Computer Organization and Design

中山大学

计算机学院

郭雪梅

Email:guoxuem@mail.sysu.edu.cn

Operands and Addressing Modes 操作数和寻址方式

- 数据放在哪?
- 是数据地址?
- 是名字表示还是值?

- 间接寻址?

Reading: Ch. 2.3, 2.10,2.14

指令格式

※机器指令的要素

- 操作码(Operation Code)：指明进行的何种操作（如 ADD, I/O）
- 源操作数地址(Source Operand Reference)：参加操作的操作数的地址，可能有多个。
- 目的操作数地址(Destination Operand Reference)：保存操作结果的地址。
- 下一条指令的地址(Next Instruction Reference)：指明下一条要运行的指令的位置，一般指令是按顺序依次执行的，所以绝大多数指令中并不显示的指明下一条指令的地址，也就是说，指令格式中并不包含这部分信息。只有少数指令需要显示指明下一条指令的地址。

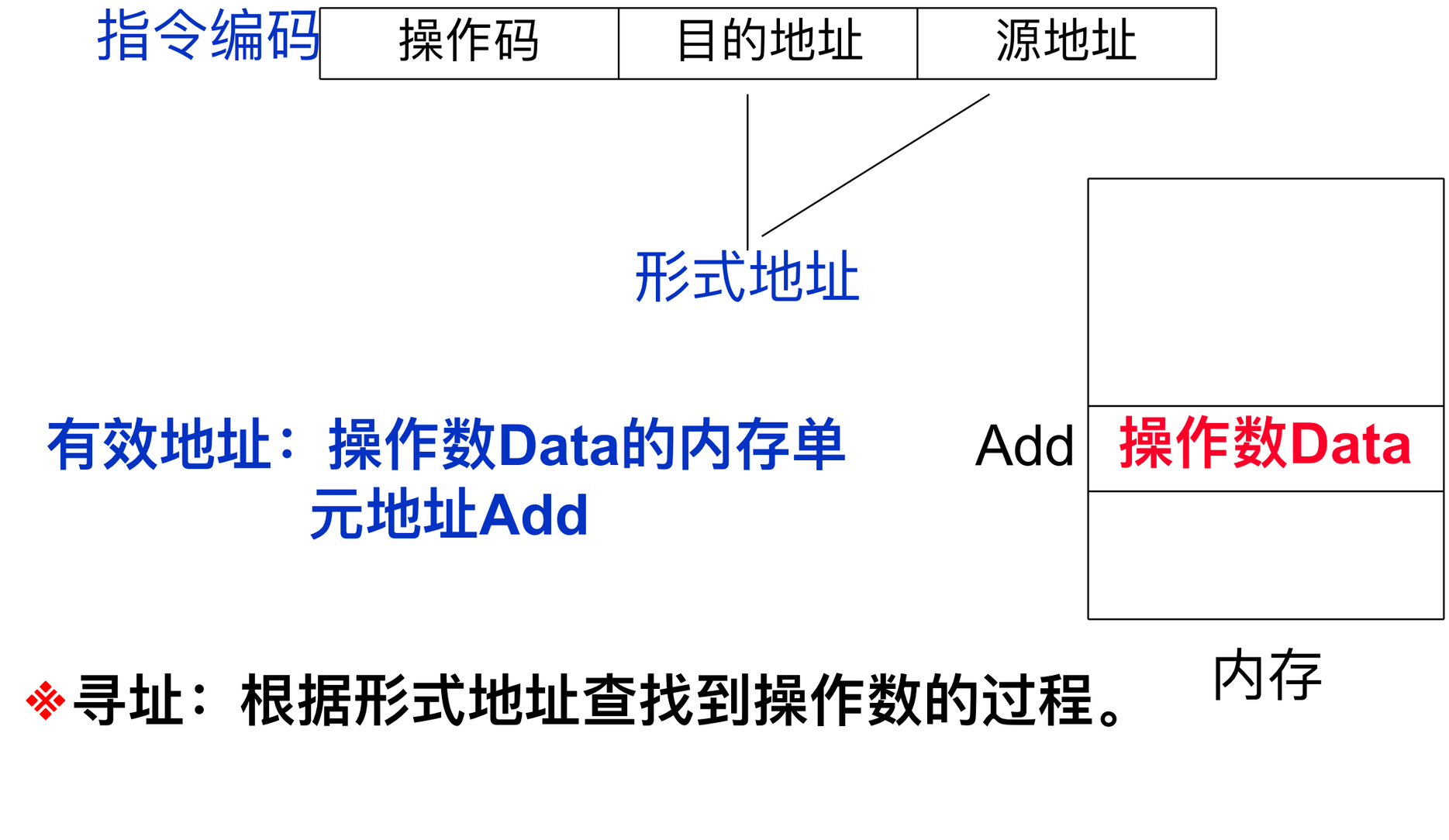
※操作数的来源

- 存储器（存储器地址）
- 寄存器（寄存器地址）
- 输入输出端口（输入输出端口地址）

寻址方式

❋形式地址与有效地址

- ➤形式地址：指令中直接给出的地址编码。
- ➤有效地址：根据形式地址和寻址方式计算出来的操作数在内存单元中的地址。



有效地址：操作数Data的内存单元地址Add

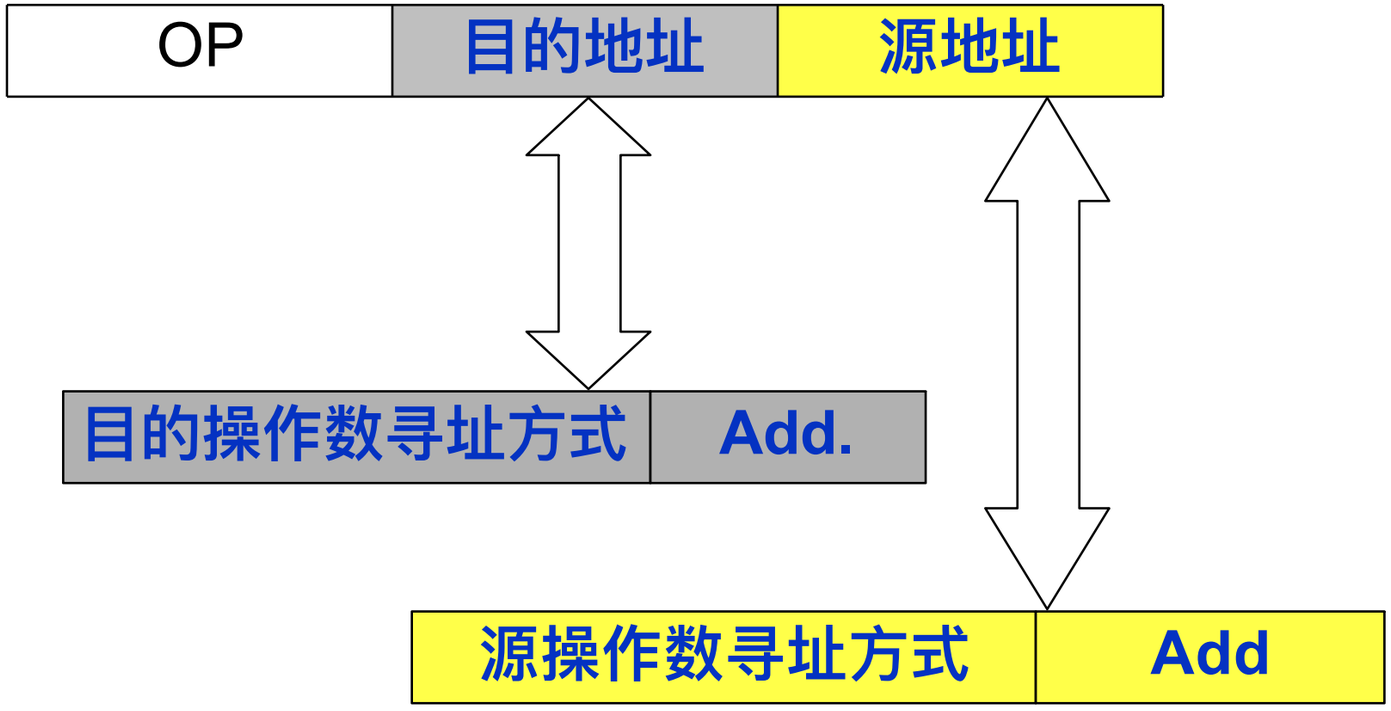
❋寻址：根据形式地址查找到操作数的过程。

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操作数地址的计算方式。

➤指令中每一个地址字段均有其寻址方式编码（或隐含寻址方式）



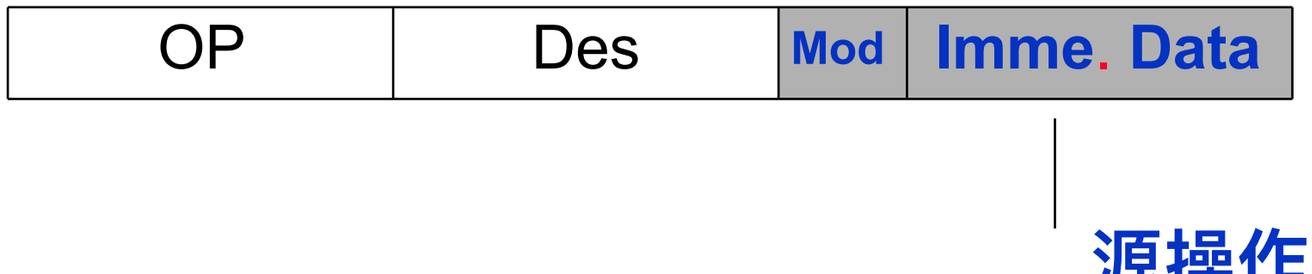
寻址方式

- 指令代码和寻址描述中有关缩写的约定
  - OP：操作码
  - Des：目的操作数地址
  - Sur：源操作数地址
  - A或Add： 形式地址（内存地址）
  - Mod：寻址方式
  - Rn： 通用寄存器
  - Rx： 变址寄存器
  - Rb： 基址寄存器
  - SP：堆栈指针（寄存器）
  - EA： 有效地址
  - Data： 操作数
  - Operand： 操作数
  - （X）：表示对象X的内容（值），如（Rn）表示寄存器Rn的内容（值），（A）：内存中地址为A的单元的内容。
  - Imme. Data： 立即数
  - XXH：16进制数XX

寻址方式

❋立即寻址 (Immediate)

- 操作数直接在指令代码中给出。



❋说明

- 立即寻址只能作为双操作数指令的源操作数。
- Operand = Imme. Data
- 例：MOV AX,1000H

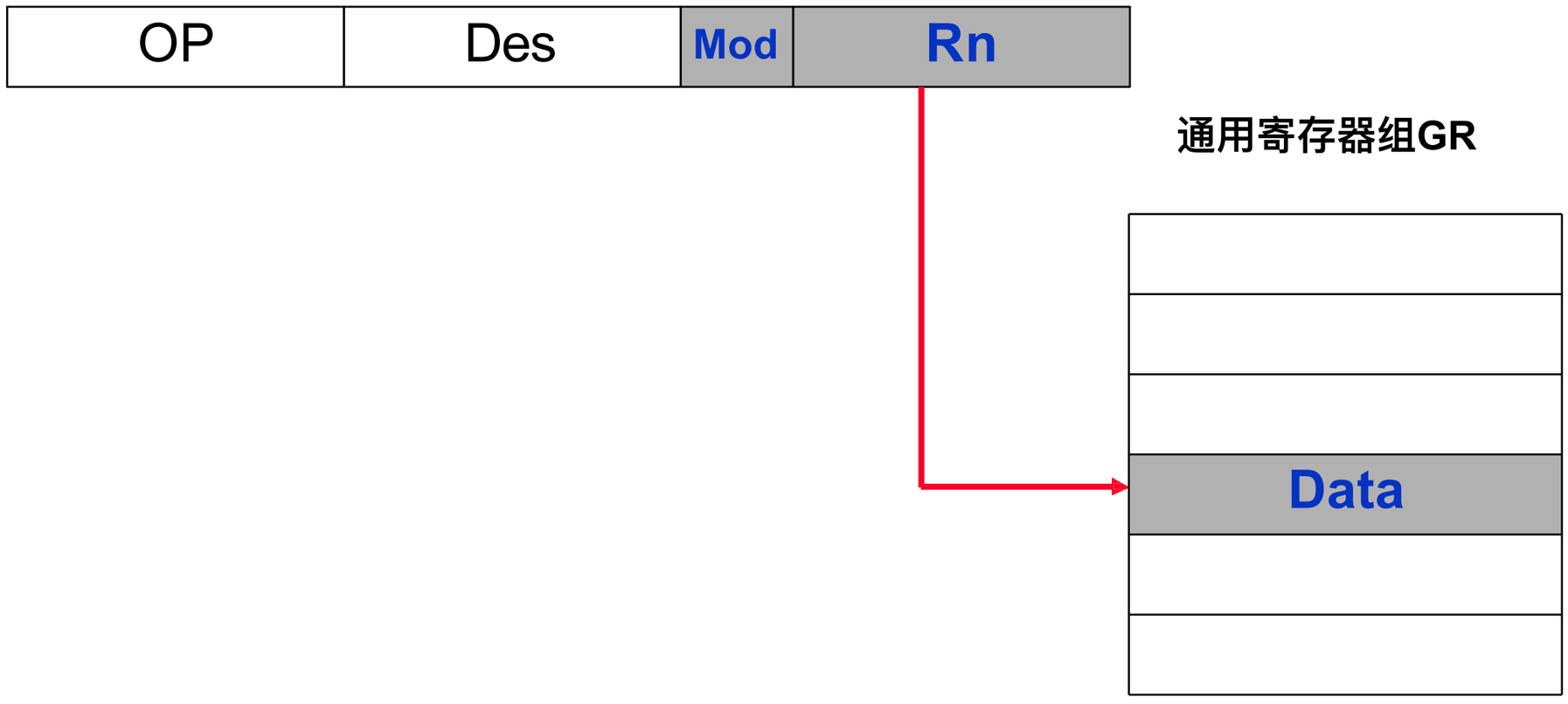
❋思考

- 立即寻址的操作数在什么地方，存储器 or 寄存器?
- 立即数的地址?

