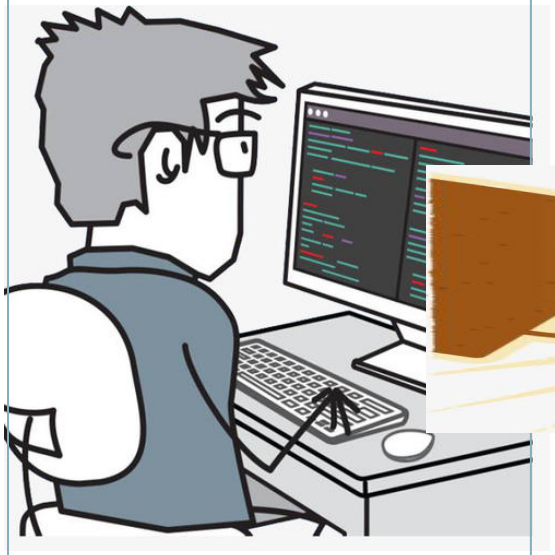
# 第四讲
# 指令与单周期CPU
# （第2部分：指令分类和寻址方式）

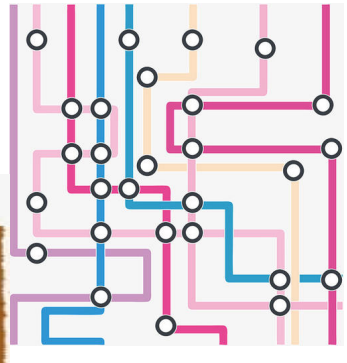
虚拟世界

协议

物理世界

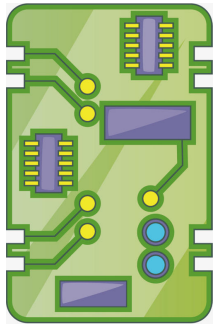


```
-u
0B98:0000 B80000        MOV     AX,0000
0B98:0003 BB0000        MOV     BX,0000
0B98:0006 EB03          JMP     000B
0B98:0008 050100        ADD     AX,0001
0B98:000B 40            INC     AX
0B98:000C 008946F6      ADD     [BX+DI+F646],CL
0B98:0010 A1A407        MOV     AX,[07A4]
0B98:0013 398672FF      CMP     [BP+FF72],AX
```



VS





程序

指令

微体系结构

# 内容

- **概述**
- **指令格式**
- **指令和数据的寻址方式**

# 内容

- **概述**
- 指令格式
- 指令和数据的寻址方式

# 指令

- 指令
  - 控制计算机执行某种操作的命令
- 指令分类
  - 按计算机系统的层次结构
    - 微指令：属于硬件，和微体系结构相关，一条指令由若干条微指令构成
    - 机器指令（简称为指令）：介于硬件与软件之间，每一条指令可完成一个独立的算术运算或逻辑运算操作
    - 宏指令：属于软件，由若干条机器指令组成的软件指令
  - 按指令集规模
    - 复杂指令系统计算机（CISC）
    - 精减指令系统计算机（RISC）

# 指令长度

- **指令字长**
  - 指令中包含的二进制的位数
  - 要满足规整性要求
- **机器字长**
  - 计算机能够直接处理的二进制数据的位数
  - 决定了计算机的运算精度，机器字长一般与主存（存储）单元的位数一致

# 指令长度

- 分类（依据机器字长）
  - 单字长指令：指令字长等于机器字长的指令
  - 半字长指令：指令字长等于半个机器字长的指令
  - 多字长指令：指令字长等于多个机器字长的指令
    - 优缺点
      - 提供足够的地址位来解决访问内存（存储）单元的寻址问题
      - 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

# 指令长度

- 分类（依据指令长度）
  - 等长指令字结构
    - 指令系统中，各种指令长度相等
      - 结构简单
      - 指令长度固定
      - 但会导致浪费
  - 变字长指令结构

# 指令长度

- 分类（依据指令长度）
  - 等长指令字结构
  - 变字长指令结构
    - 功能不同的指令，长度可变化，如单字长指令、多字长指令
      - 指令结构灵活
      - 控制复杂
      - 没有浪费

# 指令系统

- **指令系统**
  - 对应了微体系结构
  - 计算机中所有机器指令的集合
  - 不同类型的计算机有不同的指令系统
  - 指令系统的格式直接影响到计算机的硬件、软件和计算机的使用范围
  - 设计要求：完备性、有效性、规整性、兼容性
- **系列机**
  - 指令系统基本相同、系统结构基本相同的一系列计算机

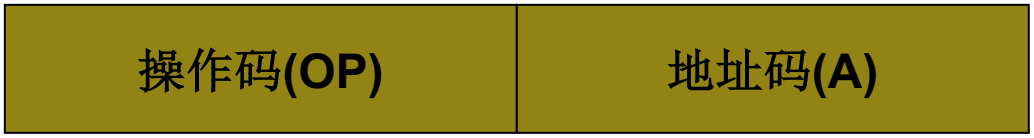
# 内容

- 概述
- **指令格式**
- 指令和数据的寻址方式

# 指令格式

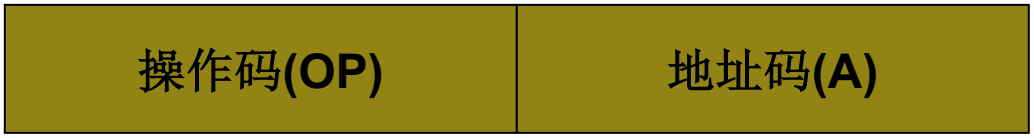
- **设计计算机时，对指令系统的每一条指令都要规定操作码和地址码。**
  - 指令的操作码表示该指令应进行什么性质的操作，如进行加法、减法、乘法、除法、取数、存数等等。不同的指令用操作码字段的不同编码来表示，每一种编码代表一种指令
  - 指令的地址码表示操作的对象，可能是数据，也可能是数据的地址

# 指令格式




- 操作码
  - 表明指令的操作特性与功能
  - 不同功能的指令，操作码字段的编码不同
  - 操作码的长度取决于计算机的指令的数量
    - $L_{OP}=\lceil \log_2 n \rceil$

# 指令格式




- **地址码**
  - 用来协助表示操作数或操作数的地址，需要显式或隐式表示：
    - 源操作数或其地址：物理地址/虚拟地址、寄存器编号、I/O端口号、立即数
    - 结果的地址
    - 下条指令地址

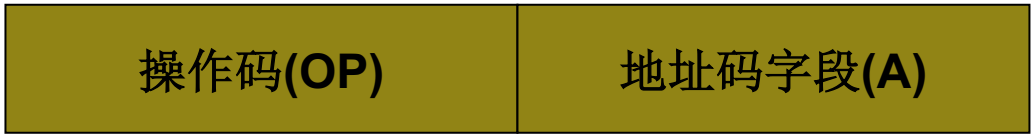
# 指令助记符

- **由于硬件只能识别1和0，采用二进制操作码是必要的，**
  - 二进制来书写程序麻烦
- **指令助记符**
  - 便于书写和阅读程序
  - 每条指令通常用3个或4个英文缩写字母来表示
  - 不同的计算机系统，规定不一样
  - 必须用汇编语言翻译成二进制代码

# 操作数类型

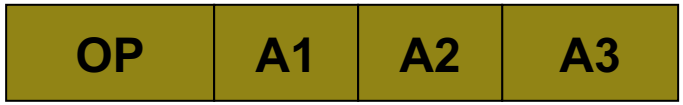
- 操作数类型
  - 地址数据
    - 作为无符号整数进行地址计算
  - 数值数据
    - 定点数
    - 浮点数
    - 压缩十进制数（一个字节用2位BCD码表示）
  - 字符数据
    - 文本数据或字符串，目前广泛使用ASCII码
    - 8字节：7位表示字符，最高位作为奇偶校验位
  - 逻辑数据
    - 数据以每1比特方式处理时，称为逻辑数据

