计算机组成原理

Principle of Computer Organization

第四章 指令系统（指令集）

Instruction Set

北京邮电大学
计算机学院

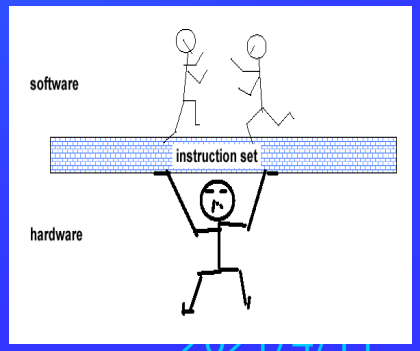
戴志涛

本章内容

- 指令系统的概念及其发展
- 指令格式
- 寻址方式
- 指令类型和典型指令介绍
- RISC

指令系统的概念

- 程序是由一系列有序且有一定意义的指令组成的
- 指令（机器指令）：命令计算机直接进行某种基本操作的二进制代码串
  - 每条指令可以完成一个独立的算术运算或逻辑运算操作，或者数据传送等基本操作
  - 直接由硬件支持、软件可控制的最小的动作单位
- 程序员按照指令格式编写软件而不必考虑机器如何实现指令的功能
- 指令系统（Instruction Set）：一台计算机能直接理解与执行的全部指令的集合
- 指令是软件和硬件之间的接口



指令概念的引申

- 机器指令（指令）：每条指令完成一个独立的算术运算或逻辑运算
- 微指令：微程序级的命令，软件不可见
- 宏指令：由若干条机器指令组成的机器指令序列，硬件不可见

计算机指令系统的发展过程

- 50年代：
  - 最基本的指令：定点加减、逻辑运算、数据传送、转移等
  - 指令数目十几至几十条
- 60年代后期：
  - 增加乘除运算、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
  - 寻址方式多样化
  - 出现系列计算机

计算机指令系统的发展过程

- 70年代末期：
  - 指令系统多达几百条
    - 复杂指令系统计算机（CISC）
    - Complex Instruction Set Computer
  - 庞大的指令系统难以保证正确性，不易调试维护，造成硬件资源浪费
    - 精简指令系统计算机（RISC）出现
    - Reduced Instruction Set Computer

低级语言与硬件结构的关系

- 高级语言（算法语言）：语法与具体机器的指令系统基本无关
- 低级语言：面向机器，和具体机器的指令系统密切相关
  - 机器语言（二进制语言）
  - 汇编语言（符号语言）

高级语言与低级语言的性能比较

比较内容	高级语言	低级语言
对程序员的训练要求 (1)通用算法 (2)语言规则 (3)对硬件的了解	 有 较少 相对较少	 有 较多 相对较多
对机器独立的程度	独立	不独立
编制程序的难易程度	易	难
软件编程和维护所需时间	短	较长
程序执行时间	较长	短
编译过程中对计算机资源的要求	多	少

高级语言与低级语言

- 低级语言：时空效率高
- 高级语言：编程难度低，可维护性高
- 可移植性
- 硬件透明性
- 混合语言程序设计
- 趋势：高级语言的应用比重加大

指令格式

- 指令字（指令）：表示一条指令的机器字
- 指令格式：指令字用二进制代码表示的结构形式
- 机器执行一条指令所必须的全部信息都必须明显或隐含地在指令中给出：

操作码 ❑ 操作类型

地址码 ❑ 参加运算的若干个源操作数的地址（内存单元或寄存器）

地址码 ❑ 目的操作数（运算结果）存放的地址（内存单元或寄存器）

地址码 ❑ 下一条指令的存放地址（内存单元）

操作码

- 每一条指令都有一个操作码，表示该指令应进行什么性质的操作
- 不同的指令用操作码字段的不同编码表示
- 操作码字段的位数一般取决于计算机指令系统的规模
- 一个特定的机器的指令系统，不同指令字中操作码字段和地址码字段的长度可以相同，也可以不同

操作码

- 等长操作码（固定长度操作码）：
  - 操作码所占的二进制位数固定不变
  - 如果系统中所有指令的操作码都用n位二进制数表示，则系统中的指令条数不会超过2^n条
  - 有利于简化硬件设计，减少指令译码时间
  - 广泛用于字长较长的计算机的指令系统中

操作码

- 可变长度操作码：
  - 不同的指令的操作码长度不同
  - 操作码扩展技术
    - 地址数少的指令使用较长的操作码，
      地址数多的指令使用较短的操作码
    - 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较长的操作码

操作码

- 可变长度操作码：
  - 优点：
    - 有效缩短操作码的平均长度，节省存储空间
    - 缩短常用指令的译码时间，提高程序的运行速度
  - 缺点：
    - 指令译码系统比等长操作码复杂
  - 在字长较短的微型机中应用广泛
- 操作码的长度应与地址码的长度以及整个指令长度综合考虑

地址码

- 指令中的地址码用来指出该指令的源操作数地址（0至2个）、目标操作数（结果）地址，以及下一条指令的地址
- 操作数的存放位置：
  - 主存单元
  - CPU内部数据寄存器
  - I/O设备接口内的寄存器
- 如果一条指令中有n个操作数地址，则将该指令称为n操作数指令或n地址指令

四地址指令

指令格式：

- OP： 操作码

  A_1：第一地址码，存放被操作数（第一源操作数）

  A_2：第二地址码，存放操作数（第二源操作数）

  A_3：第三地址码，存放操作结果

  A_4：第四地址码，存放下条要执行指令的地址

操作：（A_1）OP（A_2）→ A_3

四地址指令

- 指令可直接寻址的地址范围与地址字段的位数有关

OP	A_1	A_2	A_3	A_4

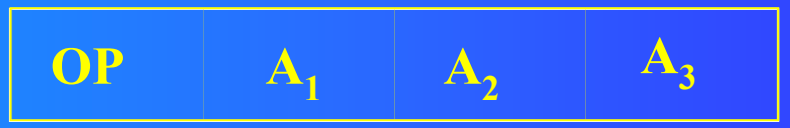
  - 例：指令字长32位，操作码占8位，4个地址段各占6位，则指令的直接寻址范围为：$2^6 = 64$
- 如果地址字段均指示主存的地址，则一条四地址指令执行时共需访存四次
- 优点：指令直观易懂，后续指令的地址可任意安排
- 缺点：指令长度长

地址的个数

- 非转移类指令的下一条指令地址：
  - 本条指令在内存中的存放首地址加上本条指令的长度
- 转移类指令：
  - 通常不需要源操作数地址和结果地址
- 有些情况下，约定某些指令的结果存放在某一个源操作数地址中
- 操作数中的一个或全部源地址可隐含给出：
  - 约定某条指令的操作数（或其中之一）固定来自CPU内的某个特定寄存器或特定的内存单元
- 有些指令无需操作数

三地址指令

指令格式：



- **操作：（A_1） OP （A_2） -> A_3**
  - **A_1：被操作数（第一源操作数）地址**
  - **A_2：操作数（第二源操作数）地址**
  - **A_3：存放结果的地址**

二地址指令

指令格式：

- 双操作数操作：（A_1） OP （A_2） -> A_1
  - A1：既作被操作数（第一源操作数）地址，又作目的操作数地址
  - A2：操作数（第二源操作数）地址
- 单操作数操作：OP （A_1） -> A_2
  - A1：被操作数（源操作数）地址
  - A2：操作数（目的操作数）地址

一地址指令（单操作数指令）

指令格式：

- 双操作数操作：(A_1) OP (X)-> X
  - A1：第一源操作数地址
  - X：第二源操作数地址和目的操作数地址，隐含使用寄存器或堆栈寻址
- 单操作数操作：OP (A_1) -> A_1
  - A1：既作被操作数地址，又作目的操作数地址

零地址指令

指令格式：

- 指令字中只有操作码，没有地址码
  - 无需操作数的指令，如空操作、停机等
  - 运算的操作数地址全部隐含
    - 源操作数和结果隐含使用寄存器、堆栈或特定操作数

地址码个数对指令系统性能的影响

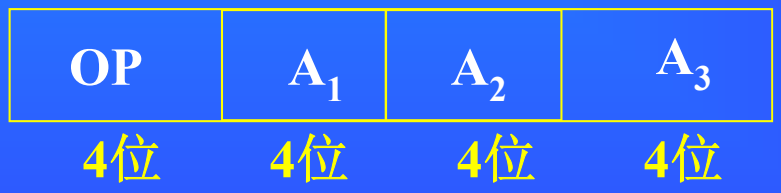
- 早期：多采用二地址指令和一地址指令
- 随着处理机字长和指令字长的增加，三地址指令已很普遍
- 地址多的指令：
  - 功能强，使用灵活
  - 占用的存放单元多，执行时间长
- 隐含地址的指令：
  - 节省了指令地址字段，但牺牲灵活性
- 一个实际的指令系统往往混合使用多种指令格式