寻址方式

❋寄存器直接寻址(Register)

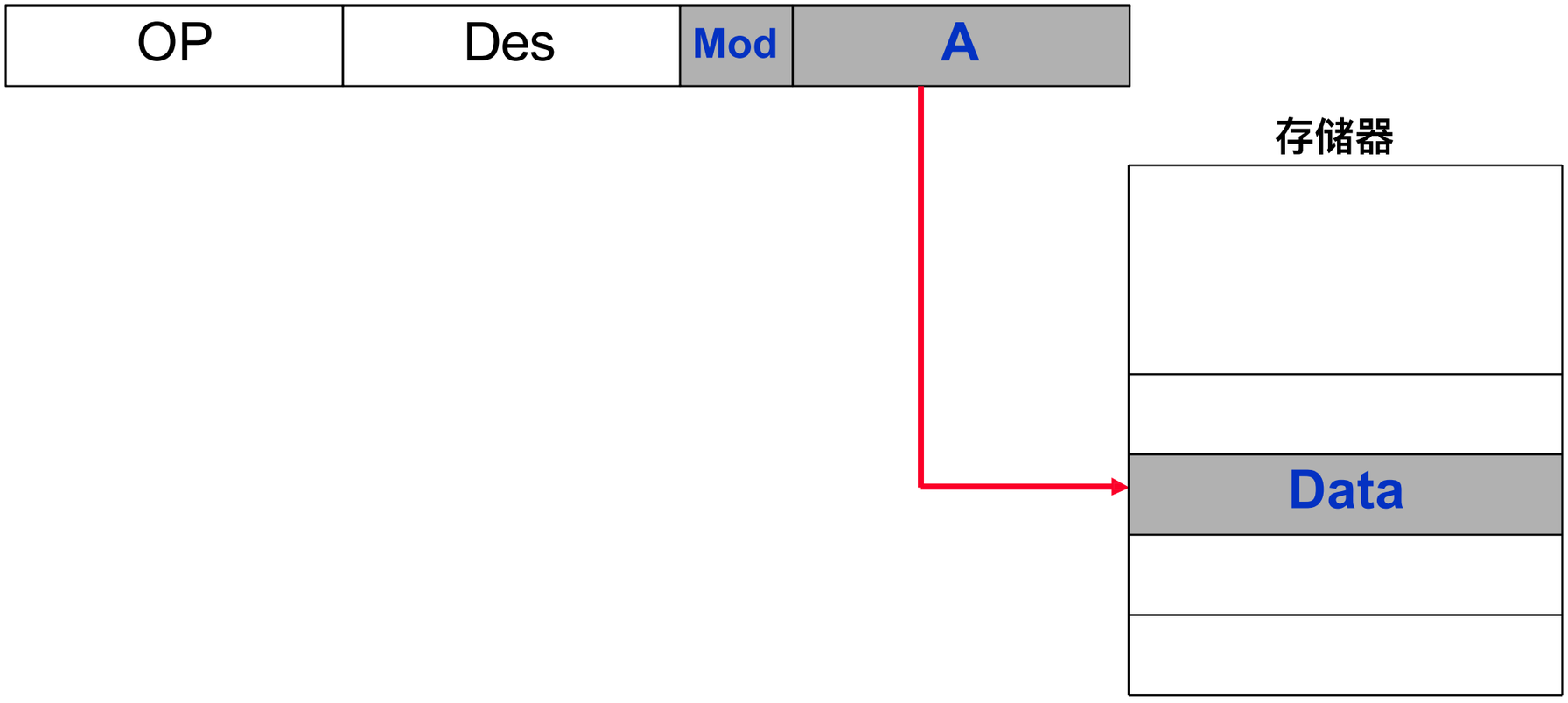
- ➤操作数在寄存器中，指令地址字段给出寄存器的地址（编码）
- ➤EA = Rn, Operand = (Rn)
- ➤例：MOV [BX], AX



寻址方式

❖存储器直接寻址 (Absolute)

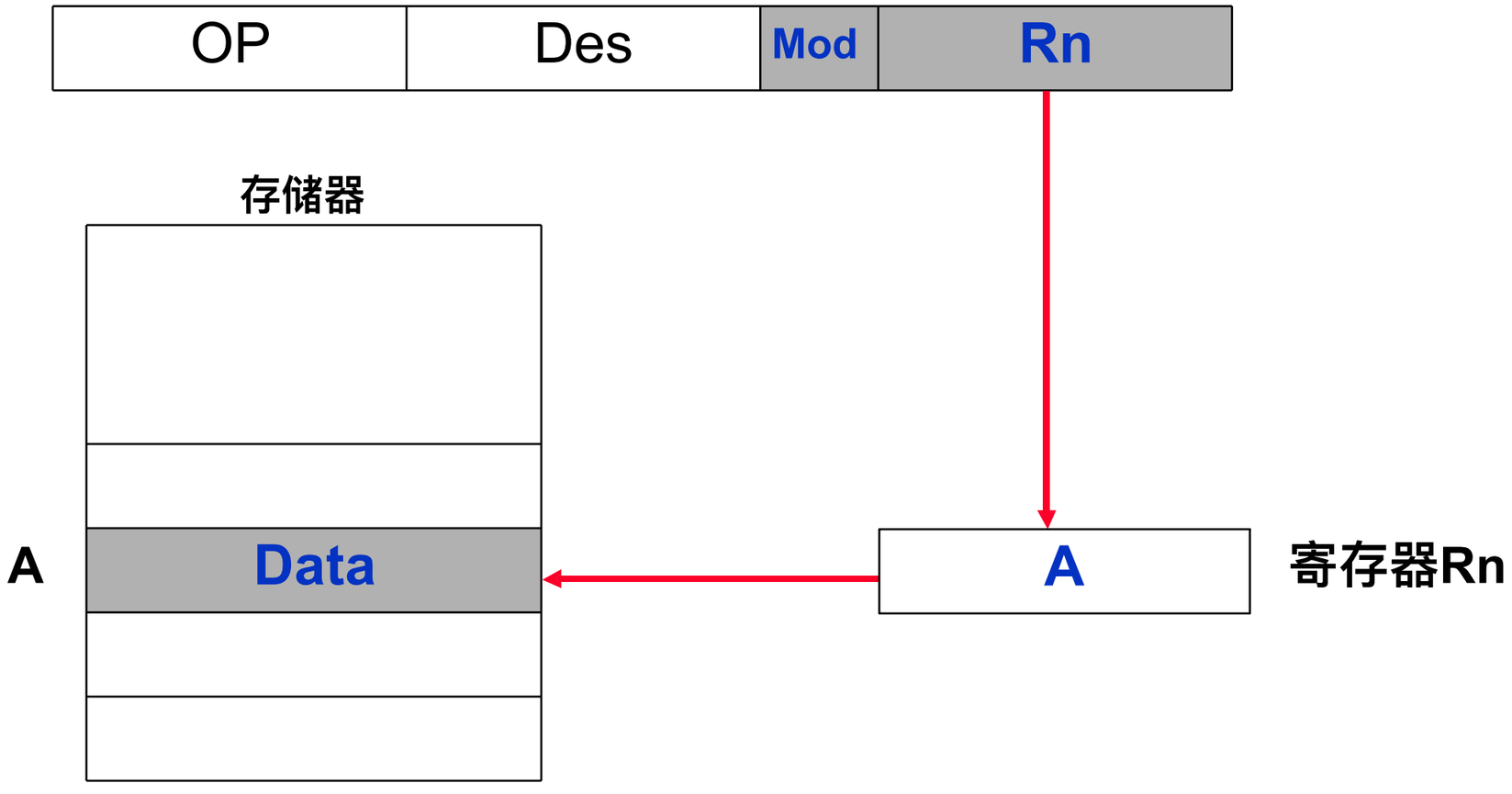
- ➤操作数在存储器中，指令地址字段直接给出操作数在存储器中的地址
- ➤EA = A, Operand = (A)
- ➤例：MOV AX, [1000H]



寻址方式

※寄存器间接寻址(Reg indirect)

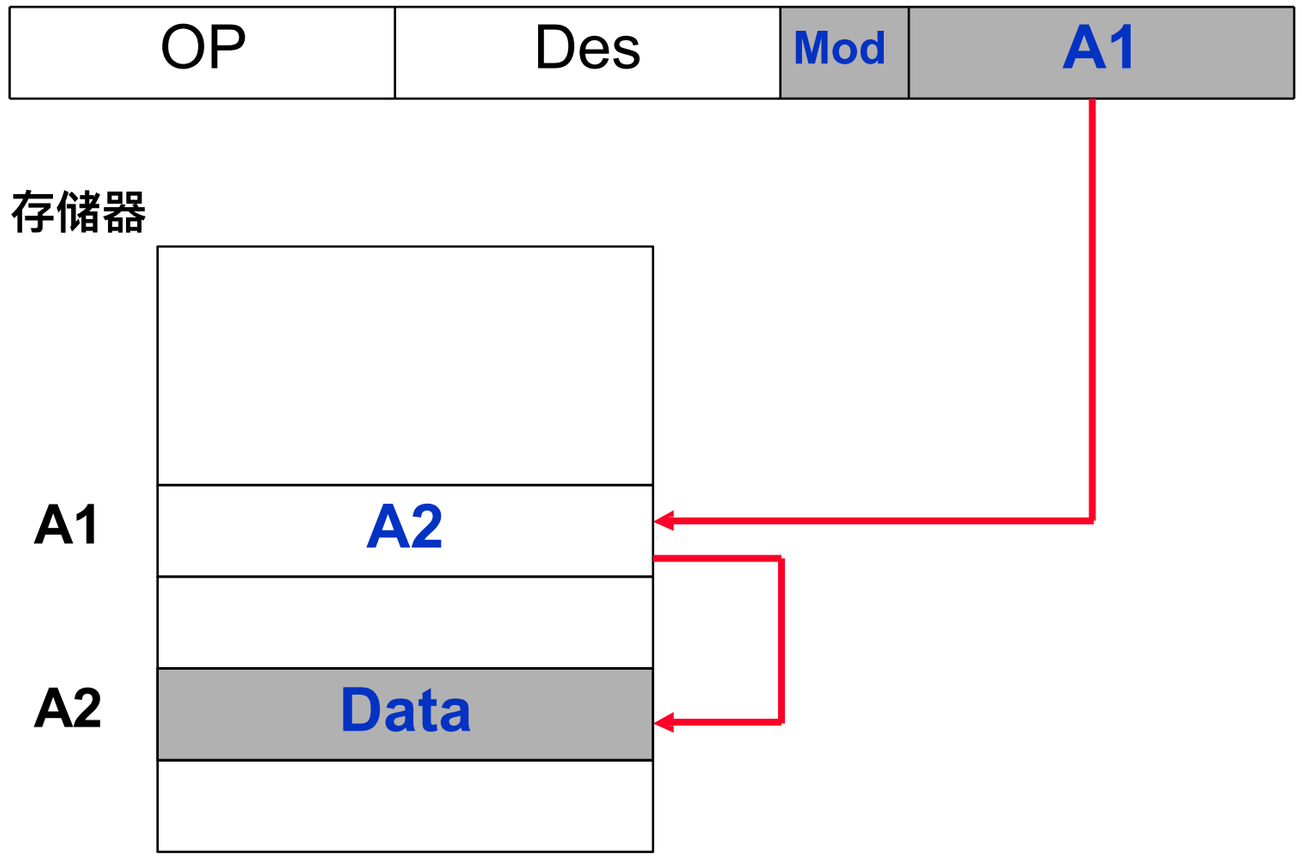
- 操作数在存储器中，指令地址字段中给出的寄存器的内容是操作数在存储器中的地址。
- EA = (Rn), Operand = ((Rn))
- 例：MOV AX, [BX]



寻址方式

❖存储器间接寻址

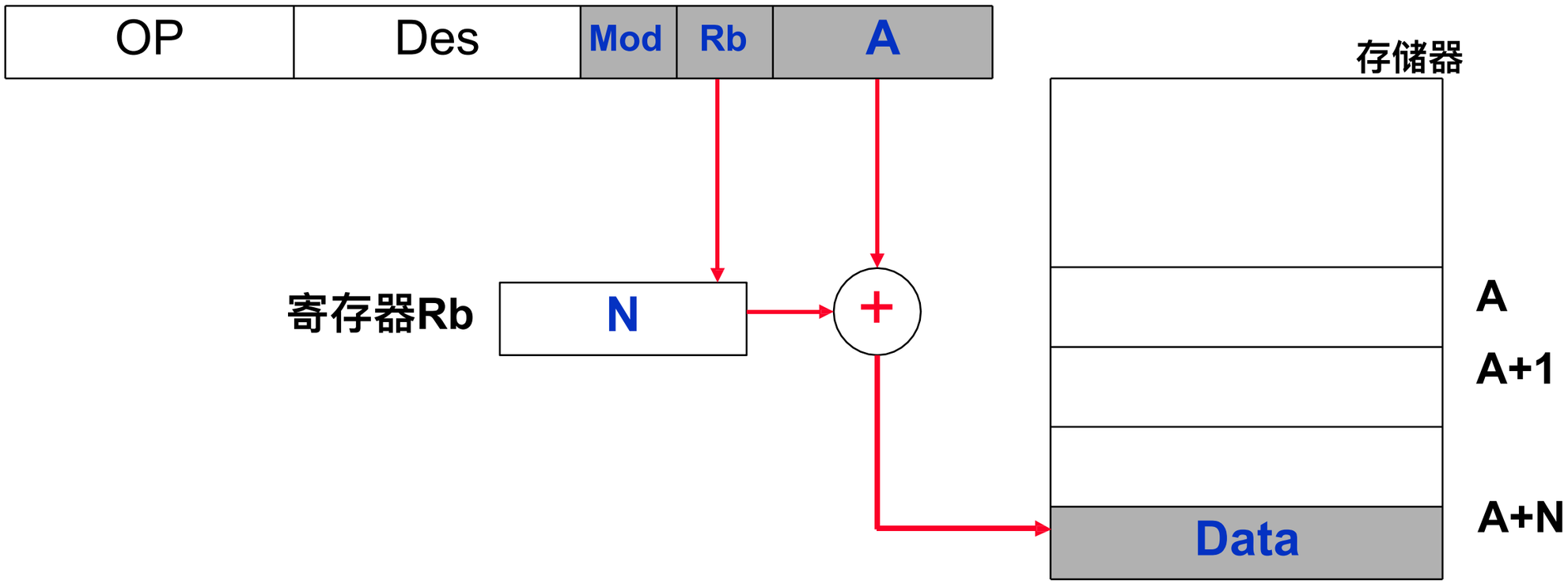
- 操作数在存储器中，指令地址字段中给出的存储器地址的单元内容是操作数在存储器中的地址。
- EA = (A1), Operand = ((A1))
- 例：MOV　　R1, @(1000H)　　PDP-11的指令



寻址方式

❖基址寻址(Displacement)

- ➤操作数在存储器中，指令地址字段给出一基址寄存器和一形式地址，基变址寄存器的内容与形式地址之和是操作数的内存地址。
- ➤EA = (Rb)+A, Operand = ((Rb)+A)
- ➤例：MOV　　AX, 1000H[BX]



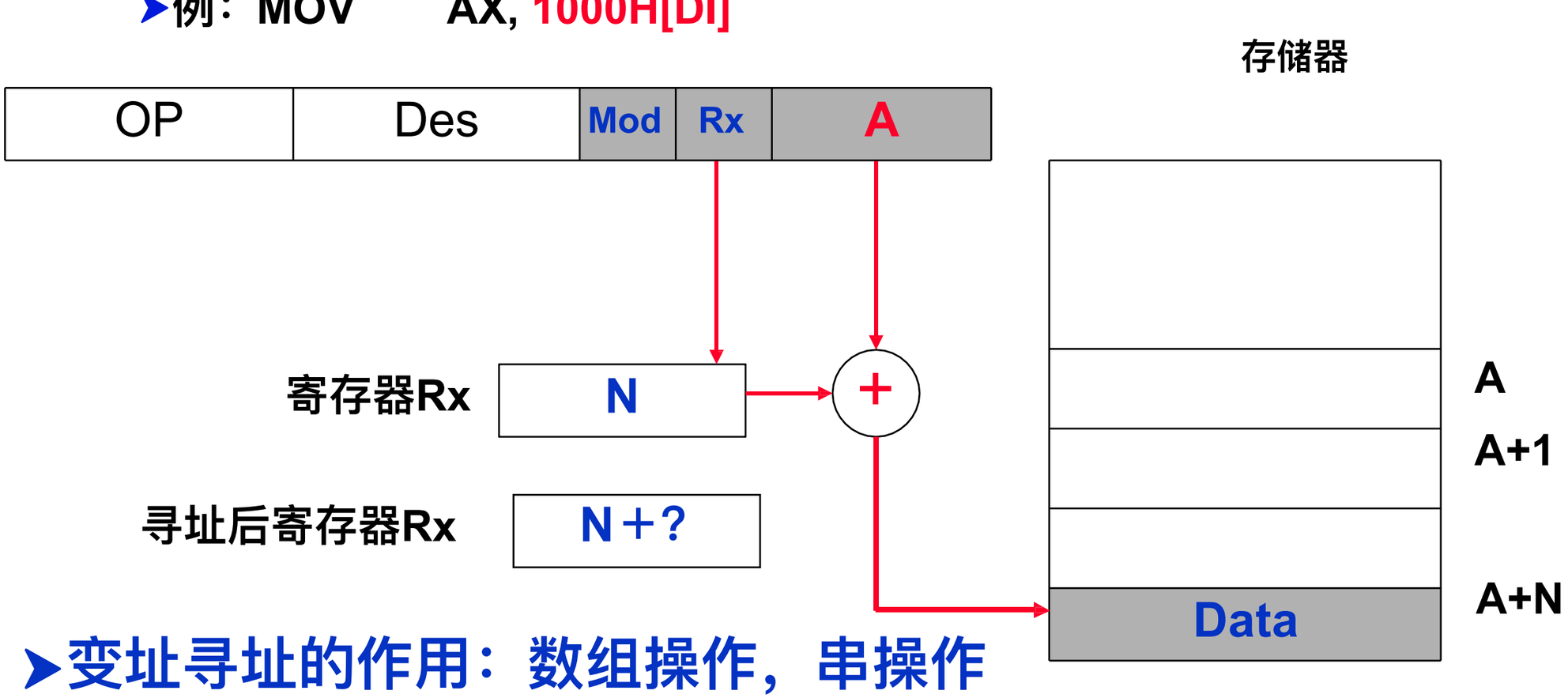
基址寻址的作用：较短的形式地址长度可以实现较大的存储空间的寻址。

寻址方式

❊变址寻址

- 操作数在存储器中，指令地址字段给出一变址寄存器和一形式地址，变址寄存器的内容与形式地址之和是操作数的内存地址。
- EA = (Rx)+A, Operand = ((Rx)+A)
- 有的系统中，变址寻址完成后，变址寄存器的内容将自动进行调整。

