ 分析三种指令的指令格式与寻址方式特点。

⑵ 分析三种指令的执行时间。

⑶ 下列指令字是否正确？分别代表什么操作？

① $(F0F1)_H$ $(3CD2)_H$　② $(2856)_H$　③ $(6FD6)_H$　④ $(1C2)_H$

三种指令的操作码分别是MOV (OP)＝$(A)_H$，STA (OP)＝$(1B)_H$，LDA (OP)＝$(3C)_H$。MOV是传送指令，STA为写数指令，LDA为读数指令。

⑶ 下列指令字是否正确？分别代表什么操作？

① $(F0F1)_H$ $(3CD2)_H$　② $(2856)_H$　③ $(6FD6)_H$　④ $(1C2)_H$

MOV (OP)＝$(A)_H$ = 001010，STA (OP)＝$(1B)_H$ = 011011，

LDA (OP)＝$(3C)_H$ =111100

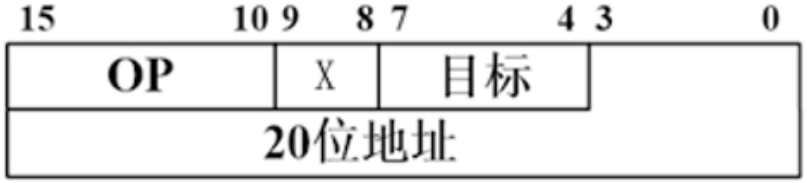
① $(F0F1)_H$ $(3CD2)_H$　双字长，1111 0000 ……　LAD

② (2856)H　单字长　0010 1000 ……　MOV

③ (6FD6)H 单字长　0110 1111 ……　错

④ (1C2)H　单字长　0000　0001……　错

四、某机器指令格式结构如下（其中 X 表示源操作数寻址特征位），问（7 分）：



1）最多可设计多少条不同的指令？

2）最多可设计几种源操作数寻址方式？

3）若某指令的操作码 OP=$(2E)_H$，三条机器指令分别为$(2E5A)_H$，$(2E30)_H(7906)_H$，$(B8F1)_H(3CD2)_H$，问这三条机器指令是否为该种指令？简要说明原因

1、OP 字段 6 位，可以设计 2^6=64 种操作　　（2 分）

2、寻址特征位 2 位，可以设计 4 种寻址方式　（2 分）

3、(2E5A)H 不是，字长不一致　　1 分

(2E30)H(7906)H 不是　OP 字段不同　其中 OP = 101110

指令 OP=00101110　1 分

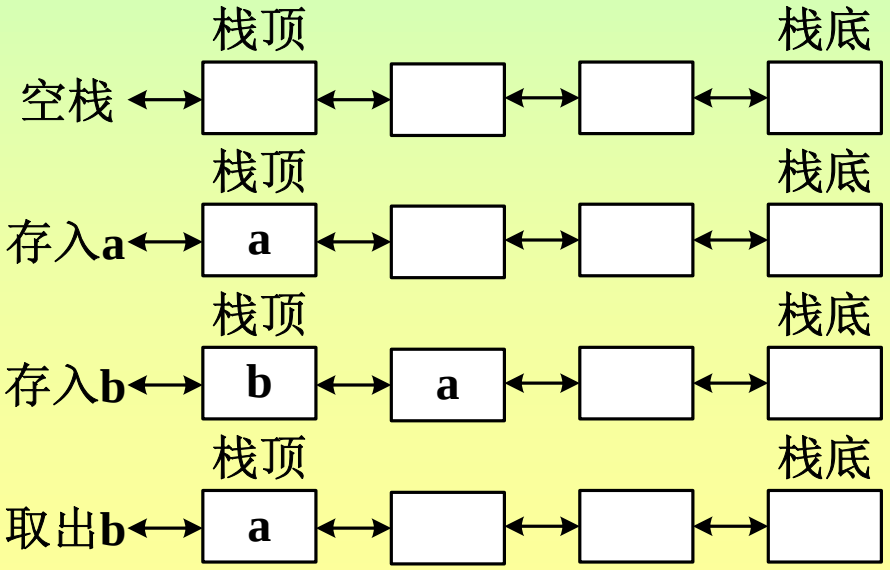
(B8F1)H(3CD2)H 是　　OP 字段相同　OP = 101110　　1 分