# 指令格式




- 按照地址字段分类

三操作数指令：




(A1 ) OP (A2) → A3

二操作数指令：


(A1) OP (A2) → A1

一操作数指令：

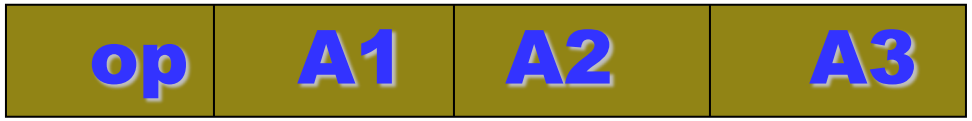

(AC ) OP (A1)→ AC

0 操作数指令：


不需要操作数（如NOP）

# 指令格式

- 三地址指令
  - 指令格式如下：




    - 操作码op
    - 第一操作数A1
    - 第二操作数A2
    - 结果A3
  - 功能描述：
    - (A1) op (A2)→A3
    - (PC) +1→PC
  - 这种格式虽然省去了一个地址，但指令长度仍比较长，所以只在字长较长的大、中型机中使用，而小型、微型机中很少使用

# 指令格式

- **二地址指令**
  - 其格式如下：


    - 操作码op
    - 第一操作数A1
    - 第二操作数A2
  - 功能描述：
    - (A1) op (A2)→A1
    - (PC)+1→PC
  - 指令执行之后，A1中原存的内容被新的运算结果替换

# 指令格式

- **二地址指令**
  - 其格式如下：


- 操作码op
- 第一操作数A1
- 第二操作数A2

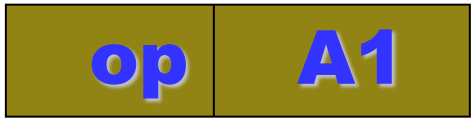
慢

二地址地址根据操作数的物理位置分为：

- SS　存储器-存储器类型
- RS　寄存器-存储器类型
- RR　寄存器-寄存器类型

# 指令格式

- **一地址指令**
  - 指令格式为：




    - 操作码op
    - 第一操作数A1
  - 功能描述：
    - (AC) op(A1) →A1
    - (PC)+1→PC
  - 单操作数运算指令，如“+1”、“-1”、“求反”
  - 指令中给出一个源操作数的地址

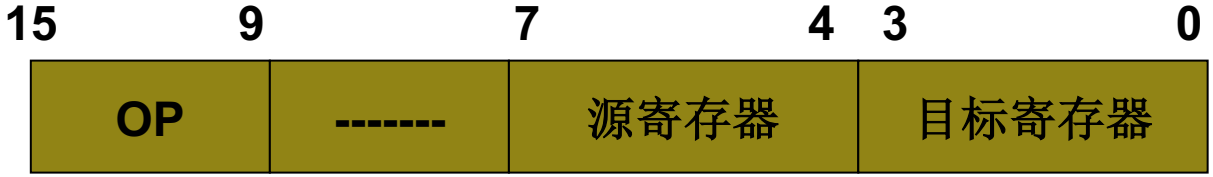
# 指令格式

- **零地址指令**
  - 指令格式为：


  - 操作码op
  - “停机”、“空操作”、“清除”等控制类指令。

# 指令格式举例（1）

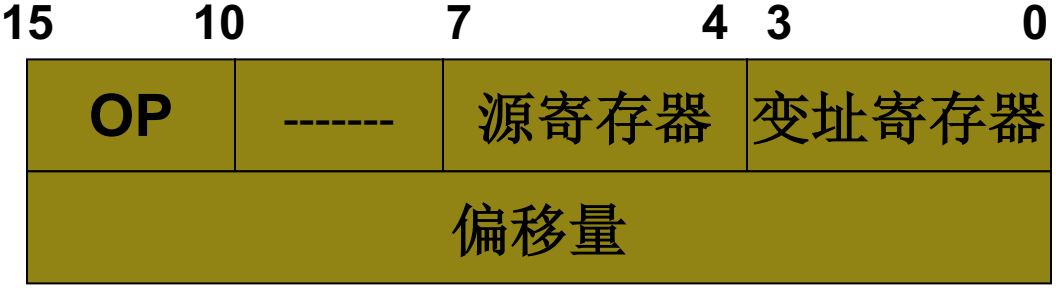
- 指令格式如下，其中OP为操作码，试分析指令格式的特点




- 分析
  - 单字长二地址指令
  - OP为7位，可以表示128条指令
  - 源操作数和目的操作数都是通用寄存器(可分别使用16个)
  - 是RR型指令.
  - 适合于算术运算和逻辑运算指令

# 指令格式举例（2）

- 指令格式如下，其中OP为操作码，试分析指令格式的特点




- 分析：
  - 双字长，二地址指令，用于访问存储器
  - OP为6位 ， 可以表示64条指令
  - 一个操作数在寄存器中（可用16个），另一个在主存中，所以是RS型指令。

# 指令格式举例（3）

某机指令字长度为16位，包括基本操作码4位和3个地址段，每个地址段长4位，其格式为：


1） 4位基本操作码若全部用于表示三地址指令，则共有多少条？

2）若三地址指令15条，二地址指令最多可有多少条？

3）若三地址指令、二地址指令和一地址指令各有15条，零地址指令16条，则共有61条指令。

# 指令格式举例（3）

| | | | | |
|---|---|---|---|---|
| 0 0 0 0 | X X X X | X X X X | X X X X | 15条三地址指令 |
| 0 0 0 1 | X X X X | X X X X | X X X X | |
| ⋮ | ⋮ | ⋮ | ⋮ | |
| 1 1 1 0 | X X X X | X X X X | X X X X | |
| 1 1 1 1 | 0 0 0 0 | X X X X | X X X X | 15条二地址指令 |
| 1 1 1 1 | 0 0 0 1 | X X X X | X X X X | |
| ⋮ | ⋮ | ⋮ | ⋮ | |
| 1 1 1 1 | 1 1 1 0 | X X X X | X X X X | |

# 指令格式举例（3）

| | | | |
|---|---|---|---|
| 1 1 1 1 | 1 1 1 1 | 0 0 0 0 | X X X X |
| 1 1 1 1 | 1 1 1 1 | 0 0 0 1 | X X X X |
| ⋮ | ⋮ | ⋮ | ⋮ |
| 1 1 1 1 | 1 1 1 1 | 1 1 1 0 | X X X X |

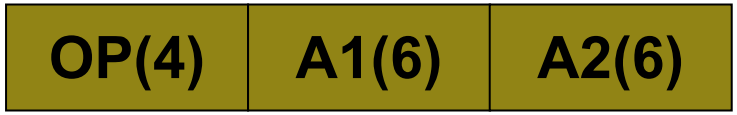
15条一地址指令

| | | | |
|---|---|---|---|
| 1 1 1 1 | 1 1 1 1 | 1 1 1 1 | 0 0 0 0 |
| 1 1 1 1 | 1 1 1 1 | 1 1 1 1 | 0 0 0 1 |
| ⋮ | ⋮ | ⋮ | ⋮ |
| 1 1 1 1 | 1 1 1 1 | 1 1 1 1 | 1 1 1 1 |

16条零地址指令

# 指令格式举例（4）

设某指令系统指令字长16位，每个地址码为6位。若二地址指令15条，一地址指令34条，则剩下零地址指令最多有多少条？




解:操作码按短到长进行扩展编码

二地址指令: (0000 – 1110) 共15条 (不扩展时为16条)

一地址指令: 1111 (000000 – 100001); (34条)

零地址指令: 1111 (100010 – 111111) (000000 – 111111)

