计算机组织与体系结构

第六讲

计算机科学与技术学院

舒燕君

Recap

- 总线的概念
  - ✓ 定义、分类
  - ✓ 特性、性能指标
  - ✓ 总线标准
  - ✓ 多总线结构
- 总线的控制
  - ✓ 总线的判优控制
  - ✓ 总线的通信控制

第4章 指令系统

4.1 机器指令

4.2 操作数类型和操作类型

4.3 寻址方式

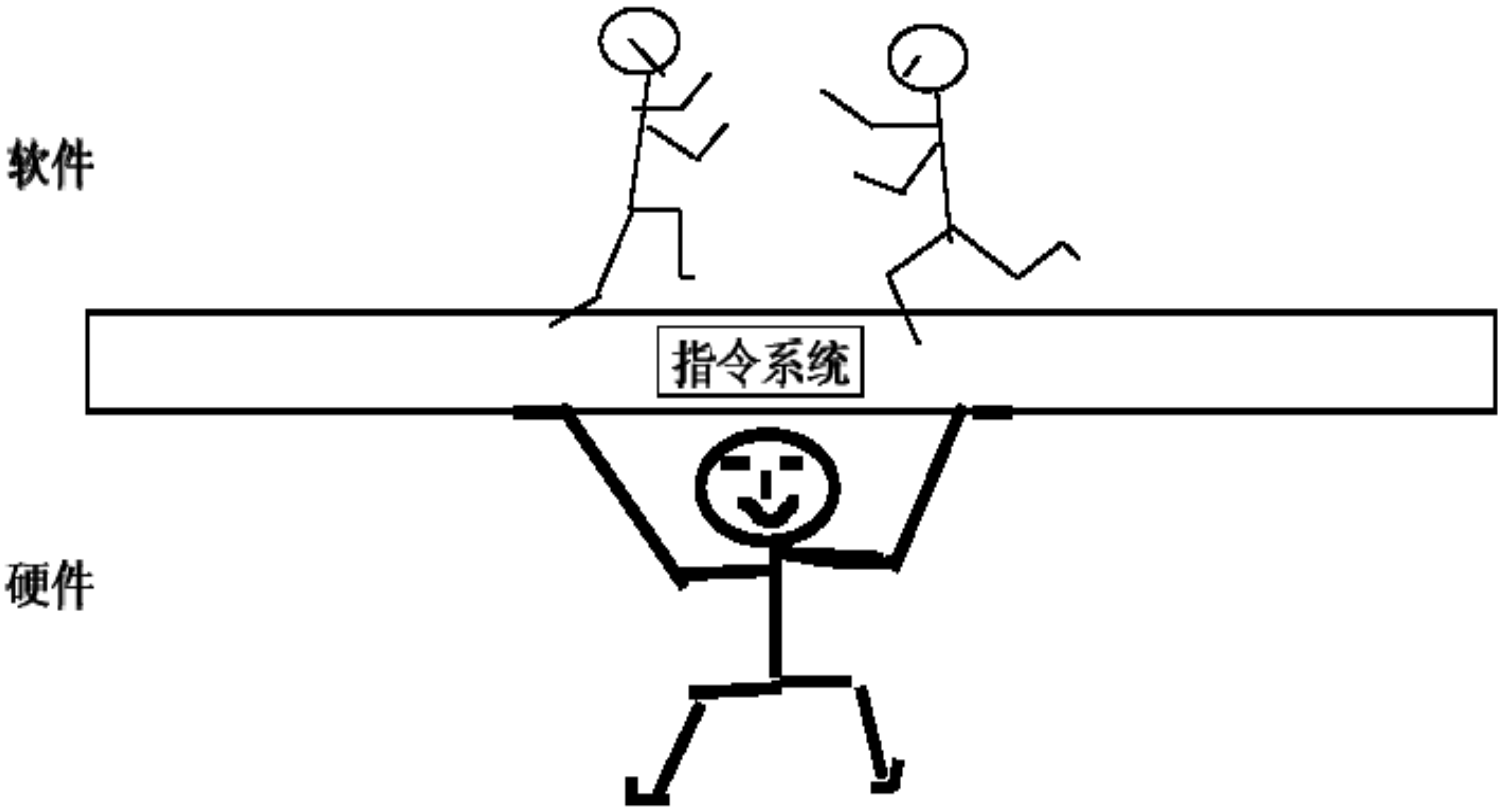
4.4 指令系统结构的分类

4.5 指令系统的设计与优化

4.6 指令系统的发展和改进

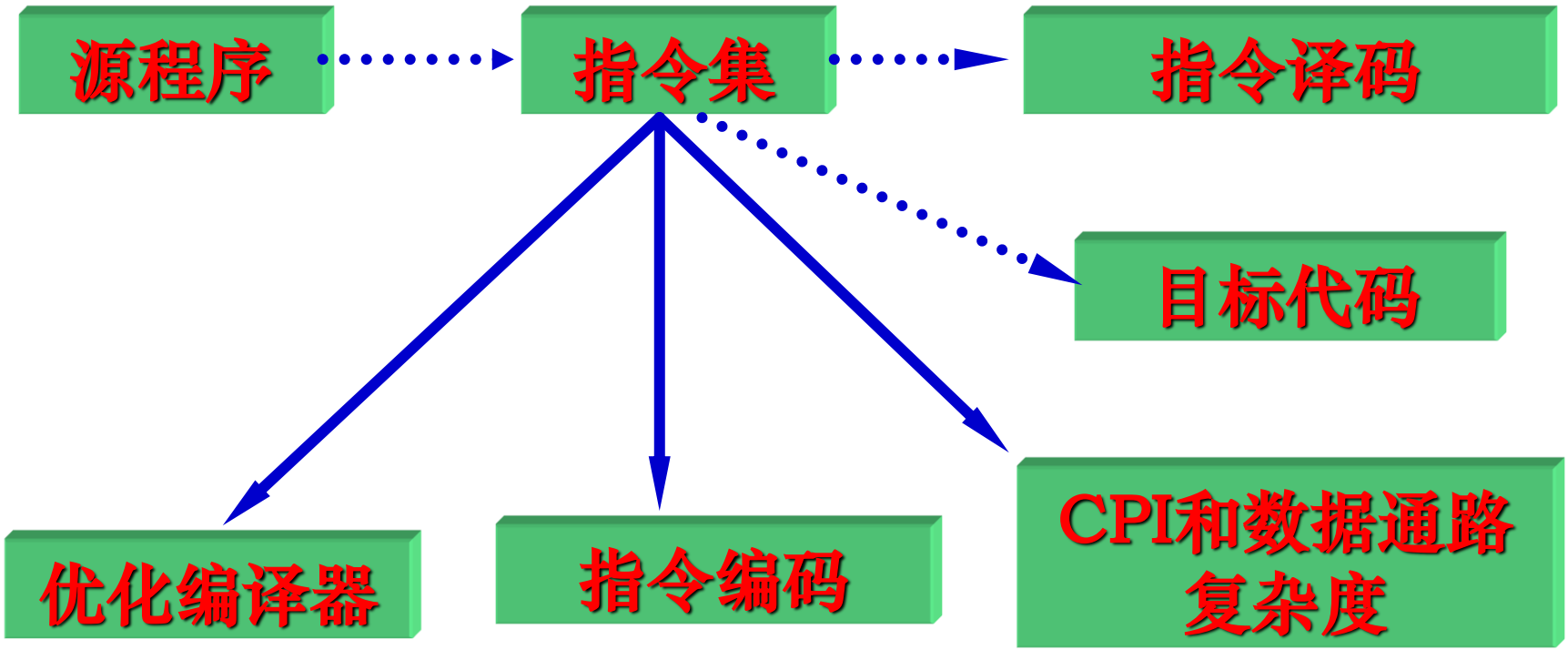
4.7 指令格式举例

指令系统在计算机中的地位



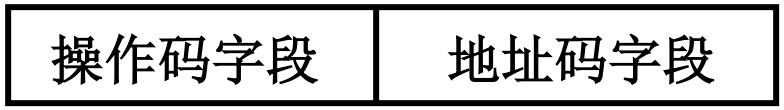
指令集与计算机的性能

$$T_{CPU} = CPI \times IC \times T_{CLK}$$



4.1 机 器 指 令

一、指令的一般格式



1. 操作码　反映机器做什么操作

(1) 长度固定

操作码集中在指令的一个字段，**RISC**

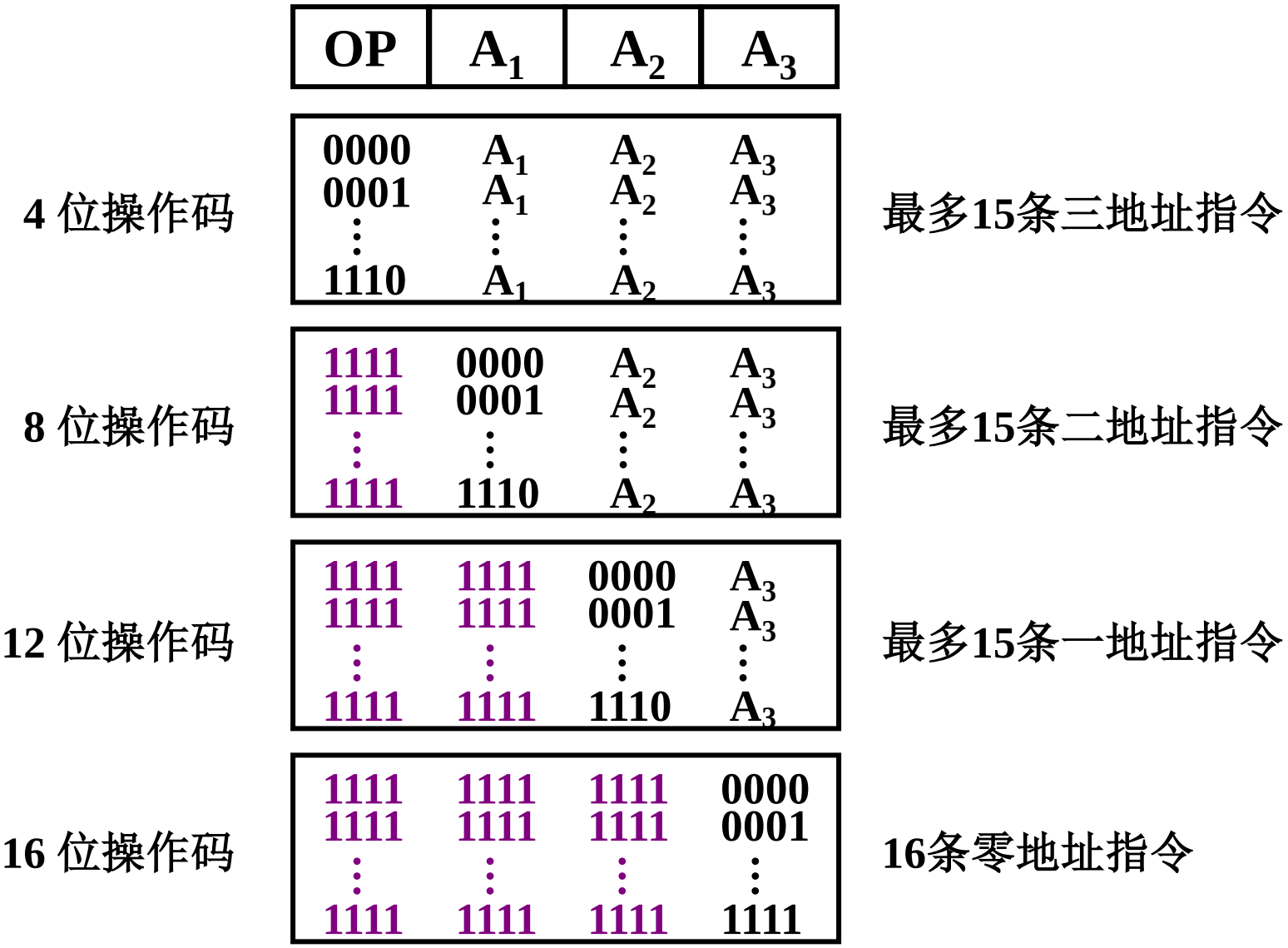
如 **IBM 370** 操作码 **8** 位

(2) 长度可变

操作码分散在指令字的不同字段中

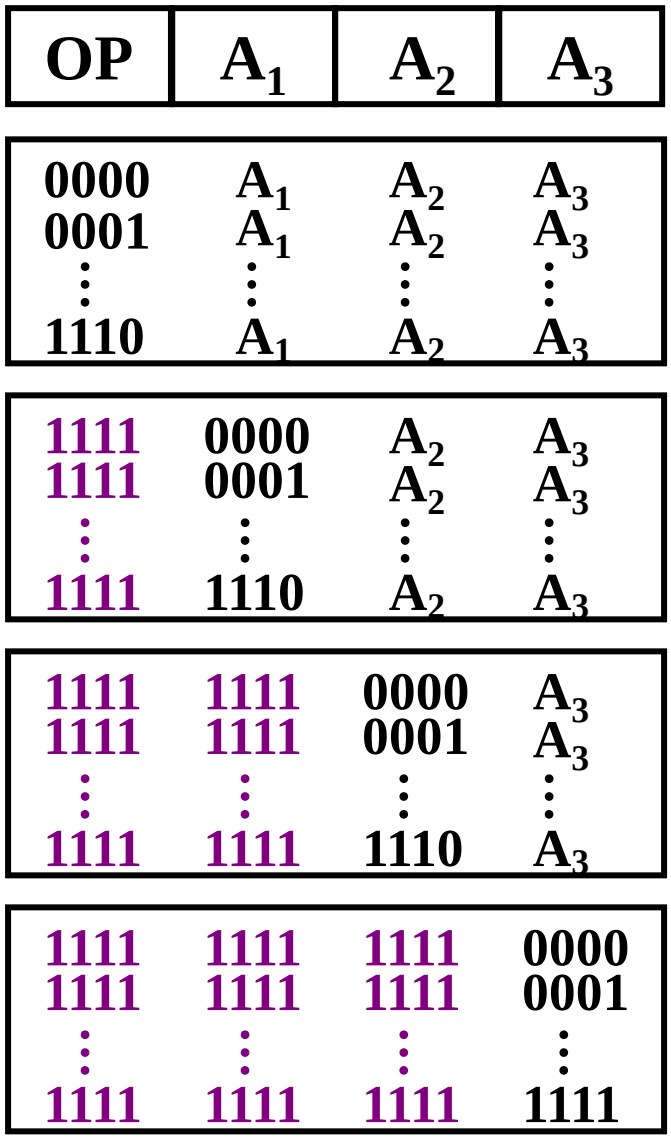
(3) 扩展操作码技术

操作码的位数随地址数的减少而增加



(3) 扩展操作码技术

操作码的位数随地址数的减少而增加

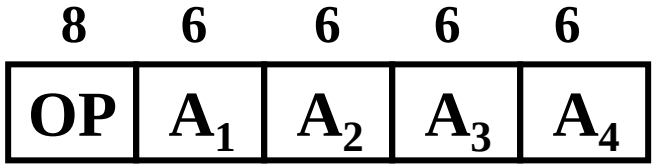


三地址指令操作码
每减少一种可多构成
2^4 种二地址指令

二地址指令操作码
每减少一种可多构成
2^4 种一地址指令

2. 地址码

(1) 四地址



A_1 第一操作数地址

A_2 第二操作数地址

A_3 结果的地址

A_4 下一条指令地址

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

设指令字长为 32 位

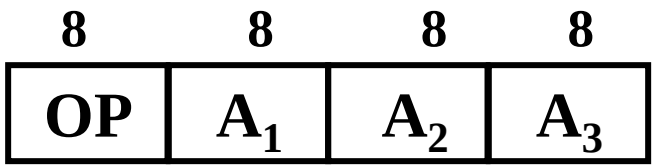
操作码固定为 8 位

4 次访存

寻址范围 $2^6 = 64$

若 PC 代替 A_4

(2) 三地址



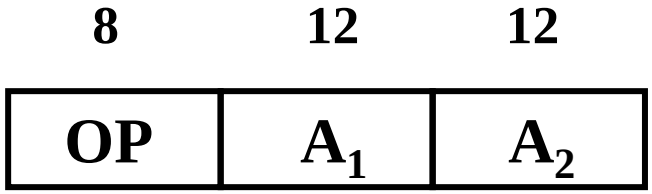
(A_1) OP $(A_2) \longrightarrow A_3$

4 次访存

寻址范围 $2^8 = 256$

若 A_3 用 A_1 或 A_2 代替

(3) 二地址



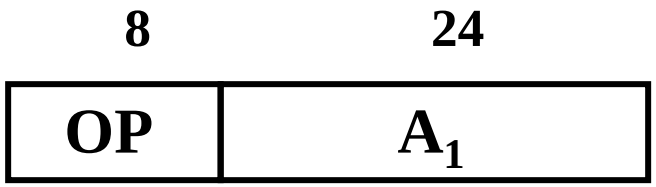