  Rx= (Rx) + ?? 操作数Data的字节数?
  - 例：MOV　　AX, 1000H[DI]

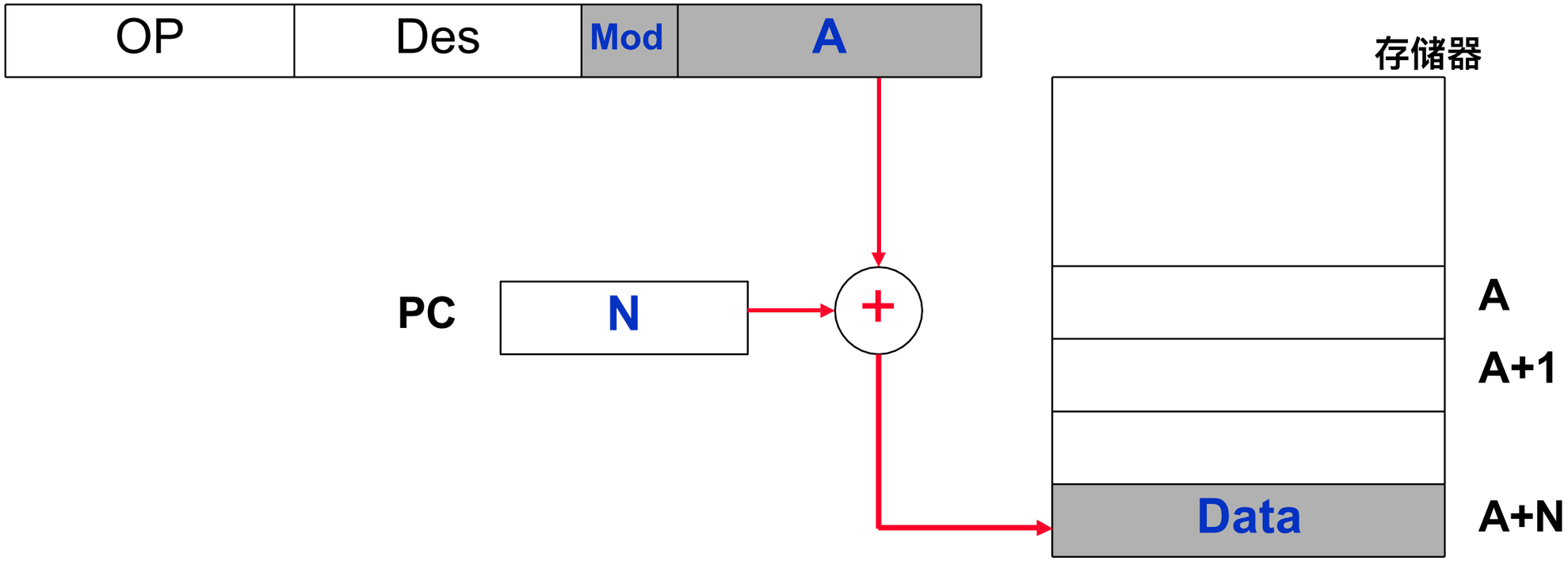


- 变址寻址的作用：数组操作，串操作

寻址方式

❋相对寻址

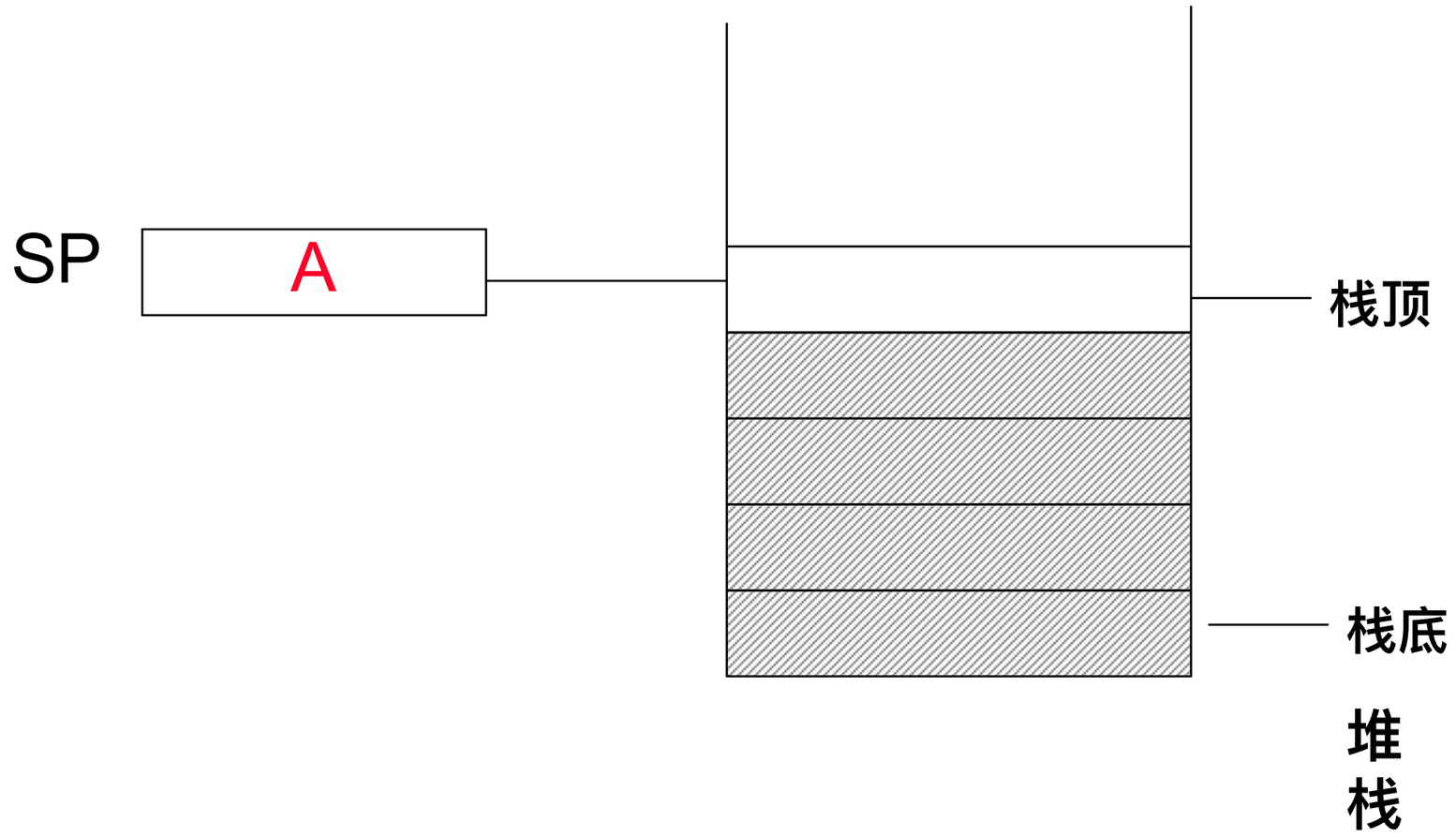
- ➤基址寻址的特例，由程序计数器PC作为基址寄存器，指令中给出的形式地址作为位移量，二者之和是操作数的内存地址。
- ➤EA = (PC)+A, Operand = ((PC)+A)
- ➤例：JNE　　A



寻址方式

❖堆栈寻址

- ➤堆栈的结构：一段内存区域。
- ➤栈底，栈顶，
- ➤堆栈指针(SP)：是一个特殊寄存器部件, 指向栈顶
- ➤堆栈操作：PUSH (从寄存器到堆栈)，POP (从堆栈到寄存器)