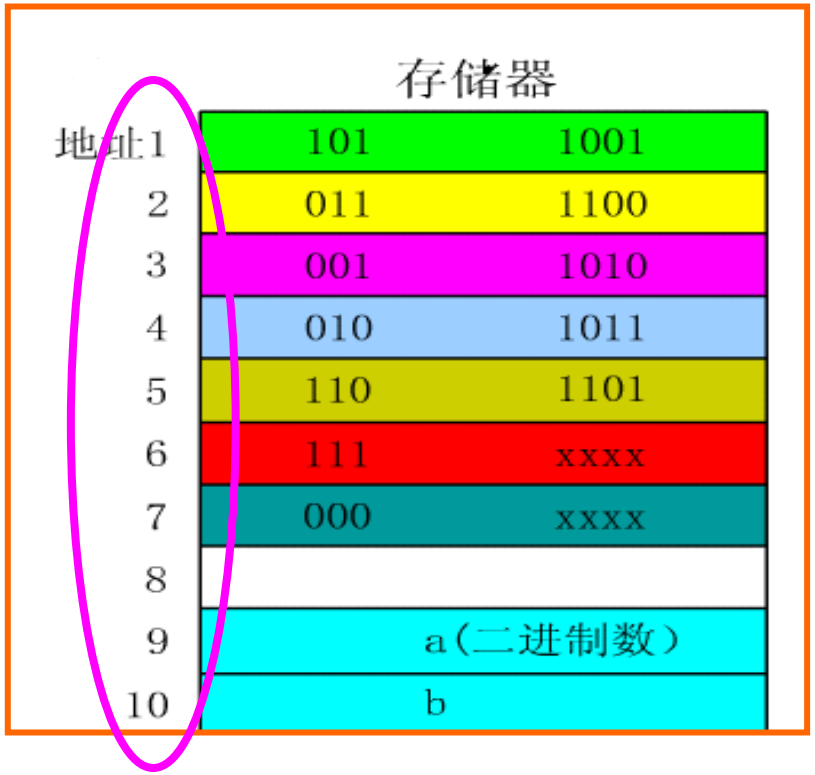
(30种扩展标志)

故零地址指令最多有$30\times2^6=15\times2^7$ 种

# 内容

- 概述
- 指令格式
- **指令和数据的寻址方式**

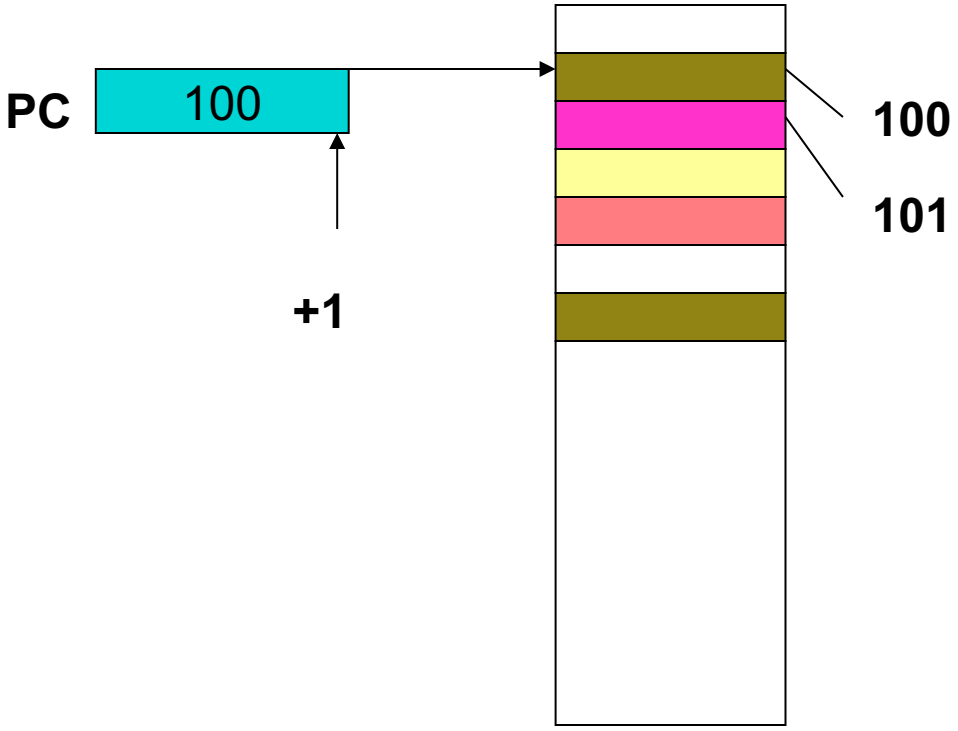
# 寻址方式




指令和操作数均存放在主存（或者地址空间对应的设备）中，寻址方式指的是生成指令或操作数地址的方式。

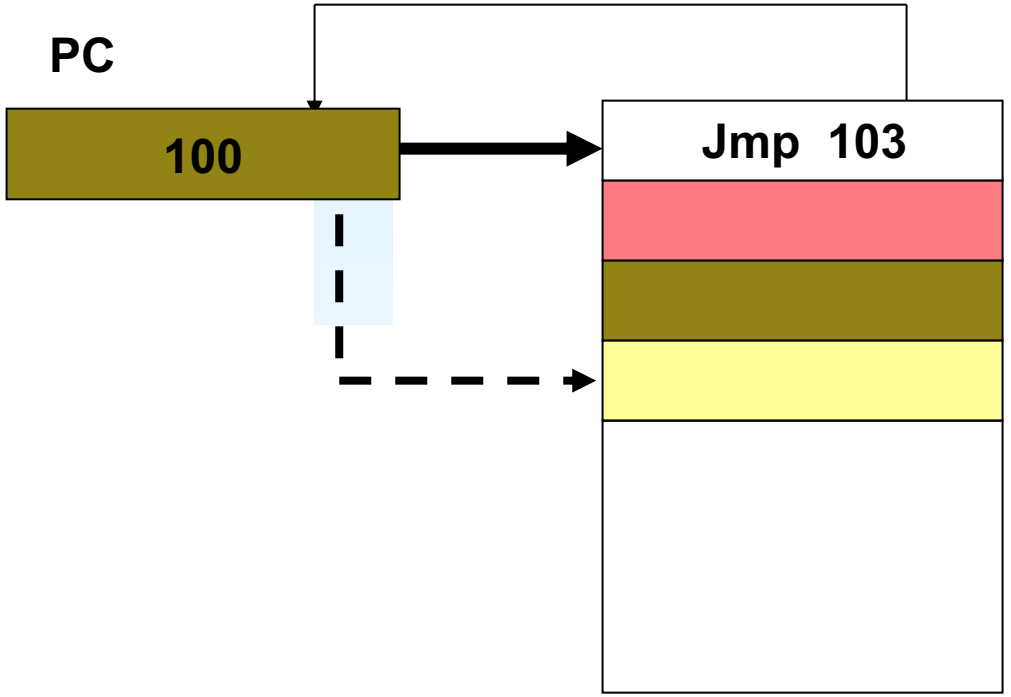
# 指令寻址（1）

- 顺序寻址
  - 下一条指令的地址在当前指令的地址基础上加“1”
  - 由程序计数器PC相关的硬件机制自动完成
  - 对上层操作系统（软件）透明



# 指令寻址（2）

- 指令直接给出或由指令执行结果决定的寻址方式，如转移指令
- 对上层操作系统（软件）不透明

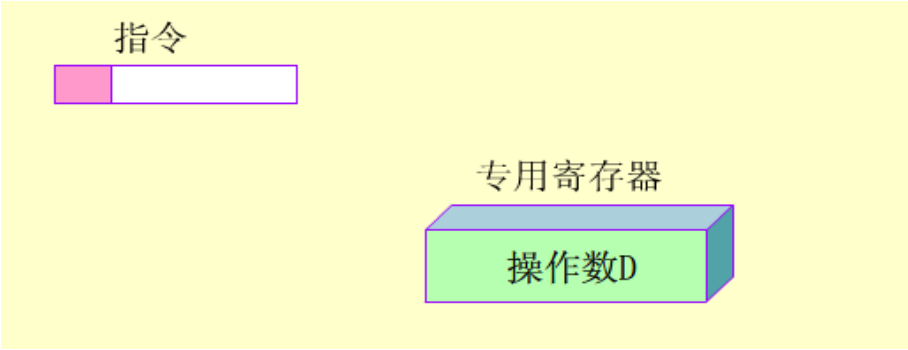


# 操作数寻址

- 有效地址——E
  - 操作数的地址（或者寄存器编号、I/O端口号等），一般用E 表示
- 取内容/取值——括号
  - (E)指地址（或者寄存器编号、I/O端口号等）E对应的内容
- 操作数寻址
  - 根据地址码和隐式信息，寻找操作数的过程