(A_1) OP $(A_2) \longrightarrow A_1$　　4 次访存

或

(A_1) OP $(A_2) \longrightarrow A_2$　　寻址范围 $2^{12} = 4$ K

若结果存于 ACC　3次访存　　若ACC 代替 A_1（或A_2）

(4) 一地址



(ACC) OP $(A_1) \longrightarrow ACC$　　寻址范围 $2^{24} = 16$ M

(5) 零地址　　无地址码

二、指令字长

指令字长决定于
- 操作码的长度
- 操作数地址的长度
- 操作数地址的个数

1. 指令字长 固定

指令字长 = 存储字长

2. 指令字长 可变

按字节的倍数变化

小结

- 当用一些硬件资源代替指令字中的地址码字段后
  - 可扩大指令操作数的寻址范围
  - 可缩短指令字长
  - 可减少访存次数
- 当指令的地址字段为寄存器时

  三地址　OP　R_1，R_2，R_3

  二地址　OP　R_1，R_2

  一地址　OP　R_1
  - 可缩短指令字长
  - 指令执行阶段不访存

4.2 操作数类型和操作种类

一、操作数类型

无符号整数

定点数、浮点数、十进制数

ASCII、字符串

图、表、树

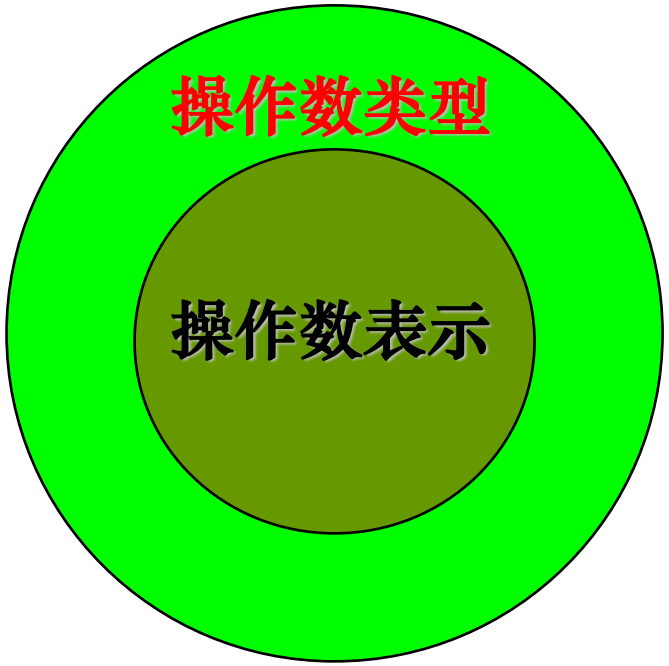
操作数类型和操作数表示也是软硬件主要界面之一。

操作数类型是面向应用、面向软件系统所处理的各种数据结构。

操作数表示是硬件结构能够识别、指令系统可以直接调用的那些结构。

一、操作数类型

- 操作数表示所表征的那些操作数类型，是应用软件和系统软件所处理的操作数类型的子集。



一、操作数类型

- 确定操作数表示实际上也是软硬件取舍折衷的问题
  - 计算机即使只具有最简单的操作数表示，如只有整数（定点）表示法，也可以通过软件方法处理各种复杂的操作数类型，但是这样会大大降低系统的效率。
  - 如果各种复杂的操作数类型均包含在操作数表示之中，无疑会大大提高系统的效率，但是所花费的硬件代价也很高。

一、操作数类型

- 整数（定点）：二进制补码表示；其大小可以是字节（8位）、半字（16位）或单字（32位）。
- 浮点：可以分为单精度浮点（单字大小）和双精度浮点（双字大小）。当前普遍采用的是IEEE 754浮点操作数表示标准。
- 字符和字符串：8位ASCII码表示。

一、操作数类型

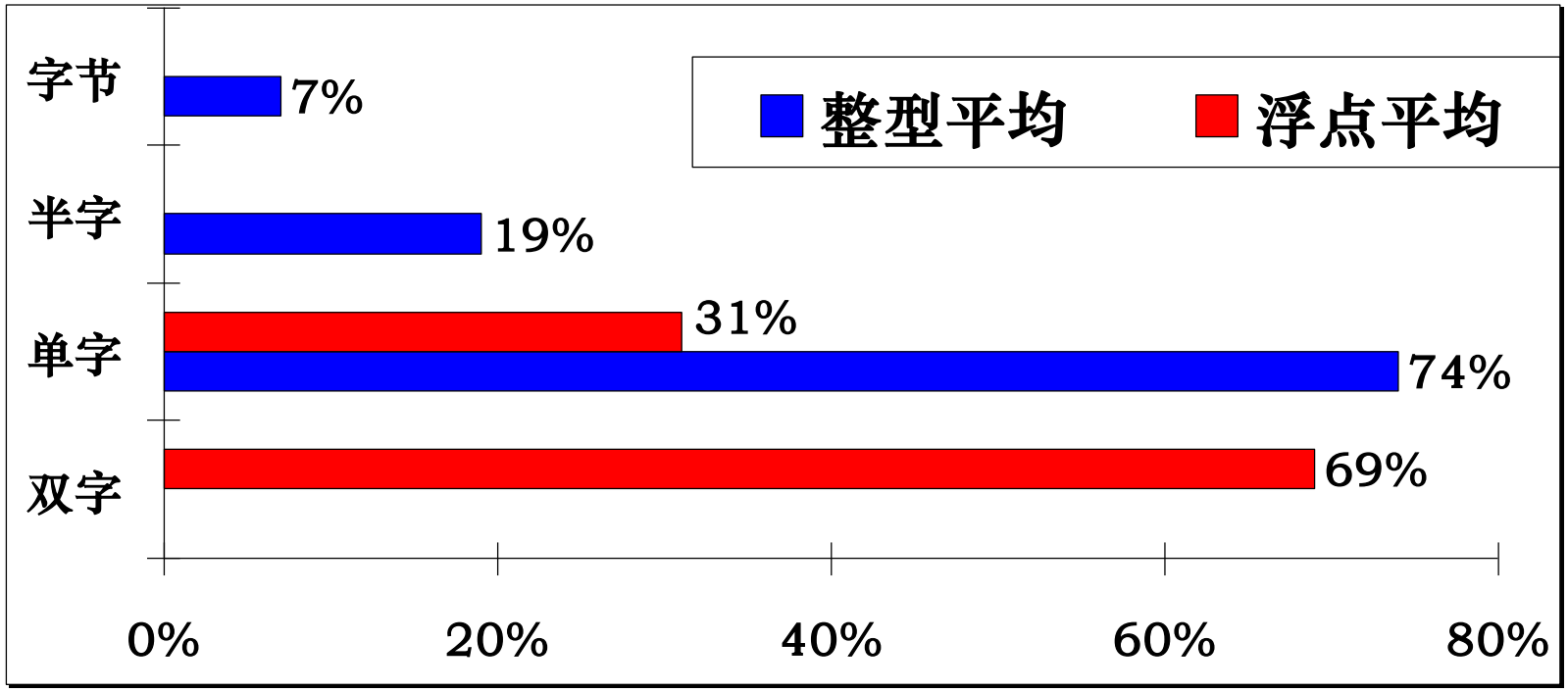
- 十进制：面向商业应用，通常采用“压缩十进制”或“二进制编码十进制（BCD）”表示。压缩十进制数据表示用4位编码数字0～9，然后将两个十进制数字压缩在一个字节中存储。如果将十进制数字直接用字符串来表示，就叫做“非压缩十进制”表示法。