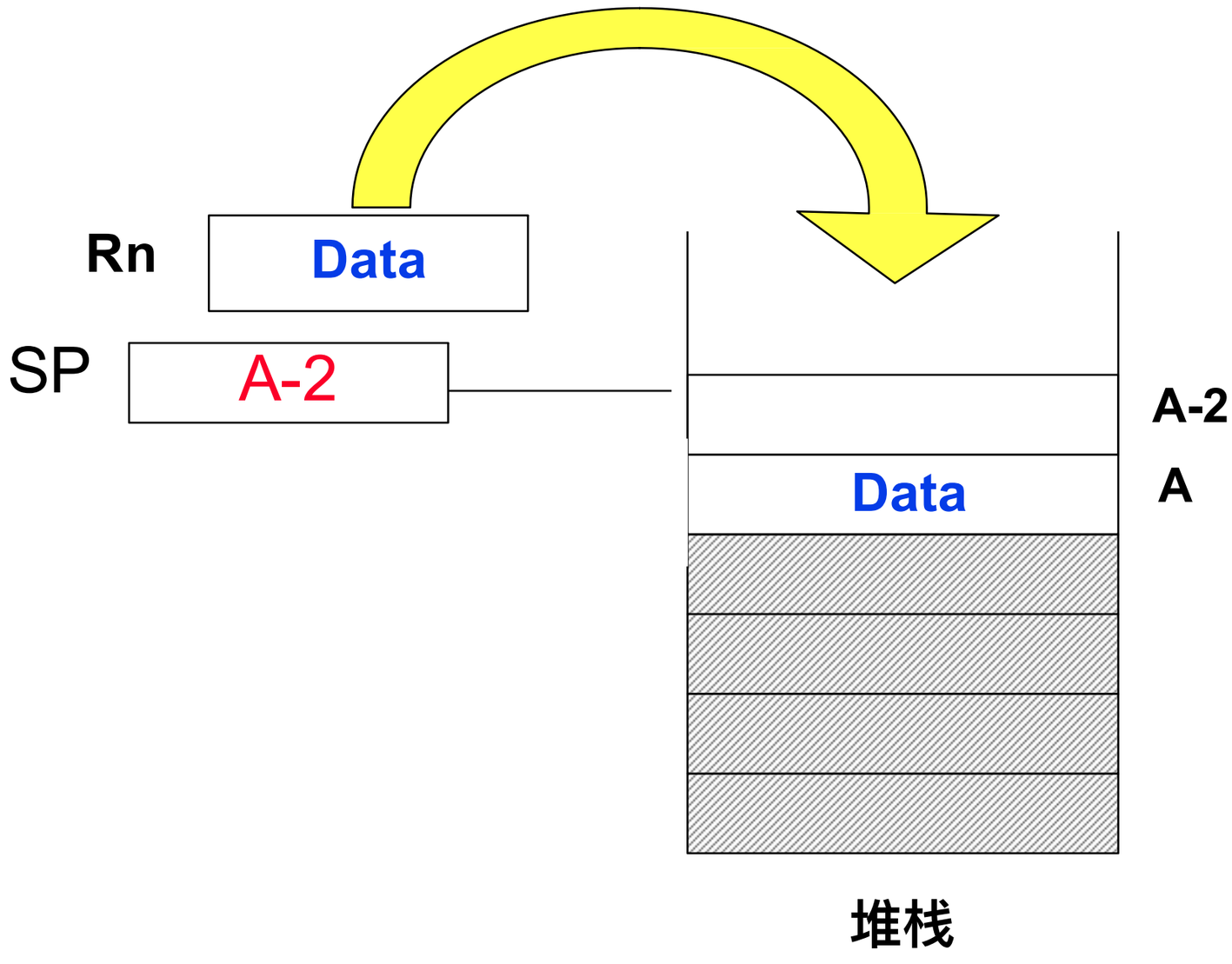


寻址方式

❋堆栈寻址

➤压栈操作：PUSH　Rn，假定寄存器Rn为16位寄存器

(SP)=(Rn)，SP=(SP)-2

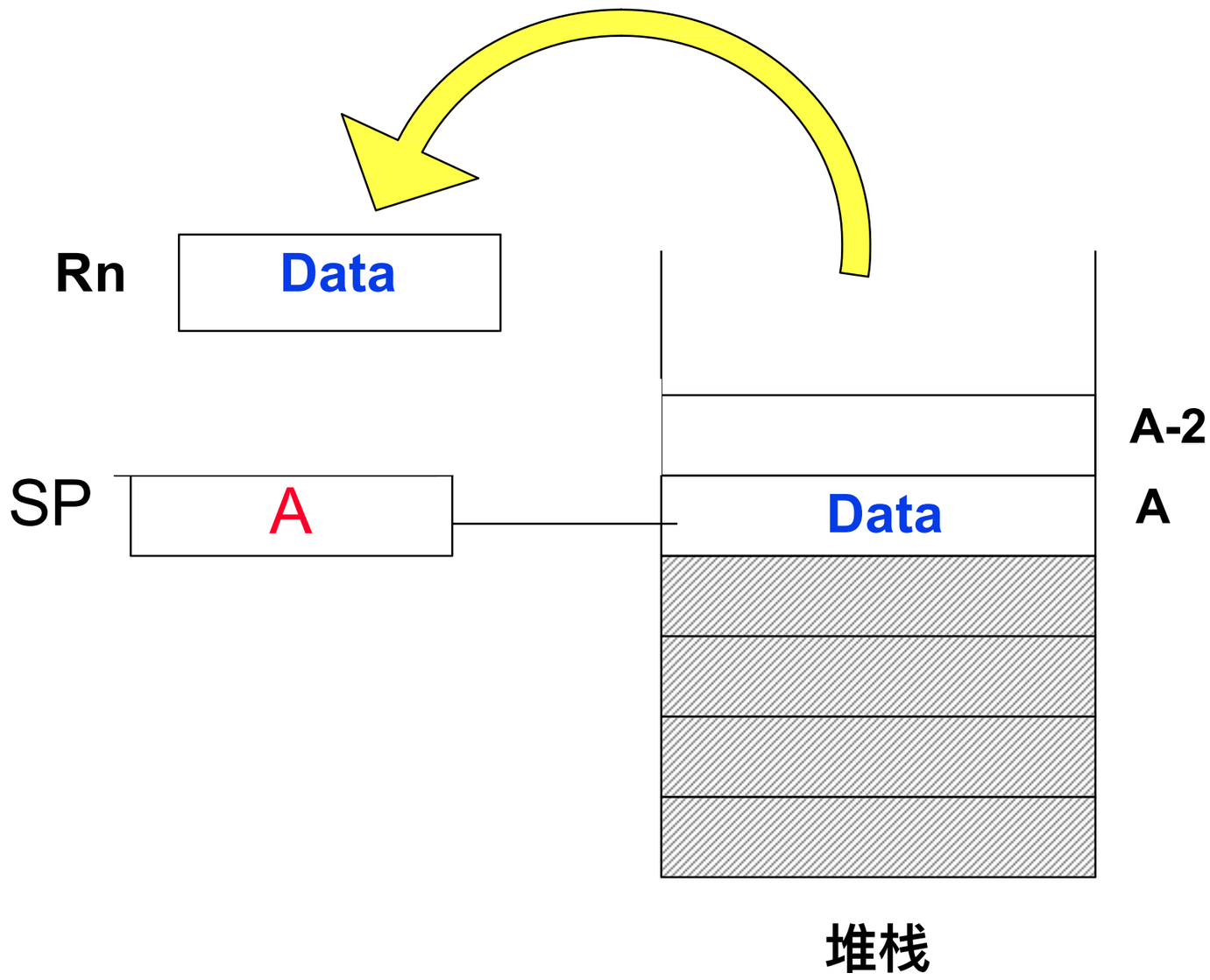


寻址方式

❋堆栈寻址

➤出栈操作：POP　Rn，假定寄存器Rn为16位寄存器

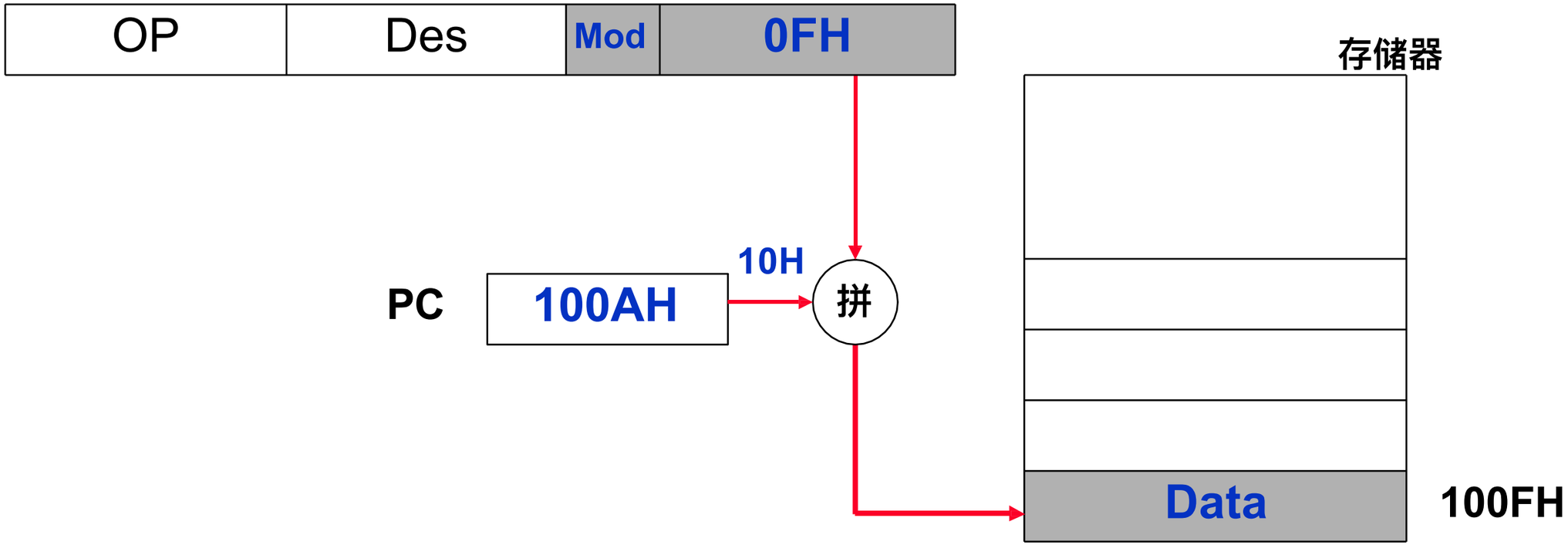
SP=(SP) + 2，　Rn=((SP))



寻址方式

❋页面寻址

- 将程序计算器PC的高位部分与形式地址拼接形成操作数的有效地址。
- EA ＝ $(PC)_H$, 拼接 A
- 内存分位若干页， $(PC)_H$,指明页地址，形式地址A表明页内的位移量



指令类型

- 数据传送指令：Move,Store,Load,Set,Clear,Exchange
- 算术运算指令：包括定点数、浮点数运算和十进制数运算
- 逻辑运算指令：And,Or,Not,Xor,Compare,Test
- 移位指令
  - 算术移位，逻辑移位，循环移位
- 程序控制类指令
  - 几个重要的寄存器：程序计数器PC，程序状态字PSW（或标志寄存器），堆栈指针SP
  - 转移指令：无条件转移指令，有条件转移指令
  - 循环控制指令（LOOP）
  - 子程序调用与返回指令（CALL，RET）
  - 程序中断指令及返回（INT，IRET）
- 串操作指令（MOVSB，MOVSW）
- I/O指令：IN，OUT
- 堆栈指令：PUSH，POP

8086／8088指令系统: x86微处理器

- 1971年1月，Intel 4004，第一枚4位微处理器芯片，标志着第一代微处理器问世。
- 1972年4月，Intel 8008，第一个8位微处理器。8008采用P沟道MOS电路，仍属第一代微处理器。
- 1973年8月， Intel 8080， 8位微处理器，以N沟道MOS电路取代了P沟道，第二代微处理器诞生。
- 1978年6月， Intel 8086/8088， 16位微处理器，标志着第三代微处理器问世，也标志着x86架构和 PC (1981年)的产生。MS的崛起。
- 1981年，Intel 80186，，AMD公司开始生产80186 CPU。
- 1982年2月，Intel 80286，，在80186发布后的几周，80286发布。
- 1985年10月，Intel 80386，，PC从16位进入32位时代。Compaq崛起
- 1989年4月，Intel 80486，，首次实破100万个晶体管，486就是80386 + 80387协处理器。 Dell崛起。
- 1993年3月，IInntteell PPeennttiiuumm ，新一代586 CPU问世，命名为Pentium以区别AMD产品。AMD推出K5微处理器。

8086/8088指令系统

❖ 8086 / 8088CPU简介

➢1978年由Intel推出，16位微处理器

➢CISC（Complex Instruction Set Computer，复杂指令集计算机） 处理器

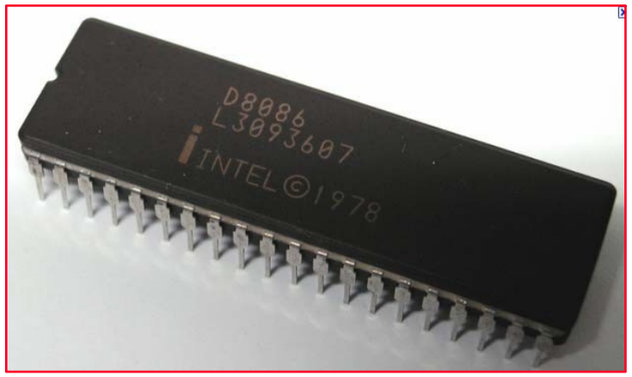
➢29000只晶体管

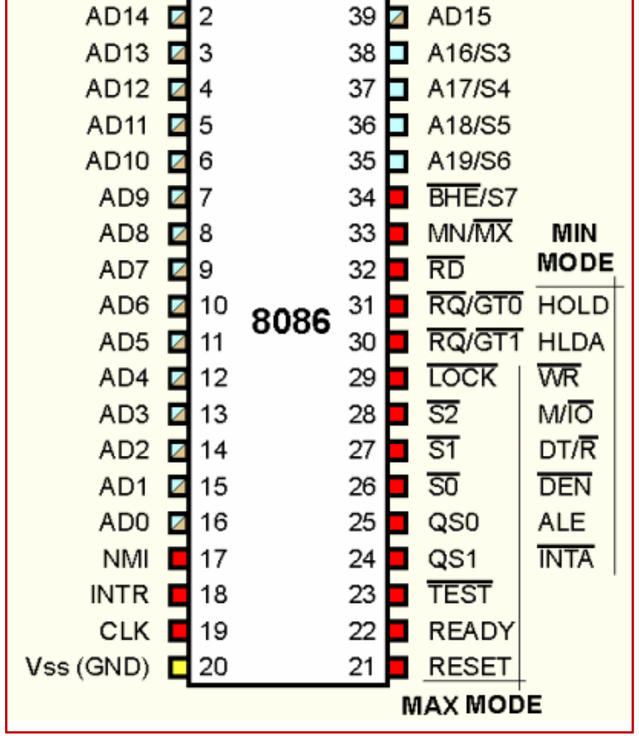
➢速度可分为5MHz、8MHz、10MHz

➢内部数据总线、外部数据总线均为16位

➢地址总线为20位，可寻址1MB内存

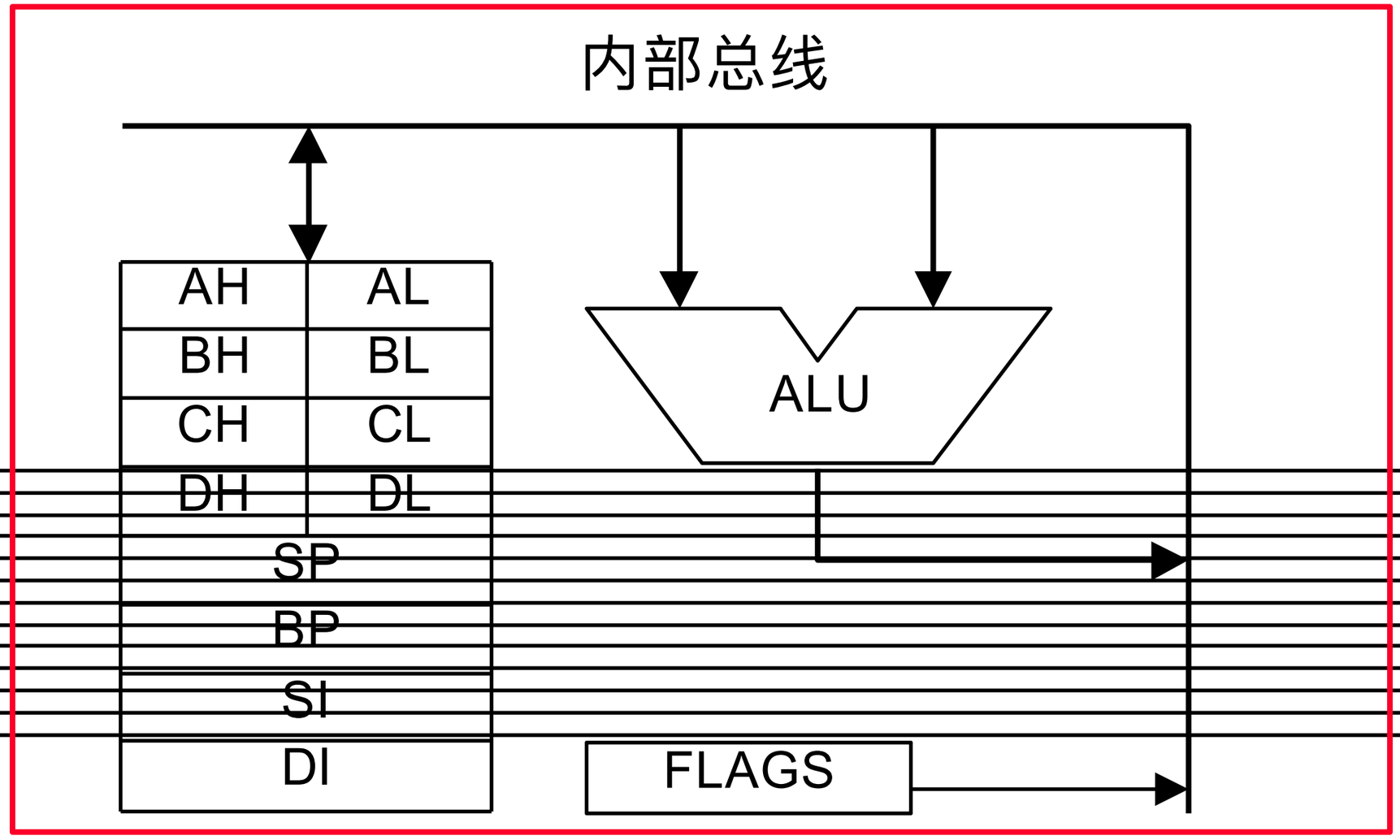
➢1981年，IBM推出首款选用8088 CPU的个人电脑IBM PC





8086／8088指令系统:CPU与存储器结构

❖8086／8088CPU寄存器结构



8086／8088指令系统：　CPU与存储器结构

❖存储器及其存储器地址结构

¬主存容量为1M（2^{20}），可直接访问的主存物理地址为20位。超过1M的存储空间通过其他方式访问。

¬8086／8088机器字长16位，所有寄存器长度位16位，数据总线16位。

¬主存采用分段的结构

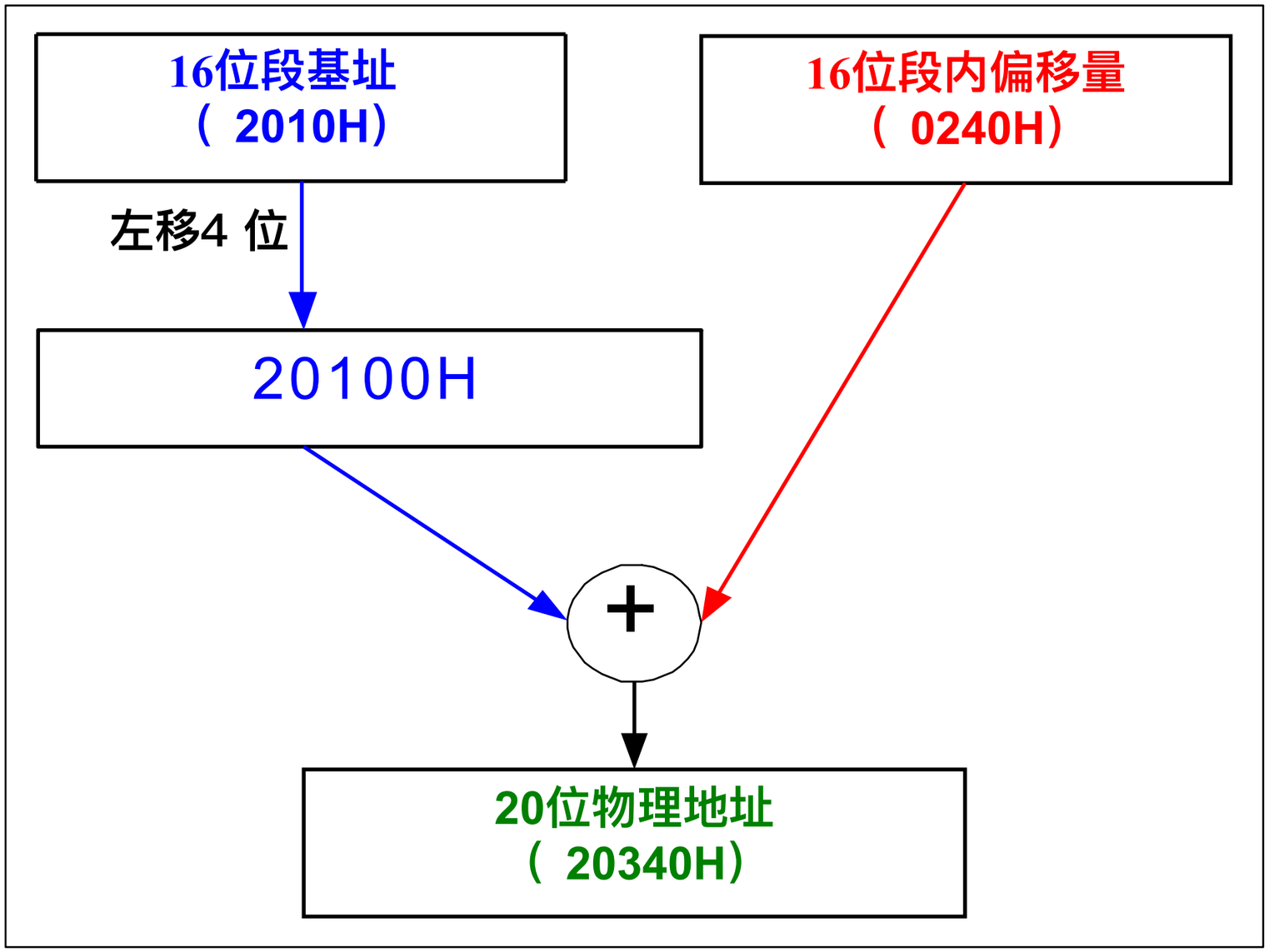
¬主存存储单元的地址构成：段基址（16 bits）：段内偏移（16 bits）

¬可执行程序（.EXE）的存储结构：代码段(Code Segment)，数据段(Data Segment)，堆栈段(Stack Segment)，扩展数据段（可选）