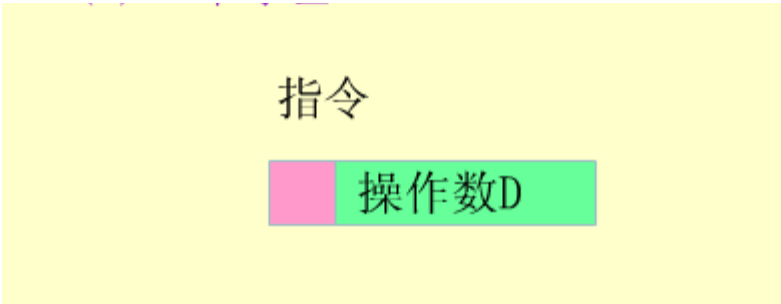
# 操作数寻址（隐含寻址）

- 隐含寻址
  - 指令中操作数的地址不直接给出，而是隐含给出
    - 二操作数地址指令就隐含AC作为操作数地址
  - 这类寻址方式中，被隐含的部分是寄存器
  - 操作数



# 操作数寻址（立即数寻址）

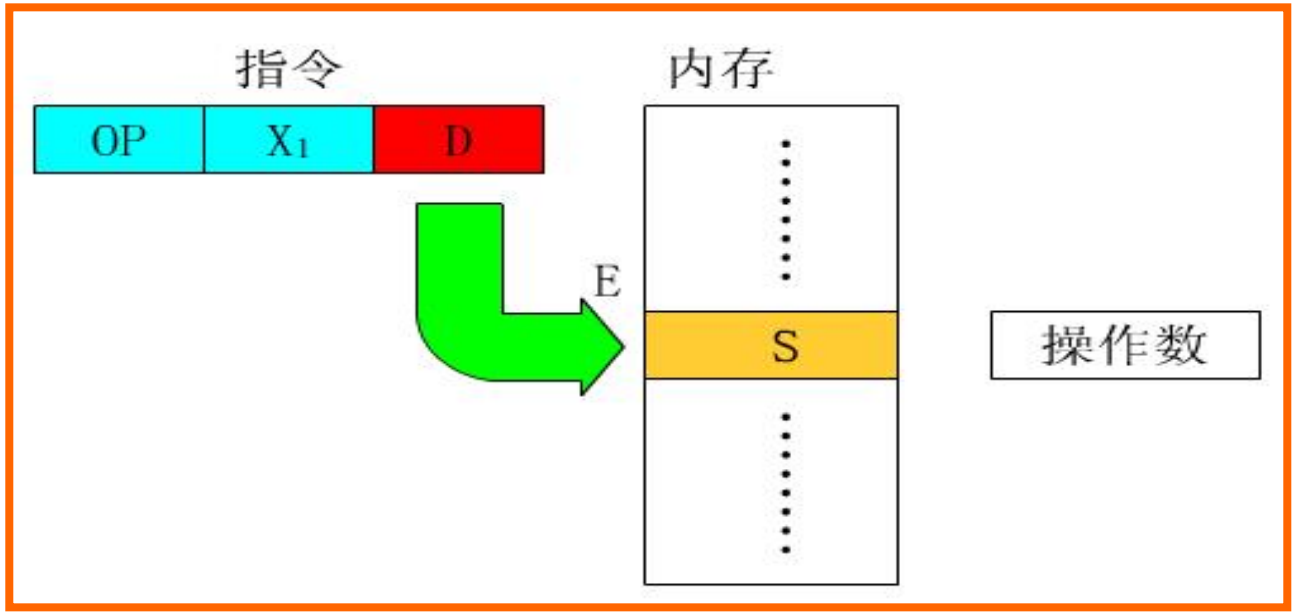
- 指令的地址码字段指出的不是地址，而是操作数本身
  - ADD　AX，2038H
  - 不计算E，S=2038H




- 优点：
  - 不需要访问存储器，执行速度快
- 缺点：
  - 数据的大小受形式地址字段的长度限制。
  - 若地址字段为16位，则表示的数据范围-32768 ~ 32767

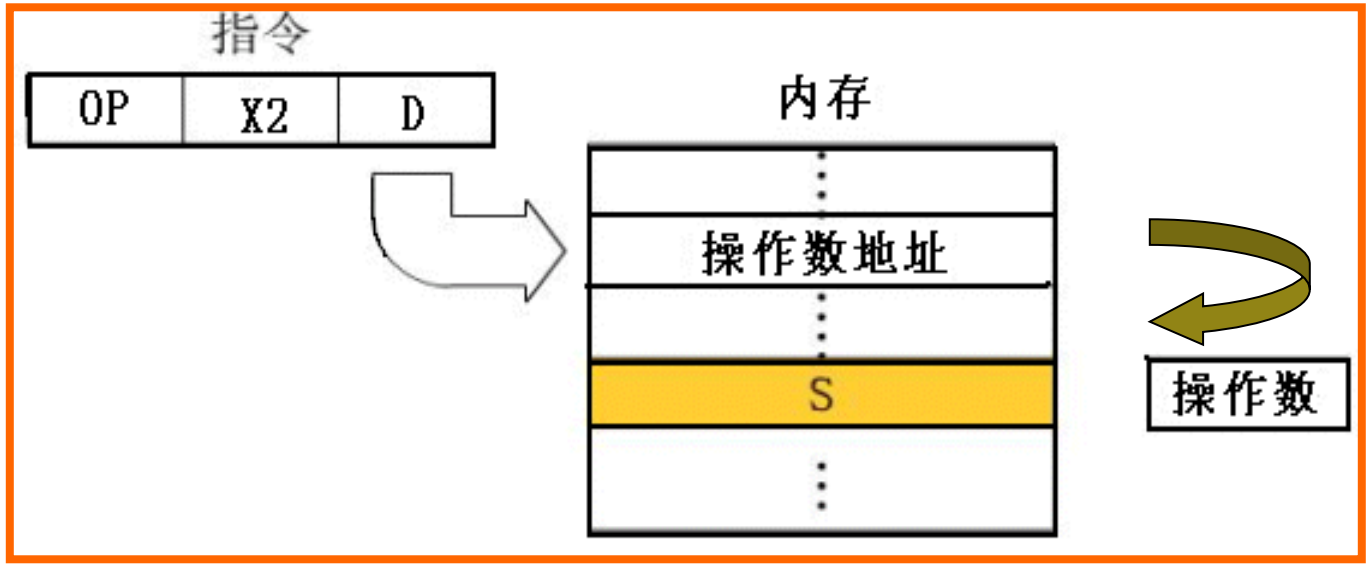
# 操作数寻址（直接寻址）

- 由指令的地址码部分直接给出操作数的有效地址
  - 操作数为（D）
  - 如：　　MOV AX,　[2038H]
- 不足
  - 寻址范围受D的位数的限制



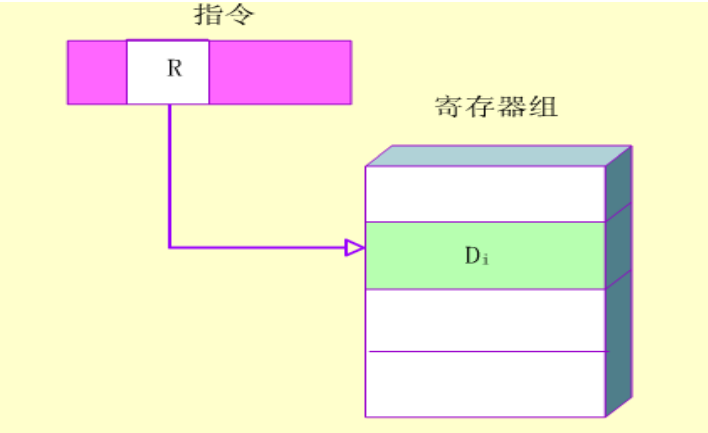
# 操作数寻址（间接寻址）

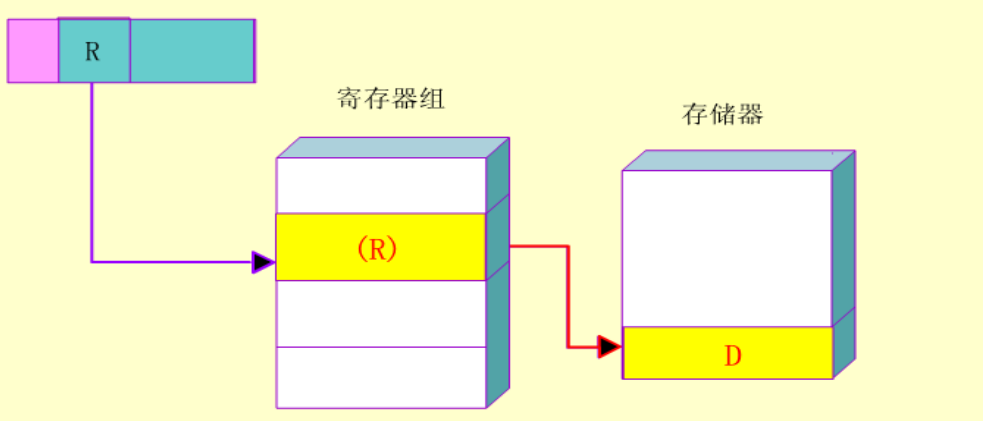
- 指令的地址码部分给出的是操作数地址的地址
  - 操作数为E=((D))
- 优点：
  - 寻址的范围不再受D的位数限制
- 不足：
  - 增加了访问内存的次数，降低了指令执行速度