串联堆栈

堆栈是指一个特定的存储区，数据的存入和取出按照后进先出（LIFO）的原则进行。

某些计算机的CPU中有一组专门的寄存器，每个寄存器可以保存一个字的数据，相邻的寄存器具有位对位的移位功能。这样的结构称为串联堆栈，也称硬堆栈。CPU通过进栈指令将数据存入堆栈，通过出栈指令把数据从堆栈中取出。因为串联堆栈在CPU内，它们是极好的暂存单元。

堆栈通过栈顶与通用寄存器进行数据交换。新数据进栈时总是保存在栈顶，将堆栈内原来的数据依次向栈底方向移动。出栈时将栈顶数据弹出，其它数据依次向栈顶移动。

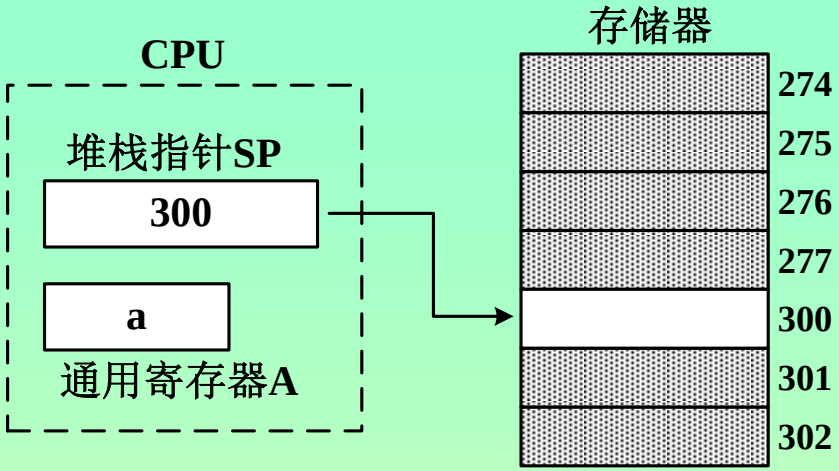


存储器堆栈

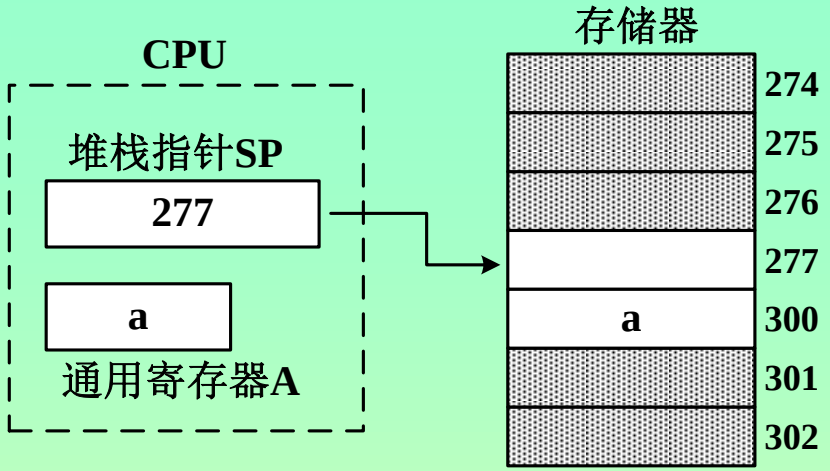
串联堆栈中寄存器的数目是有限的，而且堆栈的读出是破坏性的。通常由程序员从主存中划出一部分区域作为堆栈，称为存储器堆栈，也称软堆栈。这种堆栈的优点是：可以具有程序员要求的任意长度；可以建立多个堆栈；可以用对存储器寻址的任何一条指令对堆栈中的数据进行寻址。