¬命令程序（.COM）的存储结构：代码段，数据段和堆栈段必须是同一个段。所以命令程序最大为64KB存储空间。

8086／8088指令系统：　CPU与存储器结构

※存储器地址结构与计算



8086／8088指令系统:CPU与存储器结构

※存储器单元结构

- ➤按字节单元编址

※字节单元

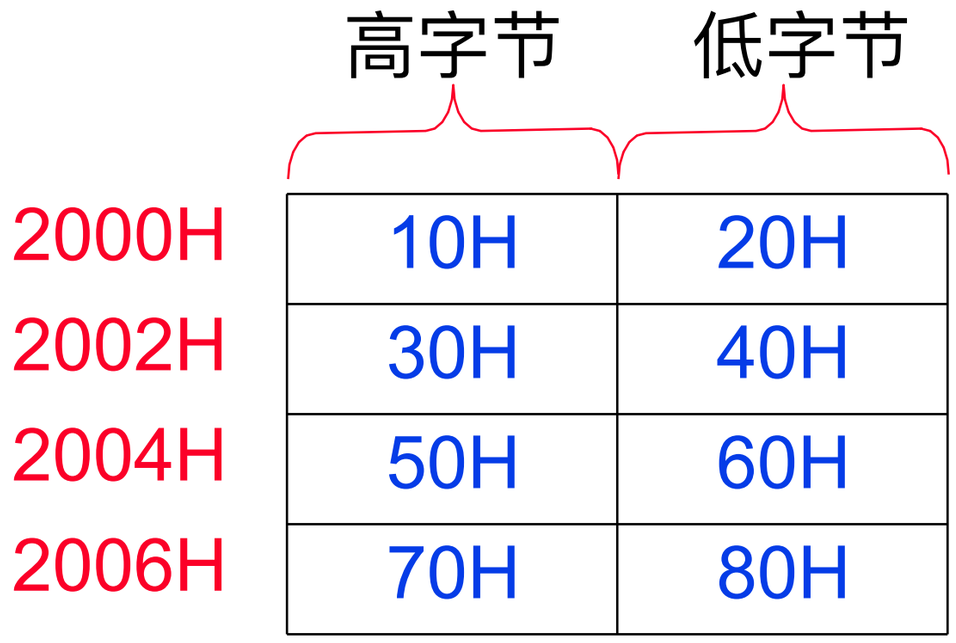
- ➤（2000H）＝20H
- ➤（2001H）＝10H

※字单元

- ➤（2000H）＝1020H
- ➤（2004H）＝5060H

※双字单元

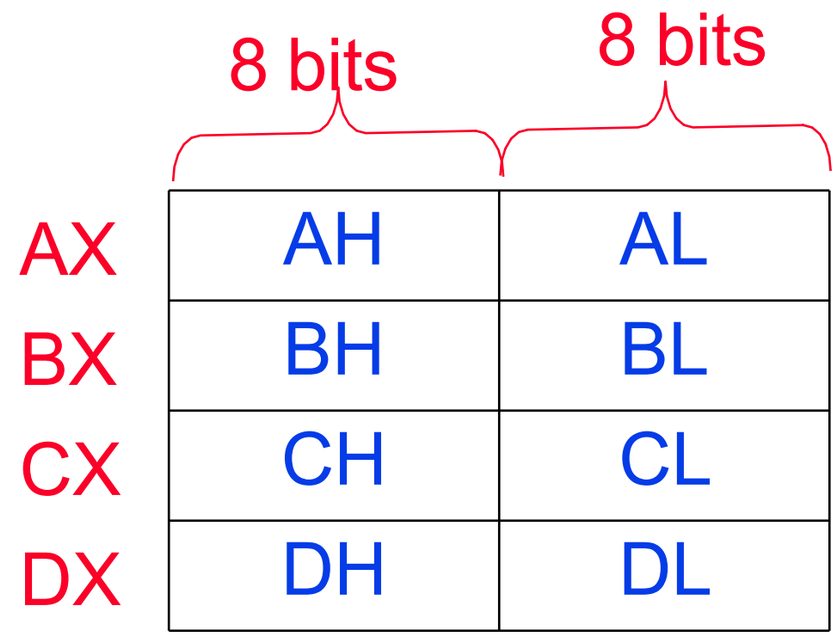
- ➤（2000H）＝30401020H



存储器

8086 / 8088指令系统:寄存器

- 通用寄存器：数据寄存器(Data Register)
  - AX，BX，CX，DX（16位）；
  - AH，AL，BH，BL，CH，CL，DH，DL（8位）
  - 各寄存器原则上没有固定的应用
  - AX：累加器
  - BX：基址寄存器
  - CX：计数器
  - DX：数据寄存器



通用寄存器

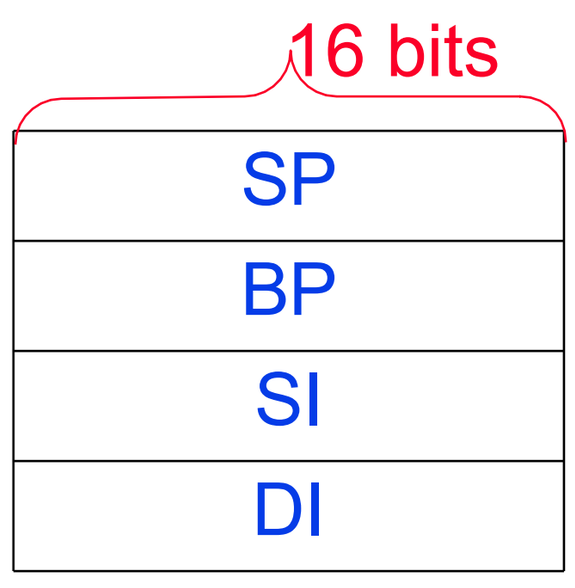
8086 / 8088指令系统：寄存器

※通用寄存器：指针寄存器(Pointer Register)

- ➤堆栈指针：SP（16 位）
- ➤基址指针：BP（16位），默认指向堆栈段

※通用寄存器：变址寄存器(Index Register)

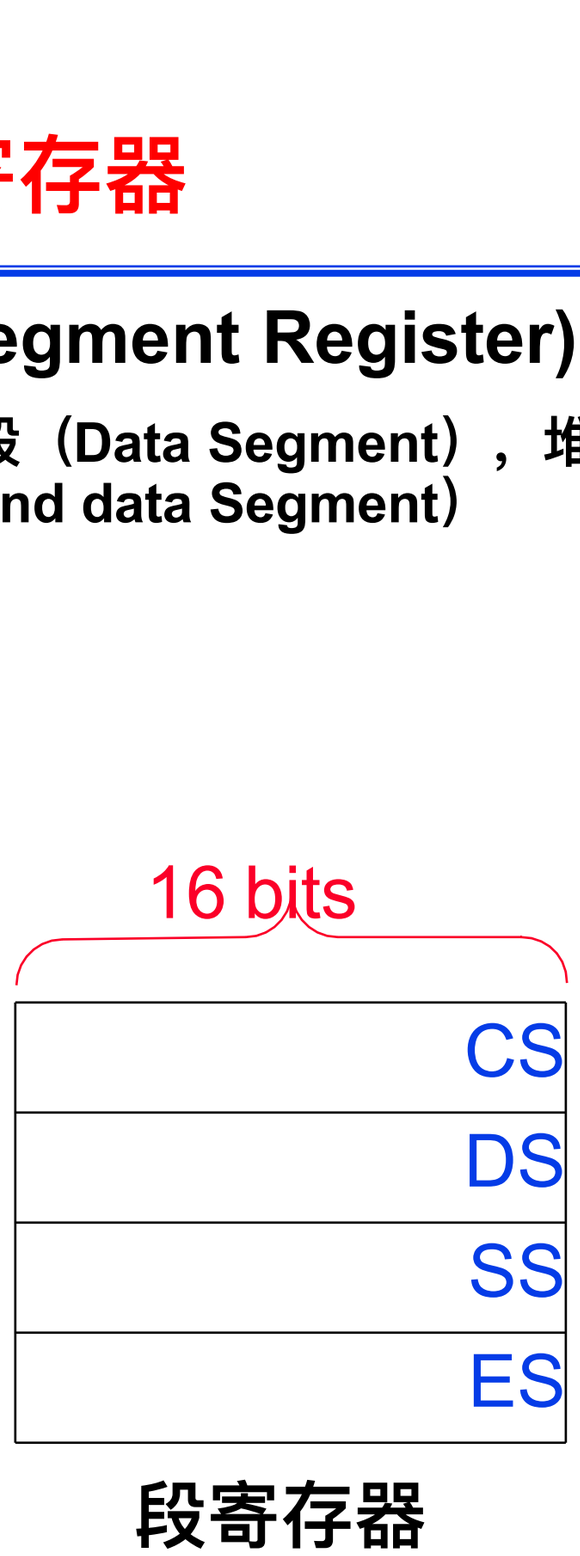
- ➤SI，DI：16位
- ➤一般情况下，二者使用上无差异，在串操作中，SI 对应源操作数，DI对应目的操作数



通用寄存器

8086 / 8088指令系统:寄存器

- 通用寄存器：段寄存器(Segment Register)
  - 代码段(Code Segment)，数据段（Data Segment），堆栈段（Stack Segment），扩展数据段（Extend data Segment）
  - 代码段寄存器：CS（16 bits）
  - 数据段寄存器：DS （16 bits）
  - 堆栈段寄存器：SS （16 bits）
  - 扩展段寄存器：ES （16 bits）



段寄存器

8086 / 8088指令系统:寄存器

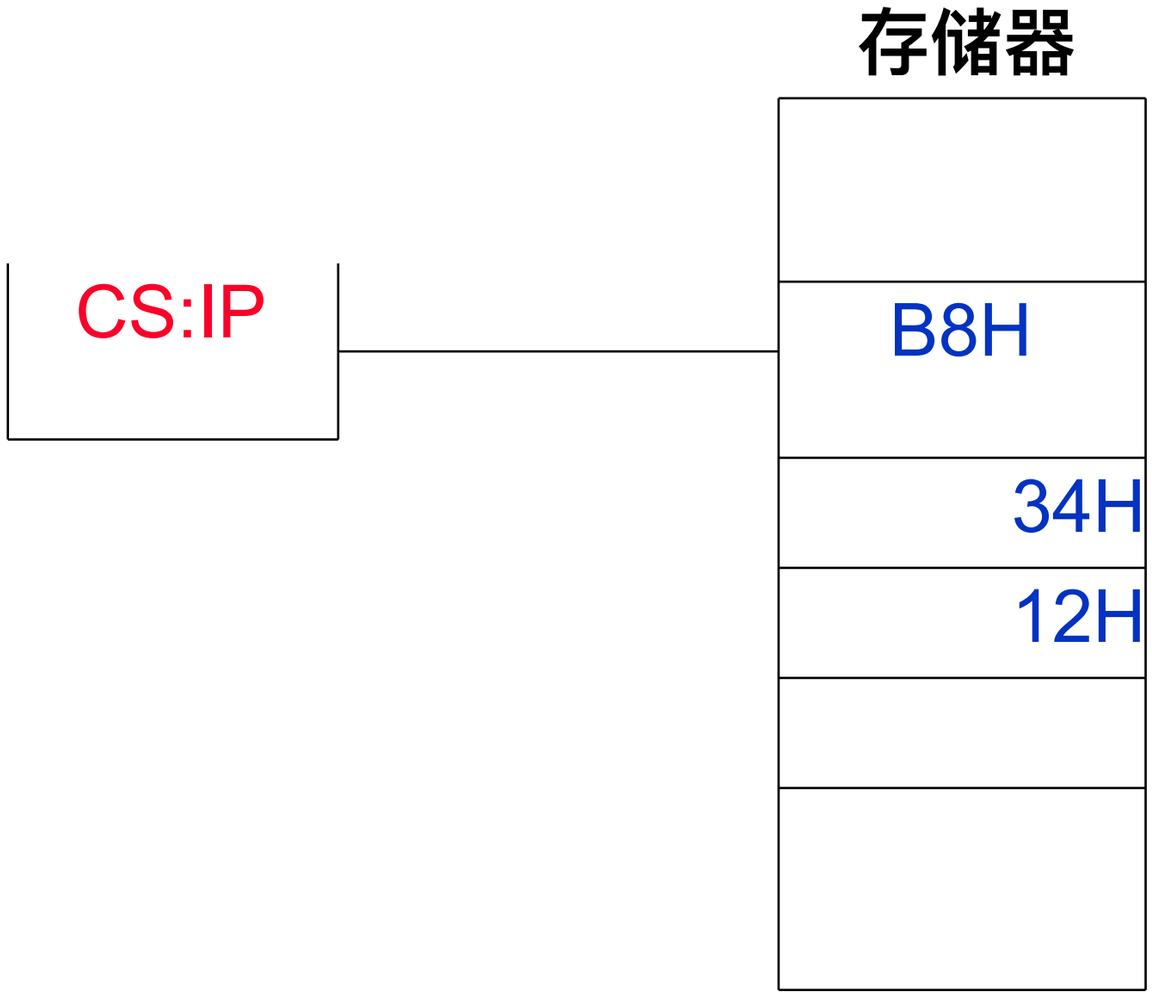
- **指令指针IP（Instruction Pointer）**
  - IP（16 bits)指向代码段中下一条要执行的指令。
  - CS:IP 形成下一次要执行的指令的内存地址。
- **标志寄存器FLAGS（Flags Register）**
  - 16位，记录当前CPU运行程序的各种状态
  - 进位标志位 CF（Carry Flag）
  - 奇偶标志位 PF（Parity Flag）
  - 辅助进位标志位 AF（Auxitiary Flag）
  - 零值标志位 ZF（Zero Flag）
  - 符号标志位 SF（Sign Flag）
  - 溢出标志位 OF（Overflow Flag）
  - 单步跟踪标志位 TF（Trace Flag）
  - 中断允许标志位 IF（Interrupt-enable Flag）
  - 方向标志位 DF（Direction Flag）

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				OF	DF	IF	TF	SF	ZF		AF		PF		CF