# 操作数寻址（寄存器寻址）

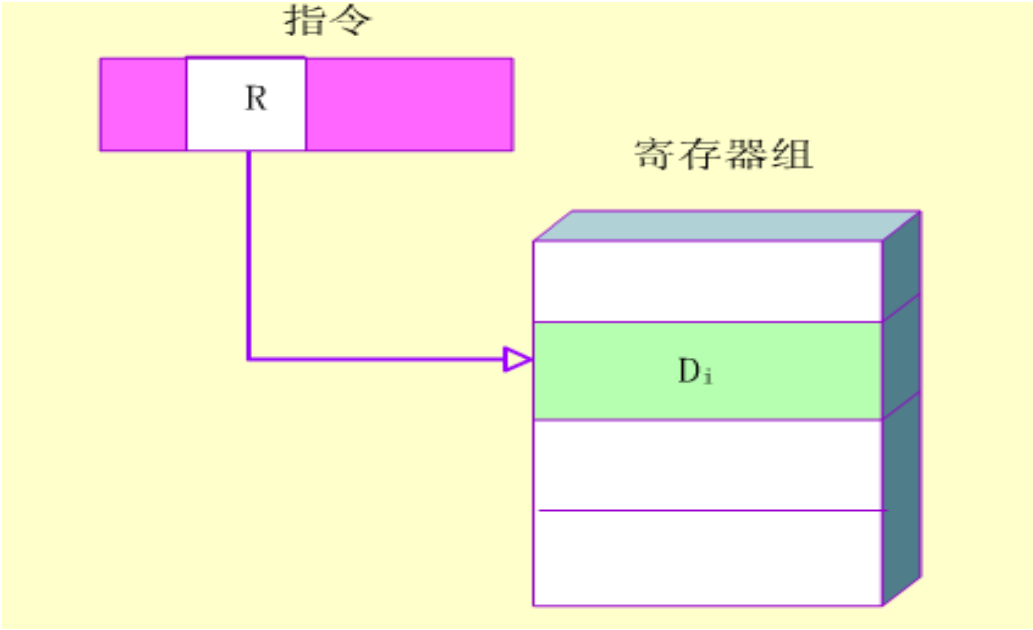
- 包含寄存器直接寻址和寄存器间接寻址
  - 操作数的值或者地址就地址码R对应的寄存器的值