双操作数运算指令的三种类型

- 存储器-存储器（SS）型指令：
  - 参与运算的操作数都放在内存中
  - 机器执行SS型指令需多次访问内存
- 寄存器-寄存器（RR）型指令：
  - 参与运算的操作数都放CPU内部的数据寄存器中
  - 机器执行RR型指令速度快，但需要多个寄存器
- 寄存器-存储器（RS）型指令：
  - 执行RS型指令时，既要访存，又要访问寄存器

例：扩展操作码

- 设某机器的指令长度为16位，包括4位基本操作码字段和三个4位地址字段
- 4位基本操作码，若全部用于三地址指令，则只能安排16种指令



扩展操作码

- 向地址码字段扩展操作码的长度：
  - 三地址指令的操作码占用4位基本操作码编码空间的0000～1110共$2^4-1=15$种组合，剩下一个编码1111用于把操作码扩展到A1，即4位扩展到8位

	OP域	A_1域	A_2域	A_3域
三地址指令15条	0000 0001 …… 1110			
二地址指令	1111 1111 …… 1111			

扩展操作码

- 二地址指令的操作码占用8位操作码编码空间的1111,0000～1111,1101 共2^4-2=14种，剩下两个编码1111,1110和1111,1111用于把操作码扩展到A2，即从8位扩展到12位

	OP域	A_1域	A_2域	A_3域
三地址指令15条	0000 0001 …… 1110			
二地址指令14条	1111 0000 1111 0001 …… 1111 1101			

扩展操作码

- 一地址指令的操作码占用12位操作码编码空间的1111,1110,0000~1111,1111,1110 共$2^5-1=31$种编码，剩下一个编码1111,1111,1111用于把操作码扩展到A3，即从12位扩展到16位

	OP域	A_1域	A_2域	A_3域
三地址指令15条	0000 0001 …… 1110			
二地址指令14条	1111 0000 1111 0001 …… 1111 1101			
一地址指令31条	1111 1110 0000 1111 1110 0001 …… 1111 1111 1110			

扩展操作码

- 零地址指令的操作码占用16位操作码编码空间的1111,1111,1111,0000~1111,1111,1111,1111共2^4=16种编码

	OP域	A_1域	A_2域	A_3域
一地址指令31条	1111	1110	0000	
	1111	1110	0001	
		……		
	1111	1111	1110	
零地址指令16条	1111	1111	1111	0000
	1111	1111	1111	0001
		……		
	1111	1111	1111	1111

扩展操作码

向地址码字段扩展操作码的长度：

	OP域	A_1域	A_2域	A_3域
三地址指令15条	0000 0001 …… 1110			
二地址指令14条	1111 0000 1111 0001 …… 1111 1101			
一地址指令31条	1111 1110 0000 1111 1110 0001 …… 1111 1111 1110			
零地址指令16条	1111 1111 1111 0000 1111 1111 1111 0001 …… 1111 1111 1111 1111			

指令字长与机器字长

- 机器字长（简称字长）：指计算机能直接处理的二进制数据的位数，通常是CPU运算器的字长
  - 字长越长，计算机的运算精度越高
  - 为便于处理字符数据及尽可能地利用存储空间，一般把机器字长定为字节长度的整数倍
- 指令字长：一个指令字中包含的二进制代码的位数
  - 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
  - 通常指令字长等于机器字长的整数倍
  - 如果机器字较长，指令字长也可是机器字长的一半的整数倍

指令字长与机器字长的关系

- 根据指令字长等于多少机器字长，可以将机器指令分为：
  - 单字长指令：指令字长等于机器字长
  - 半字长指令：指令字长等于半个机器字长
  - 双字长指令：指令字长等于两个机器字长
- 多字长指令
  - 优点：可以给操作码和地址码提供较大的空间
  - 缺点：占用了较多的内存空间和取指时间

指令字结构

- 等长指令字结构
  - 在同一个指令系统中，所有的指令字长度都是相等的
  - 指令字结构简单，控制方便，实现复杂度低
- 变长指令字结构
  - 在一个指令系统中，各种指令字长度随指令的功能而变化
  - 指令字结构灵活，能充分利用指令字长度，但指令的控制较复杂

指令助记符

- 计算机可以保存和识别的机器语言对程序员而言既繁琐又不易记忆
- 为了便于书写和阅读程序，用更具可读性的指令助记符取代机器指令的操作码部分
- 不同的指令系统中，同样或类似的操作所采用的助记符并不相同

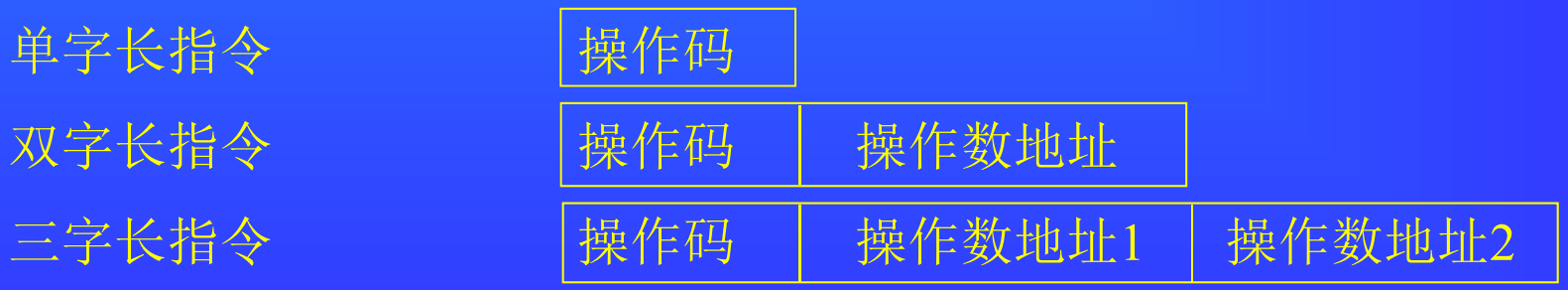
指令	指令助记符	二进制操作码举例
加法	ADD	001
减法	SUB	010
传送	MOV	011
跳转	JMP	100
转子	JSR	101
存储	STR	110
读数	LDA	111

机器语言与汇编语言

- 汇编语言
  - 用指令助记符取代机器指令的操作码部分
  - 用符号表示地址字段
    - 变量：操作数地址
    - 标号：指令地址
  - 汇编语言程序在执行前需人工或借助汇编程序自动转换成机器语言程序执行
- 机器语言与汇编语言的关系
  - 机器语言程序有助于理解计算机原理和概念，但使用不便
  - 对某一特定的机器而言，汇编语言与机器语言可以认为是一一对应的
  - 使用指令助记符简化指令的表述

指令格式举例

- Motorola M6800/Intel 8080八位微机的指令格式
  - 机器字长8位，可变字长指令结构，包含单字长、双字长、三字长指令等多种指令
  - 内存按字节编址
    - 单字长指令每执行一条指令后，指令地址加 1
    - 双字长指令或三字长指令每执行一条指令后，指令地址加2或加3



指令格式举例

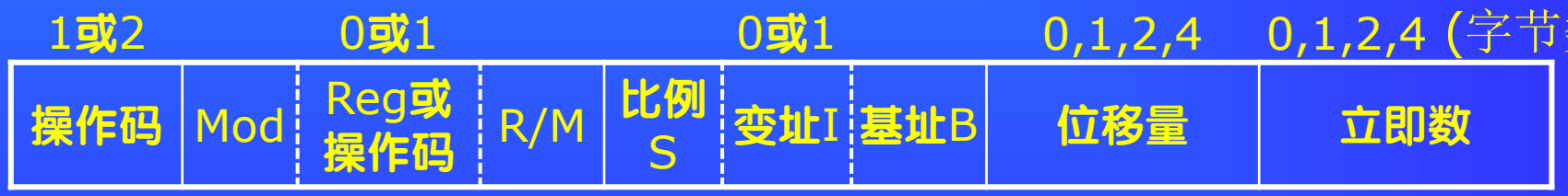
- DEC PDP/11系列小型机指令格式
  - 指令字长16位
  - 操作码字段长度不固定

指令位 / 指令类型	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
单操作数指令	操作码（10位）										目标地址(6位)					
双操作数指令	操作码(4位)				源地址(6位)						目标地址(6位)					
转移指令	操作码（8位）								位移量（8位）							
转子指令	操作码（7位）							寄存器号								
子程序返回指令	操作码（13位）															
条件码操作指令	操作码（11位）											S	N	Z	V	C

指令格式举例

- Pentium指令格式
  - 指令字长可变：从1字节到12字节，还可以带前缀

前缀



指令

课堂练习

设某指令系统字长为16位，地址码为4位。试设计指令格式，使该系统中有11条三地址指令、70条二地址指令和150条单地址指令。并指明该系统中最多还可以有多少条零地址指令。

