计算机组成原理

3.2 指令系统

2023/10/9

信息与软件工程学院
School of Information and Software Engineering

主 要 内 容

1. 指令格式
2. 指令寻址方式
3. 指令类型

引言

计算机的工作，体现为指令的执行，实际上包含以下三个步骤：

① 从主存储器中取指令（取指）；

② 在CPU内部执行该条指令（在此过程中，CPU一般还需从主存储器中获取如操作数等相关的信息）；

③ 结果送回存储器存放。

引言

指令：一系列按照某种规律有序排列的，能被CPU识别、执行的二进制代码。

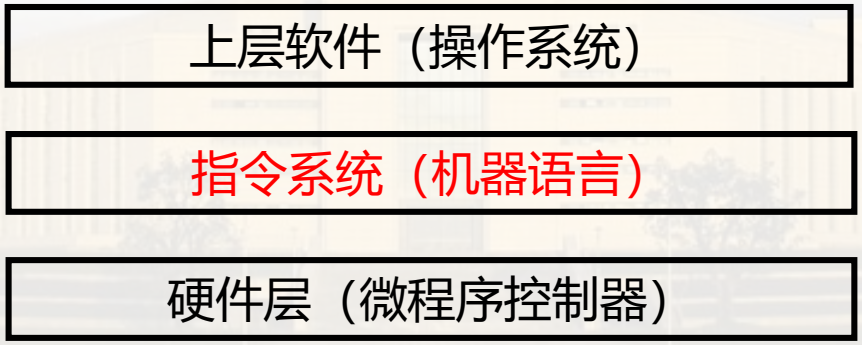
指令系统（或集）：一台计算机所能执行的全部指令。

指令系统---对应---计算机硬件功能

指令系统特点：

✓决定了计算机的硬件功能

✓计算机中软硬件的分界面



精简指令集系统 和 复杂指令集系统

CISC：复杂指令集（Complex Instruction Set Computer）

- 具有大量的指令和寻址方式。
- 大多数程序只使用少量的指令就能够运行。

RISC：精简指令集（Reduced Instruction Set Computer）

- 8/2原则：80%的程序只使用20%的指令。
- 只包含最有用的指令。
- 确保数据通道快速执行每一条指令。
- 使CPU硬件结构设计变得更为简单。

RISC 和 CISC 之间主要的差别

指标	RISC	CISC
指令集	一个周期执行一条指令，通过简单指令的组合实现复杂操作，指令长度固定。	指令长度不固定，执行需要多个周期。
流水线	流水线每周期前进一步	指令的执行需要调用一段微程序
寄存器	更多通用寄存器	用于特定目的的专用寄存器
Load/Store结构	独立的Load/Store指令完成数据在 寄存器 和 外部存储器 之间的传输	处理器能够直接处理存储器中的数据

2.1 指令格式

- 01. 指令基本格式
- 02. 指令字长
- 03. 操作码结构
- 04. 指令中的地址结构

指令基本格式

一条指令提供两方面的信息：

① 与CPU操作有关的信息---操作码（OP）；

② 与操作数有关的信息---地址码或操作数（AD）。

最基本的指令格式：

一个
（指示要执行的操作）

一个或几个
（指示操作对象）

或者

例：ADD AX, (2000H)　　地址码

OR　AX, BX　　或　寄存器

MOV AX,1000H　　或　立即数

一、指令基本格式

在指令格式设计时相应地需要考虑以下一些问题：

① 指令字长需多少位？

是固定字长还是可变字长？

② 操作码构成需多少位？

位数与位置固定还是可扩展？

一、指令基本格式

在指令格式设计时相应地需要考虑以下一些问题：

③ 地址码：一条指令的执行涉及到哪些地址？在指令中给出哪些地址？有几个？哪些地址是系统隐含约定的？

④ 寻址方式：根据地址码如何获取操作数地址？是直接给出还是间接给出？或是经过某种变换（包括计算）获取的？

二、指令字长

指令字长：

即包括操作码、操作数在内的整个指令(整条指令)的长度(单位: 位)。

指令长度对计算机的影响：

指令字长越长：

优点：指令功能越丰富，
完成工作越多；

缺点：占用存储空间大，
从主存中取指时间越长，
指令执行速度越慢。

在计算机中，根据需要，指令长度分为：可变字长指令、固定字长指令

三、操作码结构

操作码的位数决定了操作类型的多少，

位数越多所能表示的操作种类也就越多。

操作码分类
- 定长操作码
- 可变长度操作码(扩展操作码)
- 复合型操作码(方式码)