一、操作数类型

- 操作数类型的表示主要有如下两种方法：
  - 操作数的类型可以由操作码的编码指定，这也是最常见的一种方法；
  - 数据可以附上由硬件解释的标记（tag），由这些标记指定操作数的类型，从而选择适当的运算。然而有标记数据的机器却非常少见。

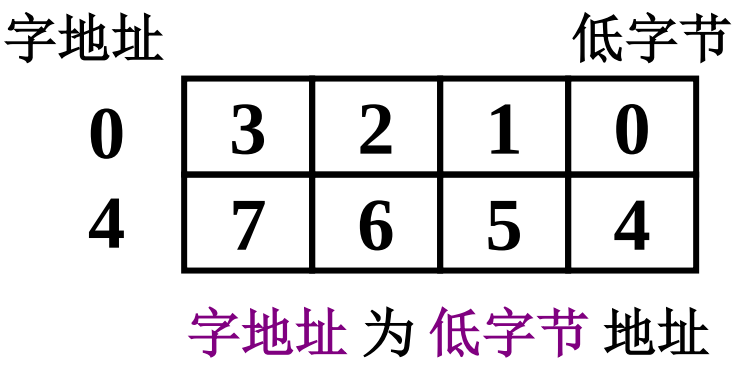
一、操作数类型

- 一般的操作数类型大小选择主要有：字节、半字（16位）、单字（32位）、和双字（64位）。

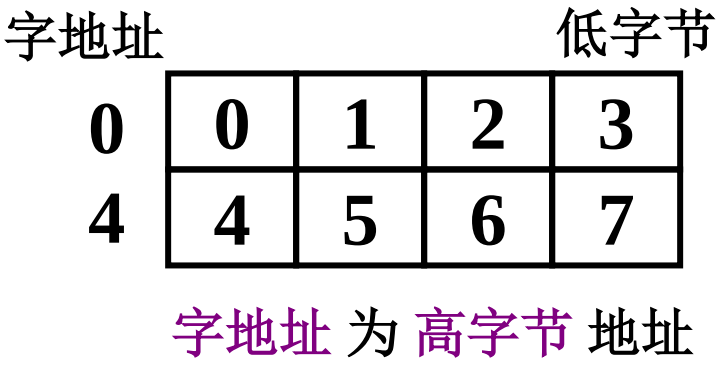


4.2 操作数类型和操作种类

二、数据在存储器中的存放方式



字地址 为 低字节 地址



字地址 为 高字节 地址

存储器中的数据存放（存储字长为32位）

边界对准

地址（十进制）

字（地址 0）				0
字（地址 4）				4
字节（地址11）	字节（地址10）	字节（地址 9）	字节（地址 8）	8
字节（地址15）	字节（地址14）	字节（地址13）	字节（地址12）	12
半字（地址18）✓		半字（地址16）✓		16
半字（地址22）✓		半字（地址20）✓		20
双字（地址24）▲				24
双字				28
双字（地址32）▲				32
双字				36

边界未对准

地址（十进制）

字(地址2)		半字(地址0)	0
字节(地址7)	字节(地址6)	字(地址4)	4
半字(地址10)		半字(地址8)	8

三、操作类型

1. 数据传送

源	寄存器	寄存器	存储器	存储器
目的	寄存器	存储器	寄存器	存储器
例如	**MOVE**	**STORE** **MOVE** **PUSH**	**LOAD** **MOVE** **POP**	**MOVE**

置“1”，清“0”

2. 算术逻辑操作

加、减、乘、除、增 1、减 1、求补、浮点运算、十进制运算

与、或、非、异或、位操作、位测试、位清除、位求反

如 **8086** **ADD SUB MUL DIV INC DEC CMP NEG**
AAA AAS AAM AAD
AND OR NOT XOR TEST

3. 移位操作

算术移位　逻辑移位

循环移位（带进位和不带进位）

4. 转移

(1) 无条件转移　**JMP**

(2) 条件转移

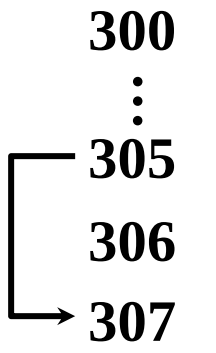
结果为零转　（Z = 1）**JZ**

结果溢出转　（O = 1）**JO**

结果有进位转（C = 1）**JC**

跳过一条指令　**SKP**

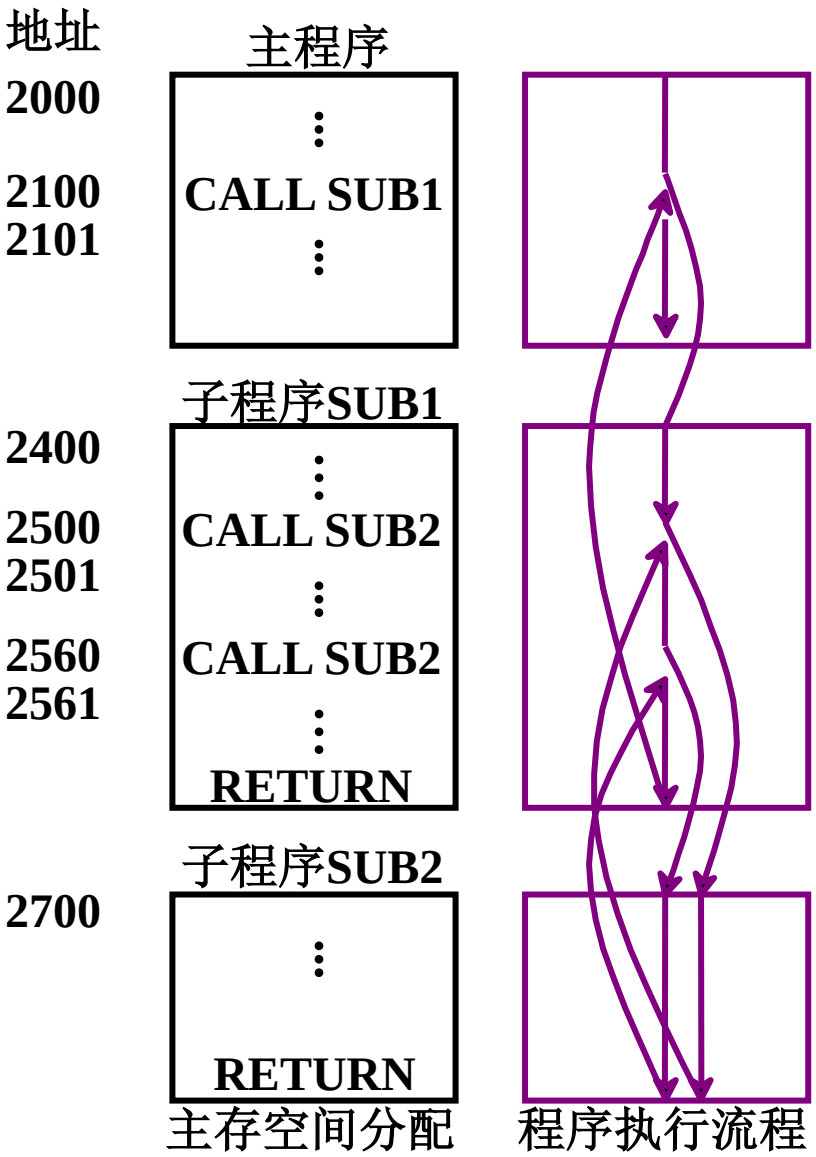
如



完成触发器

SKP DZ D = 0 则跳

(3) 调用和返回


地址
主程序
2000
2100
CALL SUB1
2101
子程序SUB1
2400
2500
CALL SUB2
2501
2560
CALL SUB2
2561
RETURN
子程序SUB2
2700
RETURN
主存空间分配
程序执行流程

