软件安全与漏洞分析

1.3 指令系统

指令系统

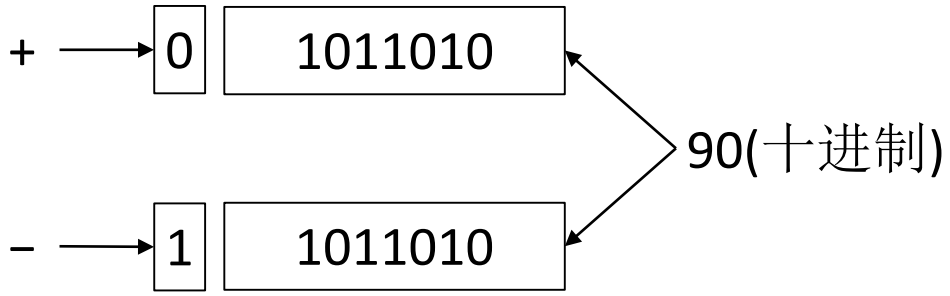
- 实际上，这是计算机原理的一部分
  - 数的机器编码及其表示
  - 指令系统的基本概念
  - 指令系统举例：8086/8088指令系统

数的机器编码及其表示

- 需要解决的问题：
  - 正数/负数的表示
  - 整数/小数的表示
  - 0的表示
  - 整数与小数的混合表示
- 解决原则：**二进制表示一切**

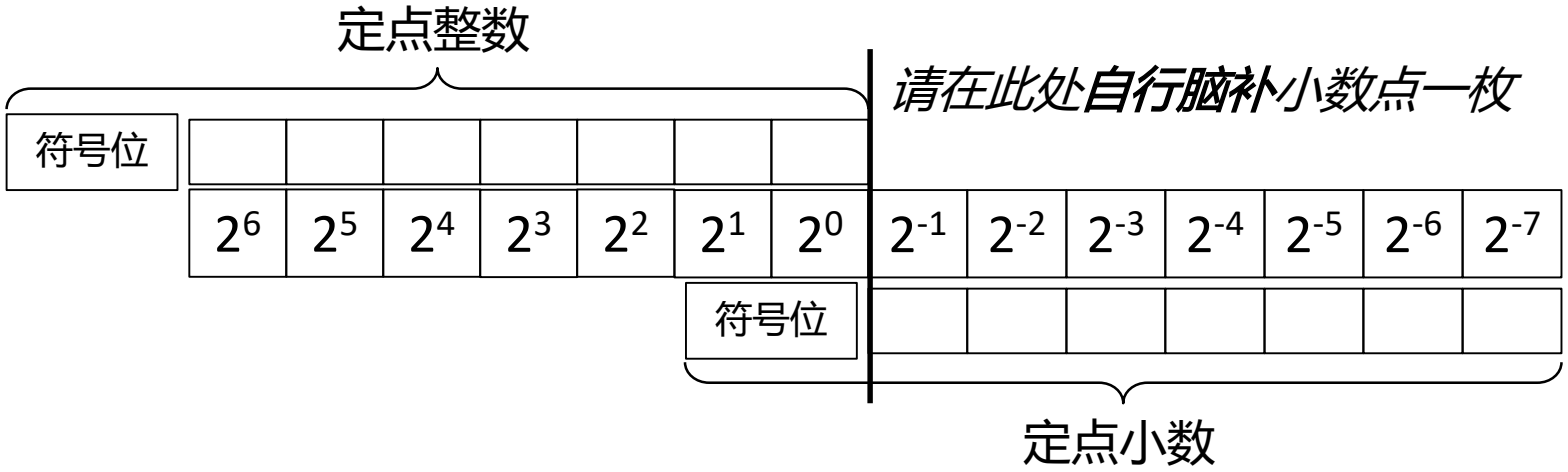
数的机器编码及其表示

- 问题1 -- 正数/负数的表示
  - 解决方法：将数的最高位bit定义为符号位
  - 符号位为0 -- 正数
  - 符号位为1 -- 负数



数的机器编码及其表示

- 问题2 -- 整数/小数的表示
  - 解决方法：二进制无法表示小数点，那么就**放弃表示小数点**
  - 定点（设小数点位置固定）整数 -- $x = a_{n-1}*2^{n-1} + a_{n-2}*2^{n-2} + \ldots + a_1*2^1 + a_0$
  - **定点小数** -- $y = a_{n-1}*2^{-1} + a_{n-2}*2^{-2} + \ldots + a_1*2^{-(n-1)} + a_0*2^{-n}$



数的机器编码及其表示

- 问题3 -- 0的表示
  - 符号位+数值0 → 两种表示（正0和负0）
  - 解决方式：采用补码实现数值的表示（补码中0的表示是统一的）
- 问题4 -- 整数与小数的混合表示
  - 解决方式：浮点数
  - 符号位+阶码k+尾数m → $x = m * 2^k$
- 若要用机器码表示十进制？--- BCD码/余3码等

指令系统

- 指令系统的作用：人指示计算机完成任务的手段和渠道
- 从穿孔纸带发展而来的计算机程序因而具有以下特点
  - 串行、顺序执行（做A → 做B → 做C）
  - 每个特定时间片内只能解读有限个比特

指令系统

- 具体到不同的指令系统：
  - 单次解读一条指令/一条指令的一个部分（复杂指令集）
  - 单次解读一条指令/多条指令（精简指令集）
- 指令顺序执行过程中的例外：
  - 指令的重复执行（循环）
  - 指令执行流的中断（子程序的调用、返回/程序中断）

指令系统

- 机器指令的要素：
  - 操作码（operation code）-- 指明进行何种操作，如mov、add等
  - 源操作数地址（source operand reference）-- 被操作对象的位置，可能有多个
  - 目的操作数地址（destination operand reference）-- 操作结果应置于何处
  - 下一指令地址（next instruction reference）-- 指出下一条指令的位置