8086 / 8088指令系统:寻址方式

❋立即寻址

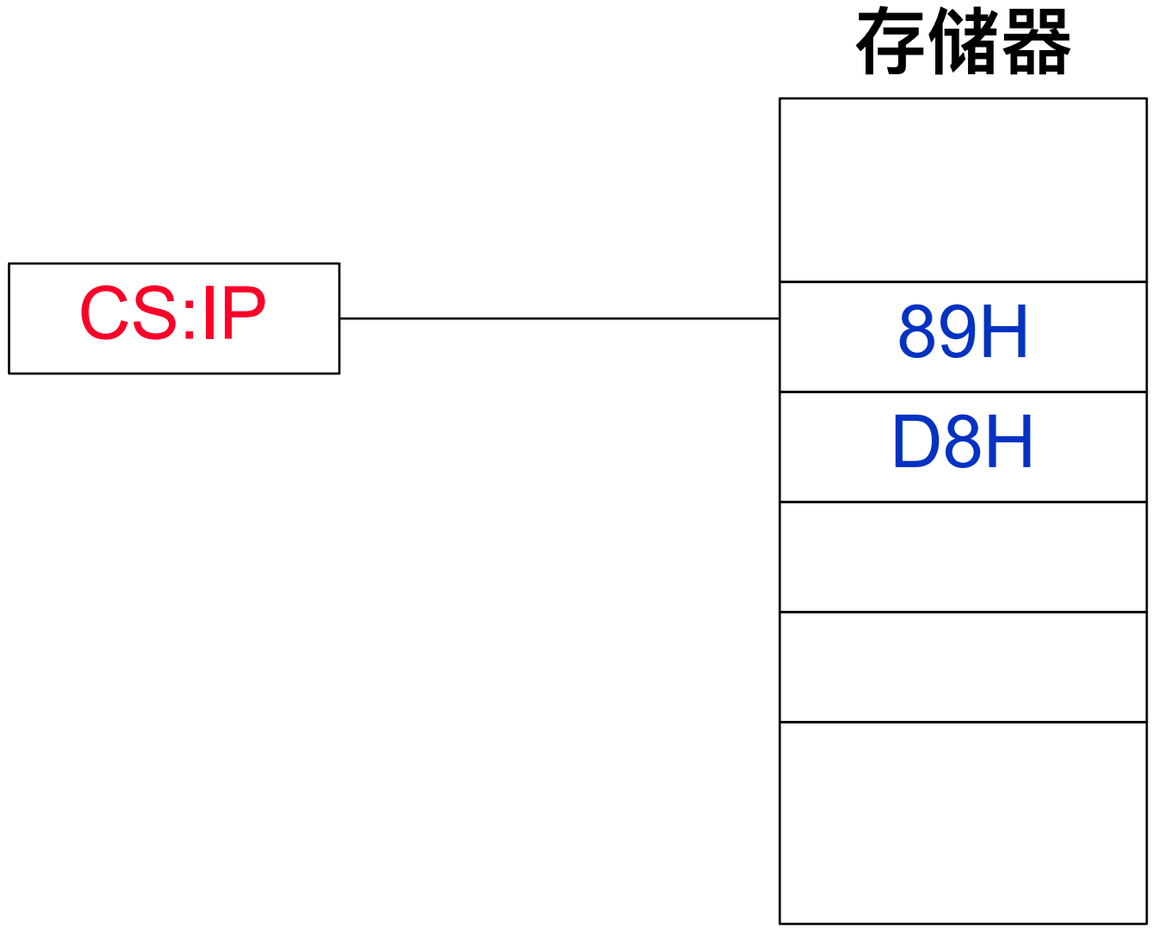
➤MOV AX,1234H　（指令代码：B83412H）



8086 / 8088指令系统:寻址方式

※寄存器（直接）寻址

➤MOV AX, BX （指令代码：89D8H）



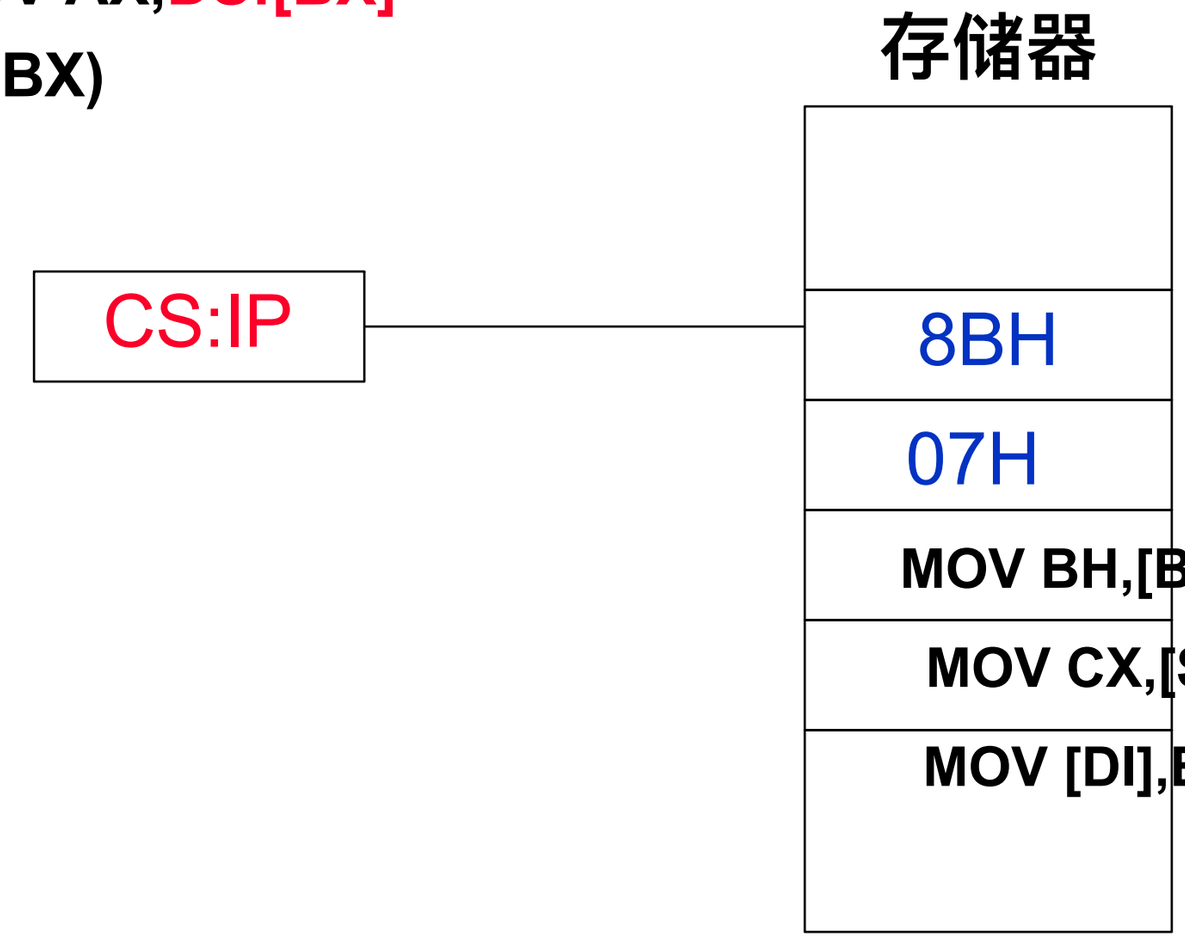
8086 / 8088指令系统:寻址方式

❖ （存储器）直接寻址

- MOV AX, [0100H] （指令代码：A10001H）
- 等价于 MOV AX,DS:[0100H]
- EA = DS:0100H

8086 / 8088指令系统:寻址方式

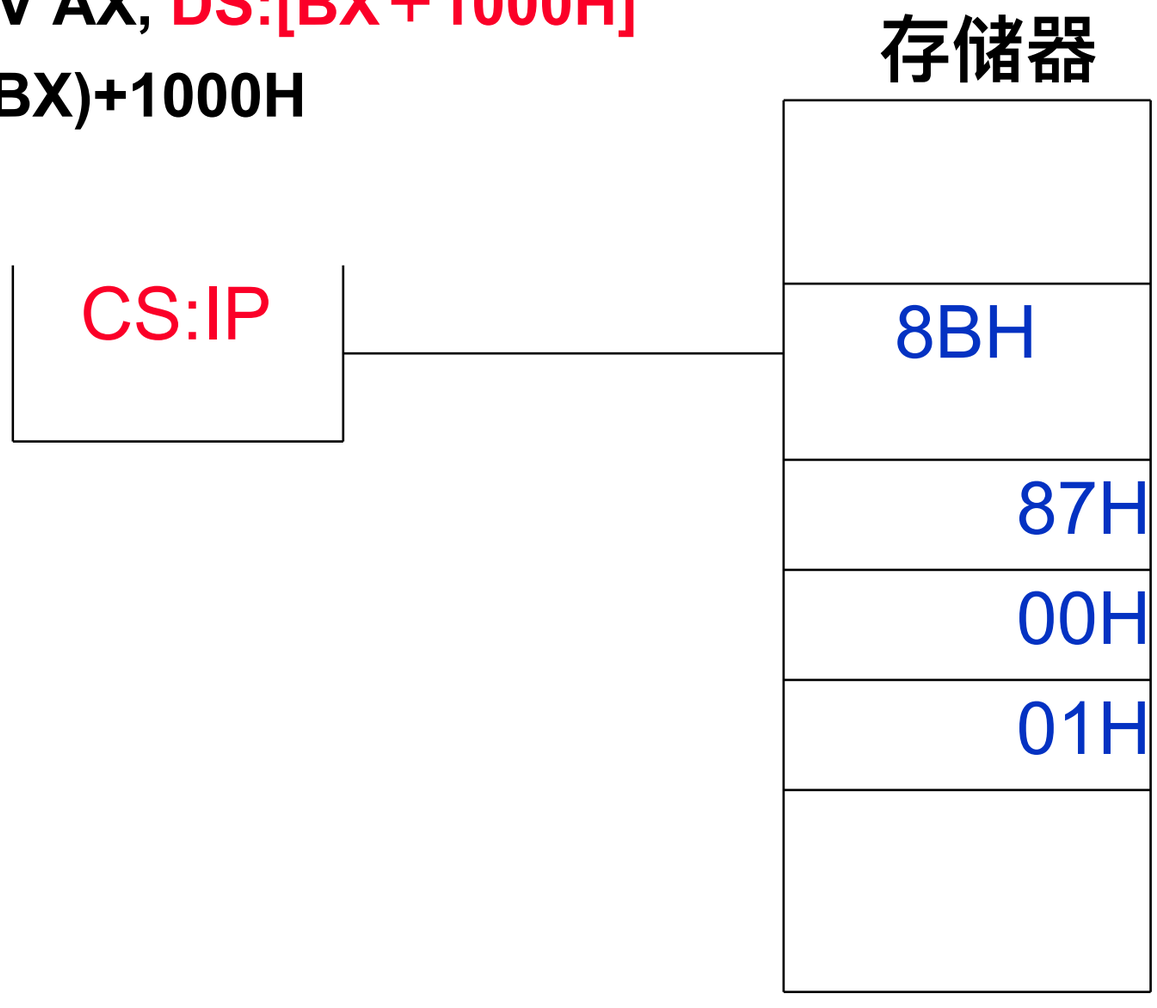
- 寄存器间接寻址
  - MOV AX, [BX] （指令代码：8B07H）
  - 等价于 MOV AX,DS:[BX]
  - EA = DS: (BX)



8086 / 8088指令系统:寻址方式

❋基址（变址）寻址

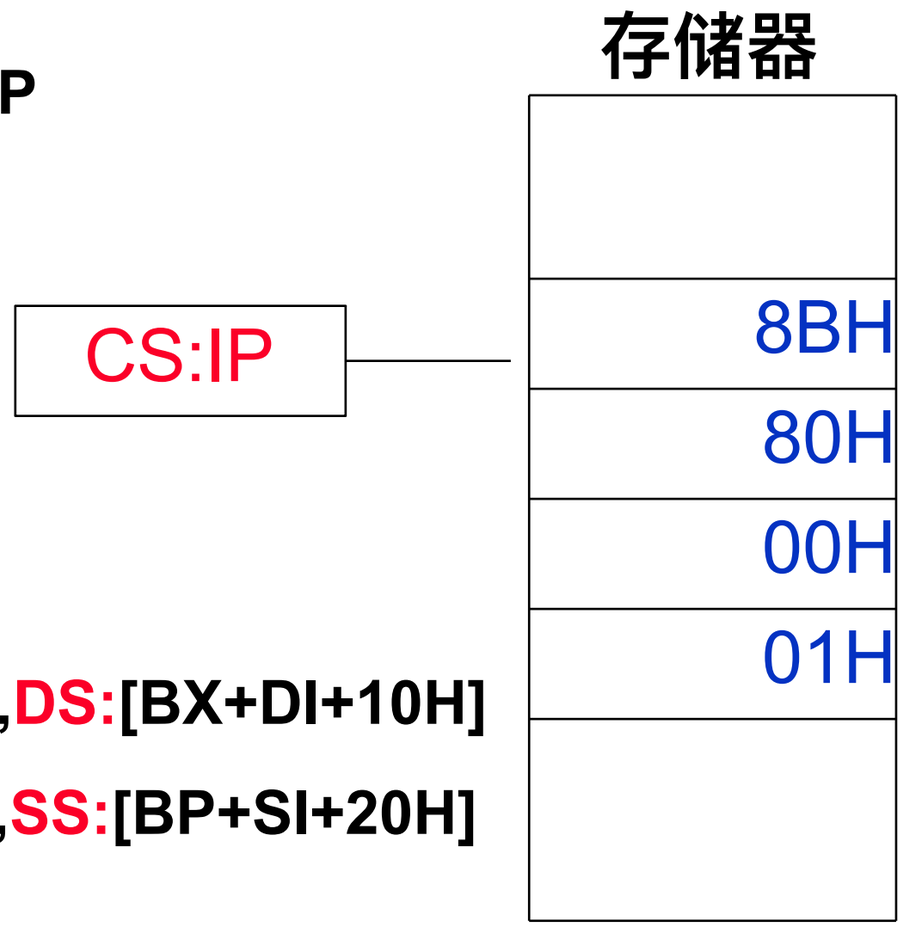
- ➤MOV AX, 1000H[BX]　（指令代码：8B870001H）
- ➤等价于 MOV AX, DS:[BX＋1000H]
- ➤EA = DS: (BX)+1000H



8086 / 8088指令系统:寻址方式

❖基址变址寻址

- ➤MOV AX, 1000H[BX][SI] （指令代码：8B800001H）
- ➤等价于 MOV AX, DS:[BX+SI+1000H]
- ➤EA = DS: (BX)+(SI)+1000H
- ➤基址寄存器只能选：BX，BP
- ➤变址寄存器只能选：SI，DI
- ➤基址寄存器BX：默认DS段
- ➤基址寄存器BP：默认SS段



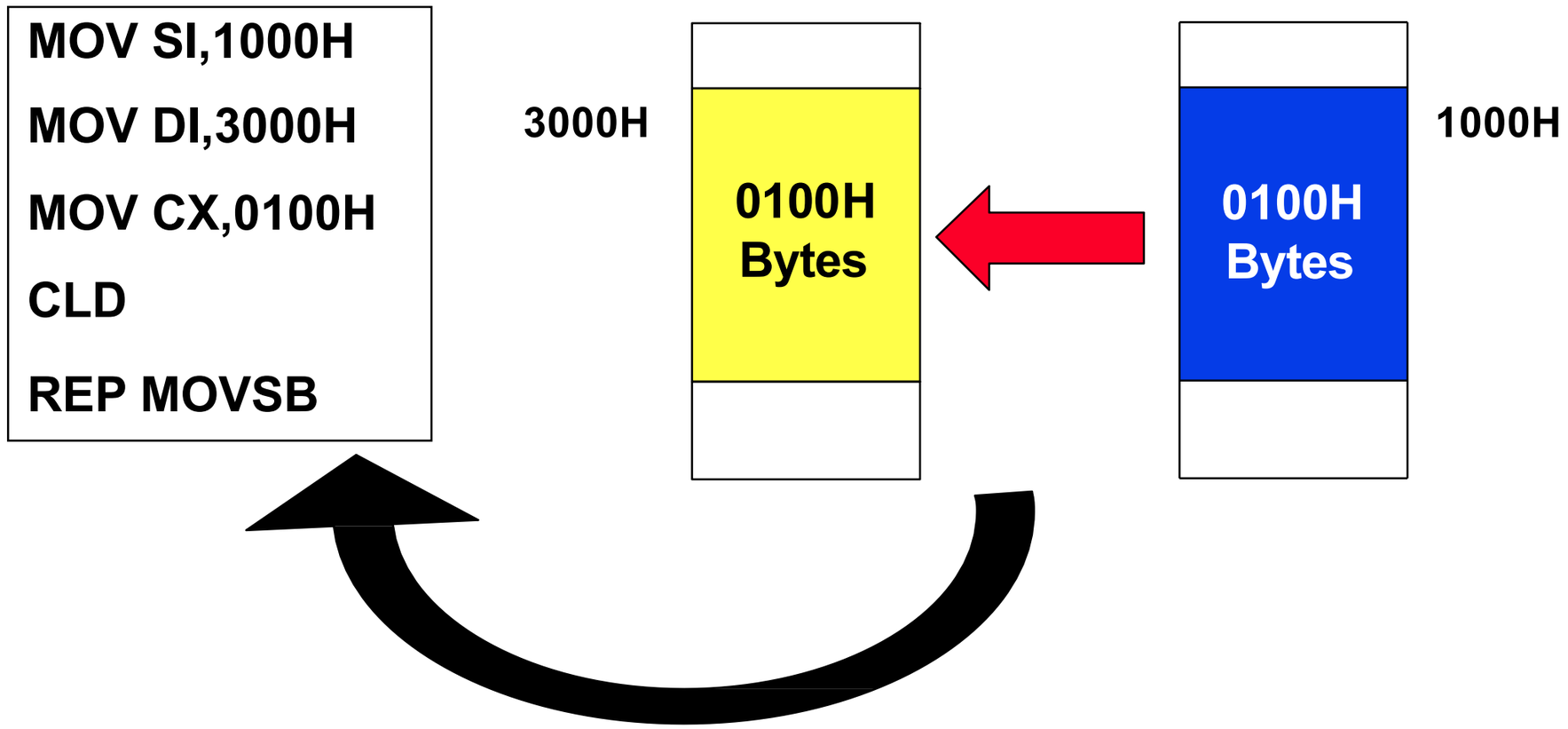
MOV AX,10H[BX][DI] == MOV AX,DS:[BX+DI+10H]