(4) 陷阱（Trap）与陷阱指令

意外事故的中断

- 一般不提供给用户直接使用

  在出现事故时，由 CPU 自动产生并执行（隐指令）

- 设置供用户使用的陷阱指令

  如 8086　　INT　TYPE　　软中断

  提供给用户使用的陷阱指令，完成系统调用

5. 输入输出

入　　端口地址 ⟶ CPU 的寄存器

如　　IN AX, *n*　　IN AX, DX

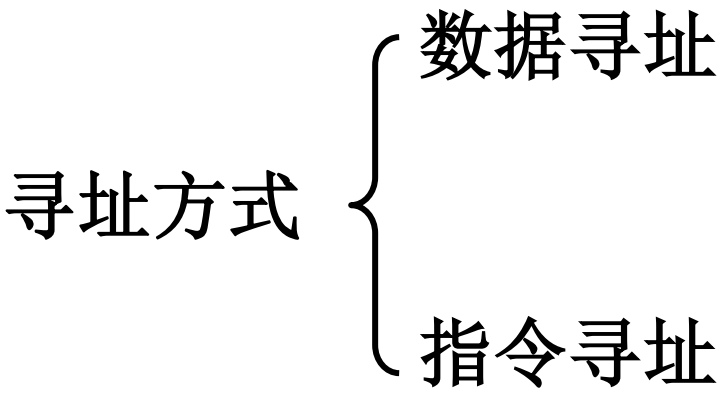
出　　CPU 的寄存器 ⟶ 端口地址

如　　OUT *n*, AX　　OUT DX, AX

4.3 寻址方式

寻址方式　确定 本条指令 的 操作数地址

下一条 欲执行 指令 的 指令地址

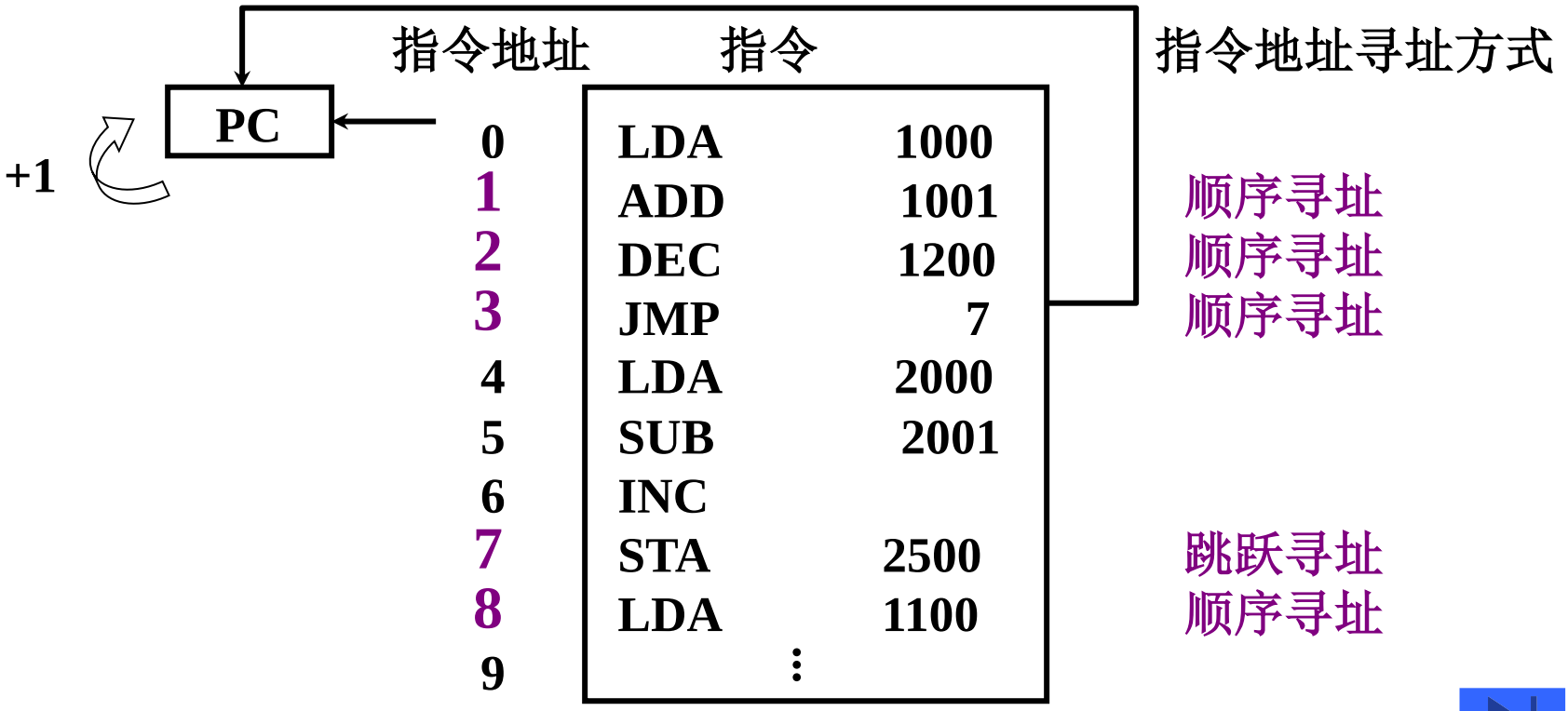


4.3 寻址方式

一、指令寻址

顺序　　(PC) + 1 ⟶ PC

跳跃　　由转移指令指出



二、数据寻址

操作码	寻址特征	形式地址 A

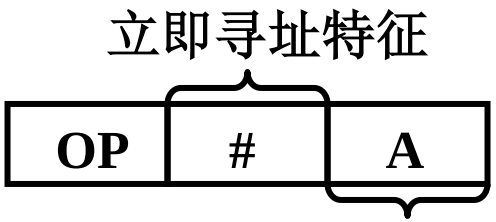
形式地址　　指令字中的地址

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（例：ADD R4, #3）

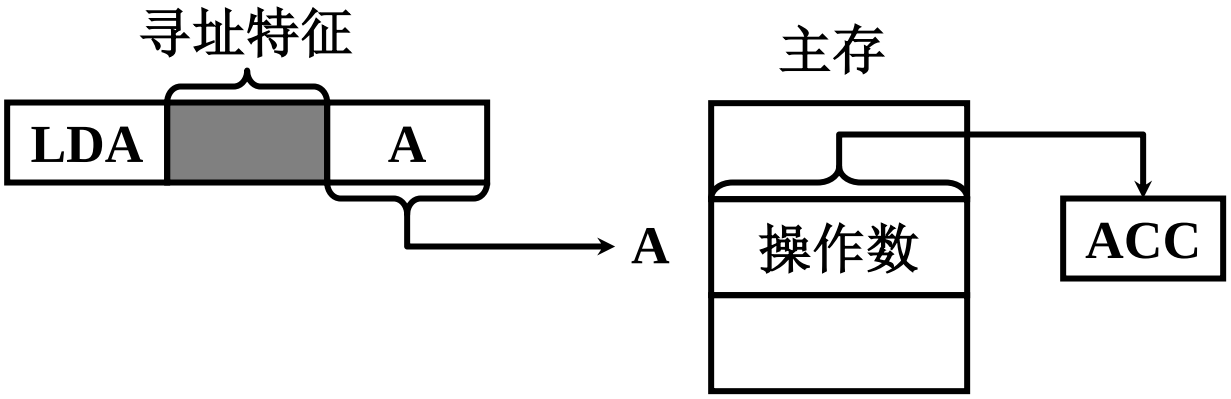


可正可负　补码

- 指令执行阶段不访存
- A 的位数限制了立即数的范围

2. 直接寻址

EA = A　　有效地址由形式地址直接给出

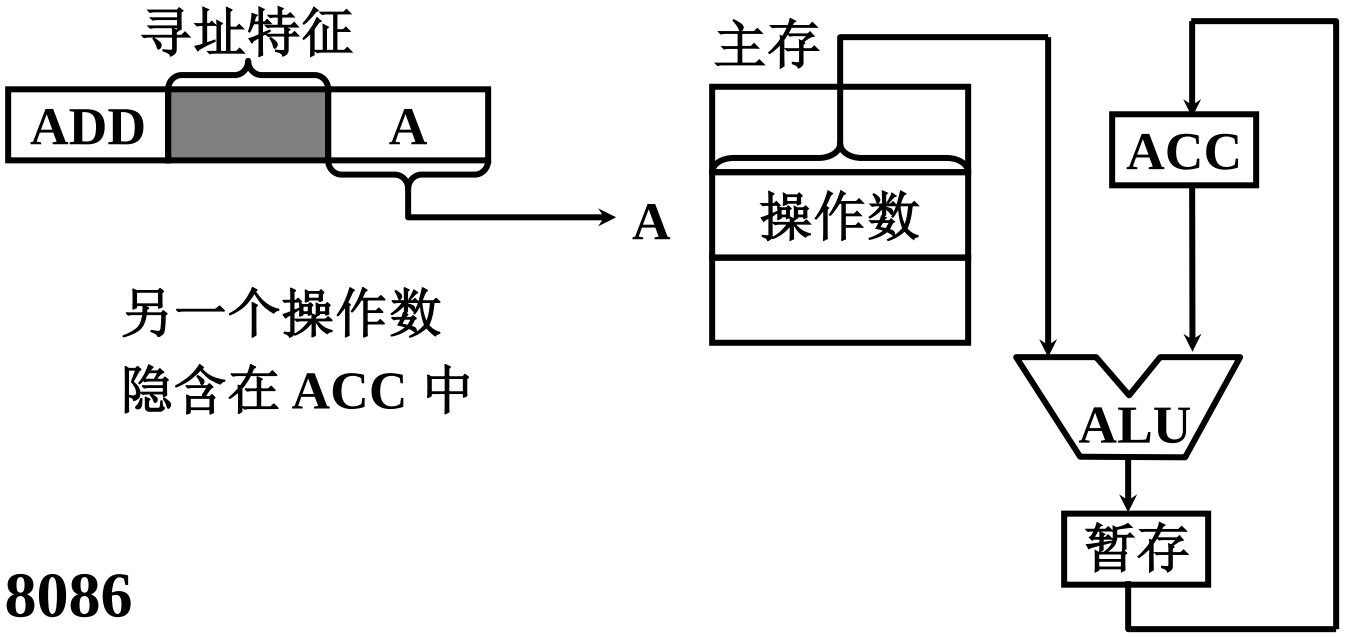


例：ADD R1, (1001)

