7.3 寻址方式

为什么要有多种寻址方式?

寻址方式　确定 本条指令 的 操作数地址

下一条 要执行 指令 的 指令地址

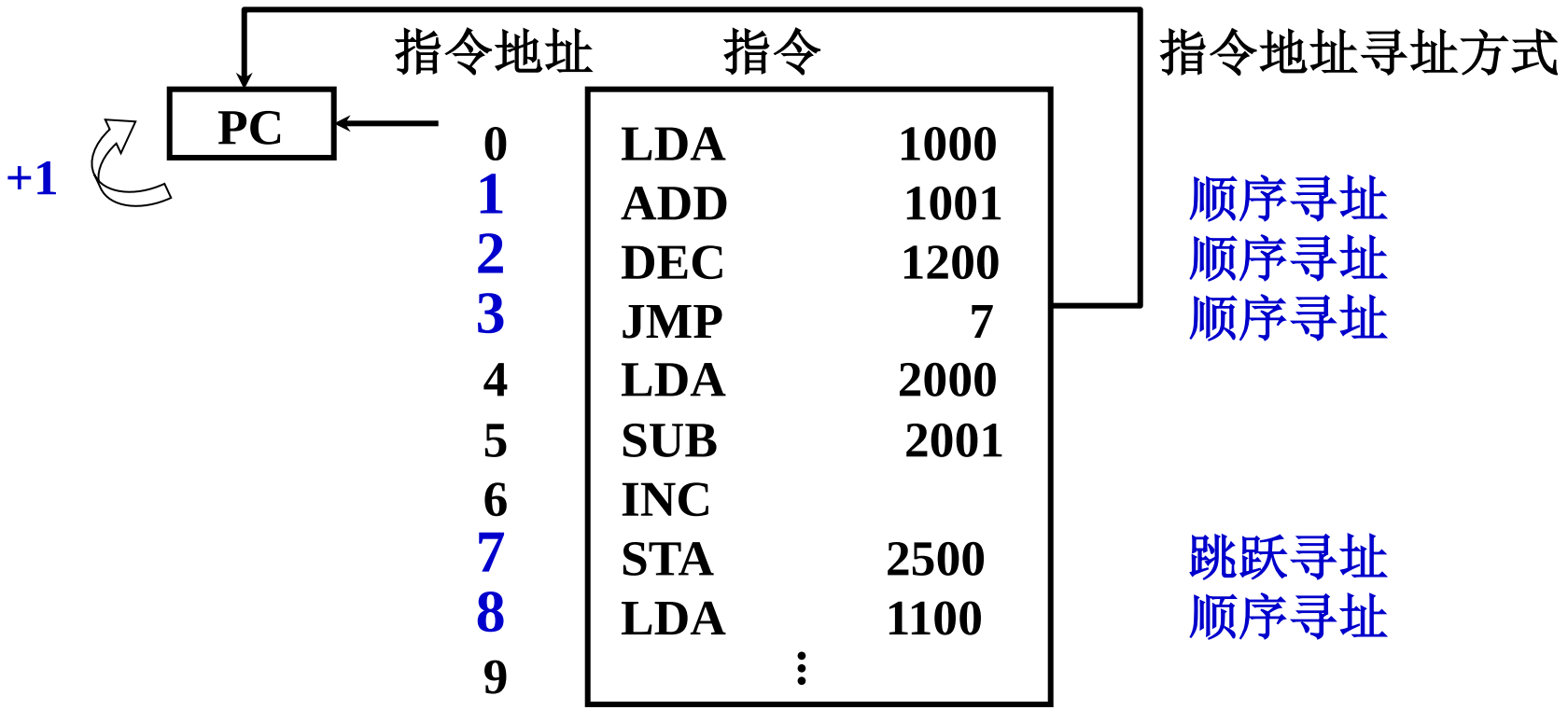
寻址方式
- 指令寻址
- 数据寻址

7.3 寻址方式

一、指令寻址

顺序　　(PC) + 1 ⟶ PC

跳跃　　由转移指令指出



二、数据寻址 7.3

操作码	寻址特征	形式地址 A

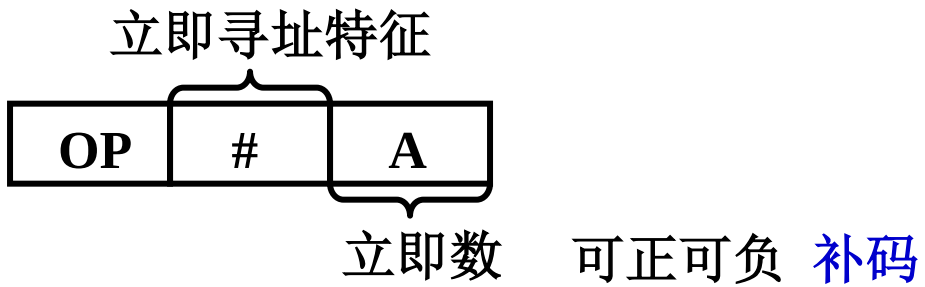
形式地址　　指令字中的地址　不是数据所在的真实地址，需和寻址特征结合求得有效地址