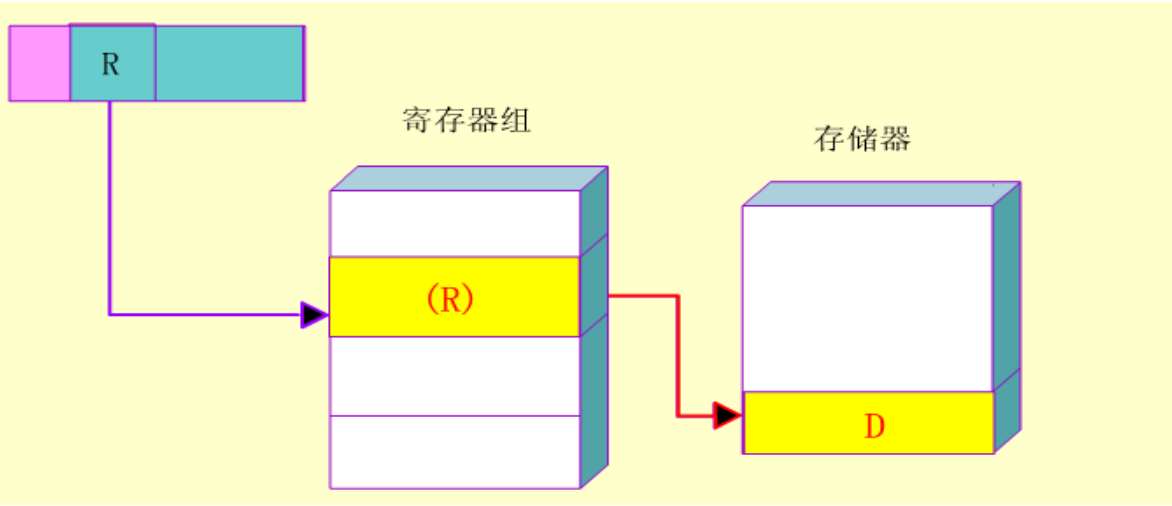
# 数寻址（寄存器寻址—寄存器(直接)寻址）

- 操作数不放在内存中，而是放在通用寄存器中，此时指令的地址码字段R的值表示的是寄存器号
  - 操作数为（R）



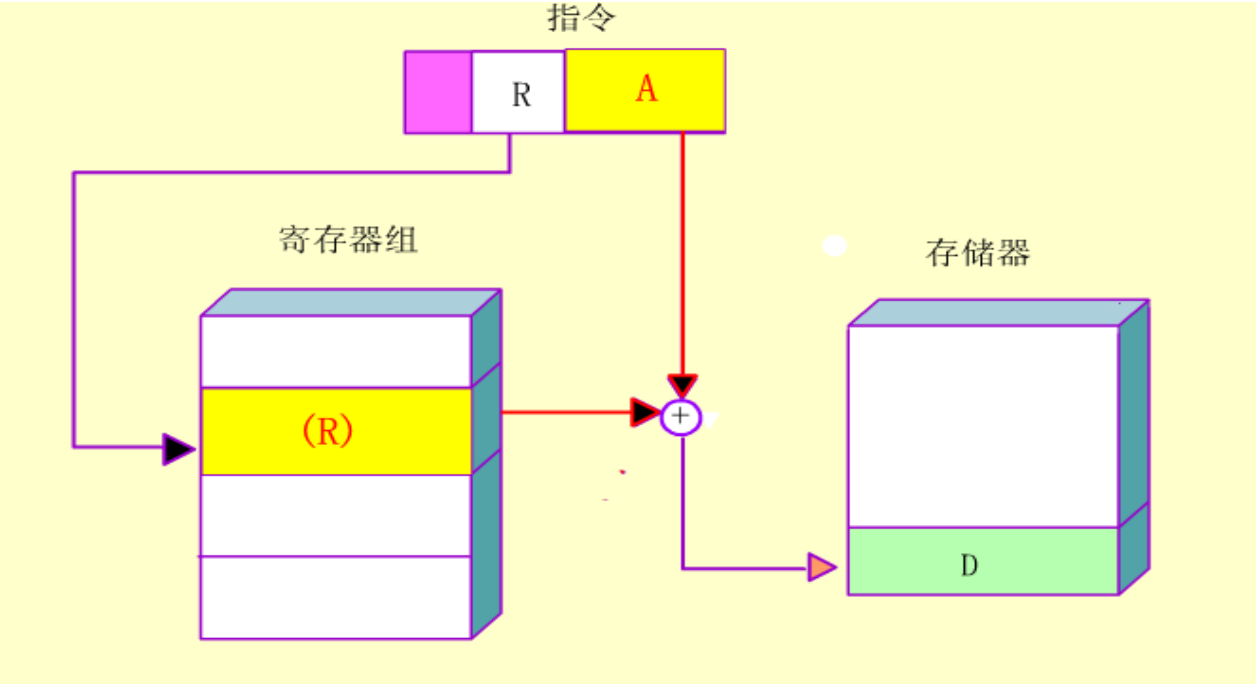
# 操作数寻址（寄存器寻址—寄存器间接寻址）

- 寄存器的值不是操作数，而是操作数所在内存单元的地址
  - 操作数（( R )）



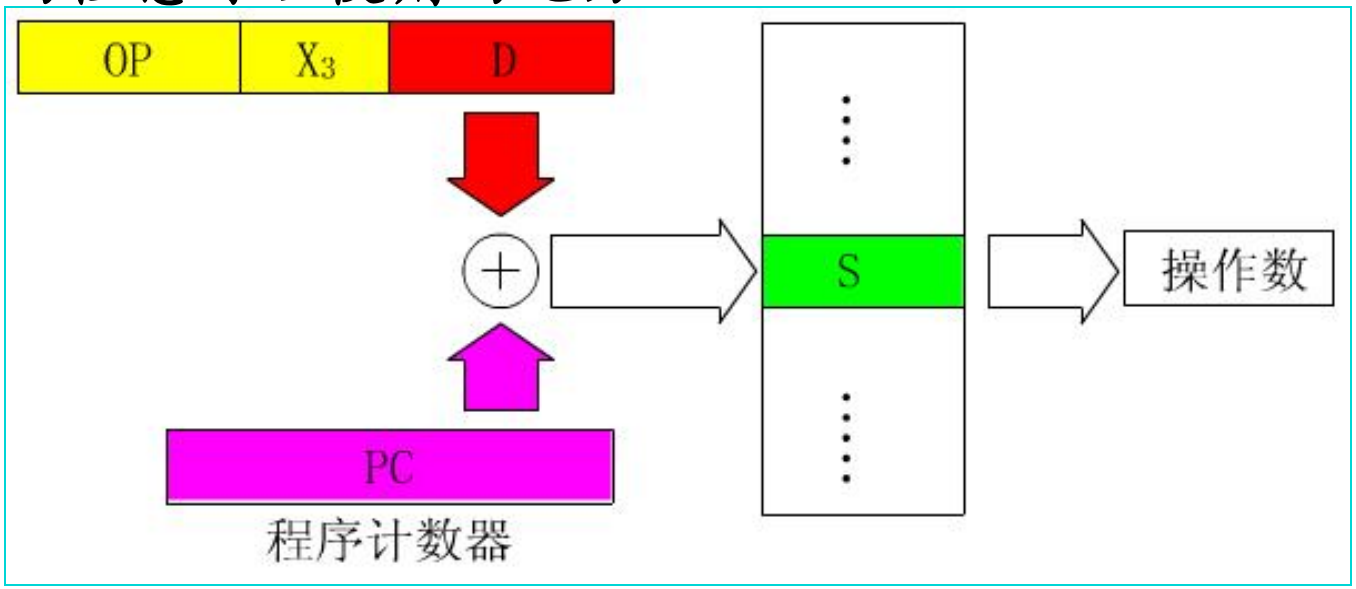
# 操作数寻址（偏移寻址）

- 包含基址寻址、变址寻址和相对寻址
  - 操作数的地址需要通过地址码D的值（偏移量）加上基准寄存器R的值的出
  - R可以明显给出，也可以隐含给出
  - 常见的寄存器为PC、基址寄存器B、变址寄存器



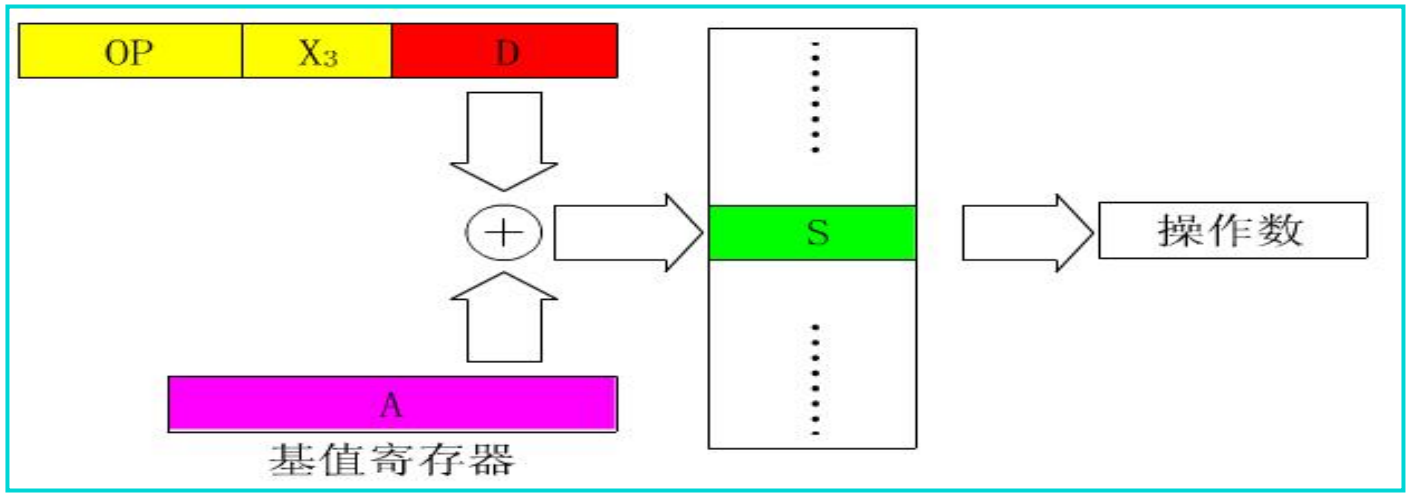
# 操作数寻址（偏移寻址—相对寻址）

- 当前指令执行位置PC的值和地址码D的值相加，作为有效地址。
  - PC的值是当前指令的还是下一条指令的?
  - 操作数为 ((PC)+D)
- 特点
  - 程序员可以使用相对地址编程,所编制的程序可以放在内存的任意可以使用的地方



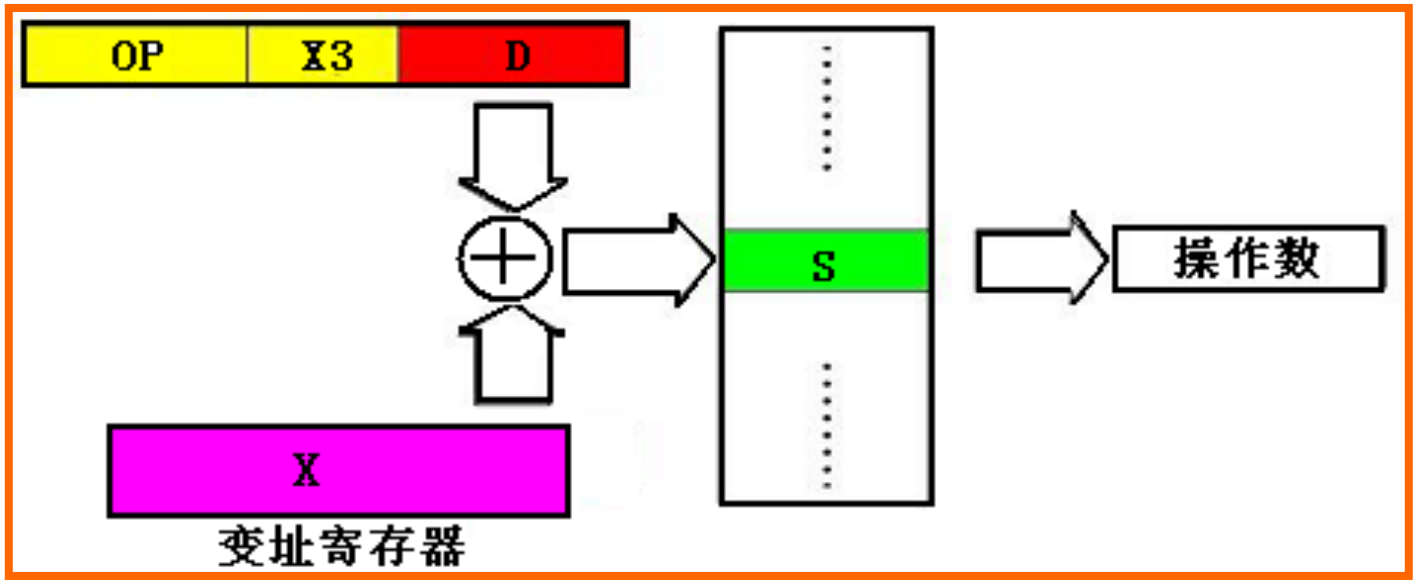
# 操作数寻址（偏移寻址—基址寻址）

- 基址寄存器的值和地址码D的值相加，作为有效地址（基址寄存器中的值不变，D中的值可变）
  - 操作数为 ((BX)+D)
- 特点：
  - 基址寄存器的位数可以设得很长，因而，可以扩大寻址能力