- 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

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

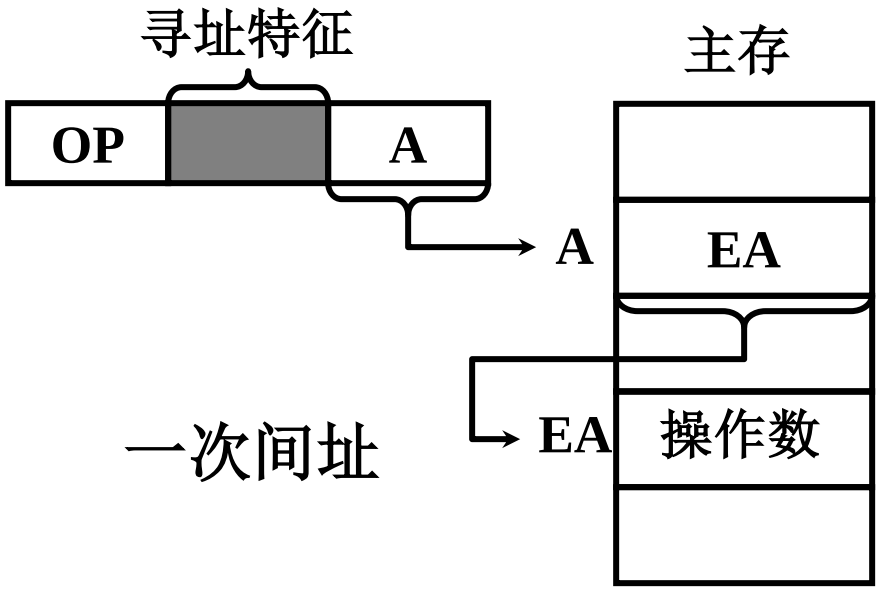
MOVS 指令　源操作数的地址隐含在 SI 中

目的操作数的地址隐含在 DI 中

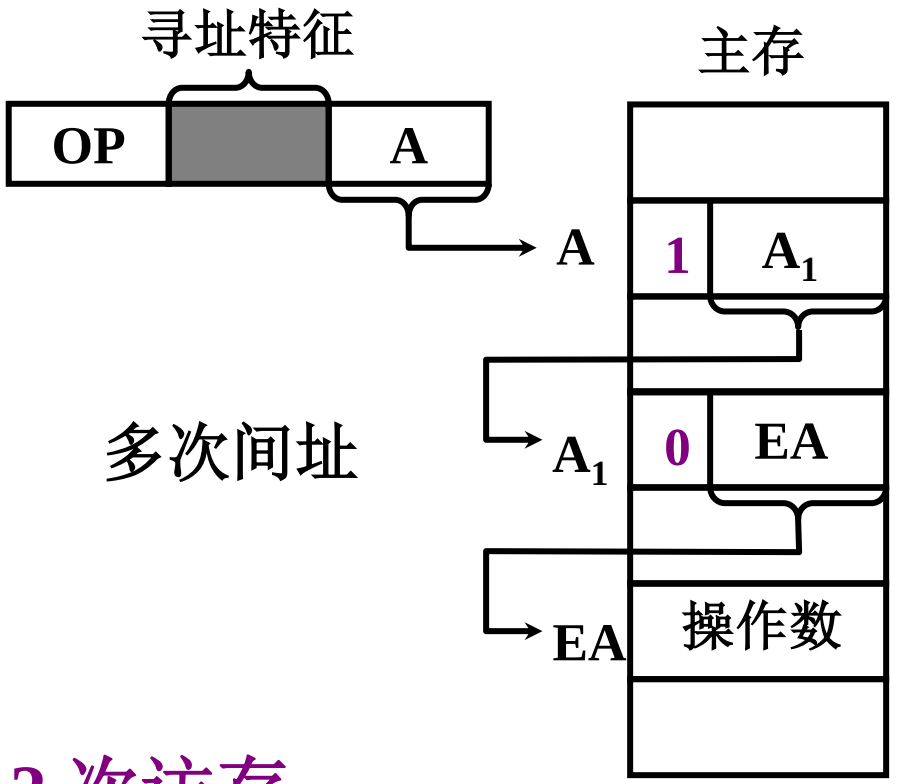
- 指令字中少了一个地址字段，可缩短指令字长

4. 间接寻址

EA =（A） 有效地址由形式地址间接提供



一次间址



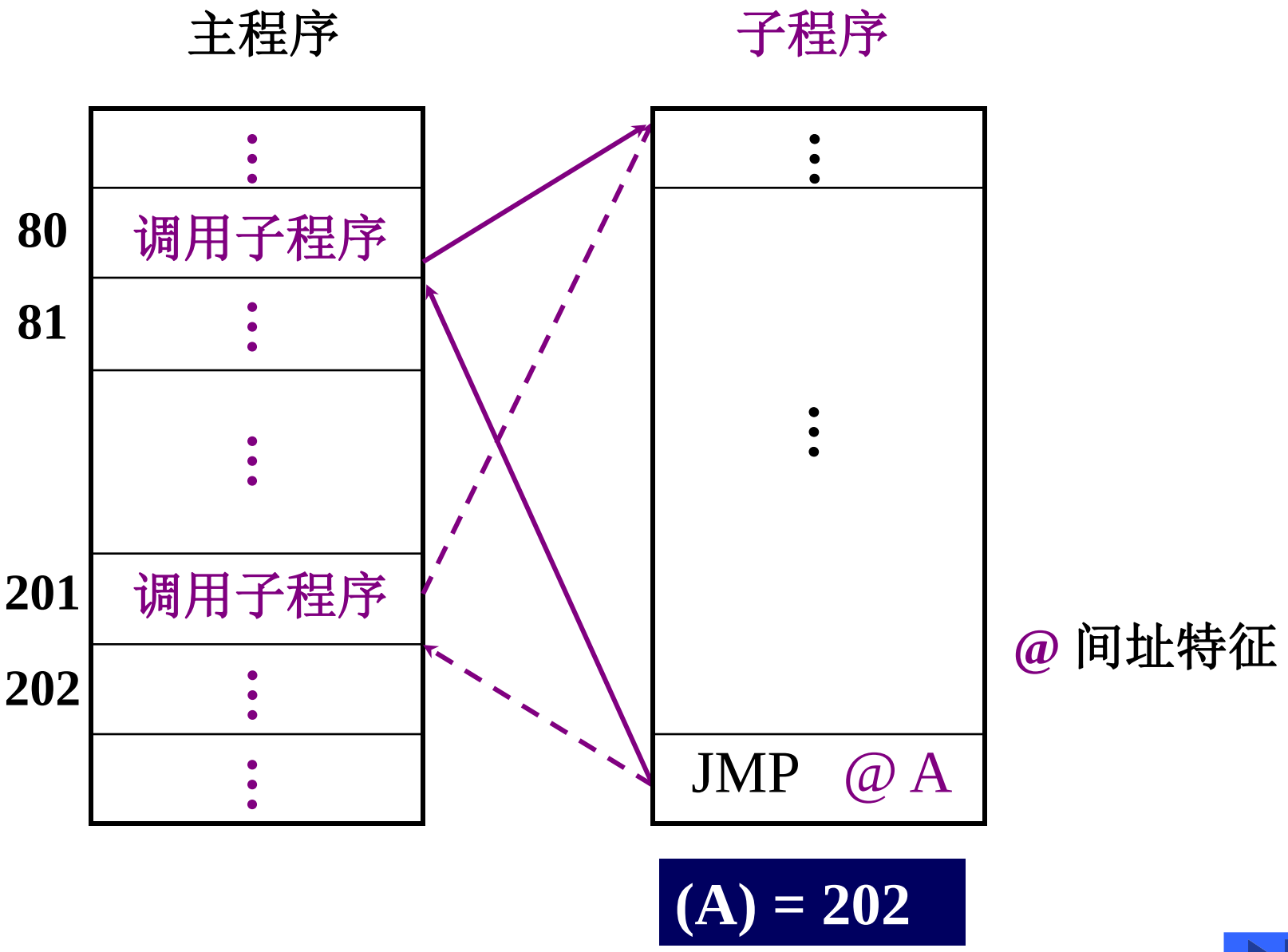
多次间址

例： ADD R1, @(1001)

- 执行指令阶段 2 次访存
- 可扩大寻址范围
- 便于编制程序

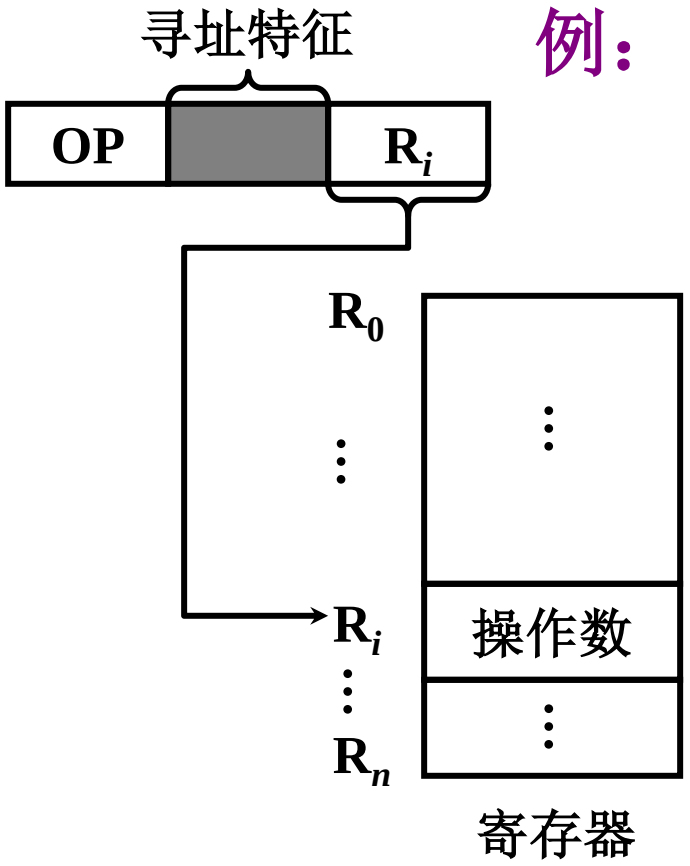
多次访存

间接寻址编程举例



5. 寄存器寻址

EA = R_i　　有效地址即为寄存器编号



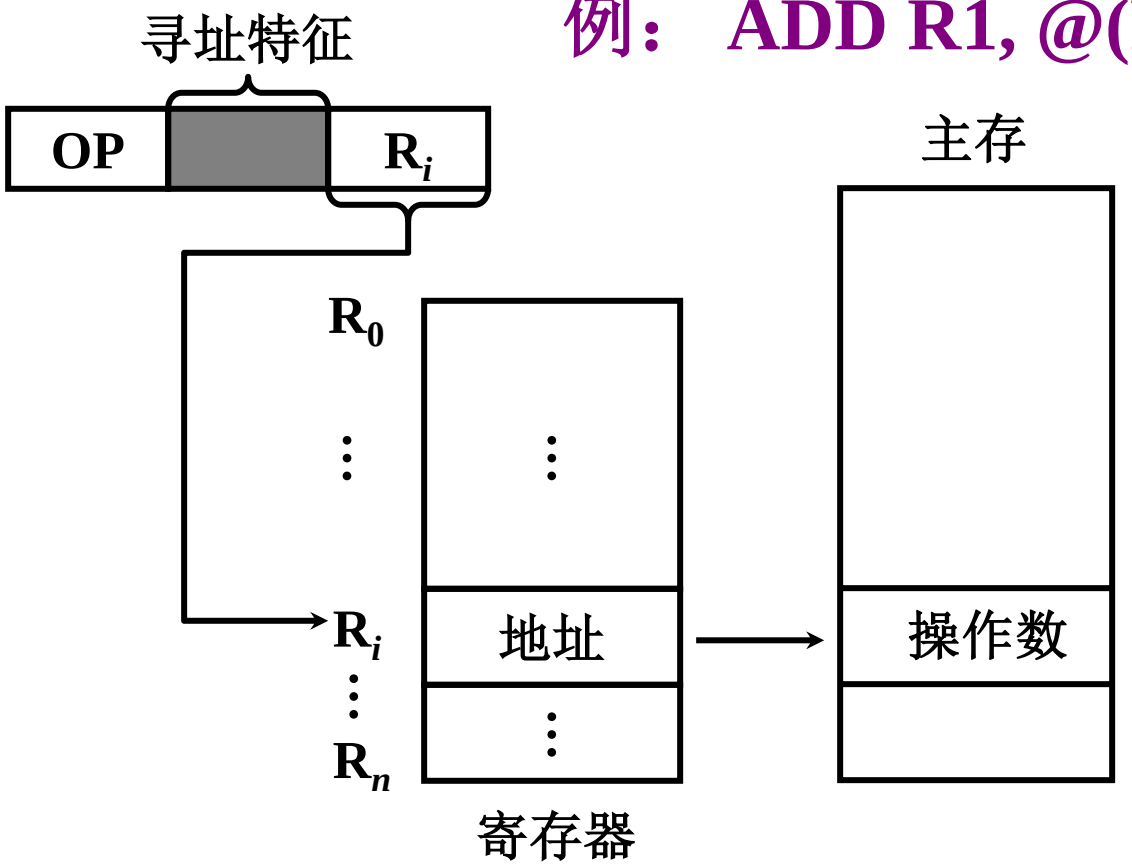
例： **ADD R1, R2**

- 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

EA = (R_i)　　**有效地址在寄存器中**

例： ADD R1, @(R2)