有效地址　　操作数的真实地址

约定　指令字长 = 存储字长 = 机器字长

1. 立即寻址

形式地址 A 就是操作数

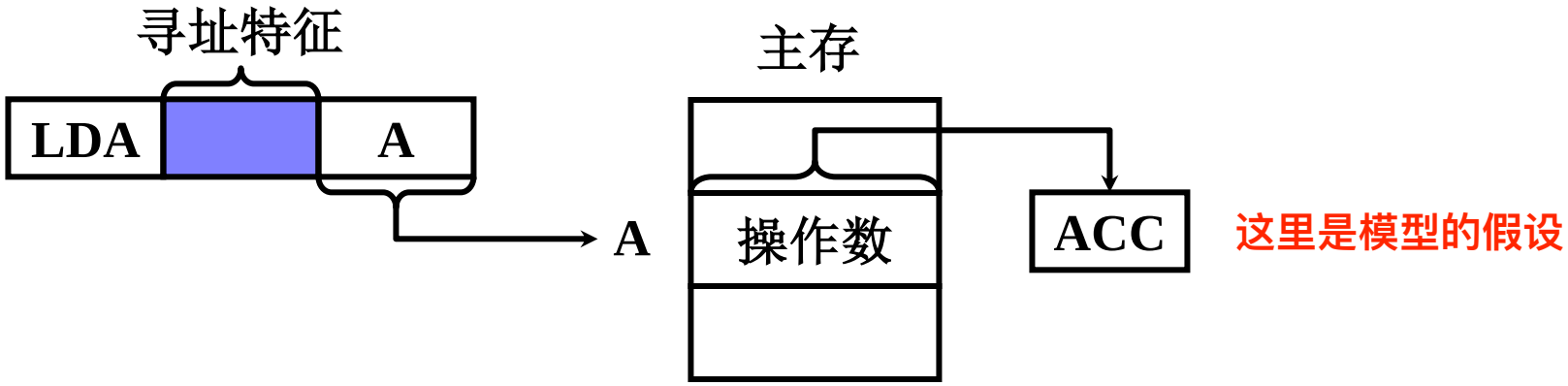


- 指令执行阶段不访存　操作数在取指令时，已经被取到了CPU中
- A 的位数限制了立即数的范围

2. 直接寻址

7.3

EA = A　　有效地址由形式地址直接给出

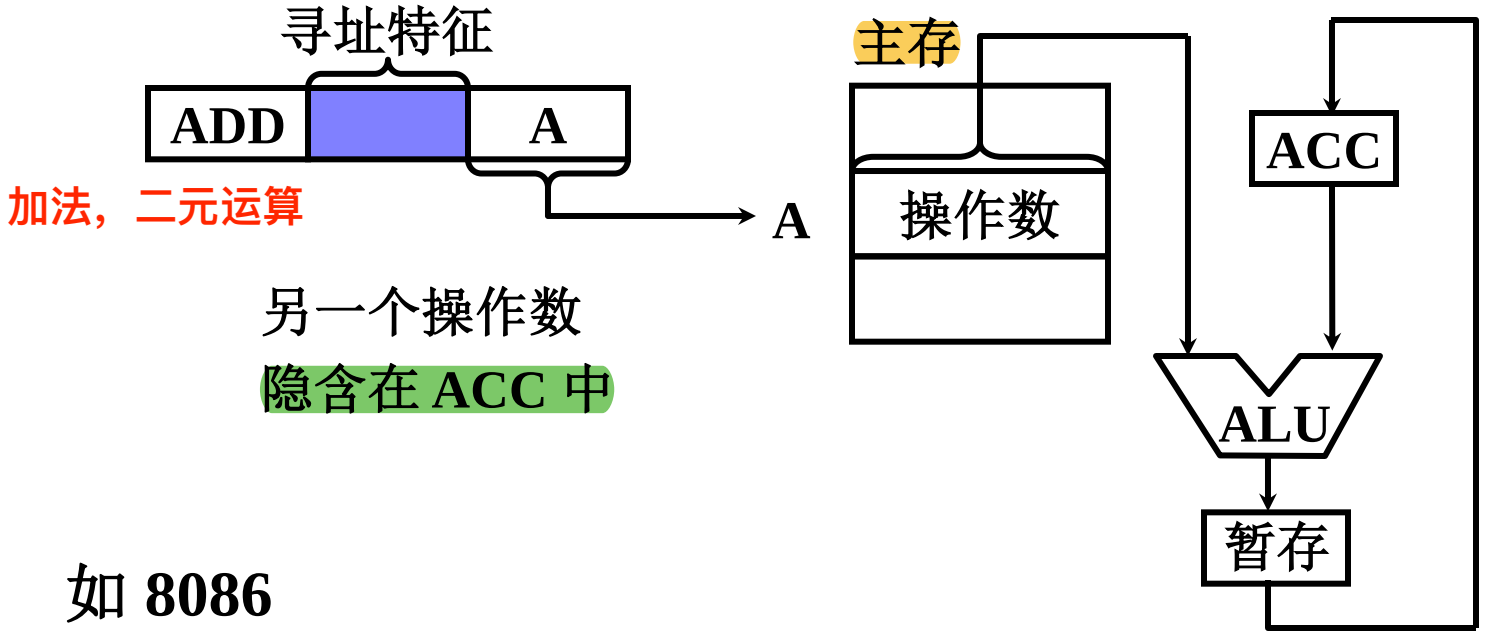


- 执行阶段访问一次存储器
- A 的位数决定了该指令操作数的寻址范围
- 操作数的地址不易修改（必须修改A）

3. 隐含寻址 去约定好的位置找

7.3

操作数地址隐含在操作码中



如 8086