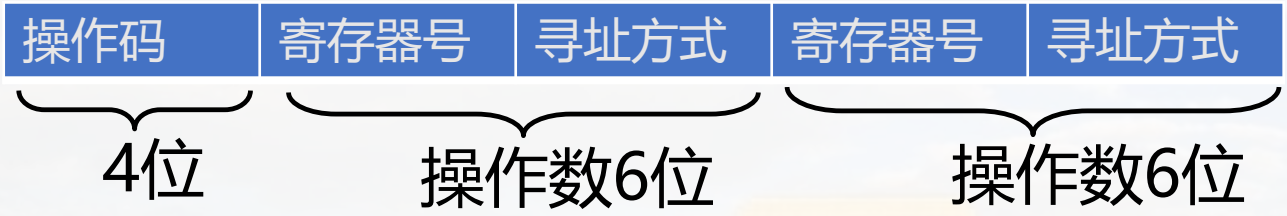
(1) 定长操作码

所有指令操作码OP的位置、位数固定相同

优点: 便于操作码译码; 译码与取操作数并行

应用: 定长操作码一般用于指令字较长的指令

例: 模型机的定长操作码



模型机的操作码: 0000 数据传输MOV
0001 加法ADD
0002 减法SUB

对加法指令 ADD R1, R2,对应的机器指令:

0001 001 000 010 000

(2)变长操作码(扩展操作码)