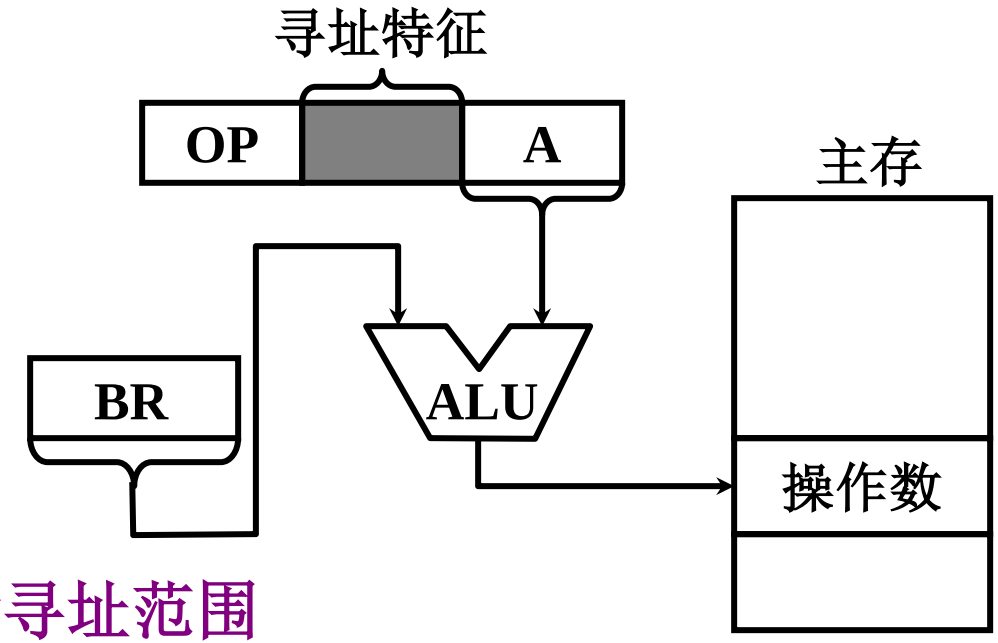


- 有效地址在寄存器中， 操作数在存储器中，执行阶段访存
- 便于编制循环程序

7. 基址寻址

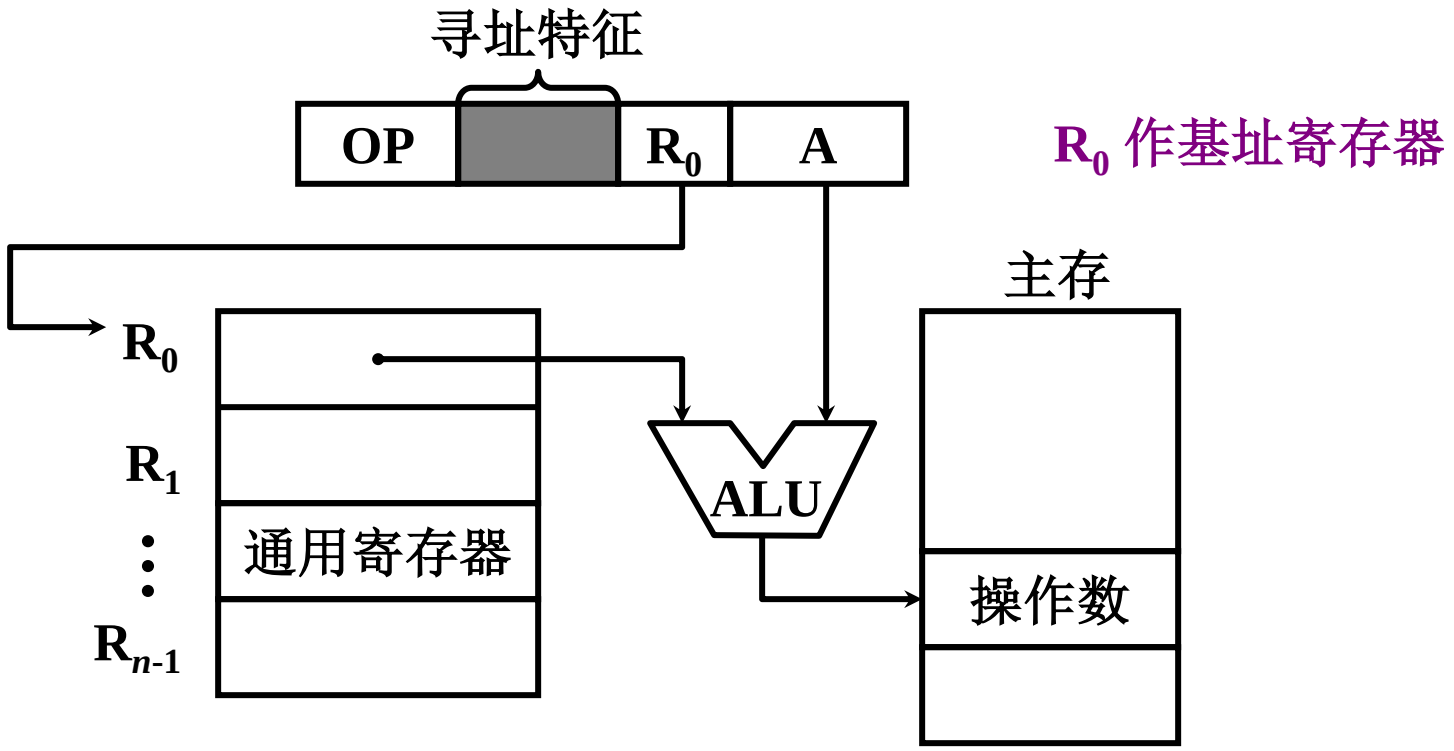
(1) 采用专用寄存器作基址寄存器

EA = (BR) + A　　BR 为基址寄存器



- 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



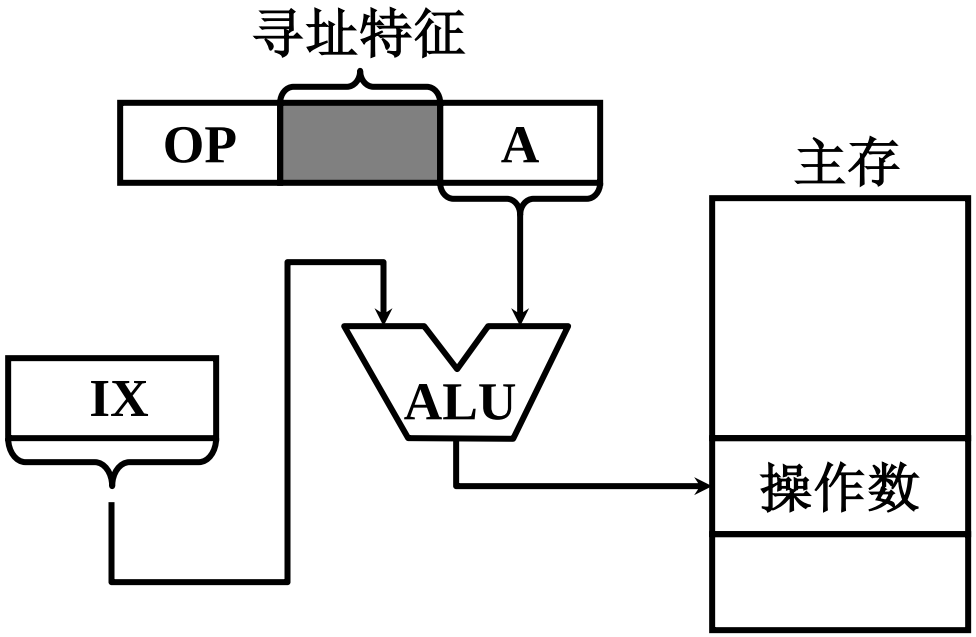
- 由用户指定哪个通用寄存器作为基址寄存器
- 基址寄存器的内容由操作系统确定
- 在程序的执行过程中 R_0 内容不变，形式地址 A 可变

8. 变址寻址

$EA = (IX) + A$

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
- 在程序的执行过程中 IX 内容可变，形式地址 A 不变
- 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

直接寻址

```
LDA    D
ADD    D + 1
ADD    D + 2
  ⋮
ADD    D + ( N -1 )
DIV    # N
STA    ANS
```

共 *N* + 2 条指令

变址寻址

```
      LDA    # 0
      LDX    # 0
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
```