  因为指令的顺序执行，故大多数指令无需明示此项，仅少数指令需要指明

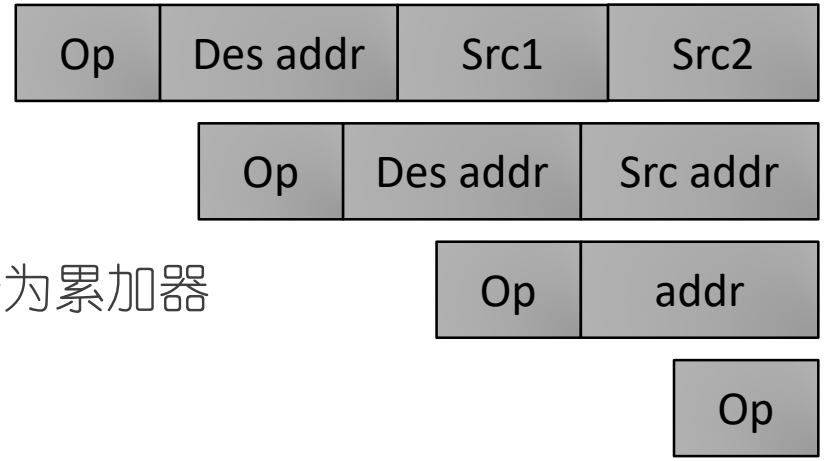
指令系统

- 操作数的可能来源：
  - 存储器/寄存器
  - 输入/输出端口
- 操作数的类型：
  - 地址（操作数地址、指令地址）
  - 数值
  - 字符
  - 逻辑型数据（true/false）

指令系统

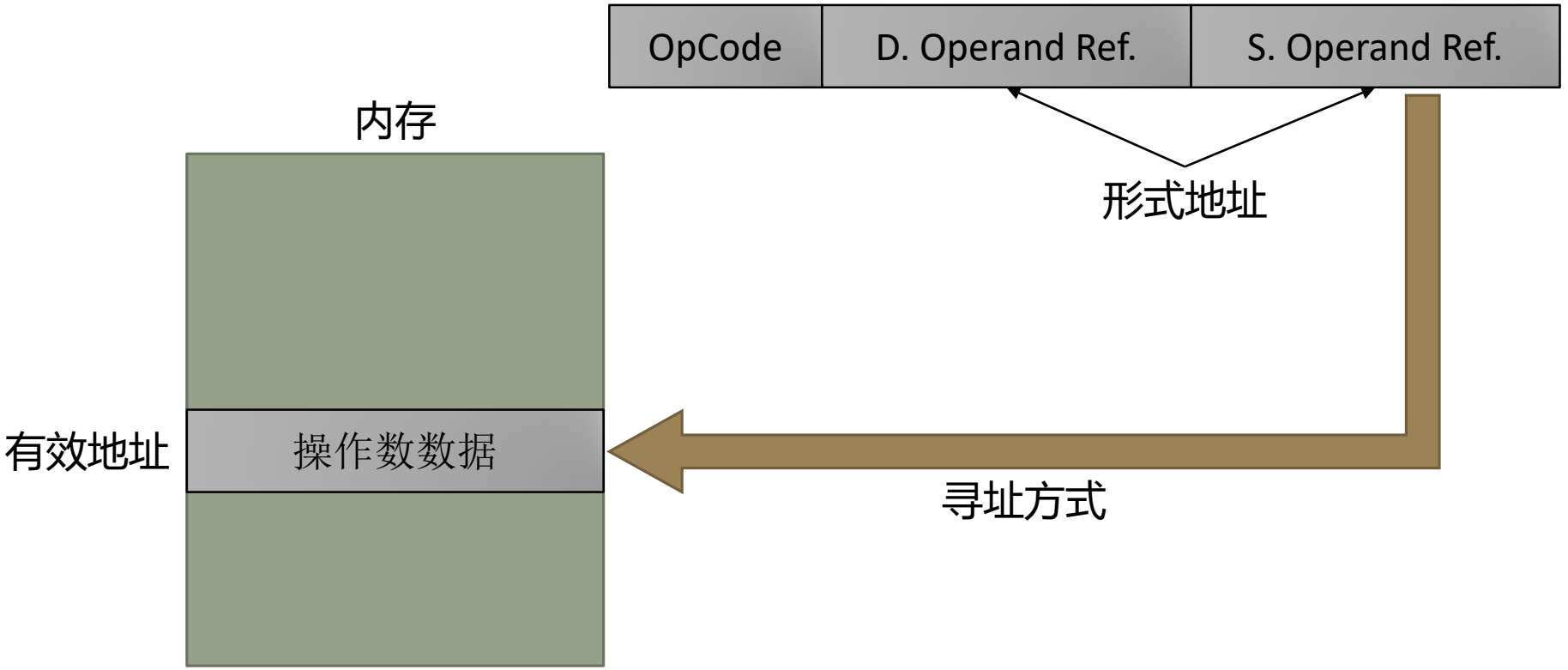
- 指令的表示：

- 地址的数目：
  - 三地址 -- Des ← (Src1) OP (Src2)
  - 双地址 -- Des ← (Src) OP (Des)
  - 单地址 -- 单操作数，或者双操作数中的一个为累加器
  - 无地址 -- 操作数隐含，或者无操作数