MUL 指令　被乘数隐含在 AX（16位）或 AL（8位）中

MOVS 指令　源操作数的地址隐含在 SI 中

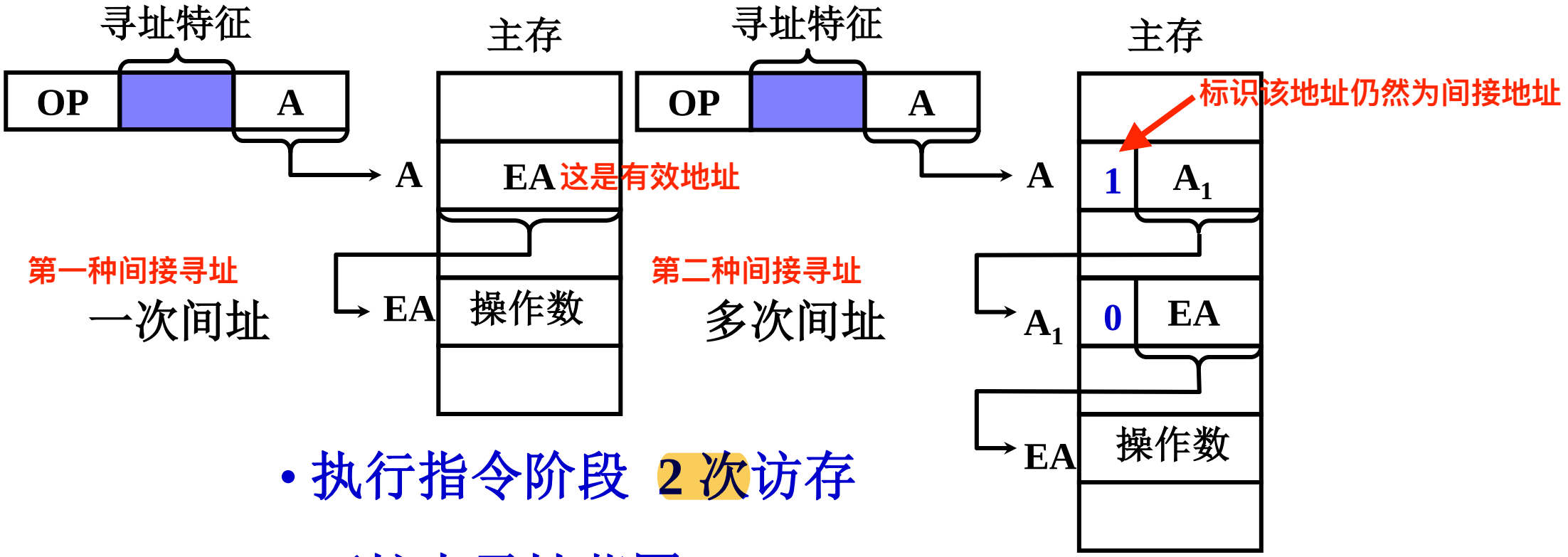
目的操作数的地址隐含在 DI 中

- 指令字中少了一个地址字段，可缩短指令字长

4. 间接寻址

7.3

EA =（A） 有效地址由形式地址间接提供

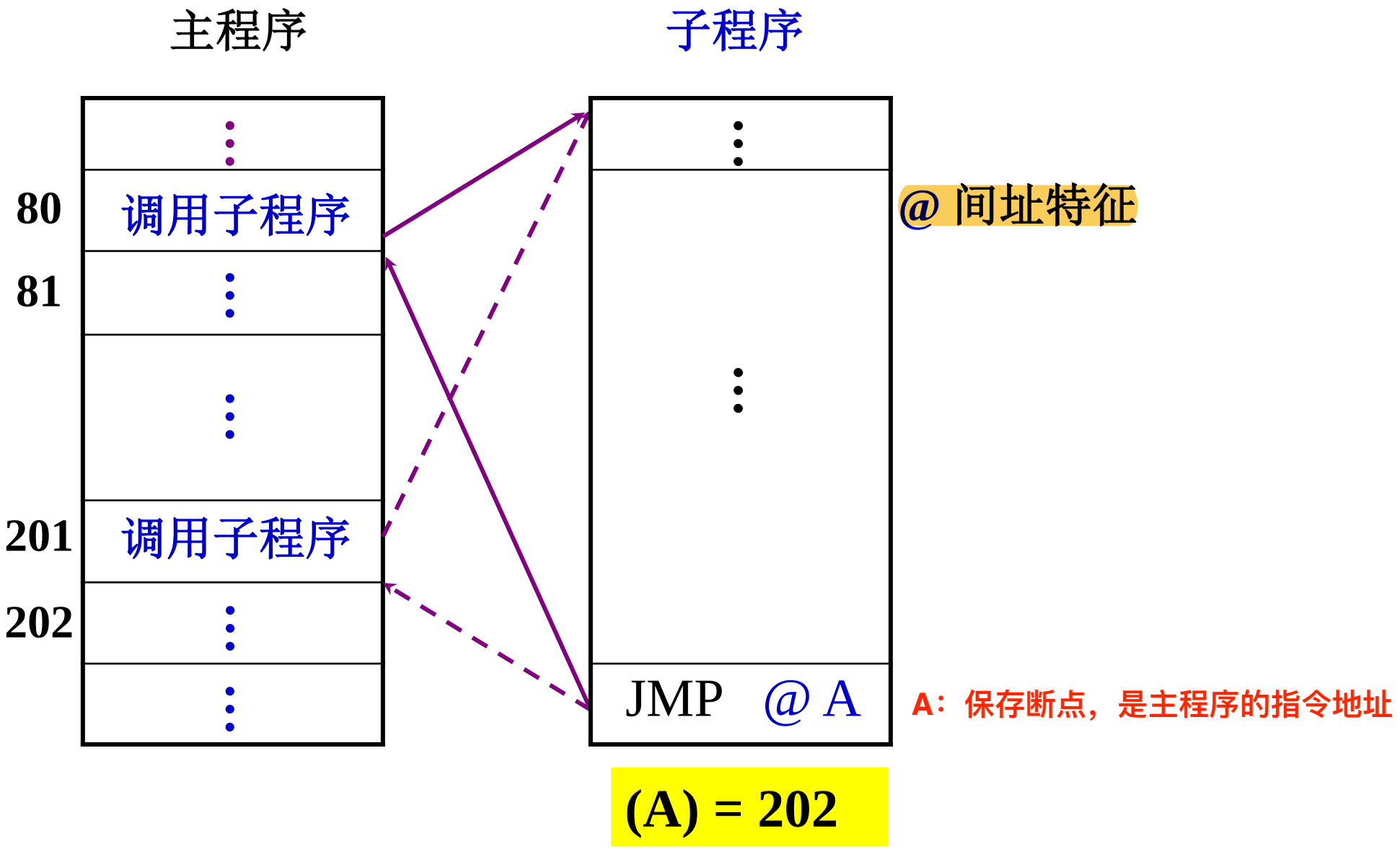


- 执行指令阶段 2 次访存
- 可扩大寻址范围
- 便于编制程序

多次访存

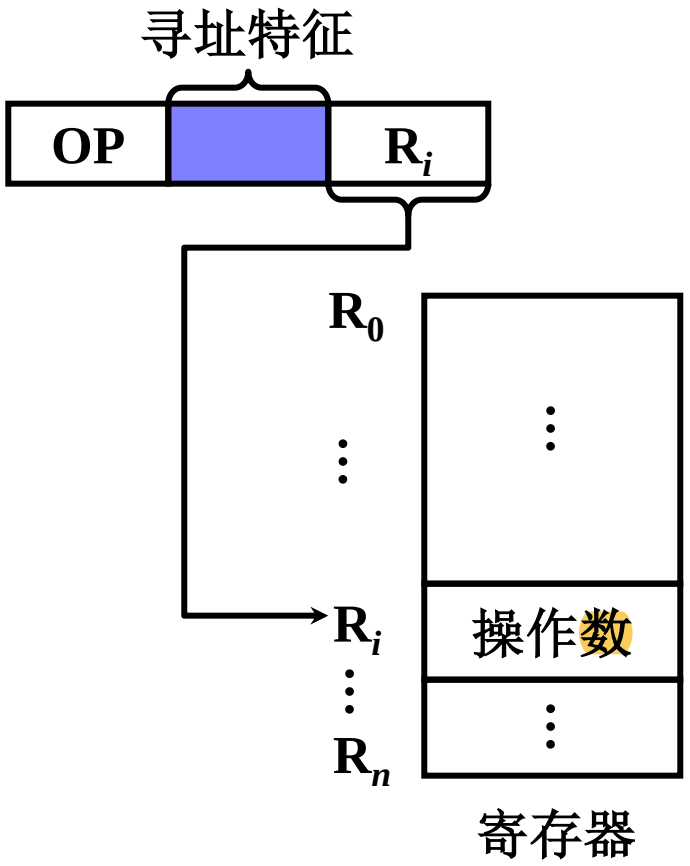
间接寻址编程举例

7.3


主程序
子程序
80
调用子程序
81
201
调用子程序