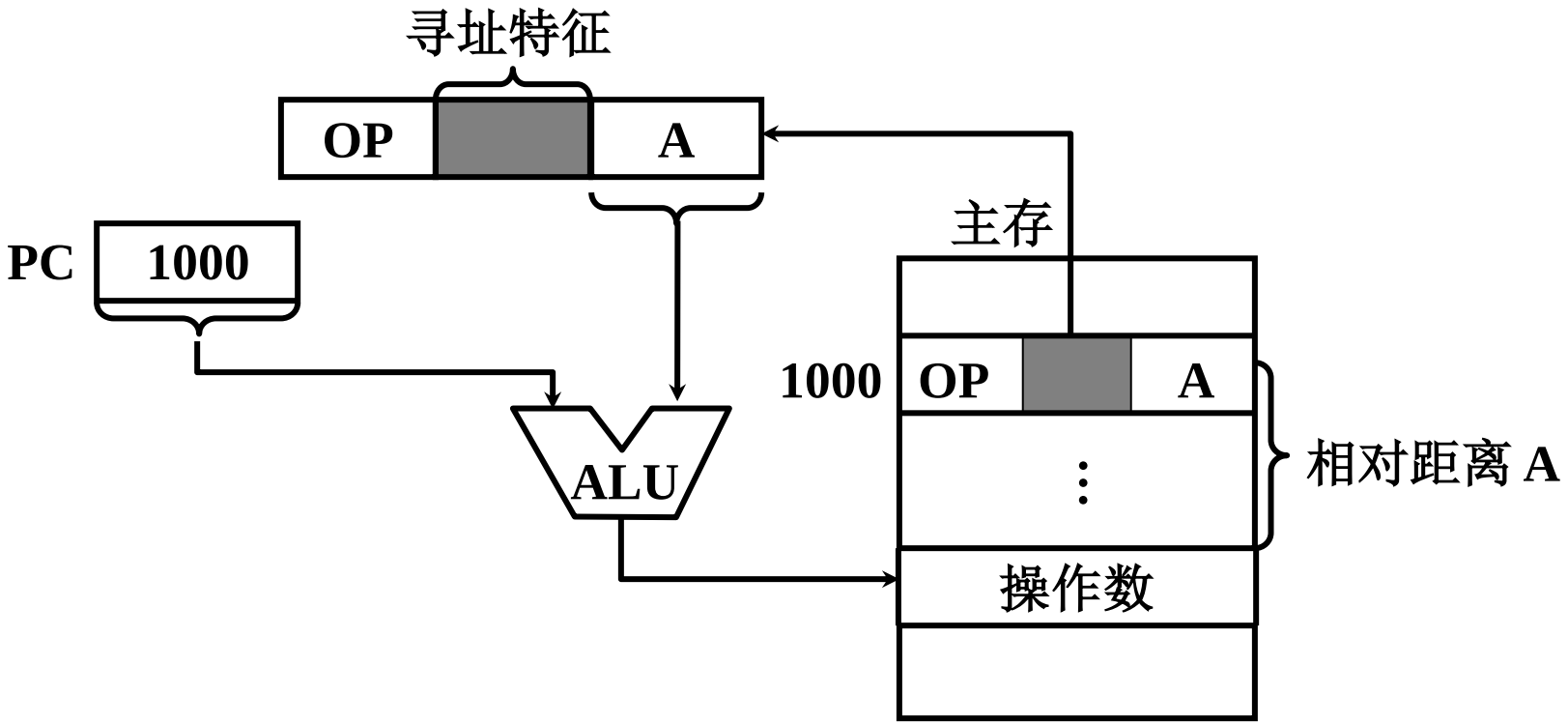
X 为变址寄存器

共 8 条指令

9. 相对寻址

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

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

✱ 相对寻址特征

M → ✱ – 3

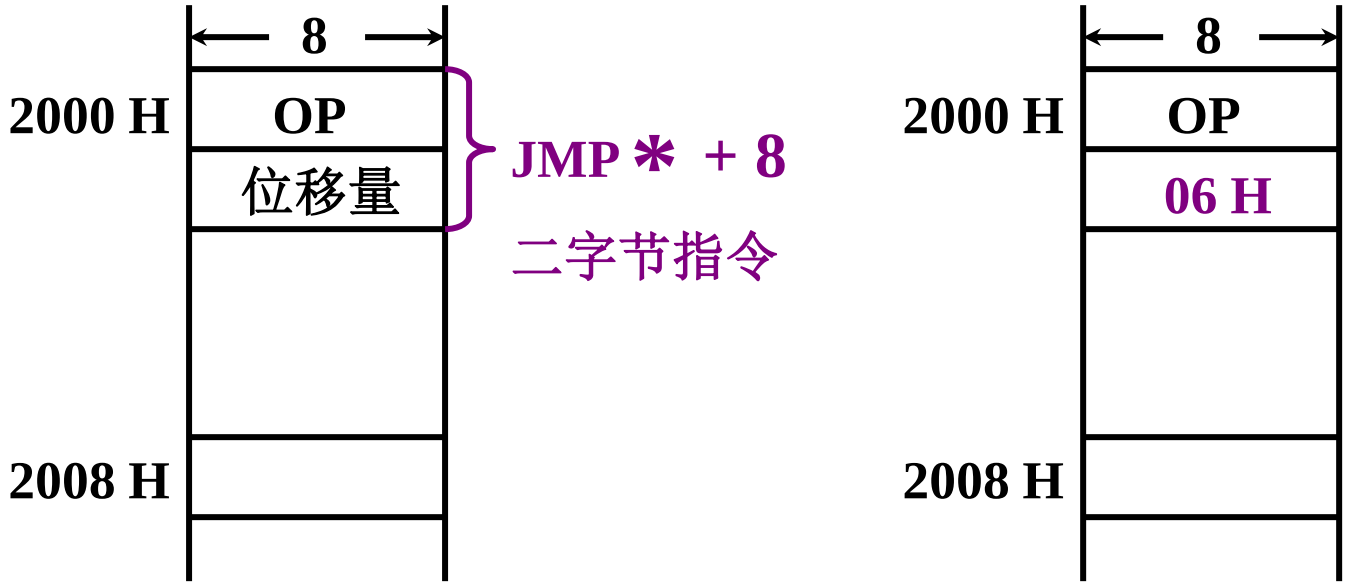
M 随程序所在存储空间的位置不同而不同

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



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**

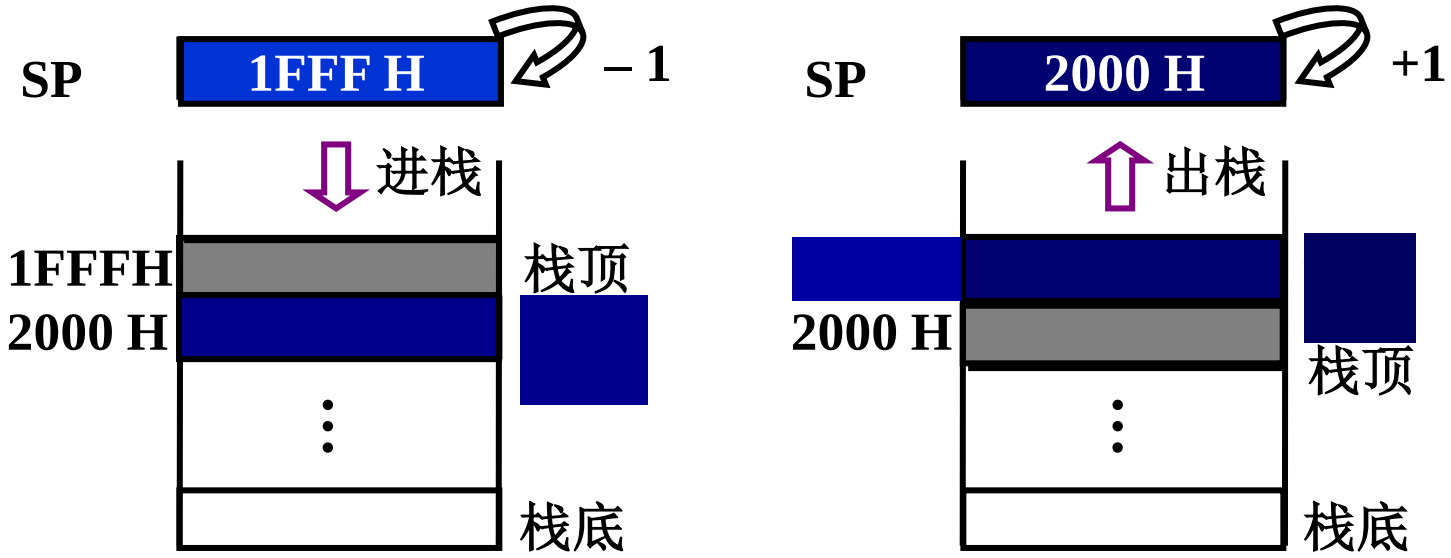
10. 堆栈寻址

(1) 堆栈的特点

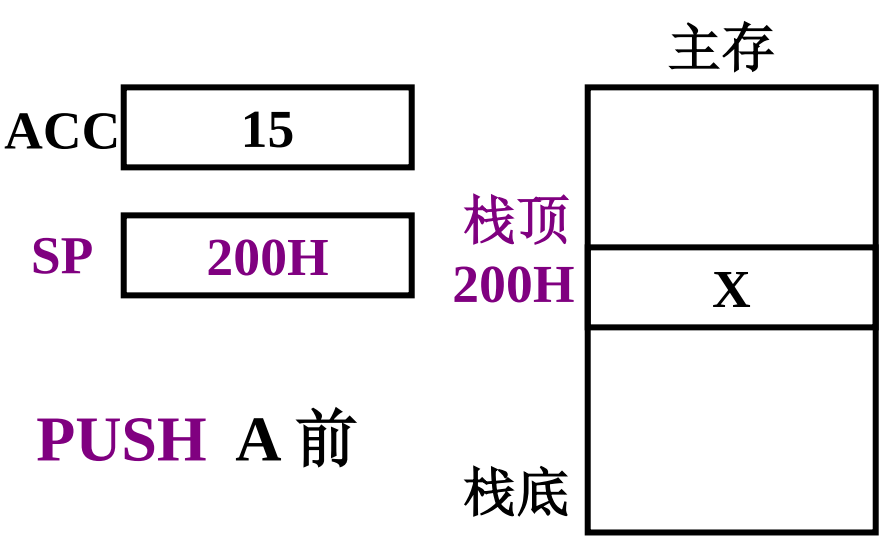
堆栈 { 硬堆栈　多个寄存器
　　 软堆栈　指定的存储空间

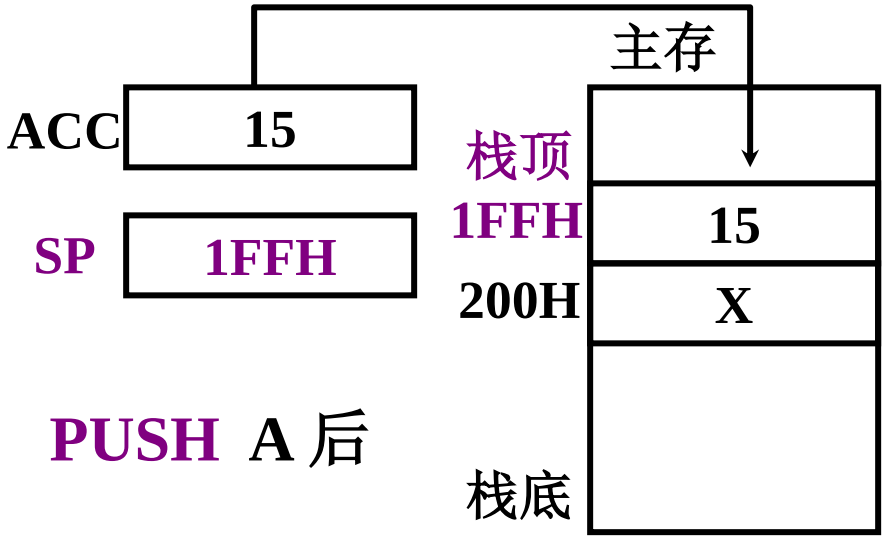
先进后出（一个入出口）　栈顶地址 由 SP 指出

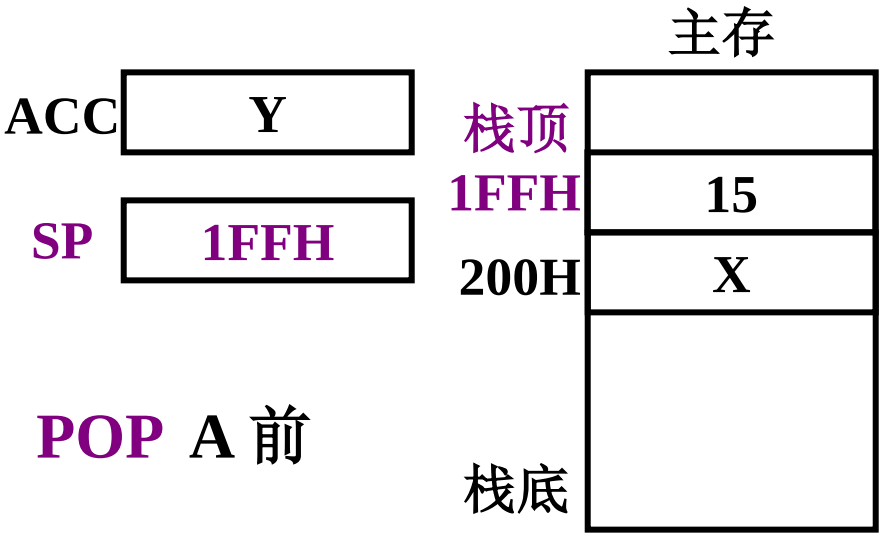
进栈 （SP）– 1 ⟶ SP　　出栈 （SP）+ 1 ⟶ SP

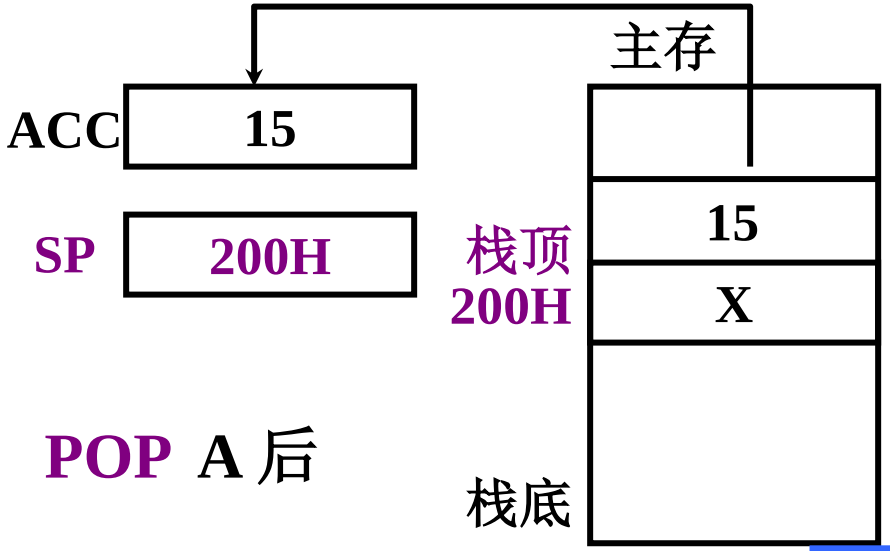


(2) 堆栈寻址举例


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
15
栈顶
200H
X
栈底
POP A 后

(3) SP 的修改与主存编址方法有关

① 按 字 编址

进栈　（SP）– 1 ⟶ SP

出栈　（SP）+ 1 ⟶ SP

② 按 字节 编址

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

基本寻址方式优缺点

方式	算法	主要优点	主要缺点
立即寻址	操作数=A	无存储器访问	操作数幅值有限
直接寻址	EA=A	简单	地址范围有限
间接寻址	EA=（A）	大的地址范围	多重存储器访问
寄存器寻址	EA=R	无存储器访问	地址范围有限
寄存器间接寻址	EA=（R）	大的地址范围	额外存储器访问
相对/基址/变址寻址(偏移寻址)	EA=A+（R）	灵活	复杂
堆栈寻址	EA=栈顶	无存储器访问	应用有限

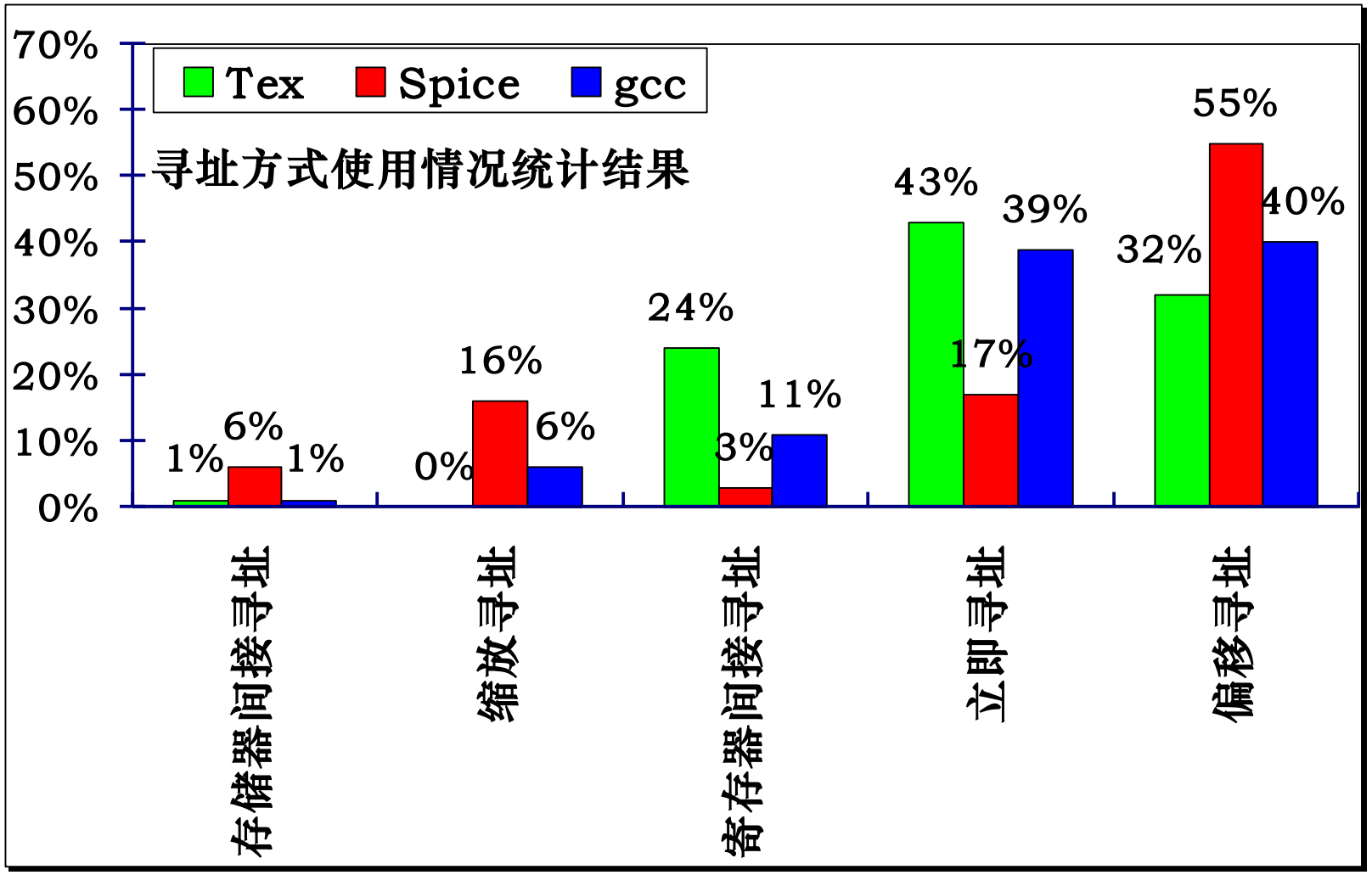
其他寻址方式

- 自增寻址
  指令实例：Add R1, (R2)+
  含义：Regs[R1]←Regs[R1]＋Mem[Regs[R2]]
  Regs[R2]←Regs[R2]＋d