解：

a)三地址指令

地址码需12位，故操作码字段应有4位

11条三地址指令 OP= 0000~1010

b)二地址指令地址码需8位，操作码字段应有8位

OP= 1011 xxxx

1100 xxxx

1101 xxxx

1110 xxxx 16×4+6=70

1111 0000

1111 0101

课堂练习

11条三地址指令、70条二地址指令和150条单地址指令。并指明该系统中最多还可以有多少条零地址指令。

解：

c)单地址指令地址码需4位，故操作码字段应有12位。

OP = 11110110 xxxx

11111110 xxxx　　16×9 + 6 = 150

11111111 0000

11111111 0101

d)零地址指令无需地址码，故操作码字段有16位。

OP= 11111111 0110 xxxx

11111111 1111 xxxx，共16×10 = 160条。

寻址方式

- 存储器既可存放数据，又可存放指令
- 在存储器中，操作数或指令字写入或读出的方式：
  - 地址指定方式
  - 相联存储方式
  - 堆栈存取方式

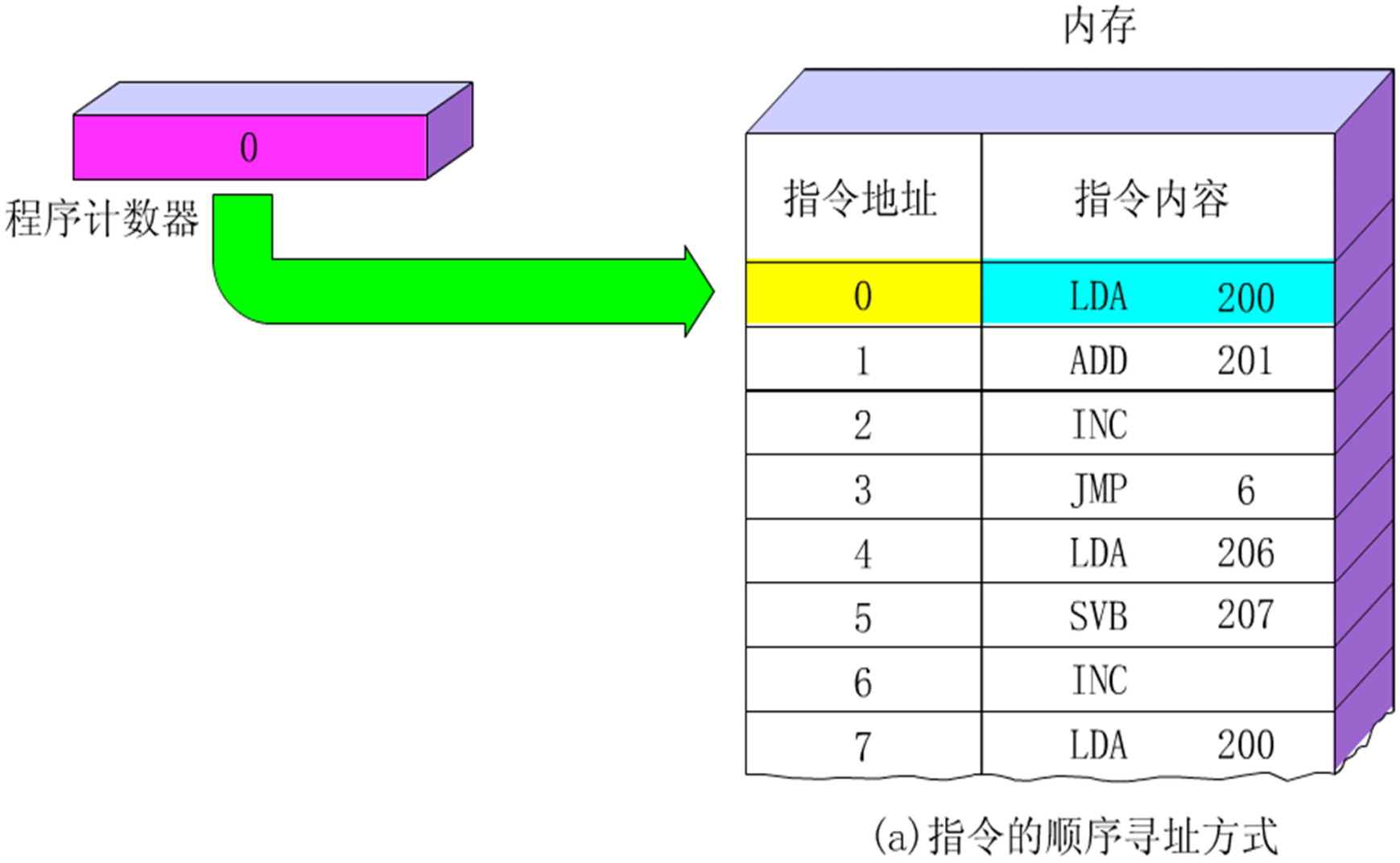
寻址方式

- 有效地址EA（逻辑地址）：实际访问存储单元的地址
- 形式地址：指令中给出的地址码
- 寻址方式：当采用地址指定方式时，形成有效地址的各种方法
- CPU访存的两个目的：
  - 取指令（fetch）
  - 存取操作数（load/store）
- 指令寻址和数据寻址是交替进行的

指令的寻址方式

- 顺序寻址方式
  - 通常情况下，指令在内存中是按顺序存放的：指令的执行顺序就是指令在内存中的存放顺序
  - 程序顺序执行过程：
    - CPU从存储器地址A1取出第一条指令，然后执行这条指令
    - 接着从存储器地址A1后面相邻地址取出第二条指令，并执行第二条指令
    - 接着再取出第三条指令
    - 依此类推……
  - 可以采用程序计数器（指令指针寄存器）PC（Program Counter）自动加一的方式自动形成下一次的取指地址

指令的顺序寻址方式

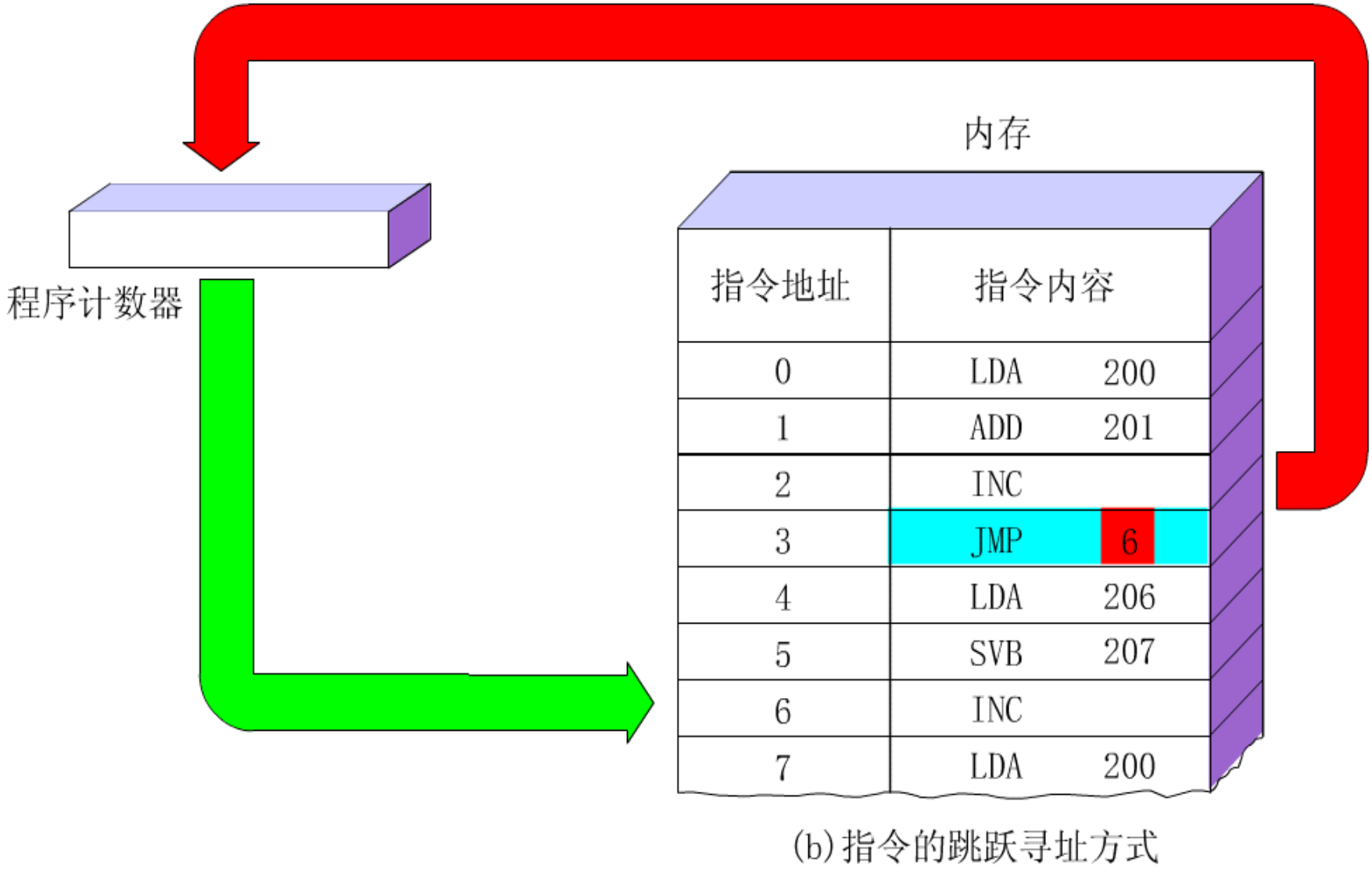


(a)指令的顺序寻址方式

指令的寻址方式

- 跳跃（转移）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
  - 转移指令的地址码字段包含下一条指令的地址
  - 执行跳跃指令时，将指令中给出的地址码送入PC中
    - 因下一次取指时仍然从PC指定的地址取指，故可实现程序的转移
    - 在遇到下一条转移指令之前，程序仍然按照PC的值顺序执行