202
JMP @ A
@ 间址特征
A：保存断点，是主程序的指令地址
(A) = 202

5. 寄存器寻址 （寄存器直接寻址）

$EA = R_i$　　有效地址即为寄存器编号

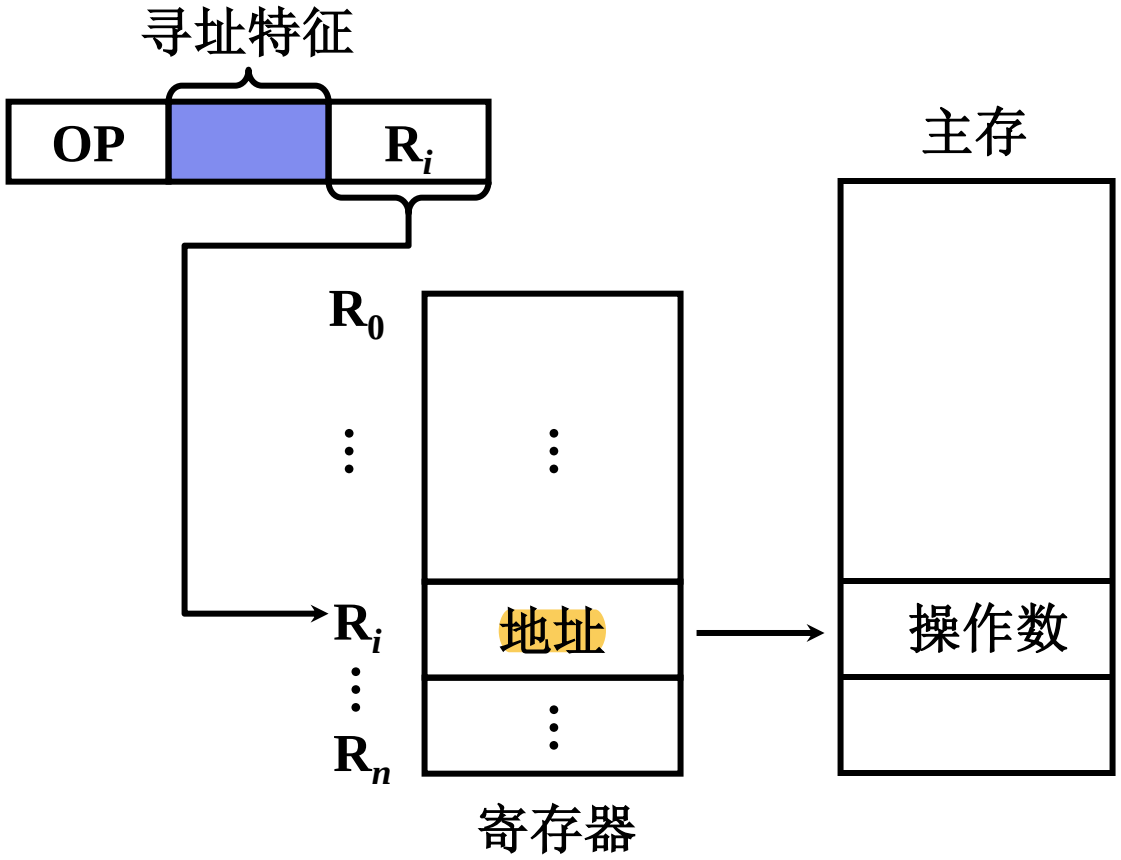


- 执行阶段不访存，只访问寄存器，执行速度快
- 寄存器个数有限，可缩短指令字长

6. 寄存器间接寻址

7.3

$EA = (R_i)$　　有效地址在寄存器中



- 有效地址在寄存器中， 操作数在存储器中，执行阶段访存
- 便于编制循环程序

7. 基址寻址 主要用于多道程序设计

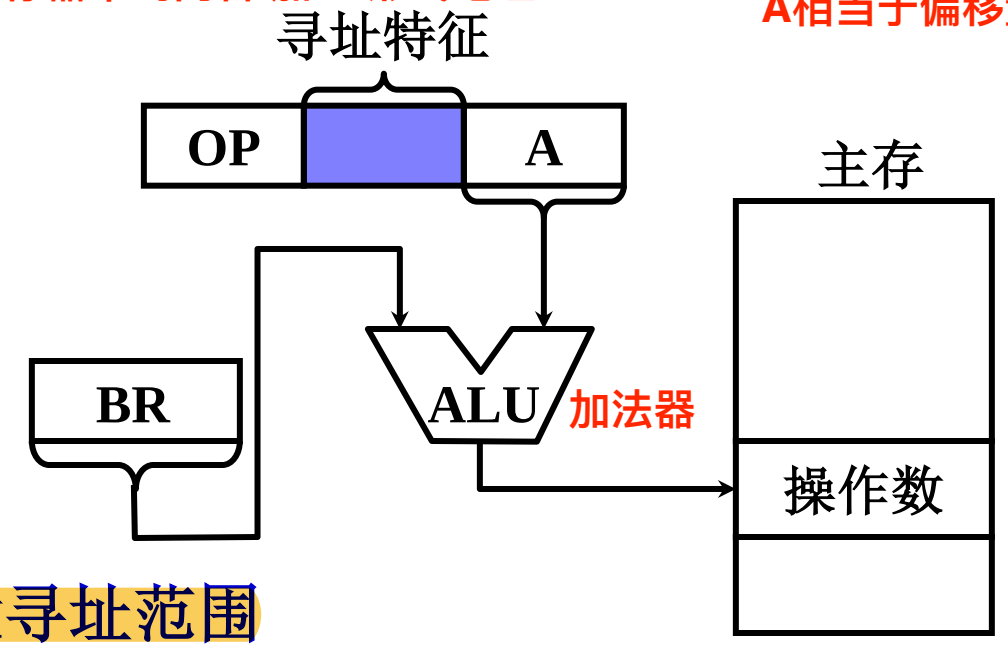
7.3

(1) 采用专用寄存器作基址寄存器