# 操作数寻址（偏移寻址—变址寻址）

- 变址寄存器的值和地址码D的值相加，作为有效地址（变址寄存器中的值可变，D中的值不变）
- 操作数为((X)+D)
- 特点
  - 不是为了扩大寻址空间，而是为了实现程序的有规律的浮动，而不改变指令.　如循环

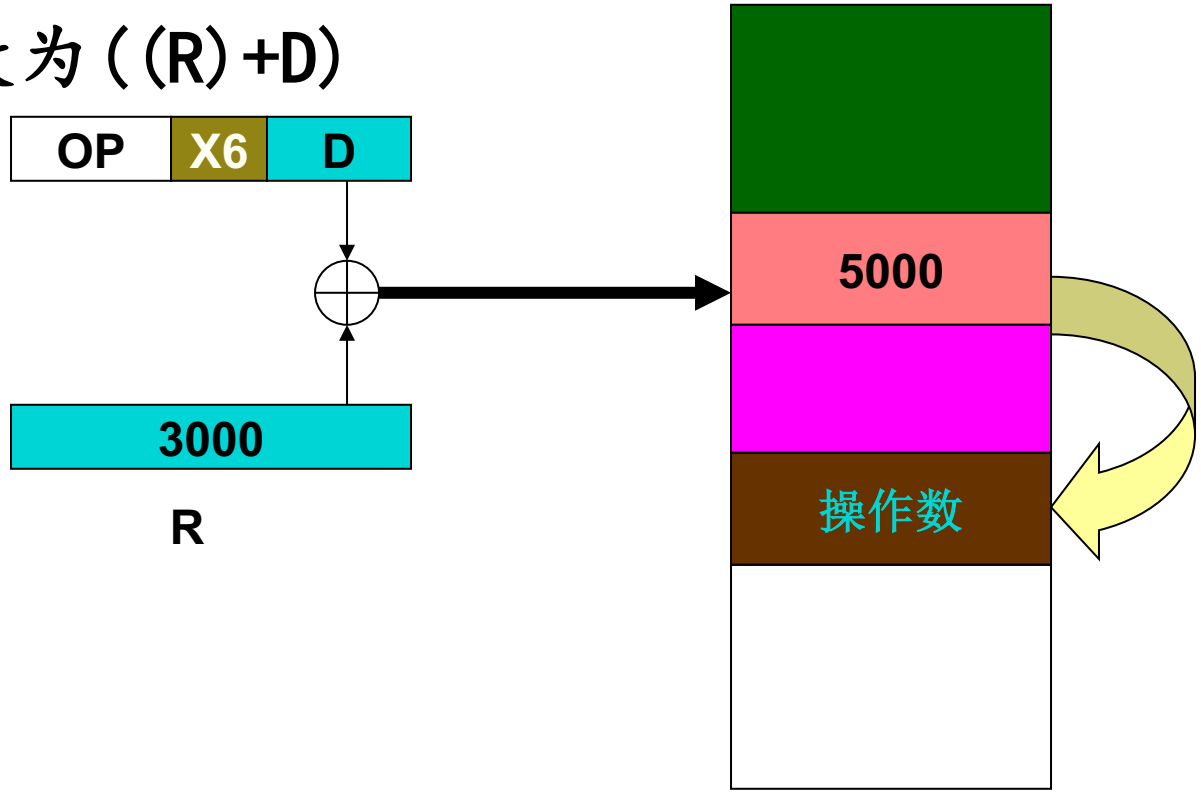


# 操作数寻址（复合寻址）

- 间址、相对、变址和基值等寻址方式组合
- 包含变址间址、间址变址和相对间址

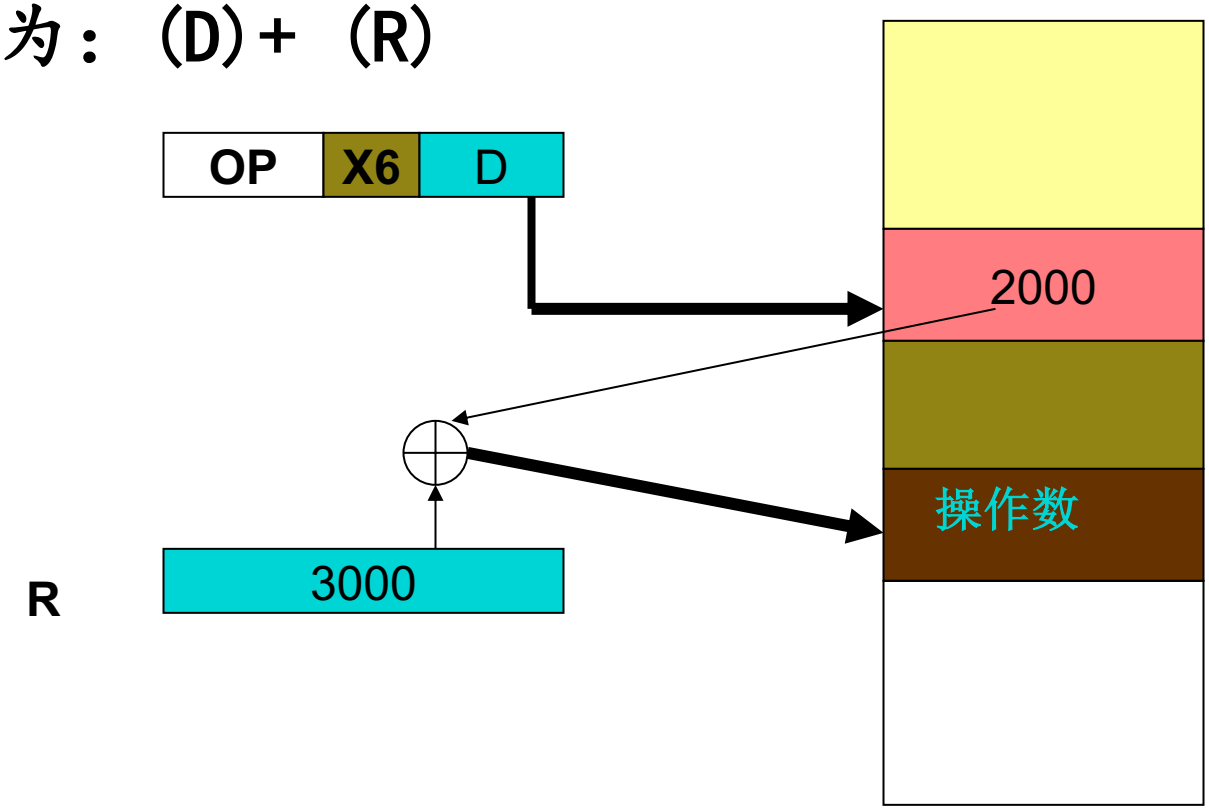
# 操作数寻址（复合寻址——变址间址）

- 变址间址
  - 先变址，后间址
  - 操作数为((R)+D)