MOV AL,20H[BP][SI] == MOV AL,SS:[BP+SI+20H]

8086 / 8088指令系统:寻址方式

❋串操作寻址

- ➤串操作指令（MOVSB / MOVSW）隐含的寻址方式
- ➤源操作数地址 DS: (SI)
- ➤目的操作数地址 ES: (DI)
- ➤寻址完成后SI，DI自动调整（根据方向标志位DF调整）



8086／8088指令系统:寻址方式

❖I/O寻址

- ➤I/O指令特有的寻址方式
- ➤IN AL,DX
- ➤OUT DX,AL
- ➤I/O端口地址：12位端口地址
- ➤I/O端口：I/O接口部件中可访问的空间（寄存器）

向COM1口发送字符‘A’

```
MOV DX, 3F8H  MOV AL, 41H
OUT DX, AL
```

8086 / 8088指令系统:指令格式与编码

❋一般双操作数指令格式与编码

- RR型或RS型，必有一个操作数在寄存器中（寄存器直接寻址）
- 长度2～6个字节（前2个字节必须）
- Opcode：操作码（6位）
- d: 方向字段（1位）。在第二个字节中，REG确定一个操作数（寄存器直接寻址），MOD和R/M确定另一个操作数的寻址方式。方向字段d表明REG确定的是源操作数还是目的操作数。
  - d=1, REG确定目的操作数，MOD+R/M确定源操作数
  - d=0, REG确定源操作数，MOD+R/M确定目的操作数
- W：字 / 字节字段（1位）：操作数是字节（8位）还是字（16位）
  - W=1，字（16位）
  - W=0，字节（8位）

6	1	1	2	3	3	8	8	8	8
Opcode	d	W	MOD	REG	R/M	Disp_Low	Disp_High	Imme. Data_Low	Imme. Data_High

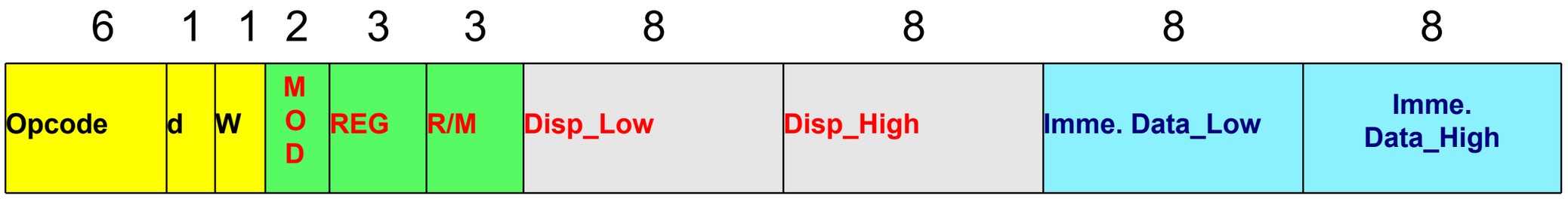
8086／8088指令系统:指令格式与编码

❋一般双操作数指令格式与编码（续）

➤REG：寄存器字段，指明两个操作数中寄存器直接寻址的那个操作数。与W字段配合使用。

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



8086／8088指令系统:指令格式与编码

※一般双操作数指令格式与编码（续）

➤MOD和R/M：确定另外一个操作数。

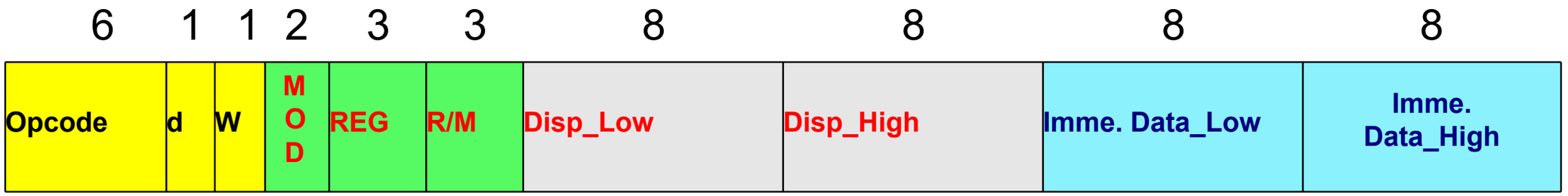
R/M	存储器操作数有效地址(EA)			寄存器操作数	
	MOD=00	MOD=01	MOD=10	MOD=11	
				W=1	W=0
000	(BX)+(SI)	(BX)+(SI)+Disp8	(BX)+(SI)+Disp16	AX	AL
001	(BX)+(DI)	(BX)+(DI)+Disp8	(BX)+(DI)+Disp16	CX	CL
010	(BP)+(SI)	(BP)+(SI)+Disp8	(BP)+(SI)+Disp16	DX	DL
011	(BP)+(DI)	(BP)+(DI)+Disp8	(BP)+(DI)+Disp16	BX	BL
100	(SI)	(SI)+Disp8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

Opcode	d	W	MOD	REG	R/M	Disp_Low	Disp_High	Imme. Data_Low	Imme. Data_High

8086 / 8088指令系统:指令格式与编码

❋一般双操作数指令格式与编码（续）

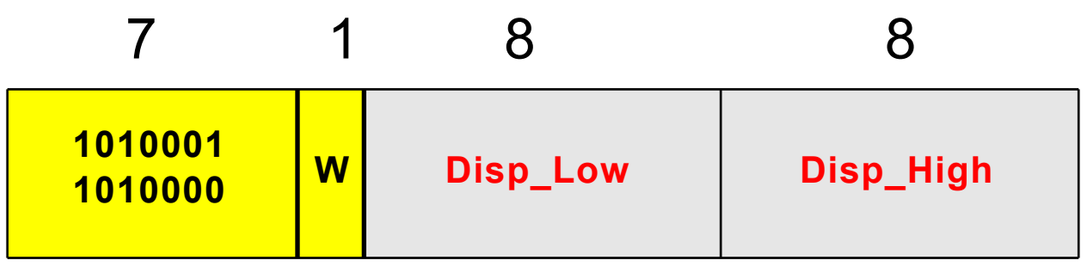
- ➤位移量部分：8位或16位，或者无。
- ➤立即数：8位或16位，或者无。



8086 / 8088指令系统:指令格式与编码

❋与累加器（AX或AL）相关双操作数的指令

- AX（AL）：寄存器直接寻址
- 另一操作数：存储器直接 / 立即数
- 指令代码中省略累加器编码字段，采用特定的操作码以区别于其他双操作数指令。



Mem/ Acc

Acc/ Mem

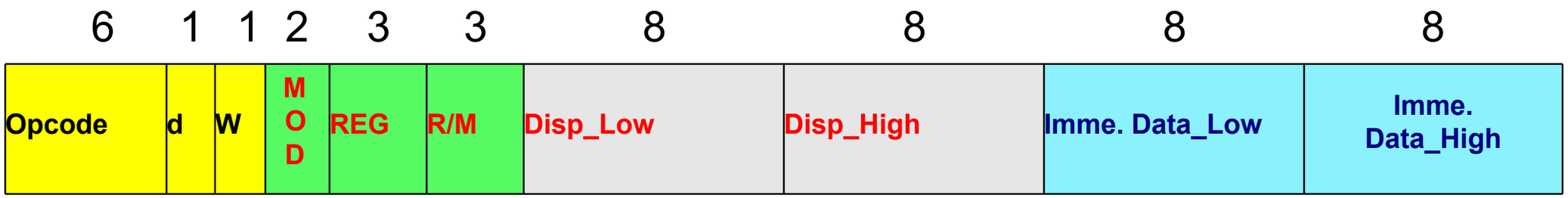
指令：MOV AX, [1000H]

指令代码：10100001 00000000 00010000

8086 / 8088指令系统:指令格式与编码

❋指令编码举例

- ➤MOV AX，1000H[BX][SI]
- ➤MOV的操作码Opcdoe＝100010
- ➤d=1
- ➤W=1
- ➤REG=000
- ➤MOD=10, R/M=000
- ➤指令前两个字节＝1000101110000000（8B80H）



8086／8088指令系统:指令格式与编码

❋其他指令格式

- ➤单操作数指令编码格式
- ➤与立即数相关的指令的特定格式

7	1	2	3	3	8	8
Opcode	W	MOD	OP	R/M	Disp_Low	Disp_High

单操作数指令

8086 / 8088指令系统:指令类型

❋指令类型

- ➤传送指令：MOV，XCHG，LDS，LEA
- ➤算术运算指令：ADD，INC，SUB，CMP等
- ➤逻辑运算指令：AND，OR，NOT，TEST等
- ➤处理器控制指令：CLC，STC，CLI，STI，CLD，NOP等
- ➤程序控制指令：CALL，RET，JMP，JNE，INT，IRET等
- ➤串指令：MOVSB，MOVSW等
- ➤I/O指令：IN，OUT

8086/8088指令系统

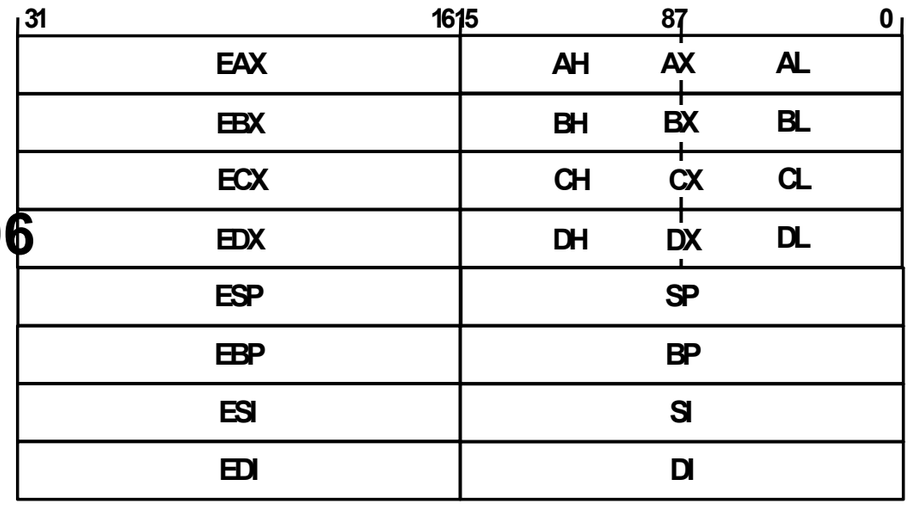
- **80286新增指令(101)**
  - 堆栈操作指令
  - 有符号数乘法指令
  - 移位指令
- **80386新增指令(14)**
  - 数据传送与填充指令
  - 堆栈操作指令
  - 取段寄存器指令
  - 有符号数乘法指令
  - 符号扩展指令
  - 移位指令
  - 位操作指令
  - 条件设置字节指令
  - 循环控制指令
  - 字符串操作指令
- **80486新增指令(12)**
  - 字节交换指令
  - 交换并相加指令
  - 比较并交换指令
  - Cache管理指令
- **Pentium新增指令(289)**
  - 8字节比较交换指令
  - 处理器特征识别指令
  - 读时间标记计数器指令
  - 读模型专用寄存器指令
  - 写模型专用寄存器指令
  - MMX(SIMD, 图形/音频)
  - SSE(图形/视频/语音)
  - SSE2

- **8086(89) -> Pentium(505)**

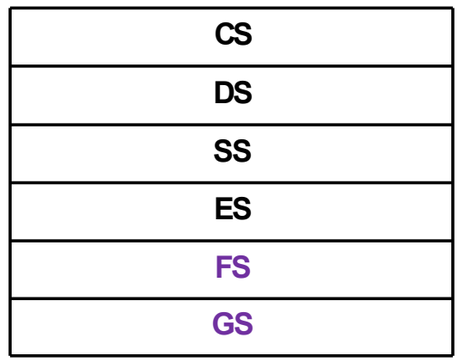
(x86) 成长之痛

- 8088、8086、80186 和 80286 机器再次取得了巨大的成功，但是，与称为工作站的新型机器相比，它们再次开始显示出局限性。

- **Intel 在 i386 和 i486 中**
  - **寄存器扩展为32-bits**
  - **地址空间不分段，“flat” 4,294,967,296 bytes**

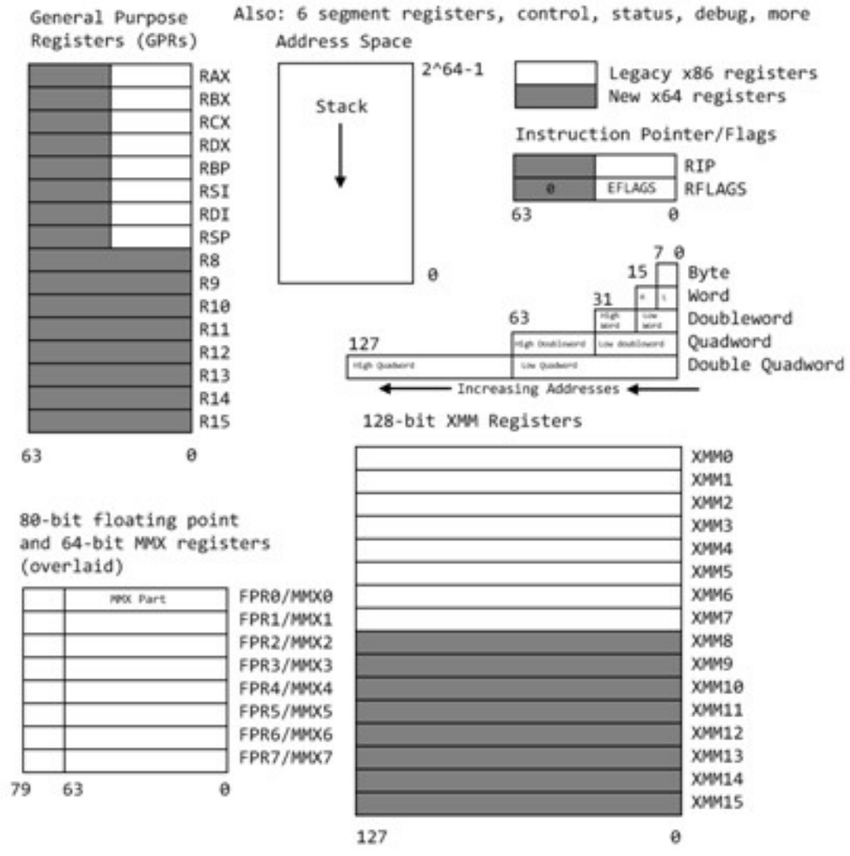


- **段寄存器**
  - **不是扩展到 32-bits. 加了两个新的段寄存器.**
  - 除了向后兼容性之外，不再强调分段寻址
  - **段寄存器承担更多的作为缩放偏移的通用用途 (eg. Mem[SI+(DS<<4)])**



Success is so Painful!