操作码的位置、位数不固定, 按需求变化。从而可以更有效的利用存储空间。

变长操作码的两种方式:扩展操作码、复合操作码。

① 扩展操作码

在指令字长较短的情况下, 利用某些指令地址位数较少产生的空闲位, 来扩展操作码。

关键在设置扩展标志。

三、操作码结构

例：假设指令字长16位，含有3、2、1、0个地址，每个地址占4位。

- 如果采用定长操作码，为了能表示出3地址指令，则操作码的长度只能有4位，最多可表示16条指令。但对2、1、 0个地址指令，则有空闲位没有得到利用，如下图所示：

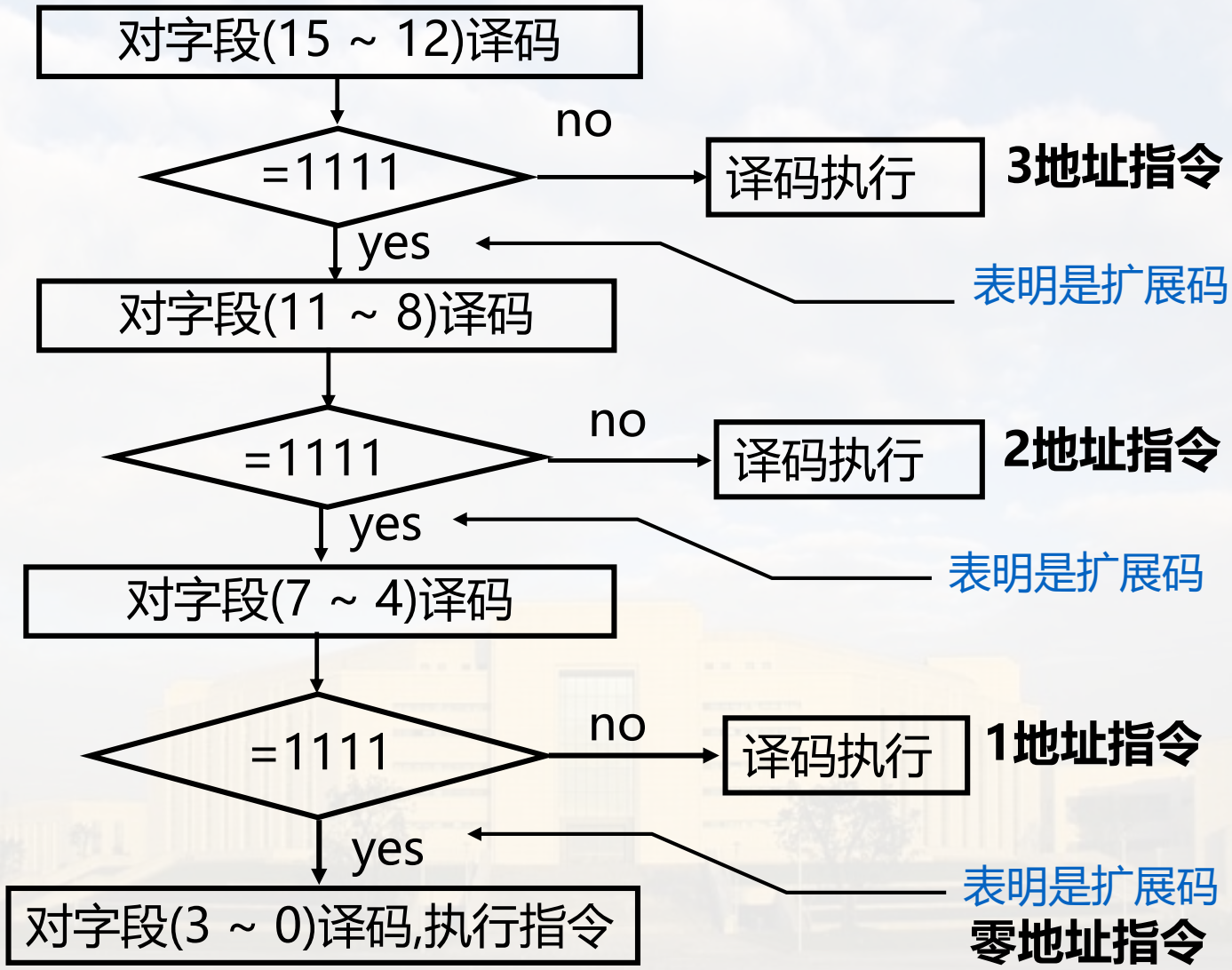
15 ..... 12 11.....8 7.....4 3.....0

4位操作码	地址3	地址2	地址1	3地址指令
4位操作码		地址2	地址1	2地址指令
4位操作码			地址1	1地址指令
4位操作码				零地址指令

三、操作码结构

- 采用变长操作码，利用2、1、0个地址指令所空出的位来扩展指令，从而表示出更多的指令。

一 扩展指令码的识别过程：



一 比较:

上例中, 共表示出61条指令(包含3、2、1、0个地址的指令, 每个地址占4位。

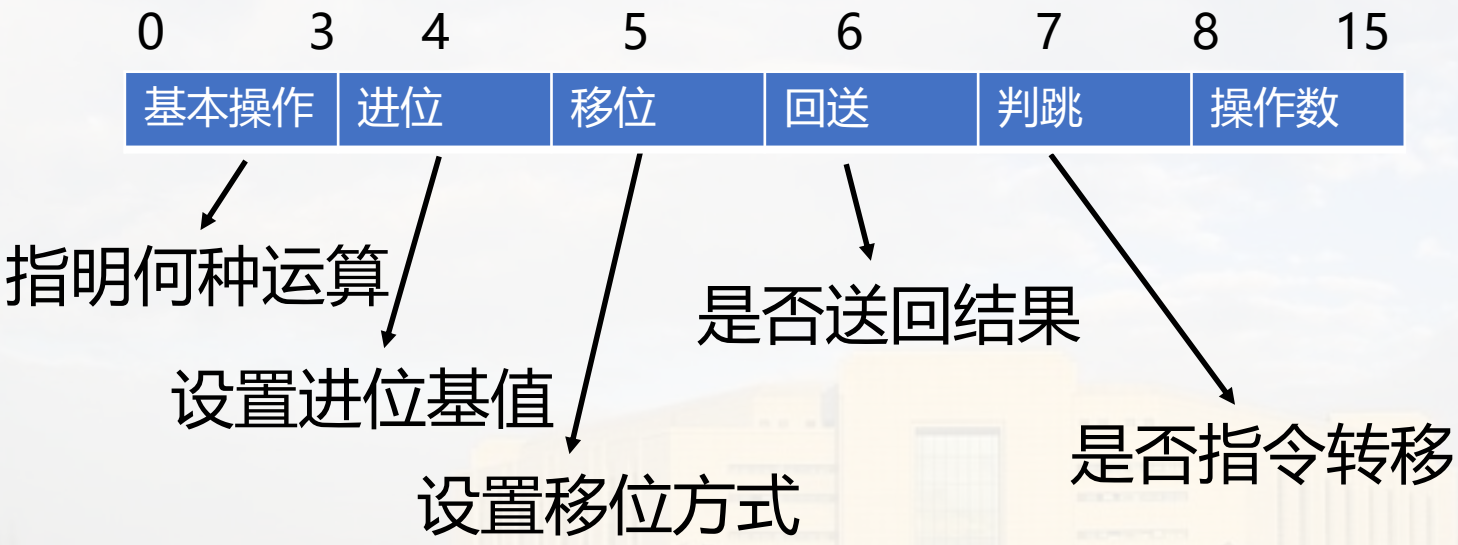
- 同样需要有3、2、1、 0个地址指令, 若采用定长操作码, 则只能表示出16条指令0000 - 1111
- 如果定长操作码超出4位长度, 要表示出全部4种不同地址的指令，则必须扩大字长。