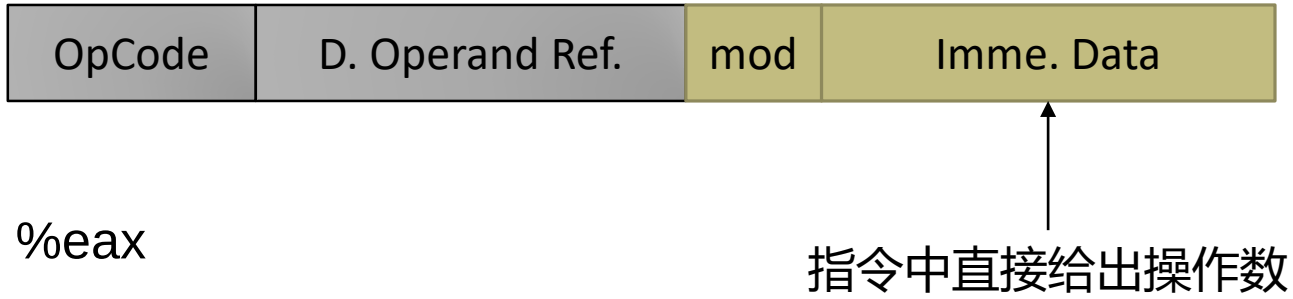
指令的寻址方式

- 形式地址与有效地址：



指令的寻址方式

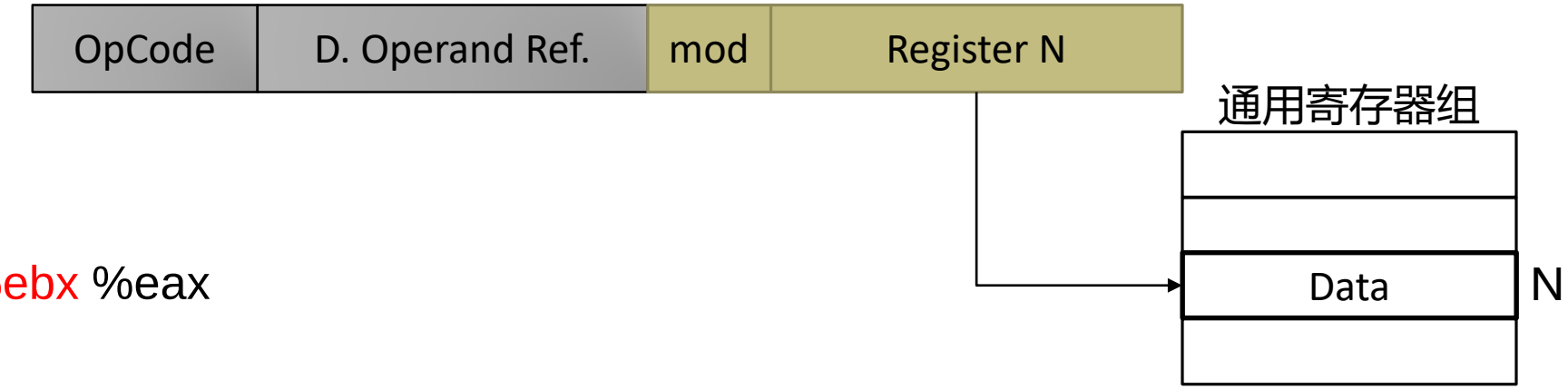
- 寻址方式
  - 定义：指令代码中地址字段的一部分，指明**操作数的获取方式**或**其地址的计算方式**
  - 指令中的每个地址字段（源、目的等）均有其寻址方式
- 立即寻址



  - 例：mov 10h %eax

指令的寻址方式

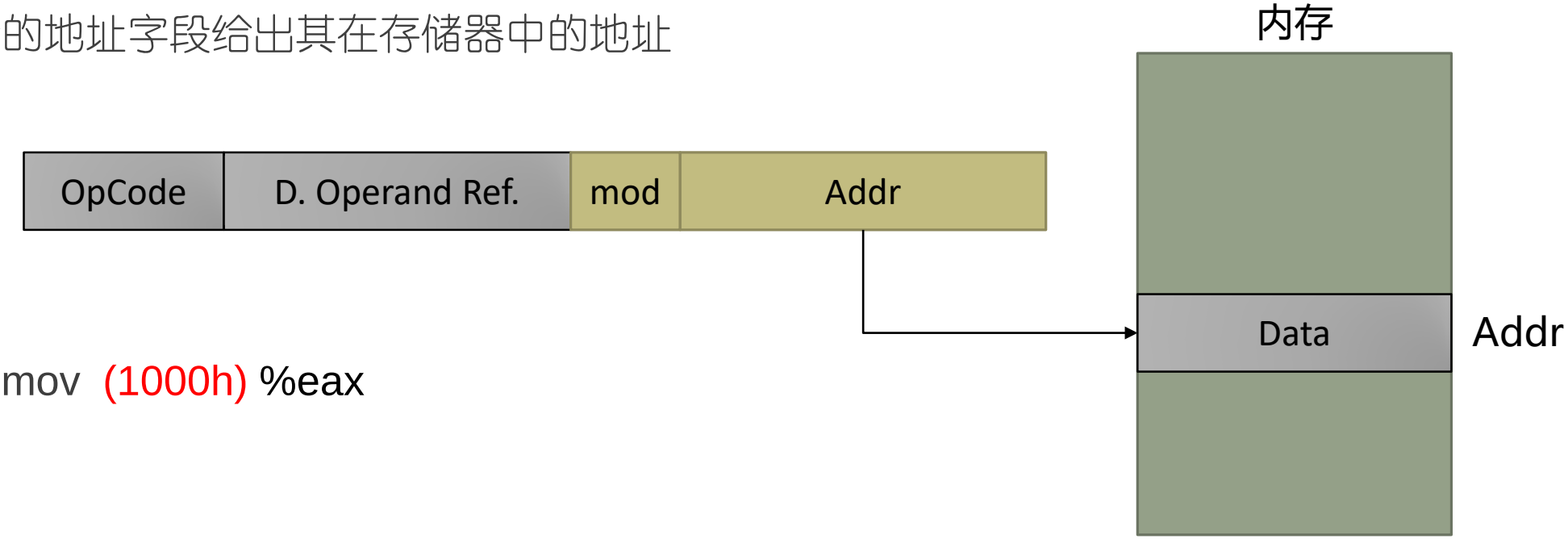
- 寄存器直接寻址
  - 操作数在寄存器中
  - 指令的地址字段给出寄存器的地址