指令的跳跃寻址方式



(b) 指令的跳跃寻址方式

操作数的寻址方式

- 操作数的三个来源：
  - 由指令中的地址码部分直接给出操作数
  - 操作数存放在CPU内的数据寄存器中
  - 操作数存放在内存的数据区（含I/O寄存器）中
    - 在指令中直接给出操作数的内存有效地址
    - 指令的地址字段给出形式地址（位移量）
      - 在指令执行时，形式地址依据某种方式变换为有效地址再存取操作数

操作数的寻址方式

- 隐含寻址
  - 指令中的某个操作数或其地址隐含在某个通用寄存器或指定的内存单元中
  - 例：Intel 8086乘法指令： mul opr
    - 约定被乘数隐含使用AX或AL寄存器

操作数的寻址方式

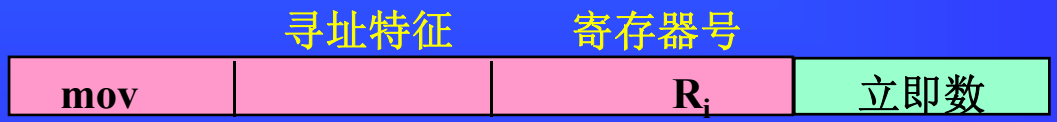
- 立即寻址
  - 指令的地址字段指出的不是操作数的地址，而是操作数本身
  - 例： Intel 8086：mov al, 7　　7——立即数

1011-0-000-00000111

操作数的寻址方式

寄存器寻址

- 操作数不放在内存中，而是放在CPU内部的通用数据寄存器中
- 例：8086：mov al, 7
- 优点：
  - 指令通常比较短
  - 指令执行时间短

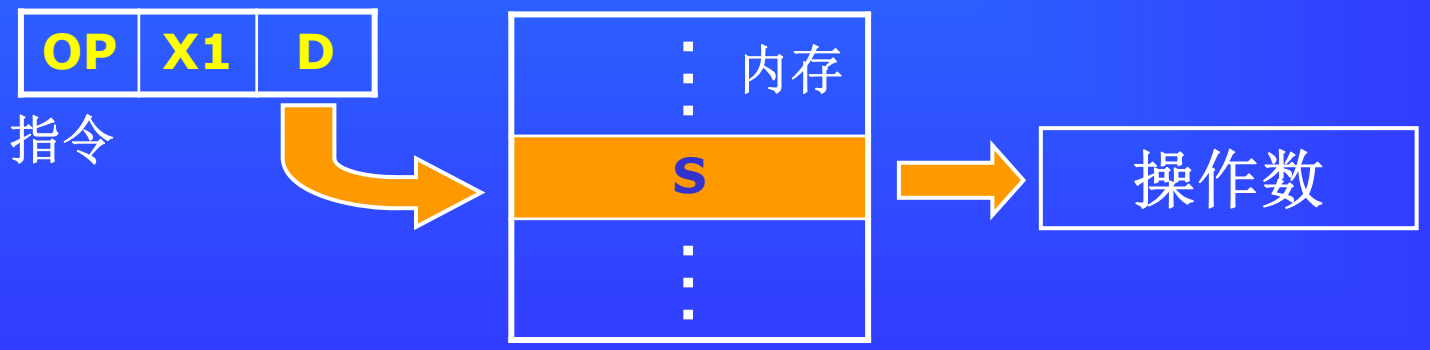


1011-0-000-00000111

操作数的寻址方式

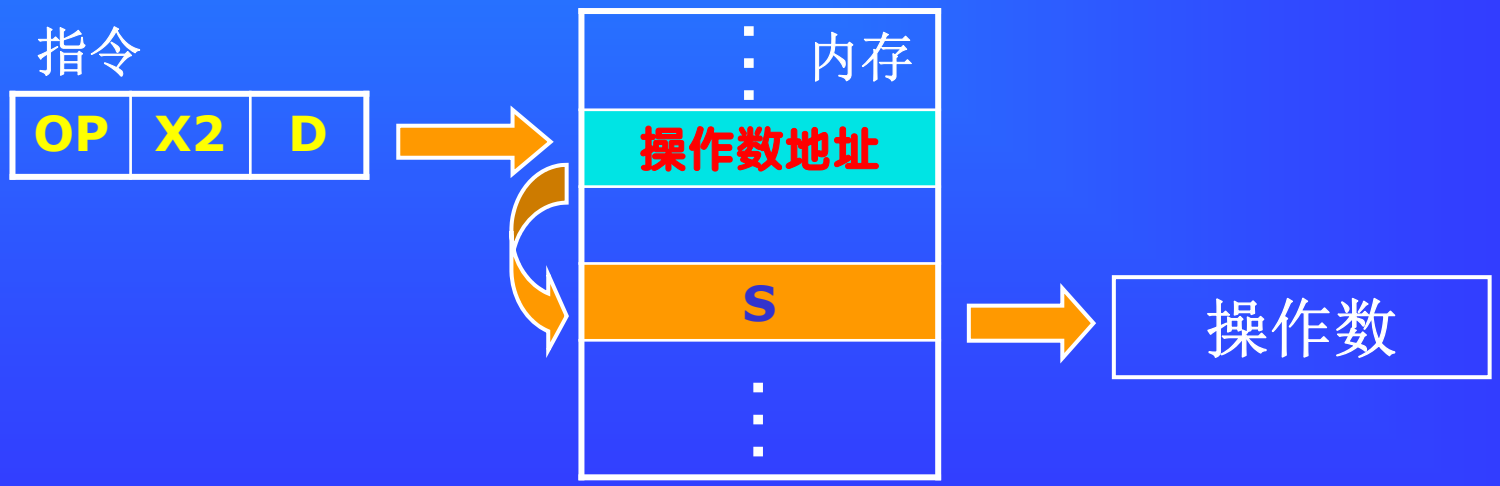
直接寻址

- 在指令中的地址码字段中给出的操作数的形式地址D就是操作数在内存中的有效地址E（E＝D）
- 通常把形式地址D称为**直接地址**
- 用S表示操作数，则S＝（E）＝（D）



操作数的寻址方式

- （内存）间接寻址（间址）
  - 指令中地址字段给出的形式地址D既不是操作数，也不是操作数的地址，而是操作数地址在内存中的存放地址（存放操作数地址的内存单元的地址）
  - D单元的内容才是操作数的有效地址



操作数的寻址方式

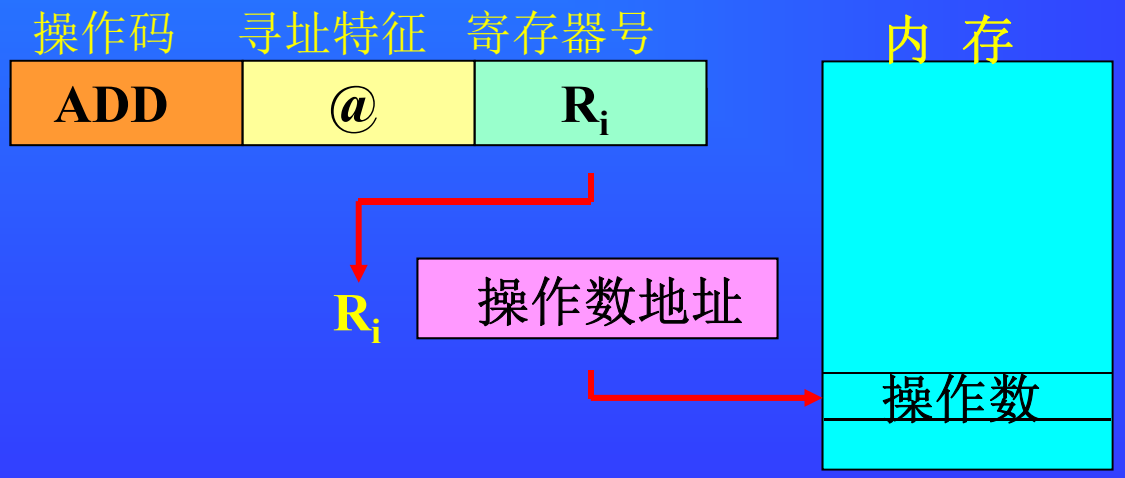
- 直接寻址和间接寻址相结合，指令有如下形式：

操作码	I	D

  - 寻址特征位
    - I＝0：直接寻址，有效地址E＝D
    - I＝1：间接寻址，有效地址E＝(D)