- i386 和 i486 再次成功，甚至与更简单、更快、更便宜的 RISC CPU 比也是如此。
- 1995 年，英特尔推出了他们的第一个流水线版本的 x86 ISA（奔腾），采用了许多 RISC 概念并添加了一些新概念.
- 尽管如此，32 位架构造成代码和数据空间不足，x86 开始再次感受到成长的痛苦。大约在 2005 年，英特尔推出了 64 位 Pentium D、Core 2、Core i3、i5 和 i7 架构



从课程中我们了解到

- 指令集架构更愿意改进进化，而不采用新的架构和发明！
  - 虽然开发新的 CPU 硬件很昂贵，但与开发软件相比，它显得相形见绌。 因此维持向后兼容一直是英特尔成功的秘诀之一。

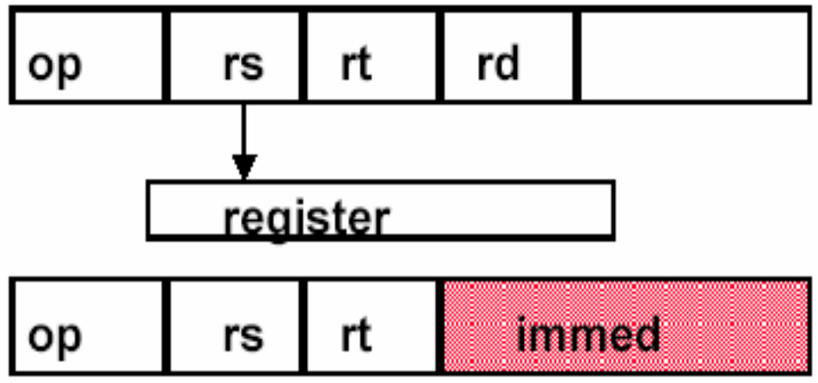
- 美不等于真！
  - 指令丑，只要它能完成任务，我们可以忽略它的丑。

- 手中的鸟不太可能成为喷气客机！
  - 运用你的聪明才智让现有客户对他们做过的事情满意，好过试图说服他们应该以更好的方式重新开始。

❖ MIPS寻址方式

add rd,rs,rt

Register (direct)

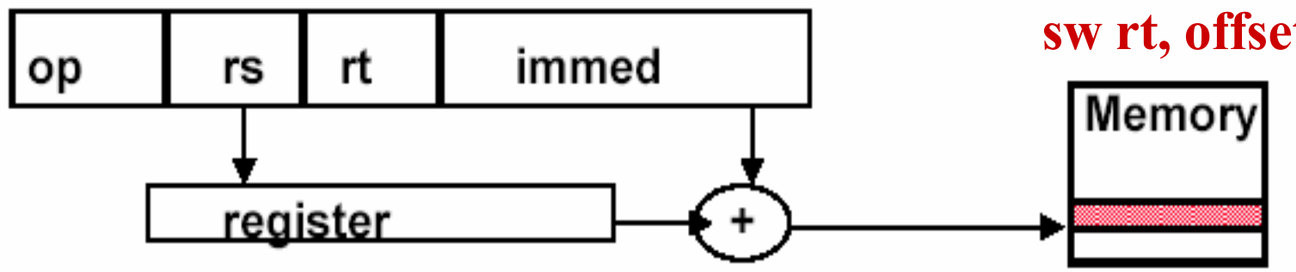


Immediate

addi rt,rs,imm

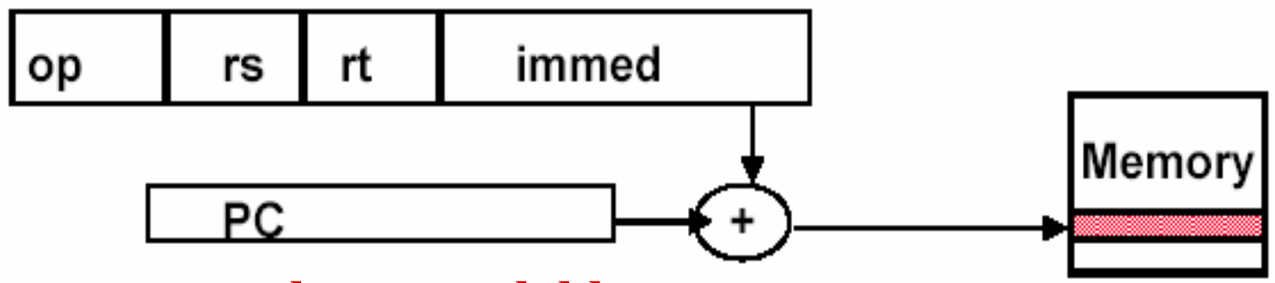
Base+index

lw rt, offset(rs)
sw rt, offset(rs)



PC-relative

由程序计数器PC作为基址寄存器，指令中给出的形式地址作为位移量，二者之和是操作数的内存地址。



beq rs,rt, lable

Mips Base+Index “寻址方式”

•MIPS 可以通过 rs 和 offset 选择构成不同的寻址方式

Absolute (Direct): **lw $8, 0x1000($0)**

- Value = Mem[constant]
- Use: accessing static data

Indirect: **lw $8, 0($9)**

- Value = Mem[Reg[x]]
- Use: pointer accesses

Displacement: **lw $8, 16($9)**

- Value = Mem[Reg[x] + constant]
- Use: access to local variables

数组访问 (base+index)

存储器直接寻址

操作数在存储器中，指令地址字段直接给出操作数在存储器中的地址

寄存器间接寻址

➤操作数在存储器中，指令地址字段中给出的寄存器的内容是操作数在存储器中的地址。

基址加变址

操作数在存储器中，指令地址字段给出一寄存器和偏移量，寄存器的内容与偏移量和是操作数的内存地址。

Absolute (Direct) Addressing 存储器直接寻址

“C”

```
int x = 10;

main() {
    x = x + 1;
}
```

“MIPS Assembly”

```
.data
.global x
x: .word 10

.text
.global main
main:
lw   $2,x($0)
addi $2,$2,1
sw   $2,x($0)
```

“After Compilation”

```
.data 0x0100
.global x
x: .word 10

.text
.global main
main:
lw   $2,0x100($0)
addi $2,$2,1
sw   $2,0x100($0)
```

- 汇编器“x” 用它的地址代替
  - e.g., 这里 data (**.data**) 从 0x100 开始
  - **x** 是第一个变量的地址, 以0x100开始

Indirect Addressing 寄存器间接寻址

What we want:

内存中的常数放在寄存器中

Examples:

“C”

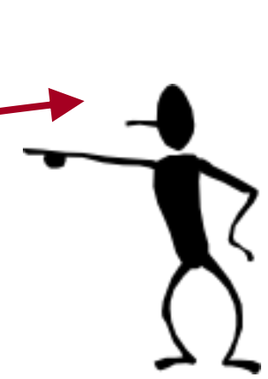
```
int x = 10;

main() {
    int *y = &x;
    *y = 2;
}
```

“MIPS Assembly”

```
.data
.global x
x: .word  10

.text
.global main
main:
    la   $2,x
    addi $3,$0,2
    sw   $3,0($2)
```



“la”不是真正的指令，这是一个方便的伪指令，通过 1 指令或 2 指令序列构造常量

```
ori   $2,$0,x

lui   $2,xhighbits
ori   $2,$2,xlowbits
```

注意

您必须确保寄存器包含有效地址（根据需要进行双精度、字或短对齐）

Note on la pseudoinstruction

la 伪指令：la $r, x

代表将变量 x 的地址“加载”到寄存器 r 中 不是实际的 MIPS 指令，但被汇编器分解成实际的指令。 如果 x 的地址很小（适合 16 位），那么单个 **ori** 即可，如果 x 的地址较大，使用组合**lui + ori**

“MIPS Assembly”

```
.data  0x0100   0x80000010
.global x
x: .word   10

.text
.global main
main:
        la   $2,x
        addi $3,$0,2
        sw   $3,0($2)
```

```
ori   $2,$0,0x100

lui   $2,0x8000
ori   $2,$2,0x0010
```

Displacement Addressing 基址加变址

What we want:

相对于寄存器偏移处的内存位置的内容 **offset relative to a register**

地址是寄存器内容与偏移量的和

Examples:

“C”

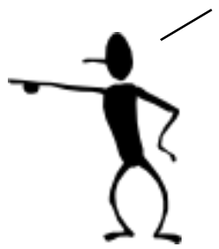
```
int a[5];

main() {
    int i = 3;
    a[i] = 2;
}
```

“MIPS Assembly”

```
.data 0x0100
.global a
a: .space   20

.text
.global main
main:
    addi $2,$0,3  // i in $2
    addi $3,$0,2
    sll  $1,$2,2  // i*4 in $1
    sw   $3,a($1)
```



为5个未初始化的整数分配空间(20-bytes)

注意

必须使地址字对齐

Displacement Addressing: 2nd example

Examples:

“C”

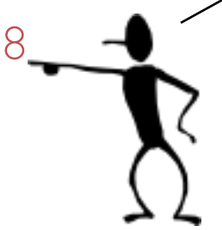
```
struct p {
    int x, y; }

main() {
     p.x = 3;
     p.y = 2;
}
```

“MIPS Assembly”

```
.data
.global p
p: .space   8

.text
.global main
main:
     la    $1,p
     addi  $2,$0,3
     sw    $2,0($1)
     addi  $2,$0,2
     sw    $2,4($1)
```



Allocates space for 2 uninitialized integers (8-bytes)

注意

结构中各个字段的偏移量是汇编器/编译器已知的常量

Real-World Addressing

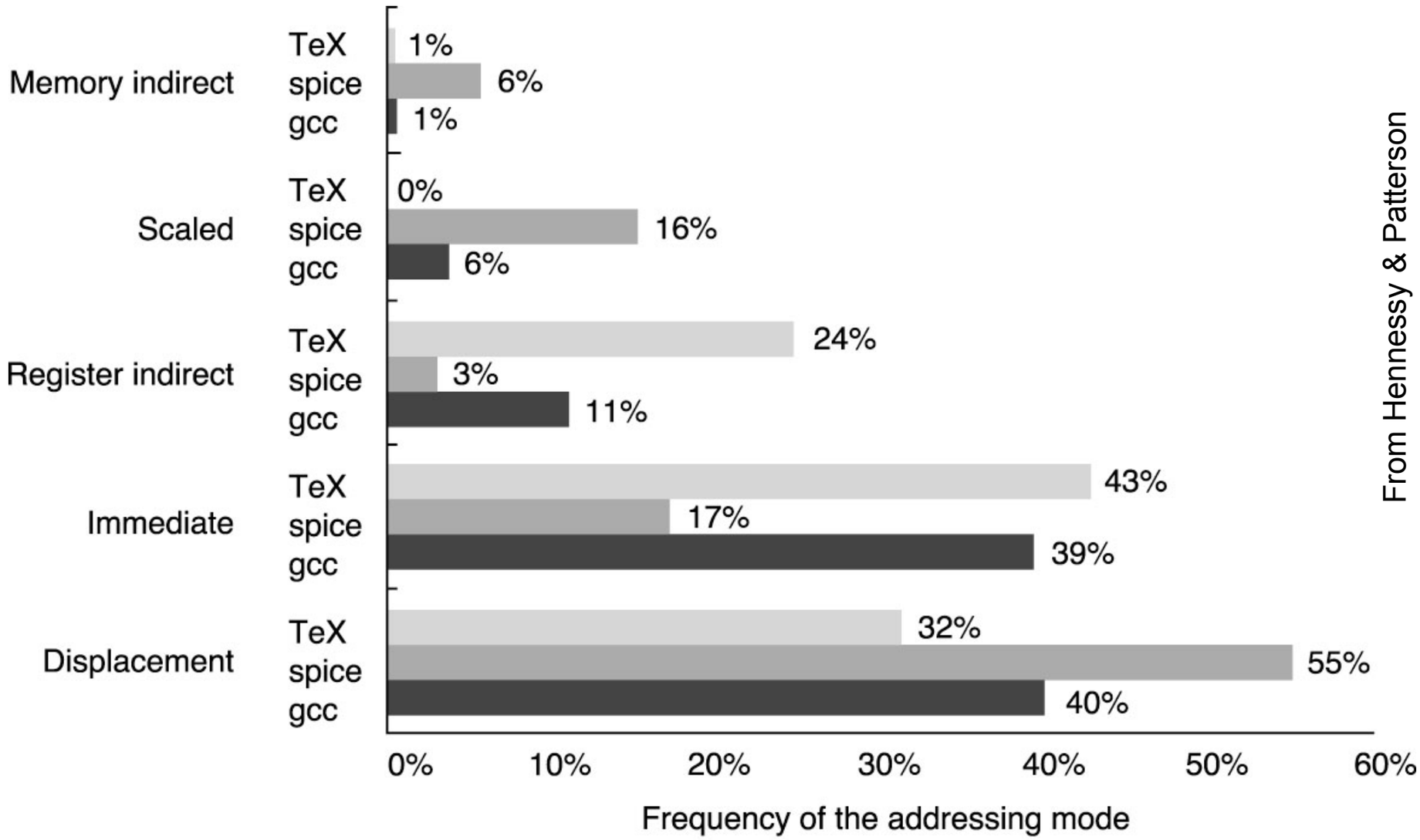
- **注意事项**
  - 在实际中，**$gp** 通常用作所有变量的基地址
  - 这种方式只能寻址首尾32K内存(offset为16Bit)
  - 有时会生成两个指令序列:

lui$1,xhighbits　　lw$2,xlowbits($1)

在上电时给gp初始化一个值，那么，所有小变量区的变量就可以通过

lw r1, offset(gp)来访问

Relative popularity of address modes



MIPS 指令系统的细节信息

°#0寄存器总是具有数值0 (即使试图对它写入其他数值，也是如此)

•°转移和跳转指令将返回地址PC+4装入链接寄存器（ link register）

•°所有指令改变目标寄存器的所有 32 位信息 (包括lui, lb, lh)，并读取源寄存器的所有32位信息 (add, sub, and, or, 等等)

•°立即数算术和逻辑指令进行如下扩展:

•逻辑立即数（logical immediates） 零扩展到 32位

•算术立即数（arithmetic immediates）符号扩展到32位

•°指令lb 和 lh 装入的数据进行如下扩展:

•lbu, lhu 是零扩展

- lb, lh 是符号扩展

°下面的算术和逻辑指令可能会出现溢出:

•add, sub, addi

•以下指令不会出现溢出： addu, subu, addiu, and, or, xor, nor, shifts, mult, multu, div, divu

总结: MIPS R3000的显著特点

- 32位固定格式的指令 (3 种格式)
- 32 个32位 GPR (R0 = 0) 和 32 个 FP 寄存器 (以及 HI LO)
- 根据软件约定划分
- 3 地址, 寄存器-寄存器算术指令
- 对于load/store指令，单一寻址模式: base+displacement
  - 没有间接寻址（indirection）
  - 16位立即数 加 LUI
- 简单的转移条件
- 与0比较，或者判断两个寄存器是否相等
- 没有条件码
- 延迟转移
- 即使转移发生，也要执行转移（或跳转）之后的指令
- (在50%的时间，编译器能够在延迟转移的时间中填充有用工作)

指令系统的设计

❖ 设计依据

➢操作特性：不同操作的数量，具体是些什么操作，复杂程度等。

➢数据类型：各种操作所处理的数据的类型。

➢指令格式：指令长度，地址数目，指令中不同字段的大小（操作码位数、地址码位数）等。

➢寄存器数目：CPU中可以直接访问的寄存器的数目及使用方法。

➢寻址(Addressing)：操作数的寻址方式。　寻址空间范围，编址粒度。

❖ 指令格式设计

- ➢ 指令格式设计首先考虑指令编码中的固定不变的部分，然后考虑可变的部分。
- ➢ 指令编码中的固定部分：指令系统方案确定后不可能发生变化的部分。
  - ▪ 操作码（数量，位数，编码）
  - ▪ 寻址方式（数量，位数，编码）
  - ▪ 寄存器（数量，位数，编码）
  - ▪ 不同指令所涉及的地址数量
- ➢ 如果代码大小至关重要：使用可变长度指令格式
- ➢ 如果性能至关重要：使用固定长度指令格式

指令编码举例

某RISC指令集每条指令16位，每个操作数占4位，有三种指令格式，其中三操作数指令15条，两操作数指令12条，剩下全为一操作数指令，请问一操作数指令有多少条？给出一种操作码编码方案。

三操作数指令15条

Op1=0001-1111	operand1	operand2	operand3

二操作数指令12条

Op1=0000	operand1	operand2	op2=0100-1111

一操作数指令

Op1=0000	operand1	Op3=0000-1111	op2=0000-0011

一操作数指令op2=0000-0011,4个编码，Op3 16个编码

可以有4*16=64个编码操作，可以编编一操作数指令64条