  - 例： mov %ebx %eax

指令的寻址方式

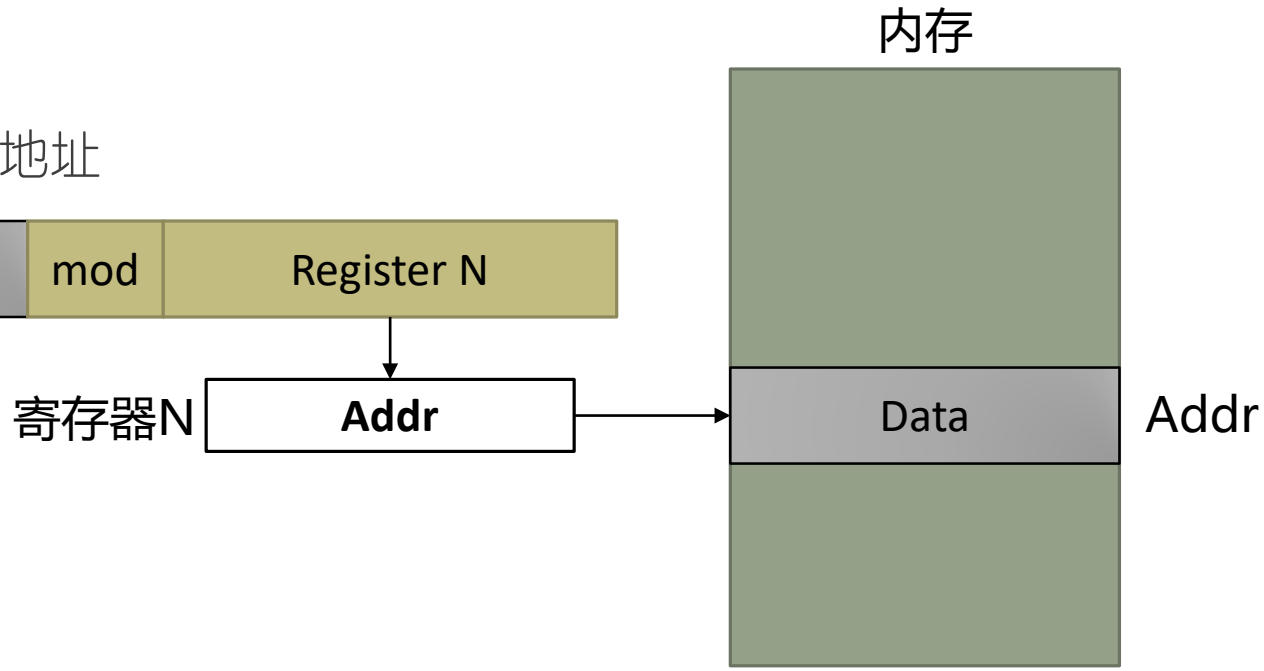
- 存储器直接寻址
  - 操作数在存储器中
  - 指令的地址字段给出其在存储器中的地址



  - 例：mov (1000h) %eax

指令的寻址方式

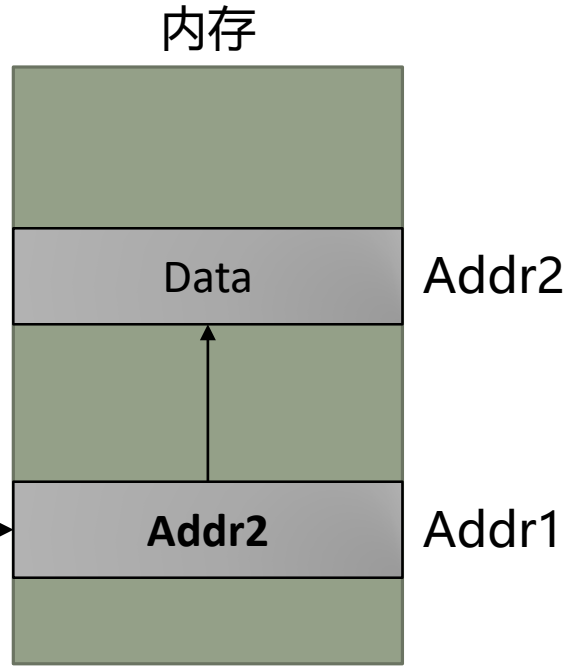
- 寄存器间接寻址
  - 操作数在存储器中
  - 指令中给出寄存器地址
  - 该寄存器的内容是操作数的存储器地址



  - 例： mov (%edx) %eax

指令的寻址方式

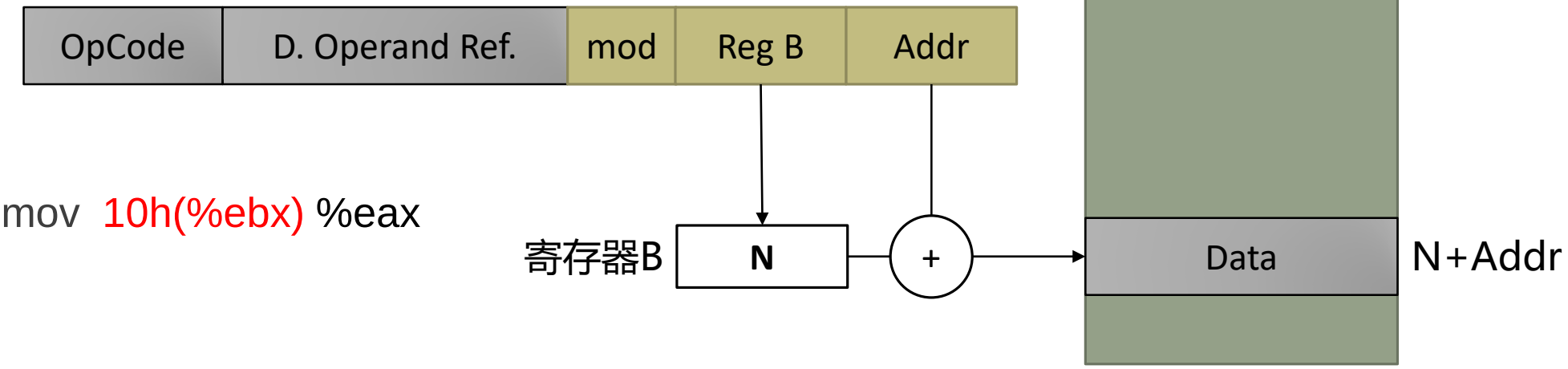
- 存储器间接寻址
  - 操作数在存储器中
  - 指令中给出一个存储器地址
  - 该地址处所存放的内容是操作数的存储器地址