EA = (BR) + A　　**BR 为基址寄存器**

有效地址 = 专用寄存器中的内容 加上 形式地址

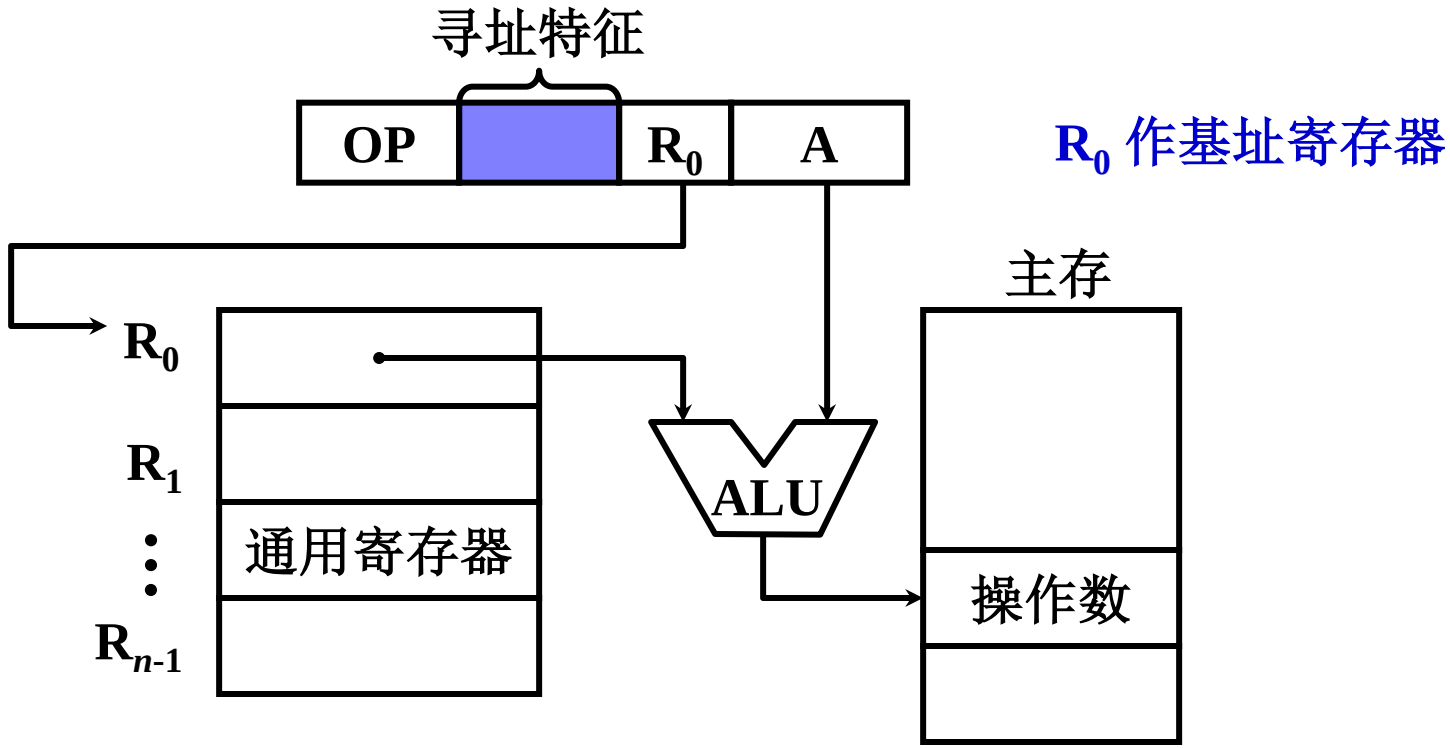
A相当于偏移量



- 可扩大寻址范围
- 有利于多道程序
- **BR** 内容由操作系统或管理程序确定
- 在程序的执行过程中 **BR** 内容不变，形式地址 **A** 可变

(2) 采用通用寄存器作基址寄存器

7.3



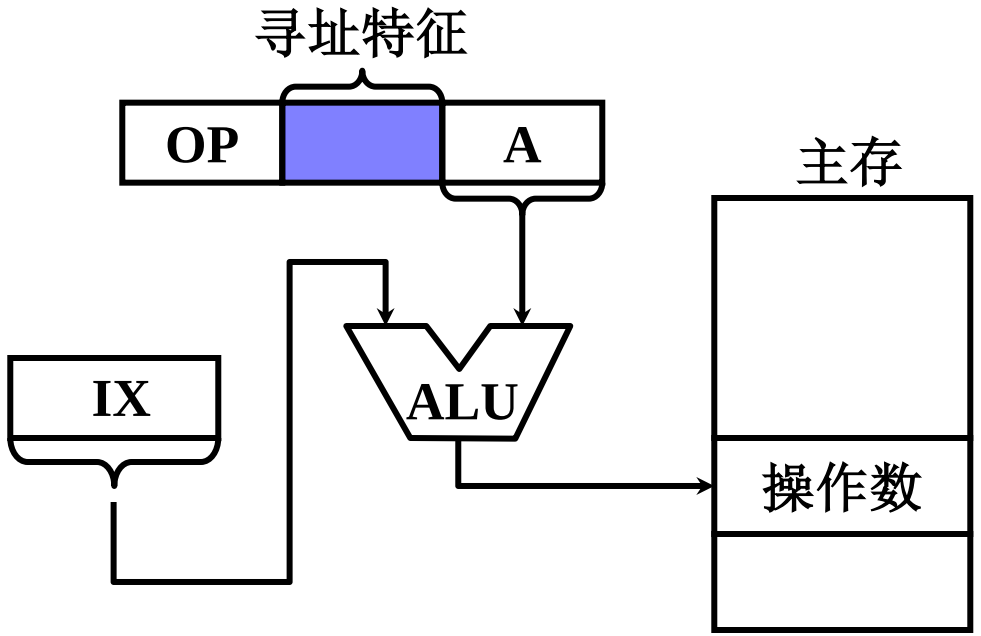
- 由用户指定哪个通用寄存器作为基址寄存器
- 基址寄存器的内容由操作系统确定　用户不能修改
- 在程序的执行过程中 R_0 内容不变，形式地址 A 可变