操作数的寻址方式

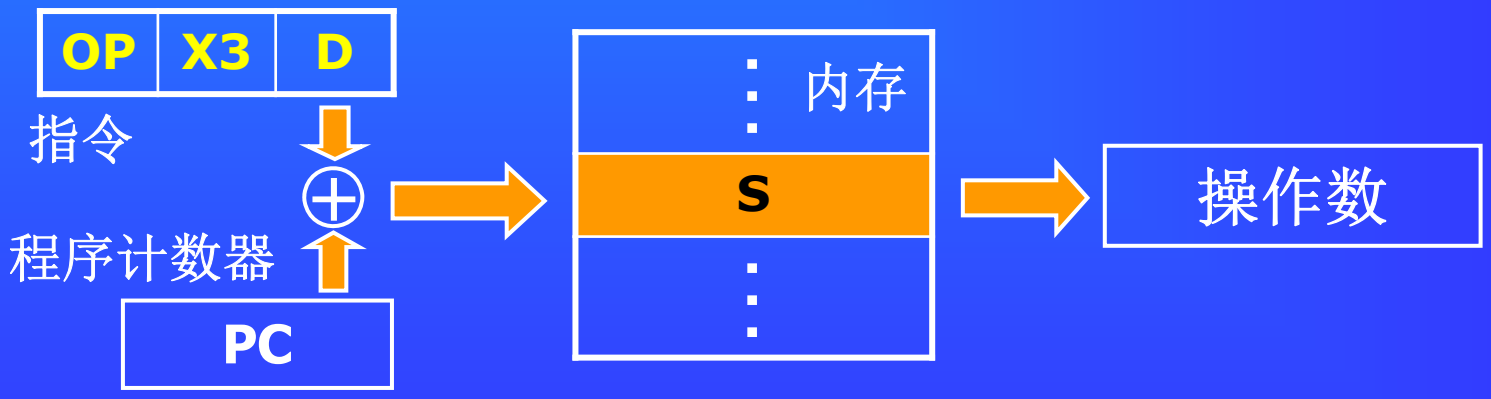
- 寄存器间接寻址 （寄存器间址）
  - 区别于寄存器寻址方式：指令格式中给出的寄存器中存放的内容不是操作数，而是操作数在内存中存放的地址
  - 间接地址在寄存器中，操作数在内存单元中



操作数的寻址方式

➢ 相对寻址方式

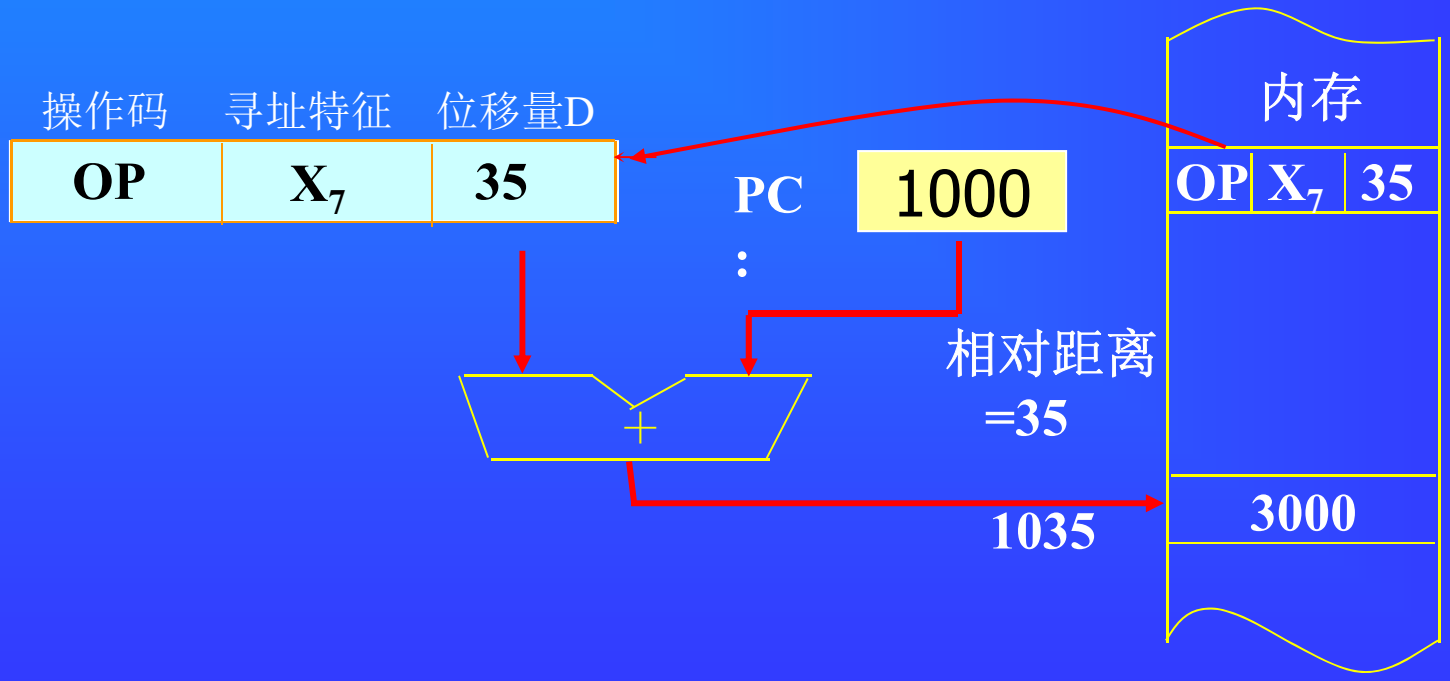
- ❑ 把程序计数器PC的内容加上指令格式中的形式地址D而形成操作数的有效地址
- ❑ 形式地址D称为位移量，其值可正可负，相对于当前指令地址进行浮动



操作数的寻址方式

相对寻址方式

- 优点：指令中不必给出绝对地址
  - 指令长度可以缩短
  - 程序可以放在内存任何位置



操作数的寻址方式

基址寻址方式

- 将CPU内的基址寄存器的内容加上指令格式中的形式地址而形成操作数的有效地址
- 优点：
  - 可以扩大寻址范围
  - 便于实现程序再定位

操作数的寻址方式

➢变址寻址方式

- ❑把CPU中某个变址寄存器的内容与位移量D相加形成操作数的有效地址
- ❑基址寻址和变址寻址的区别：
  - ✉基址寻址的目的：扩大寻址空间和实现多道程序
  - ✉变址寻址的目的：访问复合数据结构
  - ✉变址寄存器在每次使用后其值可以自动或用指令修改；基址寄存器通常是不变的

操作数的寻址方式

- 相对寻址　基址寻址　变址寻址

  有效地址=（专用寄存器）+偏移量

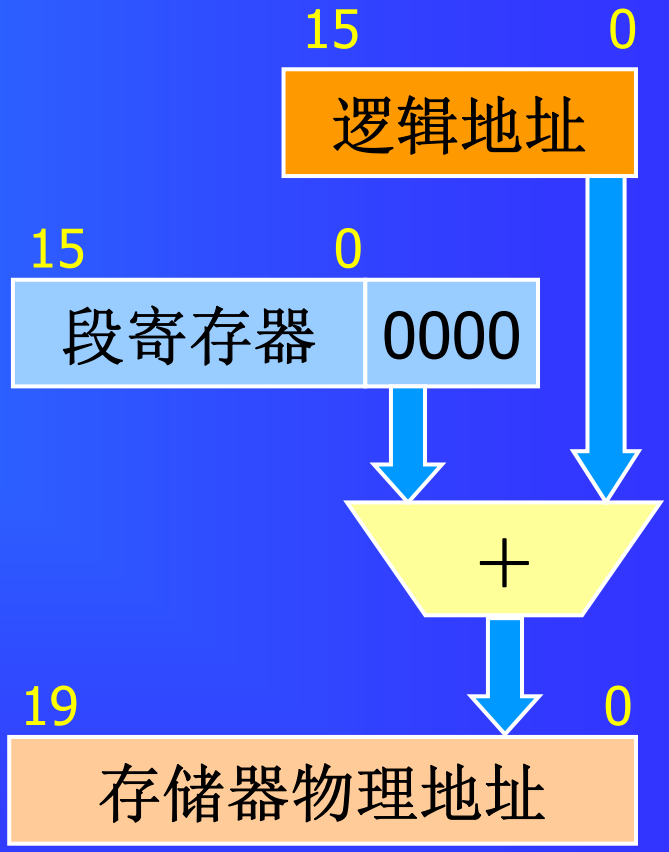
  ——偏移寻址

操作数的寻址方式

- 段寻址方式
  - 程序中的指令和数据按照逻辑结构首先分成段，每个段内均以段起始地址为基址编址
  - 例：Intel 8086
    - 将整个1M空间存储器按照最大长度64K byte划分成若干段，段内地址16位
    - 20位物理地址＝
      基地址（CPU中的段寄存器提供）+
      16位偏移量（指令中给出或是由某些寄存器提供）
    - 约定段的起始地址末四位必须为0000，故用16位段寄存器指明20位段基址