存储器堆栈需要一个堆栈指针（SP)，它是CPU中一个专用的寄存器，堆栈指针指定的存储单元，就是堆栈的栈顶。

主存各单元之间没有移位功能，因此不采用数据相对于堆栈移动的方法，而是采用堆栈顶部相对于数据进行移动的方法，堆栈指针SP随着修改。

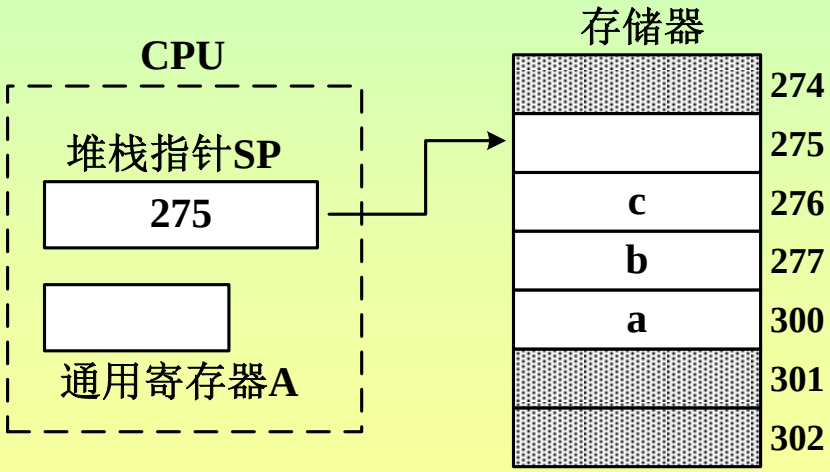


进栈前

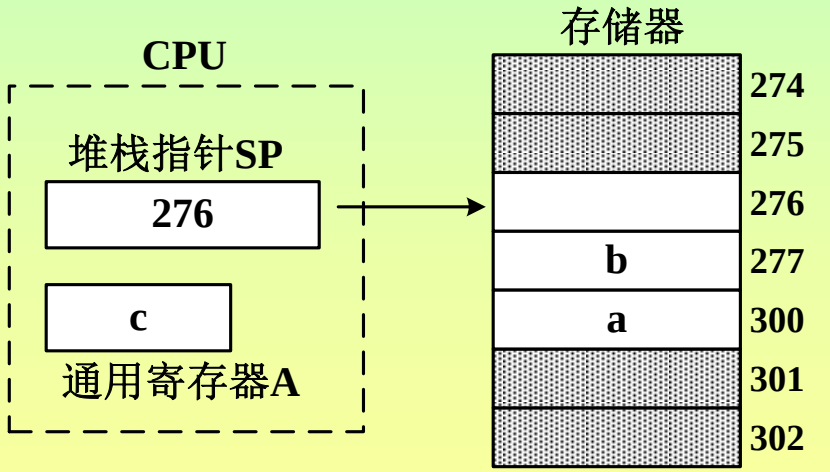


进栈后

进栈操作：(A) → (SP)，(SP)－1 → SP



出栈前



出栈后

进栈操作： (SP)＋1 → SP，((SP)) → A

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。

在堆栈计算机中，堆栈是保存操作数和运算结果的唯一场所。此时的算术逻辑类指令中没有地址码字段。参加运算的两个操作数隐含地从堆栈顶部弹出，送到运算器中进行运算，运算的结果再隐含地压入堆栈。例如：算术表达式a×b＋c÷d，运算结果送x，可以用逆波兰法表示成为ab×cd÷＋。

```
PUSH A          ；数据a压入堆栈
PUSH B          ；数据b压入堆栈
MUL             ；完成a×b
PUSH C          ；数据c压入堆栈
PUSH D          ；数据d压入堆栈
DIV             ；完成c÷d
ADD             ；完成a×b+c÷d
POP X           ；结果存入X单元
```

指令的分类

数据传送指令：主要包括取数指令、存数指令、传送指令、成组传送指令、字节交换指令、清累加器指令、堆栈操作指令等等。这类指令主要用来实现主存和寄存器之间，或寄存器和寄存器之间的数据传送。

算术运算指令：包括二进制定点加、减、乘、除指令，浮点加、减、乘、除指令，求反、求补指令，算术移位指令，算术比较指令，十进制加、减运算指令等。这类指令主要用于定点或浮点的算术运算。

逻辑运算指令：包括逻辑加、逻辑乘、按位加、逻辑移位等指令，主要用于无符号数的位操作、代码的转换、判断及运算。 移位指令用来对寄存器的内容实现左移、右移或循环移位。