8. 变址寻址

7.3

EA = (IX) +A　　IX 为变址寄存器（专用）

通用寄存器也可以作为变址寄存器



- 可扩大寻址范围
- IX 的内容由用户给定　所以叫“变”址
- 在程序的执行过程中 IX 内容可变，形式地址 A 不变　与基址寻址相反
- 便于处理数组问题

例　设数据块首地址为 D，求 *N* 个数的平均值　7.3

直接寻址

```
LDA    D
ADD    D + 1
ADD    D + 2          ┌→ M
  ⋮                   |
ADD    D + ( N -1 )   ┘
DIV    # N        立即寻址
STA    ANS        保存结果
```

共 *N* + 2 条指令

变址寻址

```
LDA    # 0        清零ACC
LDX    # 0        X 为变址寄存器   清零IX
M  ADD    X, D    D 为形式地址
   INX             (X) +1→ X
   CPX    # N      (X) 和 #N 比较
   BNE    M        结果不为零则转
DIV    # N
STA    ANS
```

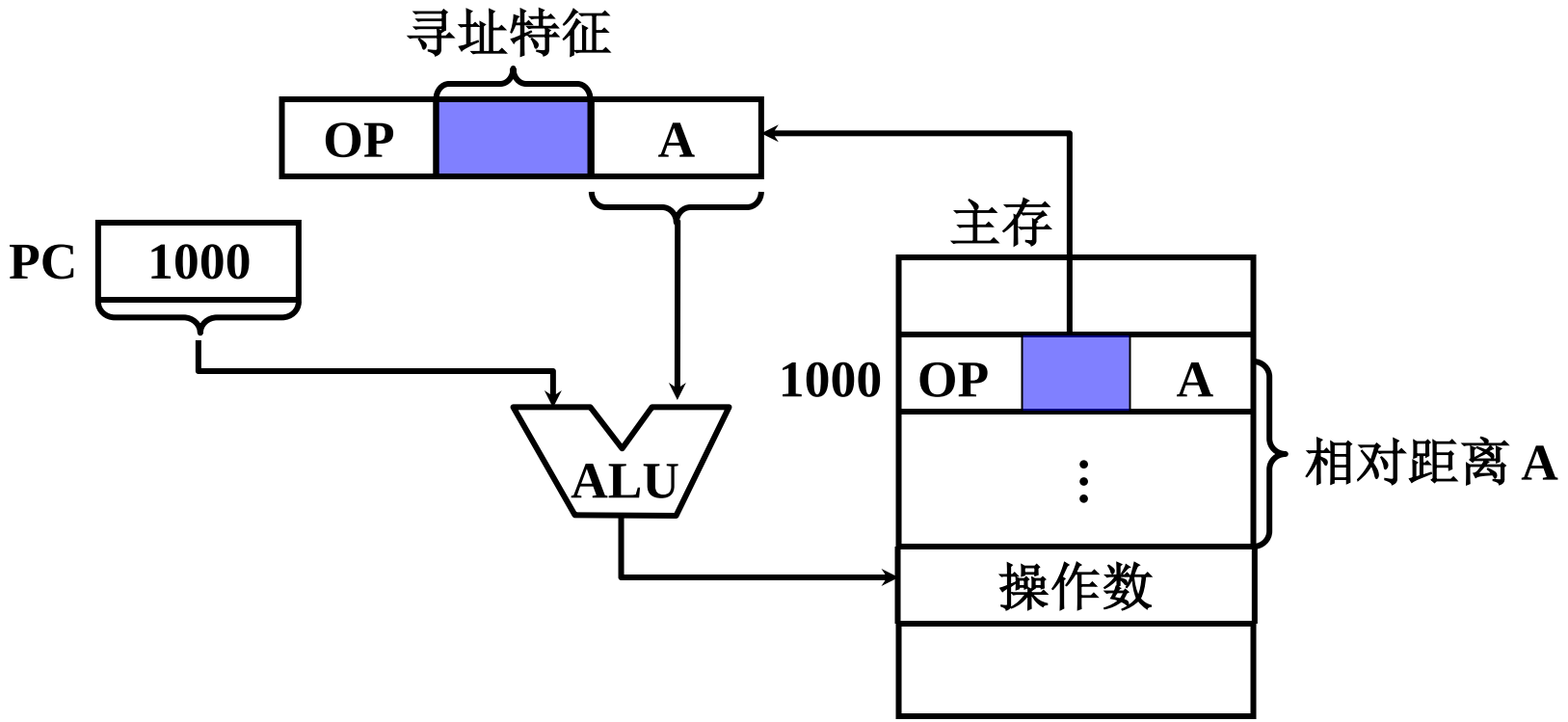
共 8 条指令

9. 相对寻址

7.3

EA = (PC) + A

A 是相对于当前指令的位移量（可正可负，补码）



- A 的位数决定操作数的寻址范围
- 程序浮动
- 广泛用于转移指令

(1) 相对寻址举例

N个数据相加，求平均值

7.3

```
        LDA    # 0
        LDX    # 0
M       ADD    X, D
M+1     INX
M+2     CPX    # N
M+3     BNE    M        ──→  * – 3
        DIV    # N
        STA    ANS
```