操作数的寻址方式

- **段寻址方式**
  - **例：Intel 8086**
    - 20位物理地址＝基地址＋16位偏移量
    - 寻址时，段寄存器中的16位数自动左移4位，与16位偏移量相加，形成20位物理地址



堆栈（stack）寻址方式

- 对堆栈的操作：
  - 向堆栈中增加数据（数据入栈，进栈）
  - 从堆栈中取走数据（数据出栈，退栈）
- CPU执行入栈和出栈操作时并不直接给出被操作的单元的地址，而是依据数据入出栈的顺序由硬件或软件自动给出地址，或是自动保证按顺序对数据进行操作
  - 数据入出栈的顺序：
    - 先进后出（FILO：First In Last Out）

存储器堆栈（软堆栈）

- 利用常规内存空间的某一部分或某几部分实现存储器堆栈
- 每个软堆栈有一个固定的栈底和一个浮动的栈顶
- CPU内部设置一个堆栈指针（堆栈指示器）SP指示当前栈顶的位置

- 堆栈指示器指定的存储器单元就是堆栈的栈顶
- 栈底的位置固定，无需存储

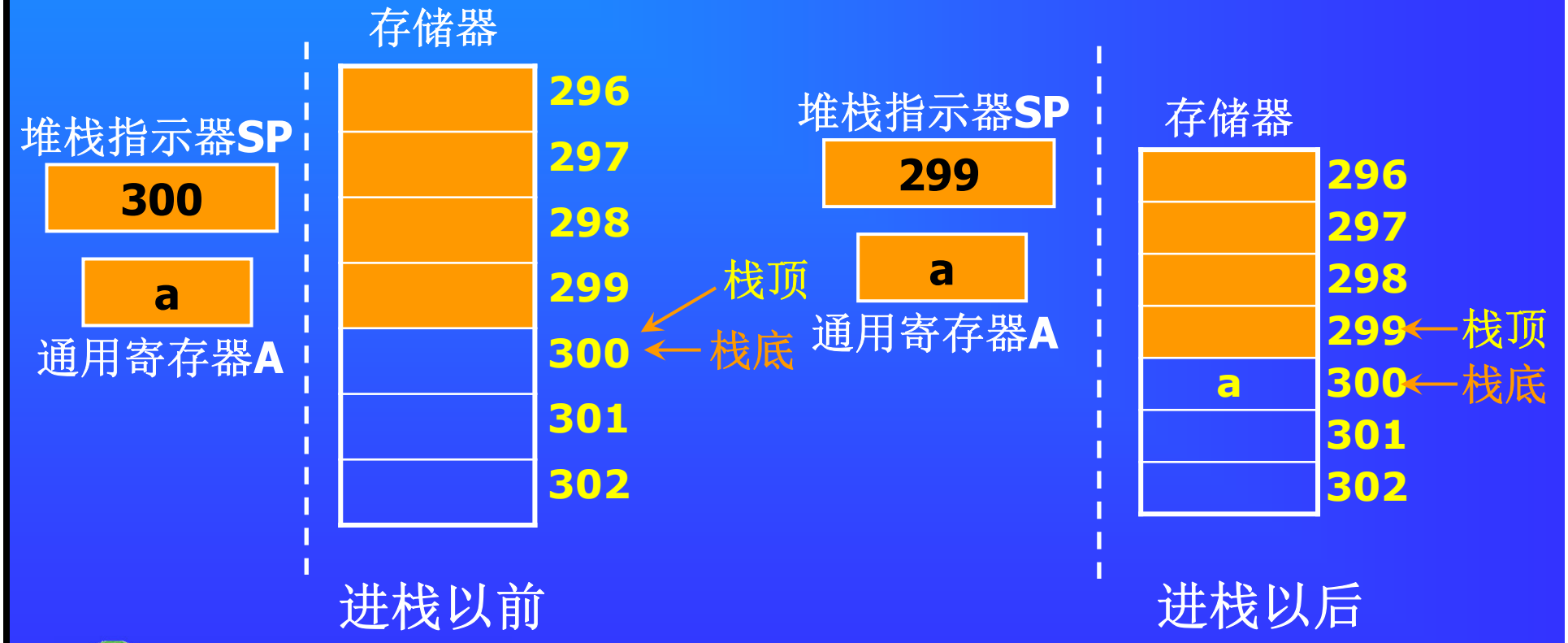
存储器堆栈入栈（进栈）操作

push A:（A）→M[sp]，（SP）－1→SP

（A）：通用寄存器/内存单元A的内容

SP：堆栈指示器

M[sp]：堆栈指针指示的存储器栈顶单元



存储器堆栈出栈（退栈）操作

pop A：（SP）+ 1→SP，M[sp]→A

（A）：通用寄存器/内存单元 A 的内容

SP：堆栈指示器

M[sp]：堆栈指针指示的存储器栈顶单元

存储器堆栈

- 栈顶指针的位置：
  - 实现方式一：总是指向下一个入栈位置（无内容的栈顶）
  - 实现方式二：总是指向下一个出栈位置（有内容的栈顶）
  - 两种实现方式入栈和出栈操作和修改堆栈指针的操作的顺序相反
- 堆栈指针的+1和-1操作：以一个存储字的长度为单位修改堆栈指针（ $+\Delta$ ； $-\Delta$ ）
- 软堆栈的两种生长方式
  - 栈顶地址比栈底地址小：向下生长的堆栈
  - 栈顶地址比栈底地址大：向上生长的堆栈

操作数的寻址方式

- 有些指令固定使用某种寻址方式
- 有些指令允许使用多种寻址方式：
  - 在指令中加入寻址方式字段指明
  - 对不同的寻址方式分配不同的操作码（看作不同指令）
  - 例：8086的mov指令
    - R-R、S-R、I-R、I-S等类型
      （R=Register、S=Storage、I=Immediate operand）
    - 各有不同的二进制操作码
    - 汇编助记符相同

寻址方式举例：Pentium

- 外部地址总线宽度36位，也支持32位物理地址空间
  - 实地址模式：段寻址方式
    - 16位段基址左移4位，加上16位段内偏移量，得到20位物理地址
  - 保护模式：
    - 32位段基地址加上段内偏移量得到32位线性地址，由存储管理部件将其转换成32位物理地址

Pentium寻址方式

序号	寻址方式名称	有效地址E	说　明
(1)	立即	（操作数在指令中）	
(2)	寄存器	（操作数在寄存器中）	指令给出寄存器号
(3)	直接	E=Disp	Disp：位移量
(4)	基址	E=(B)	B：基址寄存器
(5)	基址＋位移量	E=(B)+Disp	
(6)	比例变址＋位移量	E=(I)*S+Disp	I：变址寄存器，S：比例因子
(7)	基址＋变址＋位移量	E=(B)+(I)+Disp	
(8)	基址＋比例变址＋位移量	E=(B)+(I)*S+Disp	
(9)	相对	指令地址=(IP)+Disp	IP（或EIP）：指令指针寄存器

寻址方式举例

【例4】一种二地址RS型指令的结构如下：

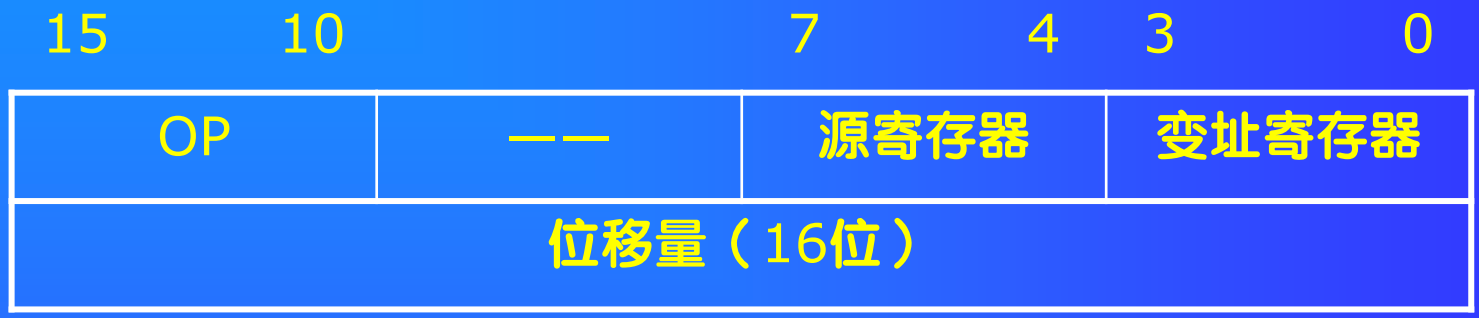
6位	4位	1位	2位	16位
OP	—	通用寄存器	I	X	位移量D

其中I为间接寻址标志位，X为寻址方式字段，D为位移量字段。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。