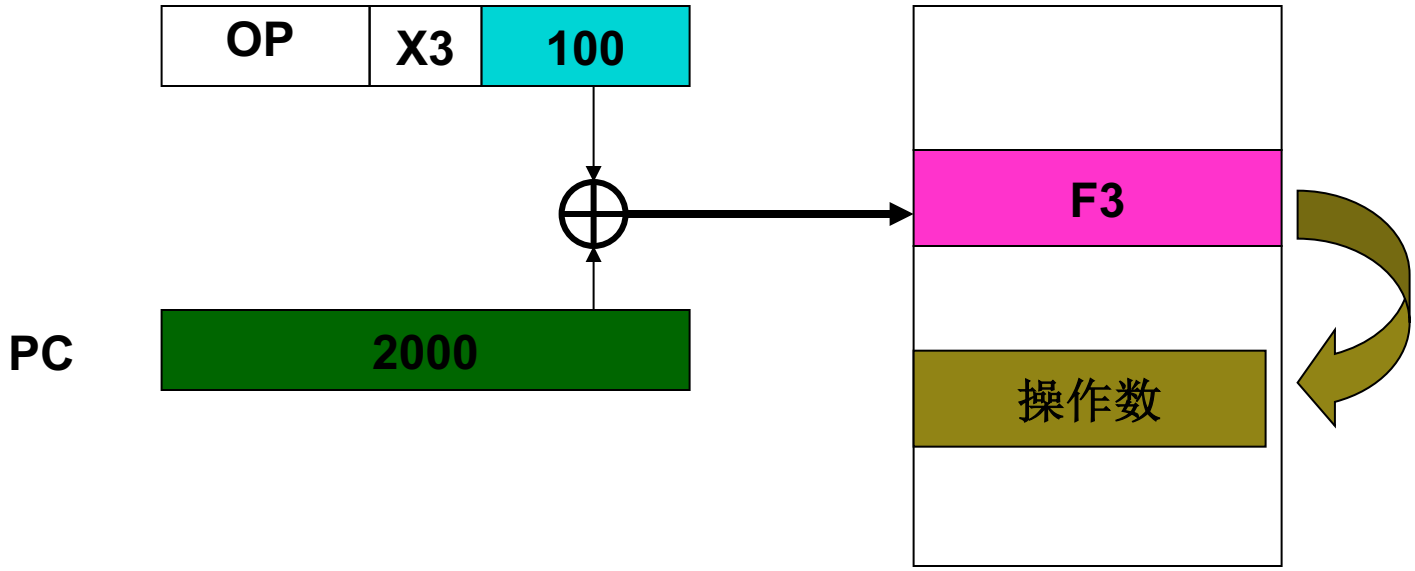
# 操作数寻址（复合寻址——间址变址）

- 间址变址
  - 先间址，后变址
  - 操作数为：(D)+ (R)

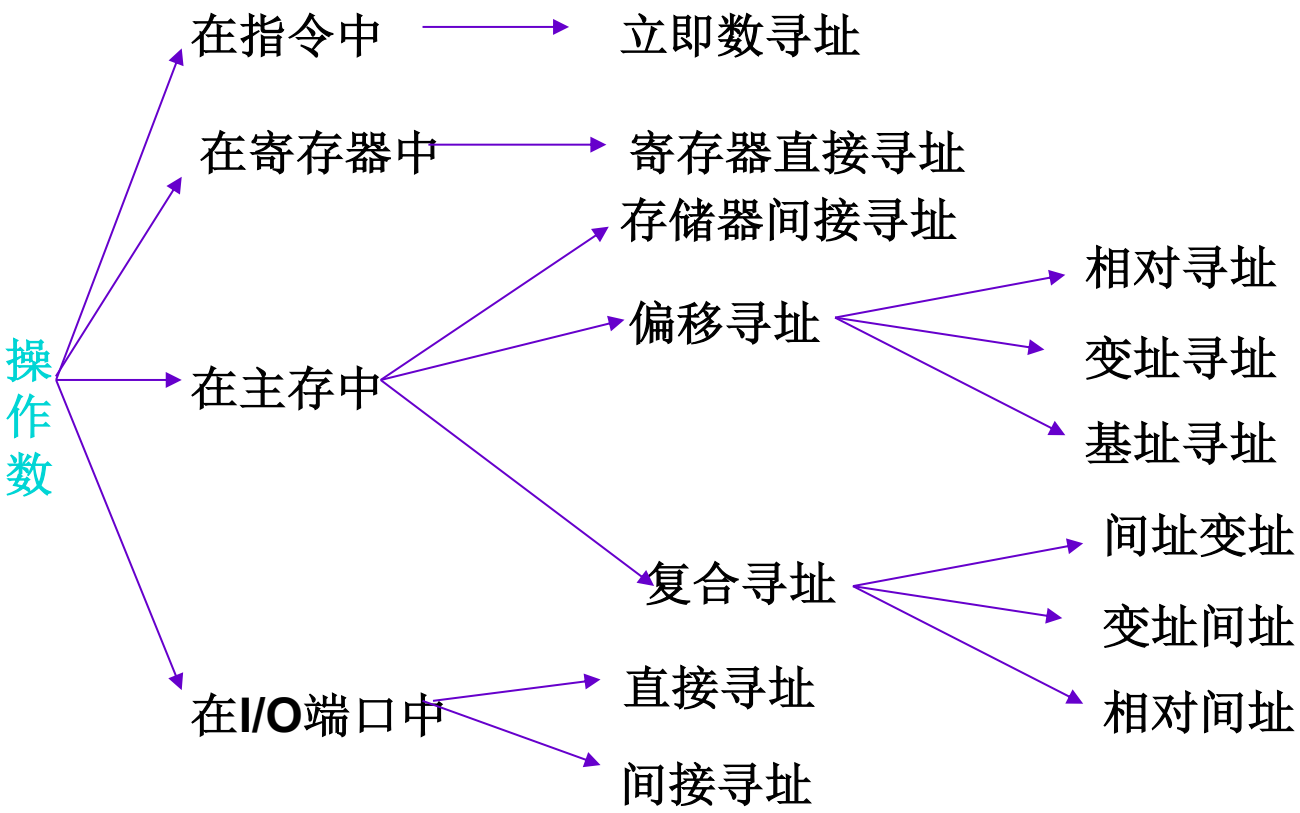


# 操作数寻址（复合寻址——相对间址）

- 相对间址
  - 先相对寻址，后间接寻址
  - 操作数为：（ (PC)+ D ）



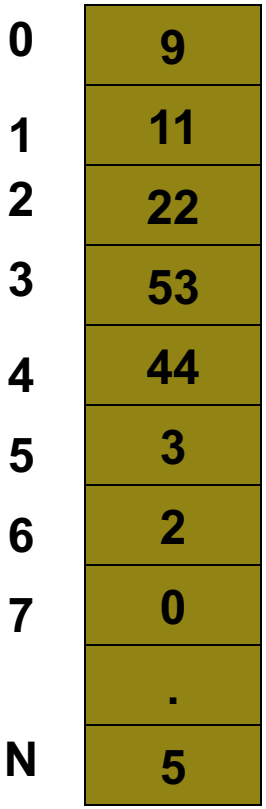
# 操作数寻址


操作数
在指令中
立即数寻址
在寄存器中
寄存器直接寻址
在主存中
存储器间接寻址
偏移寻址
相对寻址
变址寻址
基址寻址
复合寻址
间址变址
变址间址
相对间址
在I/O端口中
直接寻址
间接寻址

# 操作数寻址实例（1）

主存数据分布如图所示,若A为单元地址 (A)为A的内容,求

(( 7 )) - (N) + (( N ))+ ((( N )))的值




(( 7 )) = 9

( N )= 5

(( N )) = 3

((( N )))=53

结果 = 60

# 操作数寻址实例（2）

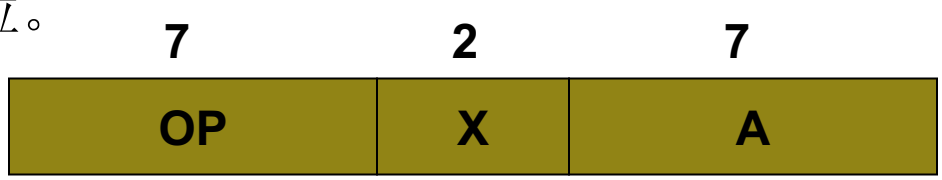
字长16位，主存64K，指令单字长单地址码，80条指令。寻址方式有直接、间接、相对、变址。请设计指令格式

解：　80条指令　⇒　OP字段需要7位( $2^7$=128 )

4种寻址方式 ⇒ 寻址方式特征位2位

16位字长指令还余7位作为地址位，由于是单地址指令，所以地址位的长度也是7位。

指令格式如下：




PC为16位　　变址寄存器16位

- 相对寻址　E=（PC）+D ,寻址范围为：64K
- 变址寻址　E=（R）+D，　寻址范围为：64K
- 直接寻址 E = D，寻址范围为128
- 间接寻址 E = （D），　寻址范围为64K

# 问题和讨论