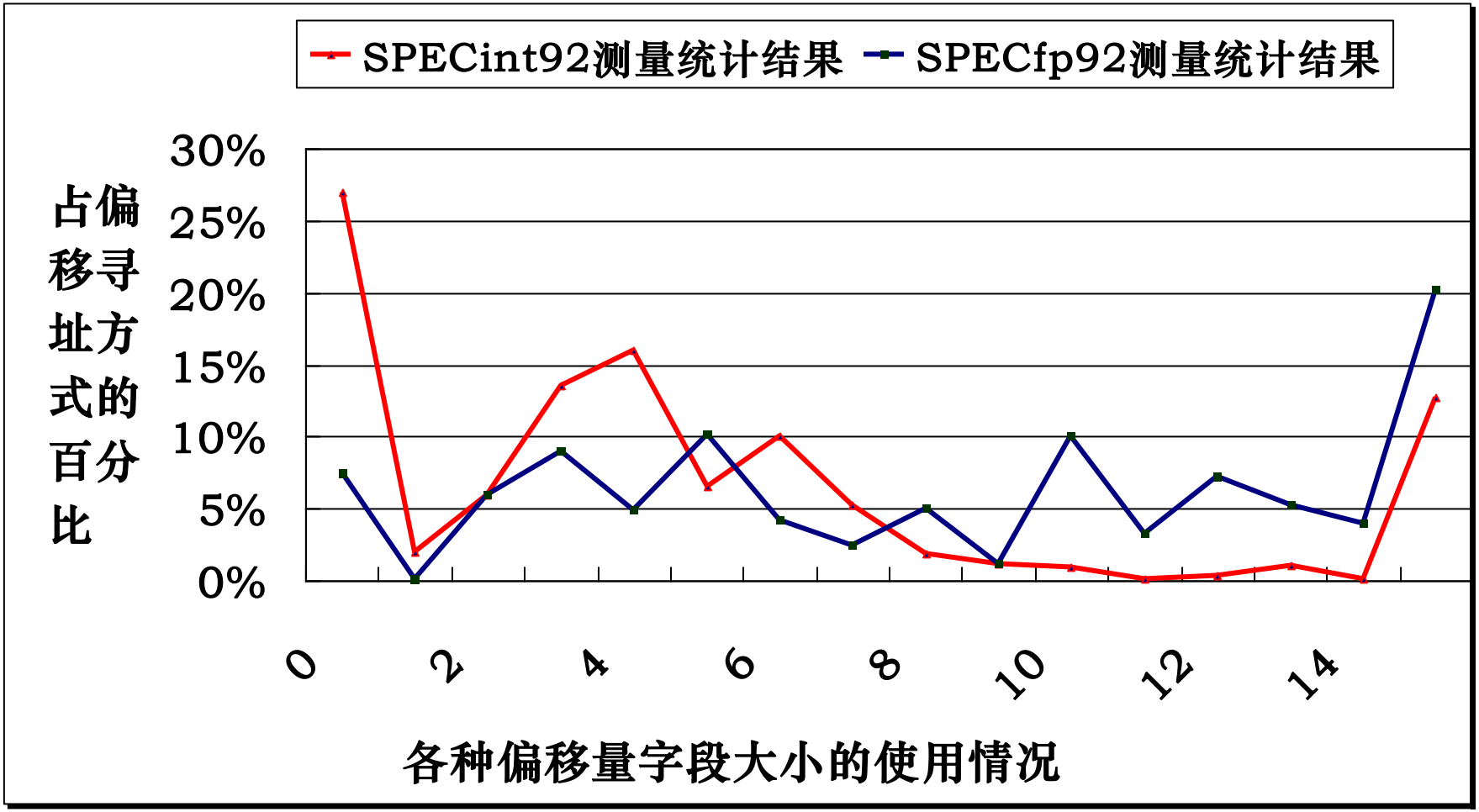
- 自减寻址
  指令实例：Add R1, -(R2)
  含义：Regs[R2]←Regs[R2]-d
  Regs[R1]←Regs[R1]+Mem[Regs[R2]]

- 缩放寻址
  指令实例：Add R1, 100(R2)[R3]
  含义：
  Regs[R1]←Regs[R1]＋Mem[100＋Regs[R2]＋Regs[R3]*d]

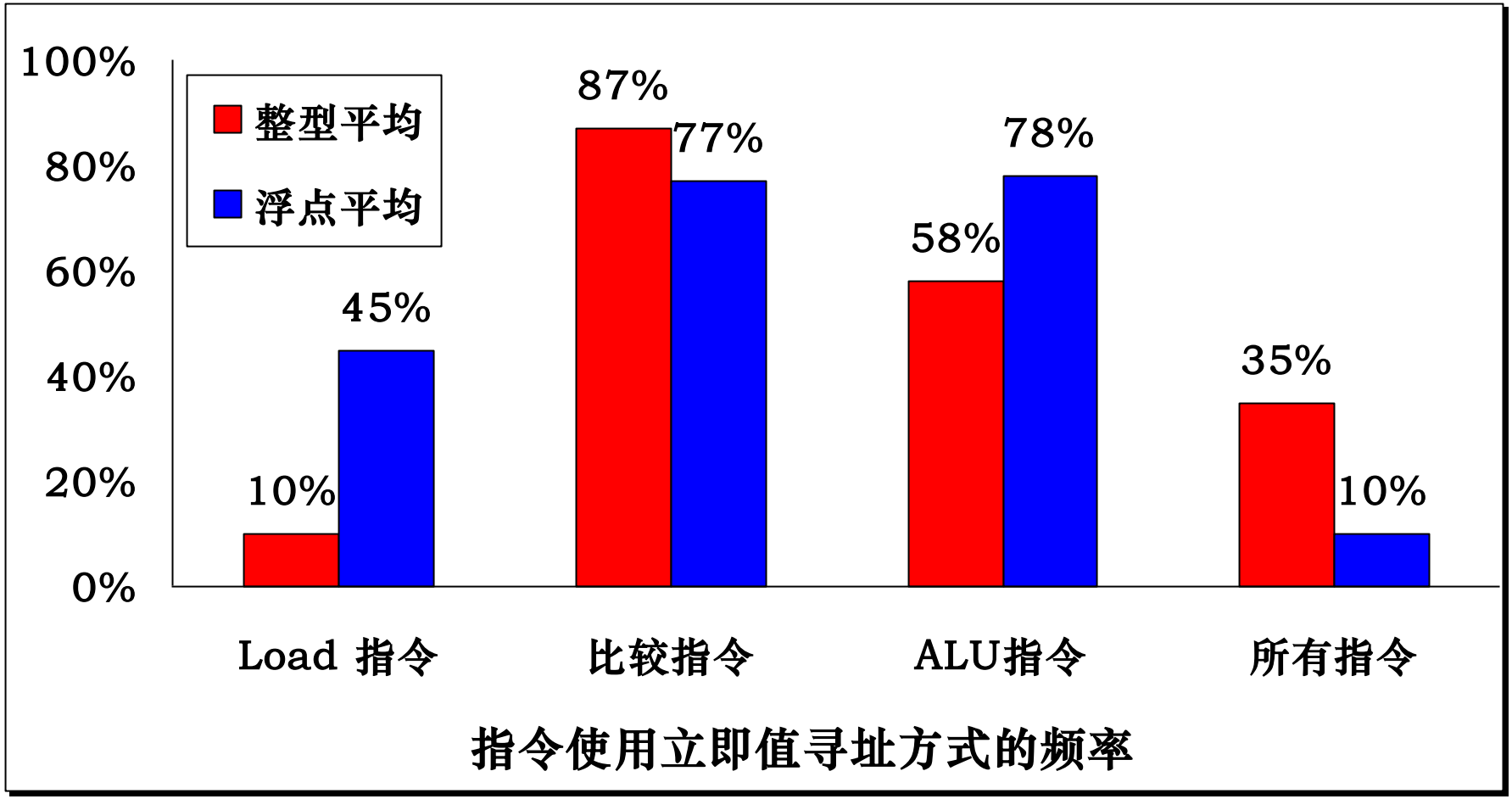
常用的一些操作数寻址方式



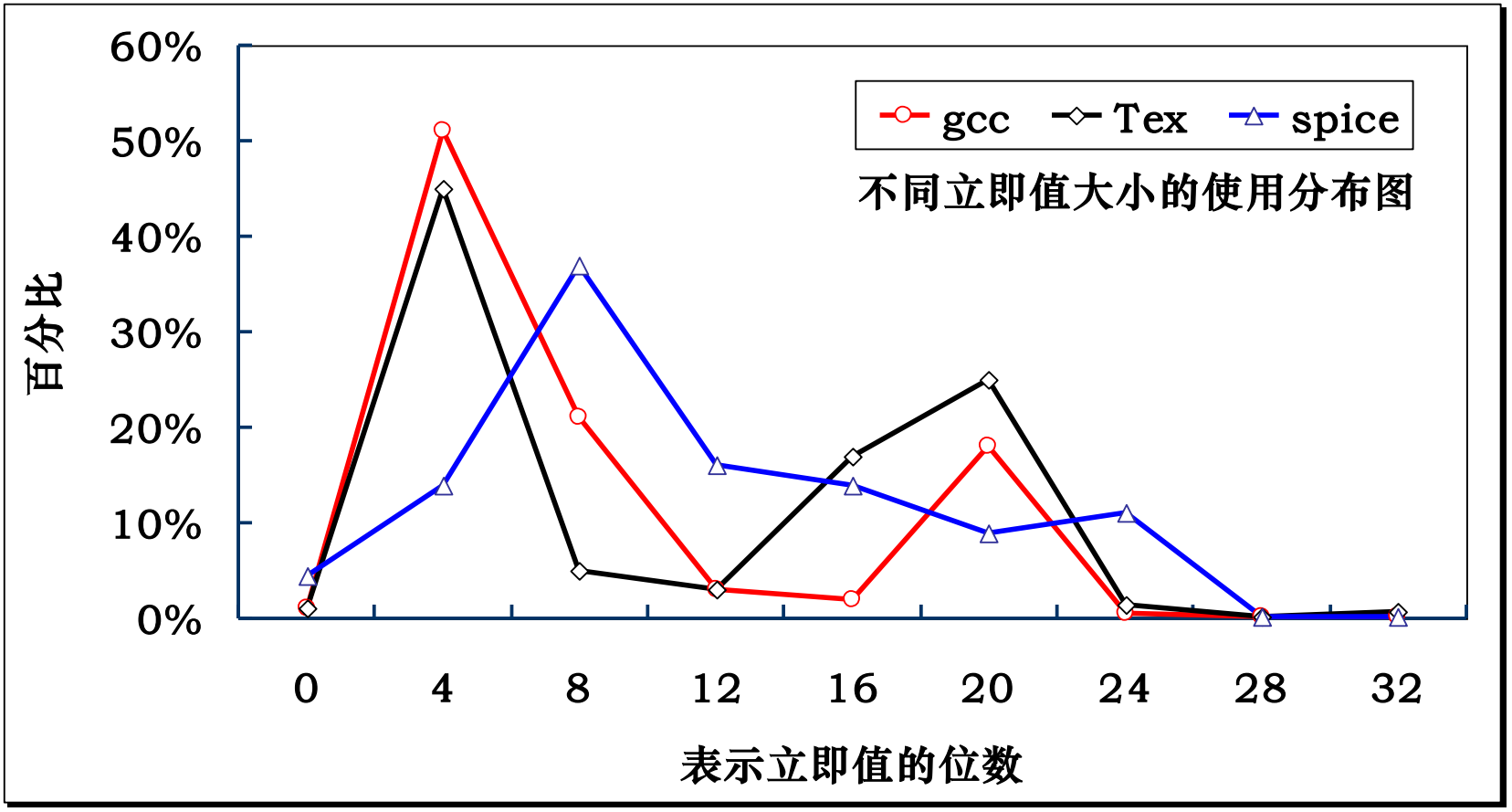
偏移寻址


SPECint92测量统计结果
SPECfp92测量统计结果
占偏移寻址方式的百分比
30%
25%
20%
15%
10%
5%
0%
0
2
4
6
8
10
12
14
各种偏移量字段大小的使用情况

立即寻址


100%
80%
60%
40%
20%
0%
整型平均
浮点平均
10%
45%
87%
77%
58%
78%
35%
10%
Load 指令
比较指令
ALU指令
所有指令
指令使用立即值寻址方式的频率

立即寻址


60%
50%
40%
30%
20%
10%
0%
百分比
gcc
Tex
spice
不同立即值大小的使用分布图
0
4
8
12
16
20
24
28
32
表示立即值的位数