  - 例： mov @(1000h) %eax

指令的寻址方式

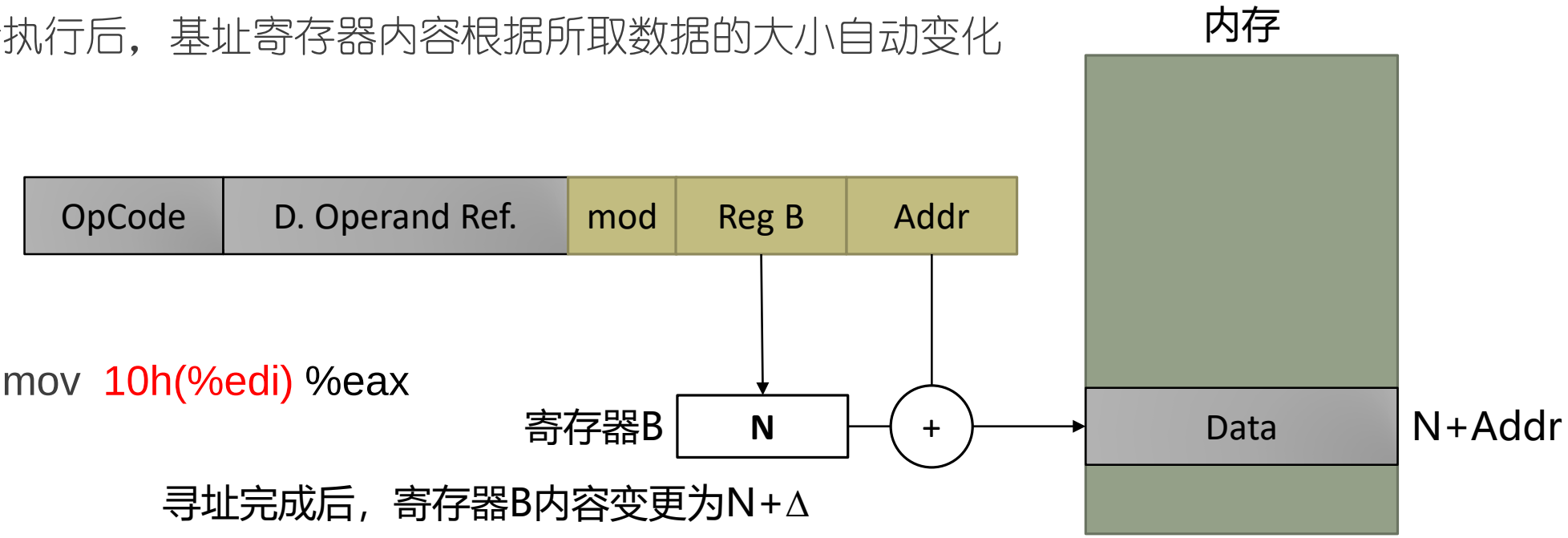
- 基址寻址
  - 操作数在存储器中
  - 指令中给出一个基址寄存器和一个形式地址
  - 基址寄存器内容与形式地址之和是操作数的存储器地址