结论:
变长指令格式 → 合理利用存储空间

② 操作码采用方式码编码 (相对于①的扩展操作码)

操作码分为几部分,每部分表示一种操作。

例. 某种计算机的算逻指令



优点: 设置更加灵活、译码更简单。

例题：

某台计算机采用32位字长的指令，地址码为12位，若定义250条两地址指令，则一地址指令最多有（　　）条。

A、4K　　B、8K　　C、16K　　D、24K

解：地址码12位，则两地址指令地址码需要24位，用于操作码编码的则只有8位。

则8位操作码一共可以有256条两地址指令。

题目要求两地址指令是250条，则前8位还有6种编码方式可以用于一地址指令的编码。

在前8位固定的前提下，可用于一地址指令编码的位数还剩下12位（32-12-8=12），则可以有4K种可能编码。

因此 前8位固定的6种编码方式下，一共有6*4K=24K种编码方式。

四、指令中的地址结构

地址结构：在指令中明确给出几个地址，给出哪些地址。

地址结构的主要内容：指令给出地址的方式

和指令中地址数量的设置

指令中提供的地址码 { 存储单元地址码

寄存器编号

(1) 指令提供地址的方式

1) 显地址方式： 指令中明显指明地址

→ 直接或间接给出

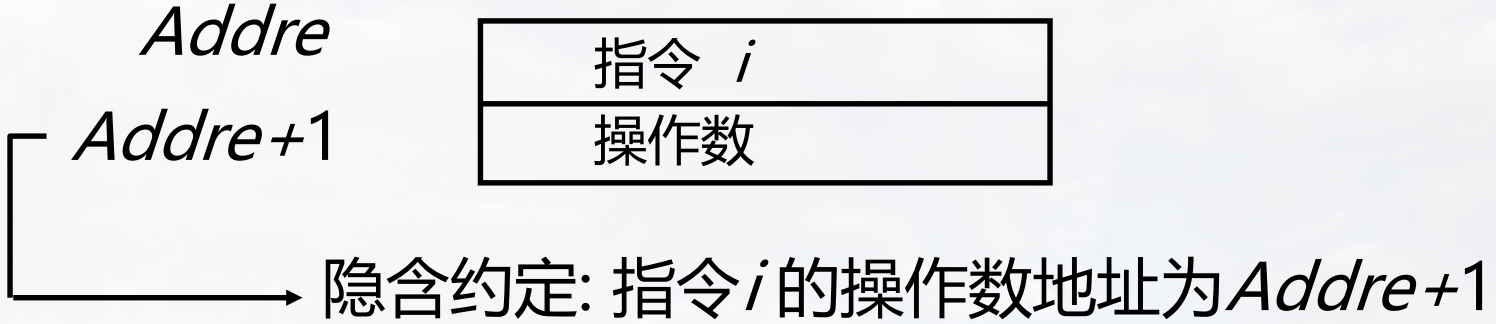
例：MOV AX, (1000H)　直接给出地址

MOV AL, (BX)　间接给出地址

2) 隐地址方式: 地址隐含约定, 不出现在指令中

优点: 可减少指令中的地址数, 简化地址结构

例1:



例2: PUSH AX ⟶ AX压入堆栈指针所指单元

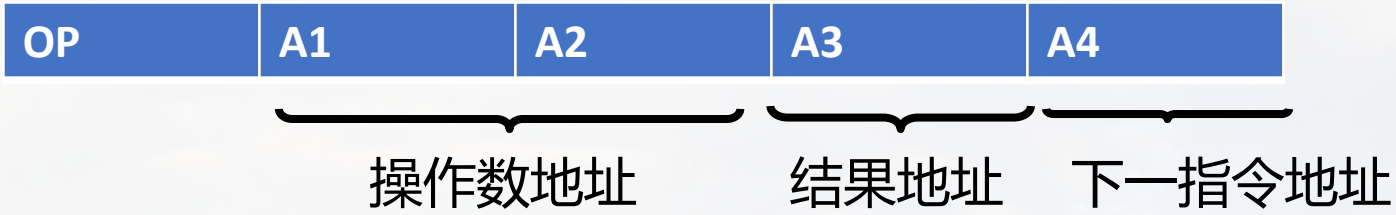
POP AX ⟶ 堆栈指针所指单元弹出到AX

使用隐地址可以减少指令中的地址数，简化地址结构。

(2)地址结构的简化

- **四地址结构指令**

格式：



功能：(A1) OP (A2) → A3

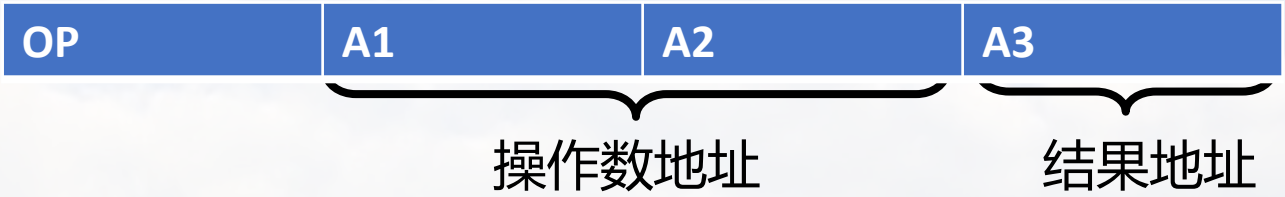
(A4) 下一指令地址

简化：用指令计数器PC指示下一条指令地址。

一、指令格式

●三地址结构指令

格式：



功能：(A1) OP (A2) → A3

(PC) + n → PC

下条指令地址

转移时，用转移地址修改PC内容。

（在模型机中，为简化起见，令n=1）

一、指令格式

●二地址结构指令

格式：

OP	A1	A2
	源/目的	目的/源

功能：(A1) OP (A2) → A2/A1

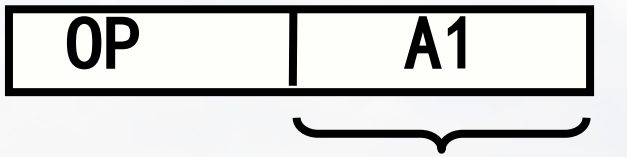
(PC) + n → PC

在绝大多数情况下，两个操作数运算后至少有一个今后不再使用，因而不需要保留。

●一地址结构指令

a.隐含约定目的地的双操作数指令

格式：



功能：　(A1) OP (AC) ⟶ AC

下条指令地址PC：　(PC) + n ⟶ PC

例：

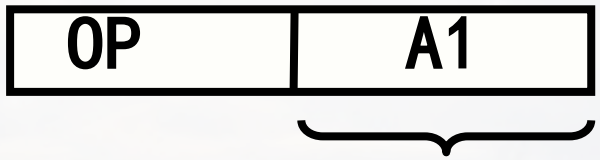
字节乘法：

MUL DL；　DL×AL→AX

●一地址结构指令

b.只有目的操作数的单操作数指令

格式：



功能：　OP (A1) ⟶ A1

下条指令地址PC：（PC） + n ⟶ PC

例：NEG　BL　；求负

NOT　BL　；求非

四、指令中的地址结构

●零地址指令

格式：

如果指令中只给出操作码而没有显地址，则这种指令被称为零地址指令。

四、指令中的地址结构

●零地址指令

a.对只有目的操作数的指令，隐含在指定寄存器内进行操作

例：PUSHF ；FLAGS→入栈

POPF ；出栈到→ FLAGS

LAHF ；FLAGS的低8位→AH

SAHF ；AH→FLAGS的低8位

- **零地址指令**

b.不需要操作数的指令。

例： NOP ； 空操作指令

它本身没有实质性运算操作，执行这种指令的目的就是消耗时间以达到延时的目的。

例: HLT ； 停机指令

它也不需要操作数。

c.对堆栈栈顶单元内容进行操作

如指令PUSHF (压入堆栈)、POPF (弹出堆栈)。

四、练习