* 相对寻址特征

M 随程序所在存储空间的位置不同而不同

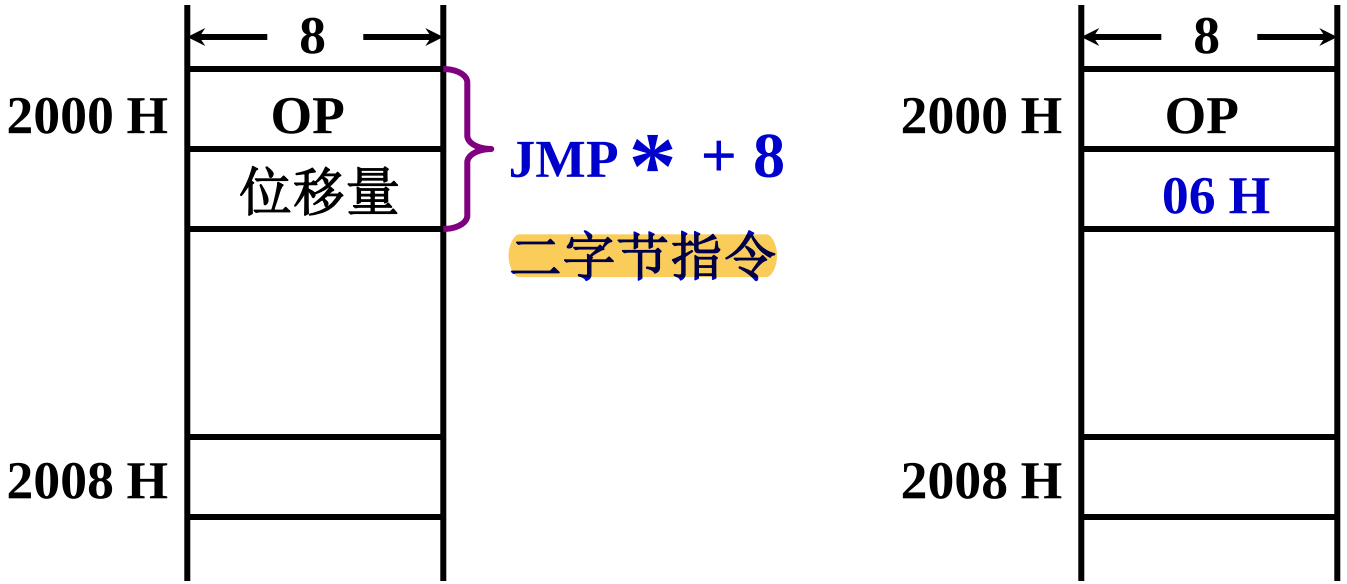
而指令 BNE *– 3 与 指令 ADD X, D 相对位移量不变

指令 BNE *– 3 操作数的有效地址为

EA = (M+3) – 3 = M

(2) 按字节寻址的相对寻址举例

7.3



设 当前指令地址 **PC = 2000H**

转移后的目的地址为 **2008H**

因为 取出 **JMP * + 8** 后 **PC = 2002H**

故 **JMP * + 8** 指令 的第二字节为 **2008H - 2002H = 06H**

10. 堆栈寻址

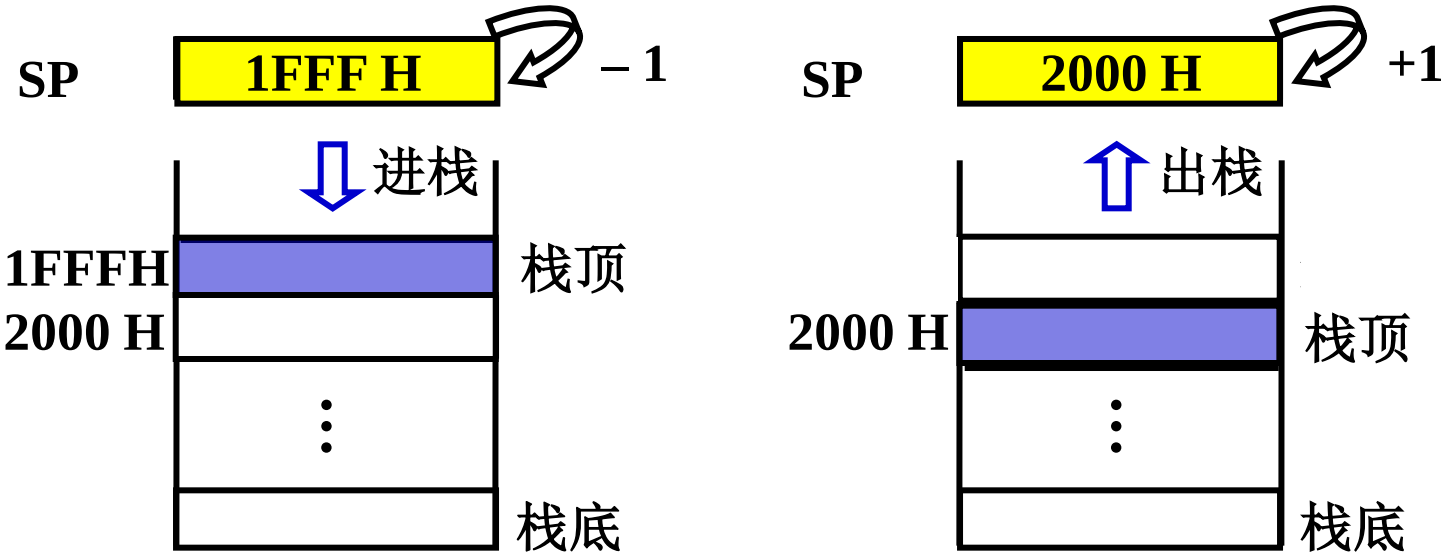
7.3

(1) 堆栈的特点

堆栈 { 硬堆栈　多个寄存器
　　 软堆栈　指定的存储空间

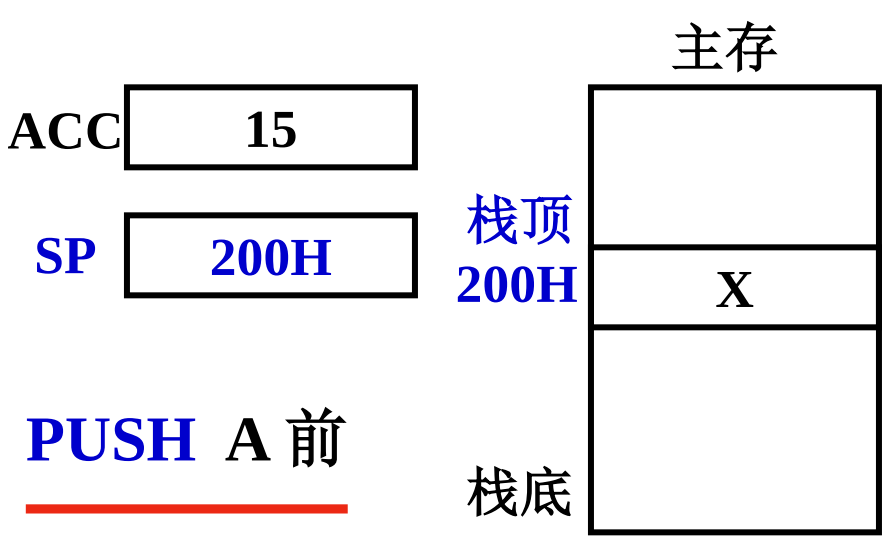
先进后出（一个入出口）　栈顶地址 由 SP 指出

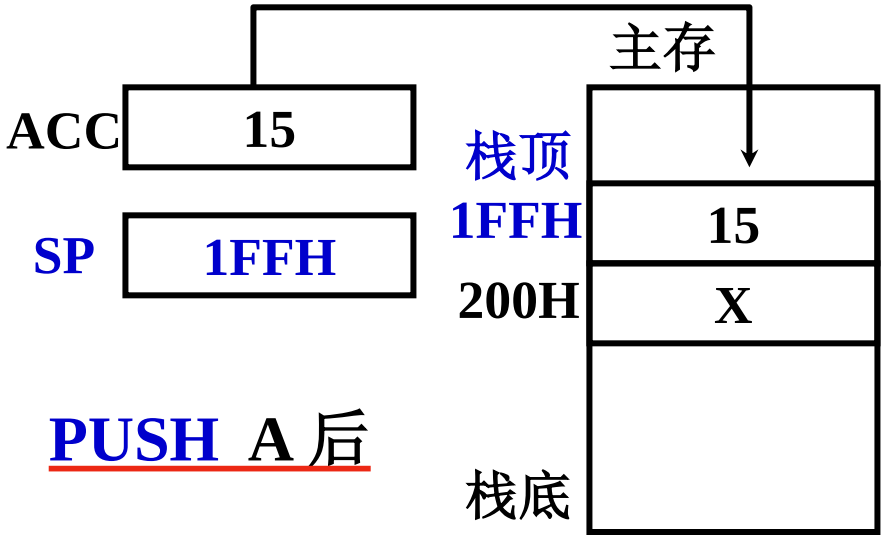