  - 例： mov 10h(%ebx) %eax

指令的寻址方式

- 变址寻址
  - 前同基址寻址
  - 指令执行后，基址寄存器内容根据所取数据的大小自动变化
  - 例：mov 10h(%edi) %eax



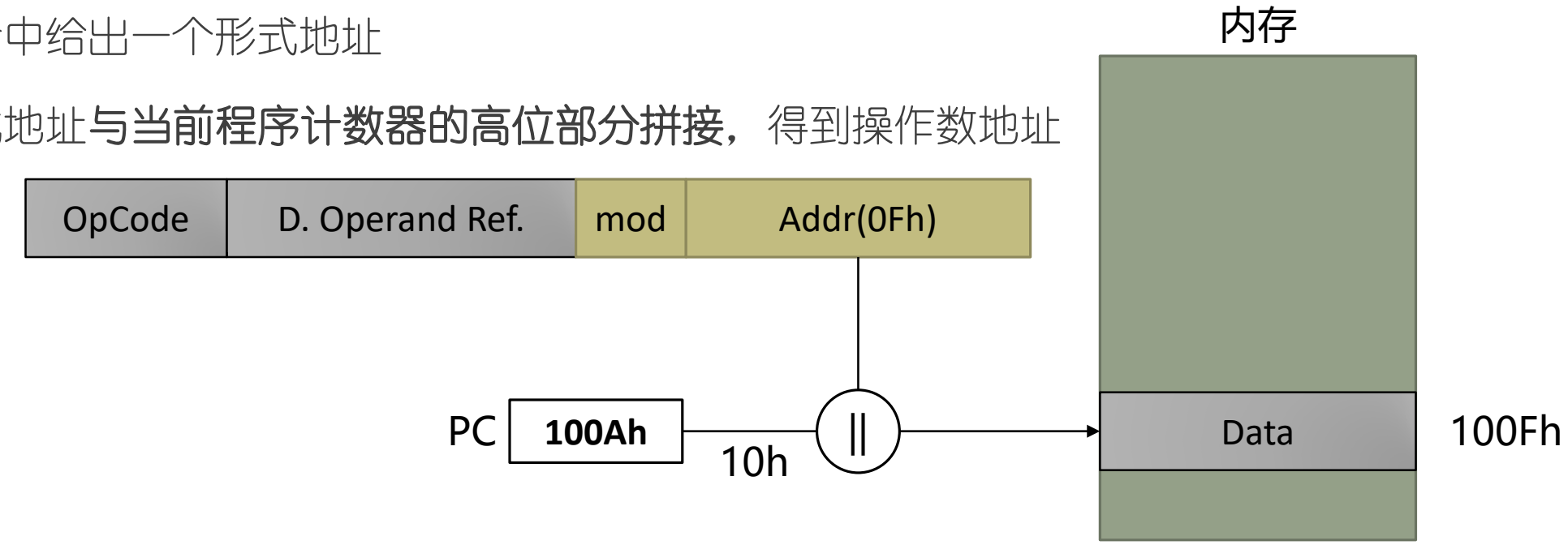
指令的寻址方式

- 相对寻址
  - 操作数在存储器中
  - 指令中给出一个形式地址
  - 形式地址与当前程序计数器之和是操作数的存储器地址

  - 例： jne addr

指令的寻址方式

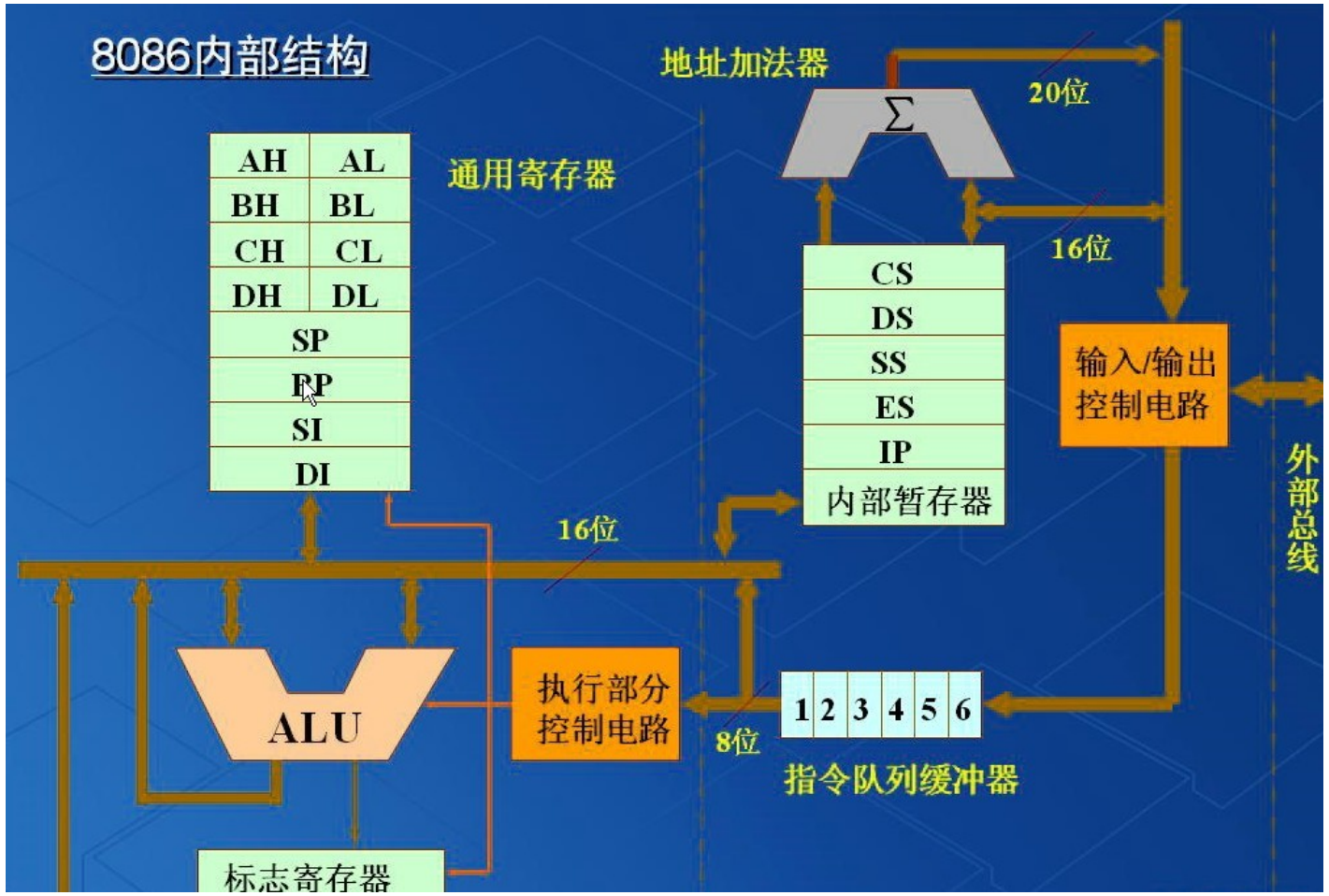
- 页面寻址
  - 操作数在存储器中
  - 指令中给出一个形式地址
  - 形式地址**与当前程序计数器的高位部分拼接，**得到操作数地址