程序控制指令：条件转移指令、无条件转移指令、转子程序指令、返回主程序指令、中断返回指令等。

输入输出指令：输入输出指令主要用来启动外围设备，检查测试外围设备的工作状态，并实现外部设备和CPU之间，或外围设备与外围设备之间的信息传送。

字符串处理指令：字符串处理指令是一种非数值处理指令，一般包括字符串传送、字符串转换（把一种编码的字符串转换成另一种编码的字符串）、字符串替换（把某一字符串用另一字符串替换）等。

特权指令：特权指令是指具有特殊权限的指令。这类指令只用于操作系统或其他系统软件，一般不直接提供给用户使用。 在多用户、多任务的计算机系统中特权指令必不可少。它主要用于系统资源的分配和管理。

其他指令：除以上各类指令外，还有状态寄存器置位、复位指令、测试指令、暂停指令，空操作指令，以及其他一些系统控制用的特殊指令。

CISC

计算机硬件的成本不断下降，而软件的成本不断提高，使得人们热衷于在指令系统中增加更多更复杂的指令。另外，为了做到程序兼容，同一系列计算机的指令集只能扩充而不能缩减。由此，形成了CISC。其特点是：

指令系统庞大，指令数目一般有二三百条；寻址方式多样；指令格式多样，指令字长不固定；可访存指令不受限制；各种指令使用频率相差很大；各种指令执行时间相差很大。

CISC的出发点有以下几点：

使目标程序得到优化。如设置数组运算指令，把原来用一段程序才能完成的功能，只用一条指令实现。

给高级语言更好的支持。设置一些在语义上接近高级语言的指令，可以减轻编译的负担，提高编译效率。

提供对操作系统的支持。操作系统功能越来越复杂，要求指令系统提供更复杂的功能。

CISC指令系统功能很强，但不一定能提高机器的速度，原因是：

CISC采用复杂的寻址方式，计算有效地址需要花费一定时间。有的指令需要多次访问存储器，执行速度会降低。

复杂的指令系统需要复杂的控制器。控制部件多采用微程序控制方式实现，微程序控制部件执行一条机器指令需要几个微周期，降低了指令执行速度。

CISC中常采用流水线技术，但由于指令字长和执行时间不同、不同指令争用共同资源以及转移指令等，流水线效率不高。

所以，传统的CISC设计思想不利于提高计算机的速度。而且由于硬件的复杂性，使计算机的研制周期长，难以保证正确性。因此，人们开始研究指令系统的合理性问题。统计表明，最常用的是一些比较简单的指令，仅占指令总数的20%，但在程序中出现的频率却高达80%。

RISC

RISC指令系统的特点是：

指令数目较少，一般都选用使用频率较高的简单指令；

指令长度固定，指令格式种类少，寻址方式种类少；

大多数指令可以在一个机器周期内完成；

通用寄存器数量多。只有取数/存数指令(LOAD/STORE)访问存储器，其余指令都在寄存器之间进行操作。

影响较大的RISC芯片有：IBM、Apple和Motorola联合开发的PowerPC，DEC的Alpha，SUN的SPARC，MIPS的R10000，Intel的i80860等。

CISC和RISC孰优孰劣，业界有过长期的争论。目前两者越来越接近。由于芯片密度和速度不断提高，使RISC日趋复杂。而CISC也开始采用部分RISC技术，如强调指令流水线、分级Cache和多设通用寄存器。现在的CISC处理器基本上都经过RISC改良，以RISC为内核，外围是CISC界面，如Intel Pentium系列。

RISC举例

SPARC机是一个32位字长的计算机，共有75条指令，分为6类：

⑴ 算术运算 / 逻辑运算 / 移位指令

⑵ 取数（LOAD） / 存数（STORE）指令

⑶ 控制转移指令

⑷ 读 / 写专用寄存器指令

⑸ 浮点运算指令

⑹ 协处理器指令

某些指令的替代实现（SPARC约定R0的内容恒为0）：

指令	功能	替换指令	实现方法
MOVE	寄存器间传送数据	ADD	Rs＋R0→Rd
INC	寄存器内容加 1	ADD	立即数imm13＝1，作为操作数
DEC	寄存器内容减 1	SUB	立即数imm13＝1，作为操作数
NEG	取负数	SUB	R0－Rs→Rd
NOT	取反码	XOR	立即数imm13＝－1，作为操作数
CLR	清除寄存器	ADD	R0＋R0→Rd

RISC、CISC区别

RISC、CISC各自的特点？

它们之间的区别？