进栈　（SP）– 1 ⟶ SP　　出栈　（SP）+ 1 ⟶ SP

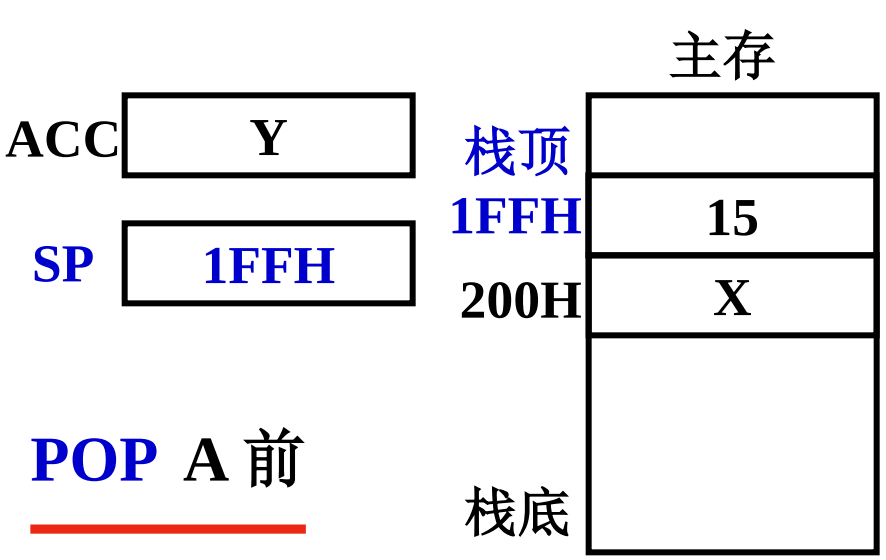


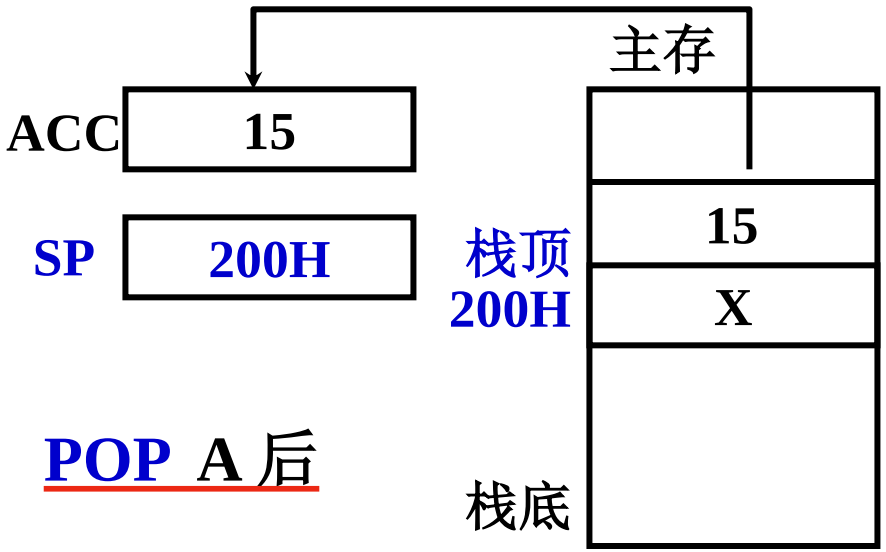
(2) 堆栈寻址举例

7.3


主存
ACC
15
SP
200H
栈顶
200H
X
栈底
PUSH A 前


主存
ACC
15
SP
1FFH
栈顶
1FFH
15
200H
X
栈底
PUSH A 后


主存
ACC
Y
SP
1FFH
栈顶
1FFH
15
200H
X
栈底
POP A 前


主存
ACC
15
SP
200H
栈顶
200H
15
X
栈底
POP A 后

(3) SP 的修改与主存编址方法有关

7.3

① 按 字 编址

进栈　（SP）– 1 ⟶ SP

出栈　（SP）+ 1 ⟶ SP

② 按 字节 编址

一个存储字占两个字节

存储字长 16 位　进栈　（SP）– 2 ⟶ SP

出栈　（SP）+ 2 ⟶ SP

存储字长 32 位　进栈　（SP）– 4 ⟶ SP

出栈　（SP）+ 4 ⟶ SP