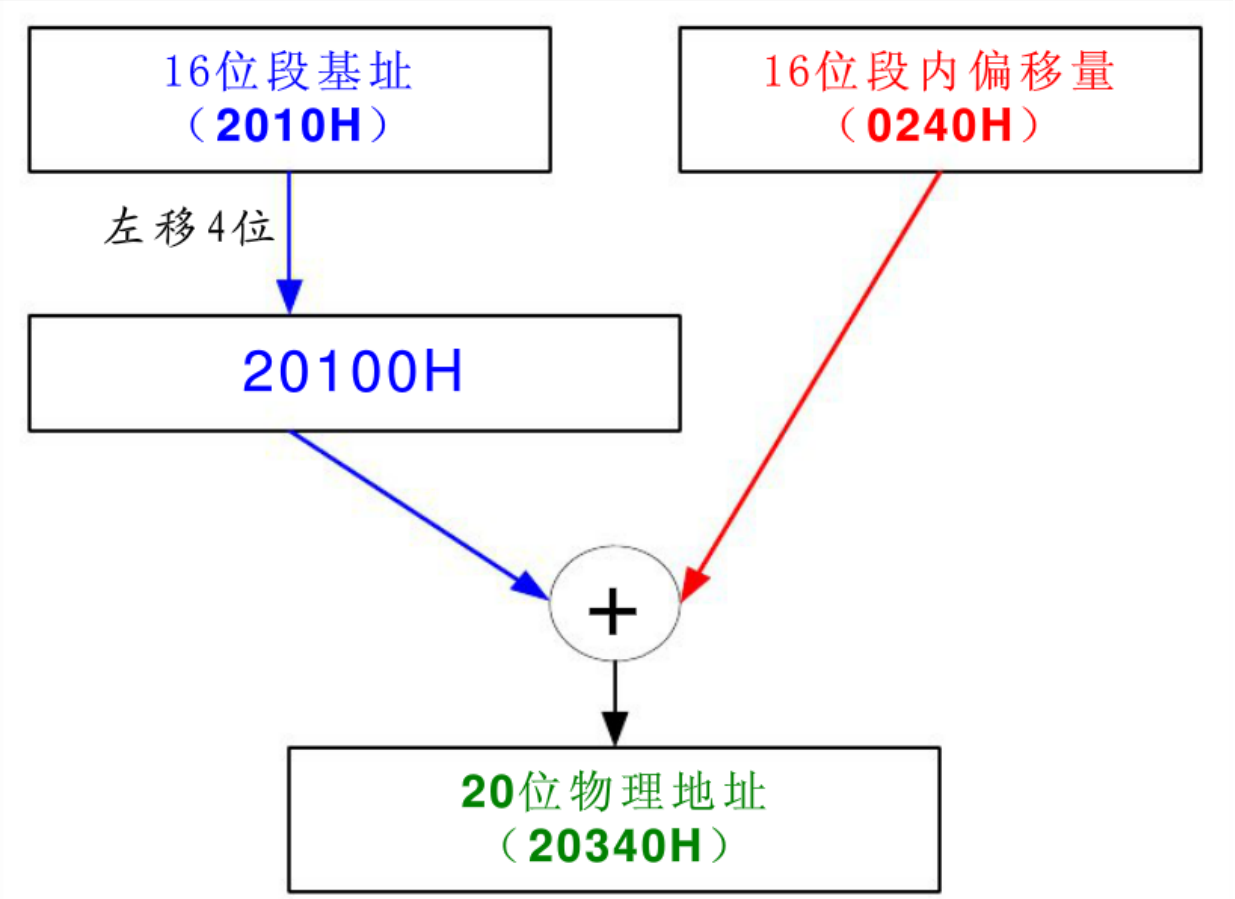
指令的寻址方式

- 栈寻址
  - 栈的结构：进程空间中的一段内存区域
  - 具有栈底（一个基地址）、栈顶
  - 栈指针（SP）：一个特殊目的寄存器，指向栈顶
- 栈操作：
  - 压栈 -- 寄存器 → 栈顶内存单元，写入开始前改写SP（减小），分配压栈所需空间
  - 出栈 -- 栈顶内存单元 → 寄存器，读取完成后改写SP（增大），释放被出栈的内存

指令系统举例：8086/8088

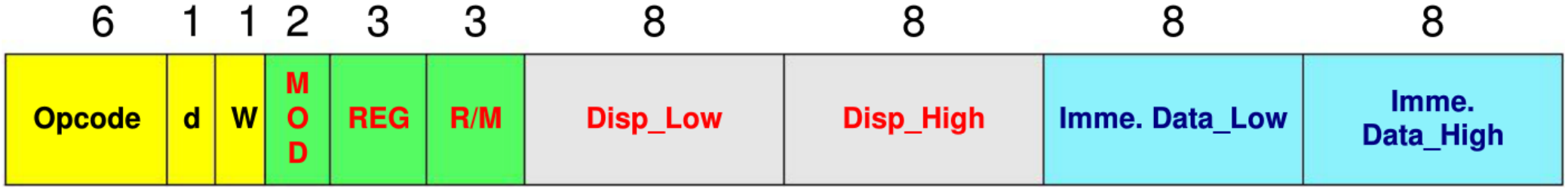


指令系统举例：8086/8088



指令系统举例：8086/8088

- 一般双操作数指令的格式与编码
  - R-R型或者R-S型，即至少应有一个操作数来自寄存器
  - 2~6字节长（其中2字节为必须）
  - 操作码长6位，结合方向字段d（1位）、字/字节字段w（1位）组成指令的首字节
  - 第二字节中，REG确定一个操作数（寄存器直接寻址）
  - 第二字节中， MOD和R/M字段共同指出另一个操作数的寻址方式
  - d=1时，REG确定目的操作数， MOD+R/M确定源操作数，vice versa
  - w=1时，操作数宽度为字（16位），否则为字节（8位）



指令系统举例：8086/8088

寄存器编码表

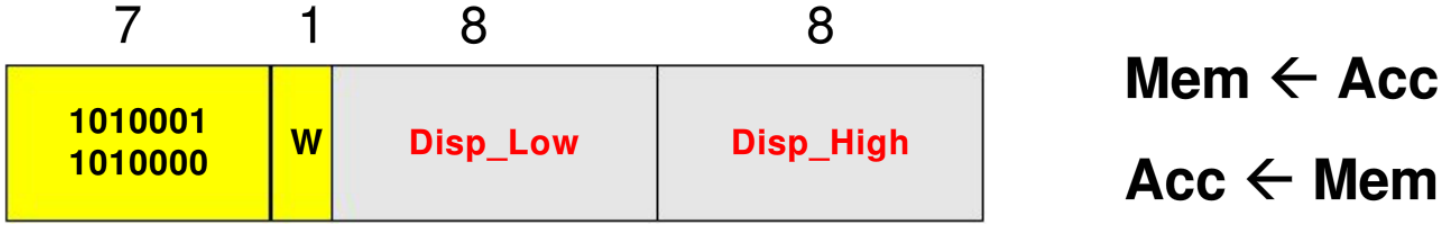
REG	W=1	W=0
000	AX	AL
001	CX	CL
010	DX	DL
011	BX	BL
100	SP	AH
101	BP	CH
110	SI	DH
111	DI	BH