	I	X	有效地址E	说明	寻址方式
(1)	0	00	E=D		直接寻址
(2)	0	01	E=(PC)+D	PC为程序计数器	相对地址
(3)	0	10	E=(R2)+D	R2为变址寄存器	变址寻址
(4)	1	11	E=(R3)		寄存器间接寻址
(5)	1	00	E=(D)		存储器间接寻址
(6)	0	11	E=(R1)+D	R1为基址寄存器	基址寻址

指令格式

【例2】已知某16位计算机指令格式如下，OP为操作码字段，试分析指令格式特点。



答：

(1)双字长二地址指令，用于访问存储器

(2)操作码字段OP为6位，如果采用等长操作码，可以指定$2^6=64$种操作

(3)一个操作数在源寄存器（共16个）中，另一个操作数在存储器中（由变址寄存器和位移量决定），所以是R-S型指令

指令的分类

- 按照指令的功能划分，一个较为完善的指令系统，通常具备下类指令
  - 数据传送指令
  - 算术运算指令
  - 逻辑运算指令
  - 移位操作指令
  - 程序控制指令
  - 输入输出指令
  - 串处理指令
  - 特权指令

指令的分类

数据传送类指令

功能

- 实现寄存器与寄存器、寄存器与存储单元以及存储单元与存储单元之间的数据传送

类型

- 取数指令（load）、存数指令（store）、传送（mov）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

指令的分类

- 算术运算指令
  - 功能：定点数或浮点数的算术运算
  - 依据机器硬件功能的不同，算术运算指令的多少各不相同：
    - 简单的CPU无乘除法指令
    - 高档的CPU支持浮点运算
    - 并行机有向量运算指令
  - 类型
    - 二进制定点加、减、乘、除指令
    - 浮点加、减、乘、除指令
    - 十进制加、减运算指令

指令的分类

逻辑运算指令

功能

- 与、或、非、异或等逻辑运算功能以及一些位操作
- 主要用于无符号数的位操作、代码的转换、判断及运算
- 各个比特之间无进位、借位关系

类型

- 逻辑加、逻辑乘、按位加（逻辑异或）、位测试、位置位、位清除、位取反

指令的分类

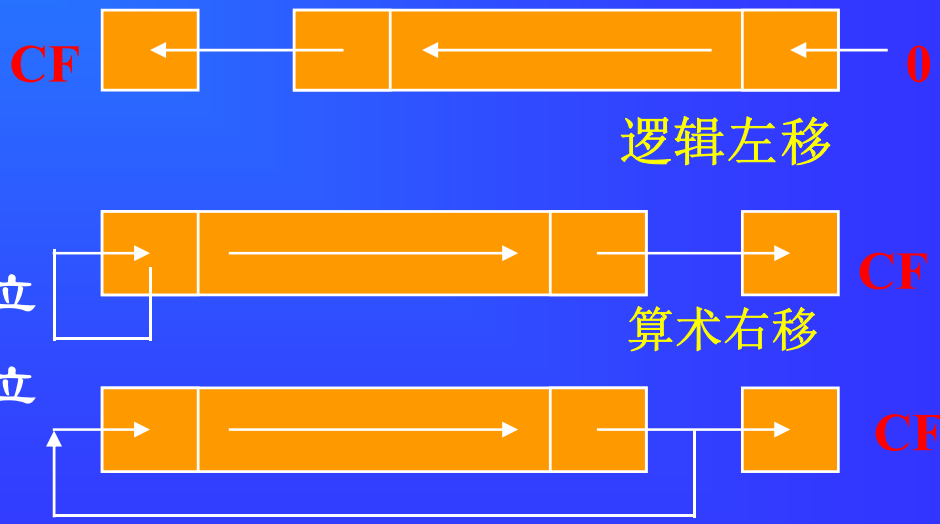
移位操作指令

功能

- 对寄存器或内存单元的内容实现左移、右移或循环移位
- 含算术移位（符号不移位）和逻辑移位（所有位均移位）指令

类型组合

- 左移、右移
- 算术移位、逻辑移位
- 单向移位、循环移位



指令的分类

- 程序控制指令（转移指令）
  - 功能
    - 控制程序的执行顺序
  - 类型
    - 转移指令：条件转移指令（进位、零、负、溢出）、无条件转移指令
    - 循环指令
    - 子程序调用（转子程序）指令、返回主程序指令
    - 中断调用与中断返回指令

指令的分类

输入输出指令

功能

- 在I/O独立编址方式中，输入输出指令实现CPU内的寄存器与外部设备接口之间的信息交换
  - 例如：启动外围设备，检查测试外围设备的工作状态等
- 在I/O与存储器统一编址方式中，由访存指令实现CPU内的寄存器与外部设备接口之间的信息交换

类型

- 输入指令、输出指令

指令的分类

- 串处理指令
  - 功能
    - 对成批数据进行处理
  - 类型
    - 串传送、串转换、串替换

指令的分类

- 特权指令（privileged instruction）
  - 功能
    - 具有特殊权限的指令，只用于操作系统或其他系统软件
    - 在多用户、多任务的计算机系统中用于系统资源的分配和管理
  - 类型
    - 存储管理指令
    - 停机指令
    - 系统状态控制指令

指令的分类

- 其他指令
  - 类型
    - 状态寄存器置位、复位指令
    - 暂停指令
    - 空操作指令
    - 自陷（TRAP）指令
    - 系统控制用的特殊指令

指令系统的发展

- 早期的计算机：
  - 结构简单，硬件功能较弱
  - 指令系统简单，指令种类少，指令系统功能弱
  - 计算机的性能较差
- 早期的发展趋势：
  - 机器性能不断提高，硬件成本下降，软件成本上升
  - 计算机系统的结构逐渐复杂，指令的种类/数目繁多
  - 同一系列计算机的指令系统越来越复杂，机器结构也越来越复杂

指令系统的发展

- 传统的设计思想：
  - 性能越高的计算机，其指令系统应该提供越多的指令种类和越复杂的指令功能
  - 依据这种设计思想实现的计算机被称为“复杂指令系统计算机”（CISC：Complex Instruction Set Computer）

指令系统的发展

- 复杂指令系统计算机的缺点
  - 统计表明，CISC的指令系统中花费很大力量增加的指令，在实际使用中，有许多指令大多数任务根本用不到
    - 通常占指令种类20%左右的指令在程序中出现的频率却占80%
  - 硬件设计费时费力、成本增加
  - 设计复杂、高效而可靠的指令系统来越难
    - 指令功能虽强，但操作繁杂，执行速度低、可靠性差
- 1975年，IBM公司的John Cocke提出了精简指令系统计算机（RISC：Reduced Instruction Set Computer）的设计思想

RISC的设计原则

- 精简指令种类
  - 选取使用频率最高的一些简单指令，去掉使用频率低而又可以通过高频率简单指令组合实现其功能的指令
- 提高指令译码速度
  - 指令长度固定，简化指令格式，减少寻址方式的种类
- 减少访存次数
  - 增加CPU内寄存器的数目
  - 尽量使用R-R操作指令：只有load和store两种指令可以访问存储器，其余指令的操作都在寄存器之间进行
- 提高指令执行效率
  - 一般采用硬布线控制方式，不采用或少采用微程序控制方式
  - 大部分指令在一个机器周期内完成，复杂指令可化为简单指令序列
  - 采用多级指令流水线结构，处理机在同一时间内可执行多条指令

RISC和CISC 的比较

- 设
  - 高级语言程序经编译后在机器上运行的机器指令数为I
  - 每条机器指令执行时所需要的平均机器周期数是C
  - 每个机器周期的执行时间为T
- 计算机执行程序的时间P计算：P=I × C × T

	I	C	T
RISC	1.2 ~ 1.4	1.3 ~1.7	<1
CISC	1	4 ~ 6	1

（I、T为比值，C为实际周期数）

RISC处理机实例：SPARC

例. 在SPARC中，有一些指令没有选入指令系统，但很容易使用指令系统中的另外一条指令来替代实现。下表左半部列出了6条指令的功能。请在表的右半部填入替代指令及实现方法。

约定：当某些指令中地址字段为寄存器R0的编号时，以立即数0（而不是R0的内容）作为操作数

解：

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simm13=1,作为操作数
DEC	寄存器内容减 1	ADD	立即数simm13= -1,作为操作数
NEG	取相反数	SUB	R0－Rs->Rd
NOT	取反码	XOR	立即数imm13= ffff,作为操作数
CLR	清除寄存器	ADD	R0+R0->Rd