指令系统举例：8086/8088

R/M	存储器操作数有效地址(EA)			寄存器操作数	
	MOD=00	MOD=01	MOD=10	MOD=11	
				W=1	W=0
000	(BX)+(SI)	(BX)+(SI)+Disp8	(BX)+(SI)+Disp16	AX	AL
001	(BX)+(DI)	(BX)+(DI)+Disp8	(BX)+(DI)+Disp16	CX	CL
010	(BP)+(SI)	(BP)+(SI)+Disp8	(BP)+(SI)+Disp16	DX	DL
011	(BP)+(DI)	(BP)+(DI)+Disp8	(BP)+(DI)+Disp16	BX	BL
100	(SI)	(SI)+Disp8	(SI)+Disp16	SP	AH
101	(DI)	(DI)+Disp8	(DI)+Disp16	BP	CH
110	Disp16	(BP)+Disp8	(BP)+Disp16	SI	DH
111	(BX)	(BX)+Disp8	(BX)+Disp16	DI	BH

指令系统举例：8086/8088

- 与累加器相关的双操作数指令
  - AX（AL）：寄存器直接寻址
  - 另一操作数：存储器直接寻址/立即数
  - 采用特定操作码，与一般双操作数指令相区别



Mem ← Acc

Acc ← Mem

- 单操作数指令

指令系统举例：8086/8088

- 指令的类型
  - 数据传送（move / store / load / exchange / set / clear）
  - 算术运算
  - 逻辑运算（and / or / not / xor / compare / test）
  - 移位指令
  - **程序控制类指令**
  - 串操作指令
  - I/O指令
  - 栈操作指令

指令系统举例：8086/8088

- 程序控制类指令
  - 几个重要寄存器：程序计数器（对应IP）、程序状态字（对应FLAGS）、栈指针SP
  - 控制转移指令：无条件跳转/条件跳转
  - 循环控制指令（loop）
  - 子程序调用/返回（call / ret）
  - 程序中断和返回（int / iret）

指令系统举例：8086/8088

- 控制转移指令
  - 无条件跳转可以为短跳转/近跳转/远跳转
  - 短跳转（偏移量8bit）/近跳转（偏移量16bit）为段内控制转移，CS不变，IP改变
  - 远跳转时，CS和IP均改变
  - 条件跳转仅存在段内短跳转一种情况

指令系统举例：8086/8088

- 子程序调用/返回指令
  - 既可以是段内近调用（返回），也可以是段间远调用（返回）
- 调用中的保护现场：
  - 近调用时将IP当前值压栈
  - 远调用时将CS、IP当前值均压栈（CS先于IP）
- 返回时的恢复现场：
  - 近返回时，仅将IP出栈
  - 远返回时，将IP、CS依次出栈（IP先于CS）

指令系统举例：8086/8088

- 子程序的定义：

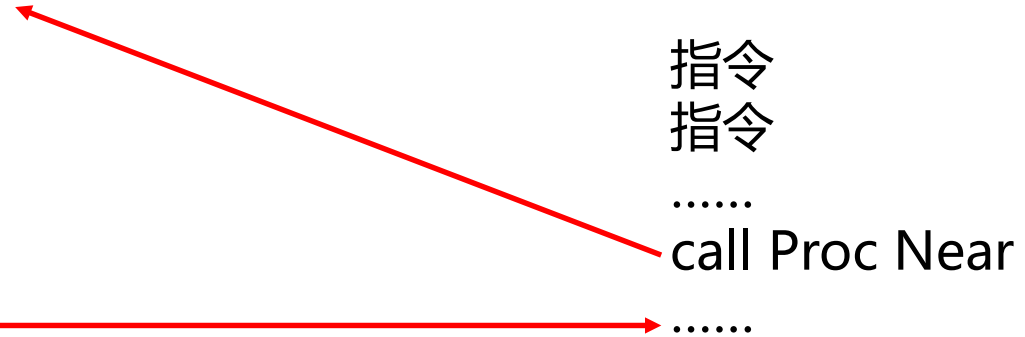
Label Proc Near

    指令
    指令
    ……
    retn

Label endProc

- 子程序的调用：

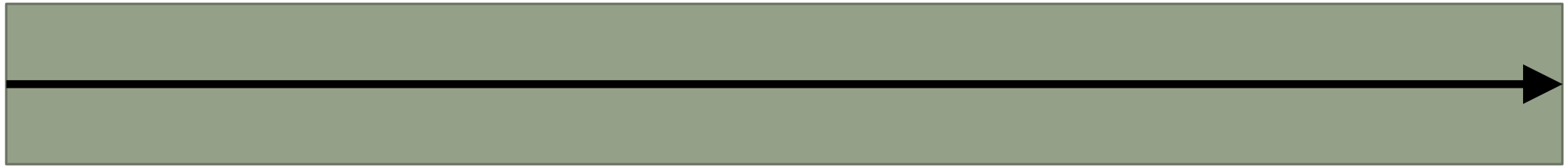
    指令
    指令
    ……
    call Proc Near
    ……
    指令



注：标签Proc Near和endProc并不实际占用空间，它们仅仅是代码内的两个地址

指令系统举例：8086/8088

- 理想中的程序执行：



- 程序执行的实际情况：