ARMv8-A架构支持的指令集

- 两种执行状态（Execution State）：
  - AArch64执行状态
    - A64指令集则使用64位工作寄存器
    - 使用64位计算模式
    - 指令长度依然保持32位（4字节）
  - AArch32执行状态
    - A32指令集使用32位工作寄存器
  - 均支持SIMD（Single Instruction Multiple Data）和浮点运算指令

ARMv8-A架构支持的指令集

- AArch64执行状态
  - 只能使用A64指令集
  - 所有指令均为32位等长指令字
- AArch32执行状态
  - A32指令集：
    - 对应ARMv7架构及其之前的ARM指令集
    - 32位等长指令字
  - T32指令集
    - 对应ARMv7架构及其之前的Thumb/Thumb-2指令集
    - 16位和32位可变长指令字结构

ARM寻址方式

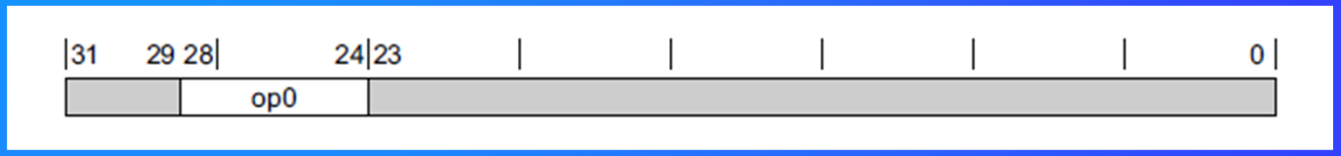
- ARM处理器共有八种寻址方式：
  - 立即数寻址
  - 寄存器寻址
  - 寄存器间接寻址
  - 基址寻址
  - 多寄存器寻址
  - 堆栈寻址
  - 相对寻址
  - 寄存器移位寻址

ARM A64指令集的指令分类

指令类型	说明
分支指令	条件跳转、无条件跳转（#imm、register）指令
异常发生指令	系统调用类指令（SVC、HVC、SMC）
系统寄存器指令	读写系统寄存器，如：MRS、MSR指令可操作PSTATE的位段寄存器
数据处理指令	包括各种算数运算、逻辑运算、位操作、移位（shift）指令
load/store访存指令	load/store（单寄存器、寄存器对儿、多寄存器等）

A64 instruction set encoding

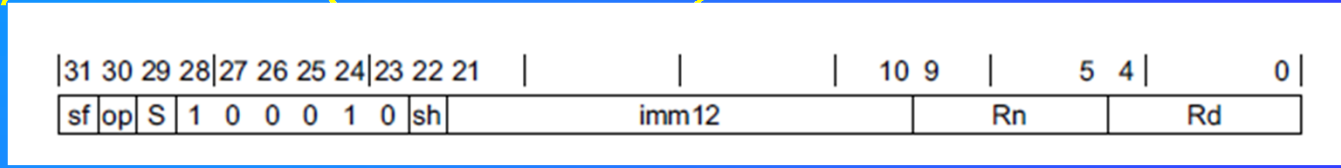
- A64 instruction set encoding



字段编码 op0	编码分组或指令类型
0000x	保留（Reserved）
00011	未分配（Unallocated）
100xx	数据处理——立即寻址
101xx	分支、异常发生后系统指令
x1x0x	Loads and Stores
x101x	数据处理——寄存器寻址
x111x	数据处理——浮点与SIMD指令

A64 instruction set encoding

- Add/subtract (immediate) instruction set encoding



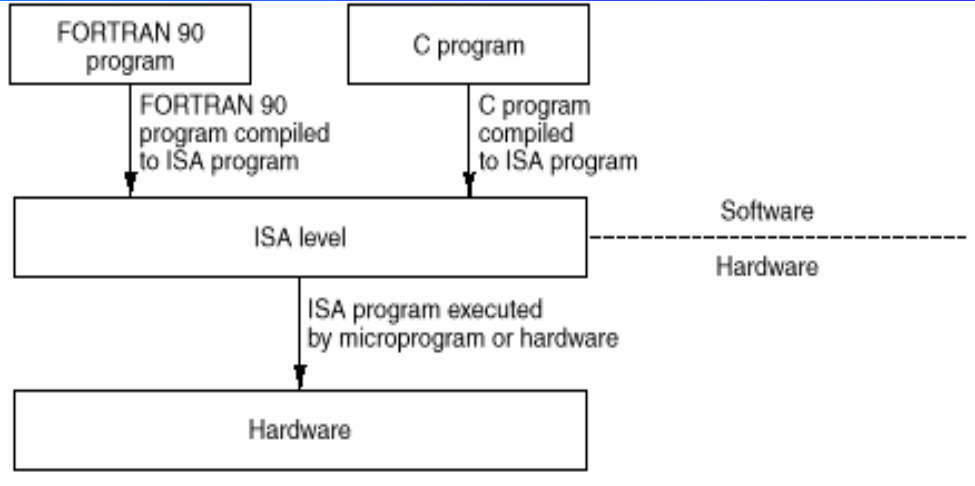
Decode fields			Instruction
sf	op	S	
0	0	0	ADD (immediate) - 32-bit variant
0	0	1	ADDS (immediate) - 32-bit variant
0	1	0	SUB (immediate) - 32-bit variant
0	1	1	SUBS (immediate) - 32-bit variant
1	0	0	ADD (immediate) - 64-bit variant
1	0	1	ADDS (immediate) - 64-bit variant
1	1	0	SUB (immediate) - 64-bit variant
1	1	1	SUBS (immediate) - 64-bit variant

S: setting flags

- Add (immediate)：**源寄存器Rn与立即数imm12移位（可选，sh）后相加，结果送入目标寄存器Rd**

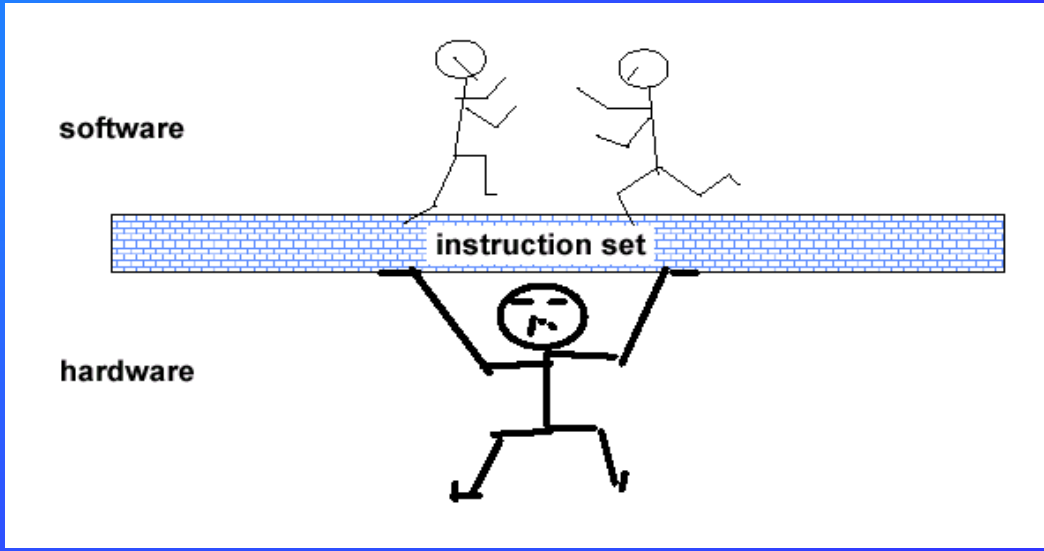
指令系统体系结构（ISA）

- 概念
  - ISA：Instruction Set Architecture
  - 与程序设计有关的计算机架构
  - 机器语言程序员看到的计算机的属性
- 内容
  - 寄存器组织
  - 存储器的组织和寻址方式
  - I/O系统结构
  - 数据类型及其表示
  - 指令系统
  - 中断机制
  - 机器工作状态的定义及切换
  - 保护机制



本章内容

- 指令系统的概念及其发展
- 指令格式
- 寻址方式
- 指令类型和典型指令介绍
- RISC



本章重点

- 指令格式
- 操作数的寻址方式
- CISC和RISC的特点

本章作业

一、某16位机主存容量位为64K×16位，采用单字长单地址指令，共有60条，一个变址寄存器。试采用直接、间接、变址、相对这四种寻址方式设计指令格式，并说明每一种寻址方式的寻址范围及有效地址计算方法。

二、某机字长32位，共有机器指令100条。指令为单字长，等长操作码。CPU内部有通用寄存器32个，可作变址寄存器用。存储器按字节编址。指令拟采用直接寻址、间接寻址、变址寻址和相对寻址等4种寻址方式。

1、分别画出采用4种不同寻址方式的单地址指令的指令格式；

2、采用直接寻址和间接寻址方式时，可直接寻址的存储器空间各是多少？

3、写出4种寻址方式下，有效地址E的表达式。

计算机组成原理

Principle of Computer Organization

第四章　指令系统

本章结束