某模型机的字长为16位，有单地址和双地址指令。若每个地址字段均为4位，双地址指令只有12条，单地址指令12条，则零地址指令最多可以有（　　）条。

- 双地址指令，两个地址占用8位，指令码可以使用剩余的8位，则双地址指令最多有2^8种。
- 这里使用的12条，则，剩余的2^8-12种可以用于单地址指令，则单地址指令最多有$(2^8-12)*2^4$条；
- 如果单地址指令使用了12条，则剩余的$((2^8-12)*2^4-12)$种指令，可用于零地址指令。每种未使用的单指令码(12位)，可以和剩余的4位配合，表示2^4个零地址指令。则零地址指令最多有$((2^8-12)*2^4-12)*2^4$

2.2 指令寻址方式

- 01. 操作数存储位置
- 02. 寻址方式

一、操作数存储位置

寻址方式是规定如何对地址字段作出解释，以找到操作数。

- 操作数的位置
  - 堆栈
    - 主存开辟（软堆栈）
    - CPU中的寄存器组组成（硬堆栈）
  - 寄存器
    - CPU中的寄存器
    - 外设接口中的寄存器
  - 存储器
    - 主存（包括cache）
    - 外存

一、操作数存储位置

能被CPU直接使用的操作数位置：
- CPU内的R
- $M_{主}$（包括CACHE、接口中的R）

结论：

① CPU能够直接访问的操作数只能存放在主存储器（包括CACHE、接口中的R）或CPU内的寄存器中；

② 由于主存储器的容量远远大于CPU内的寄存器的容量，因此CPU能够直接访问的操作数主要存放在主存储器中。

二、寻址方式

所谓寻址，就是指产生操作数的有效地址，因此将产生操作数有效地址的方式称为寻址方式。

寻址方式可分为四大类，其它的寻址方式则是它们的变型或组合。

① 立即寻址

在读取指令时也就从指令之中获得了操作数，即操作数包含在指令中。

② 直接寻址类

直接给出主存地址或寄存器编号，从CPU内或主存单元内读取操作数。

③ 间接寻址类

先从某寄存器中或主存中读取地址，再按这个地址访问主存以读取操作数。

④ 变址类

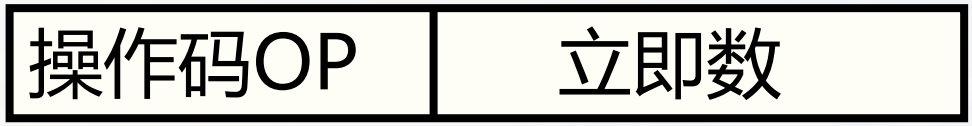
指令给出的是形式地址（不是最终地址），经过某种变换（例如相加、相减、高低位地址拼接等），才获得有效地址，据此访问主存储器以读取操作数。

1、立即数（助记符 I）

指令中直接给出操作数。

操作数在指令中，
长度固定、有限。

定长格式：



变长格式：

操作数在基本指令
之后，长度可变。

例： MOV AX，1234H

主要用于提供常数、设置初值等。
速度快，但灵活性差。

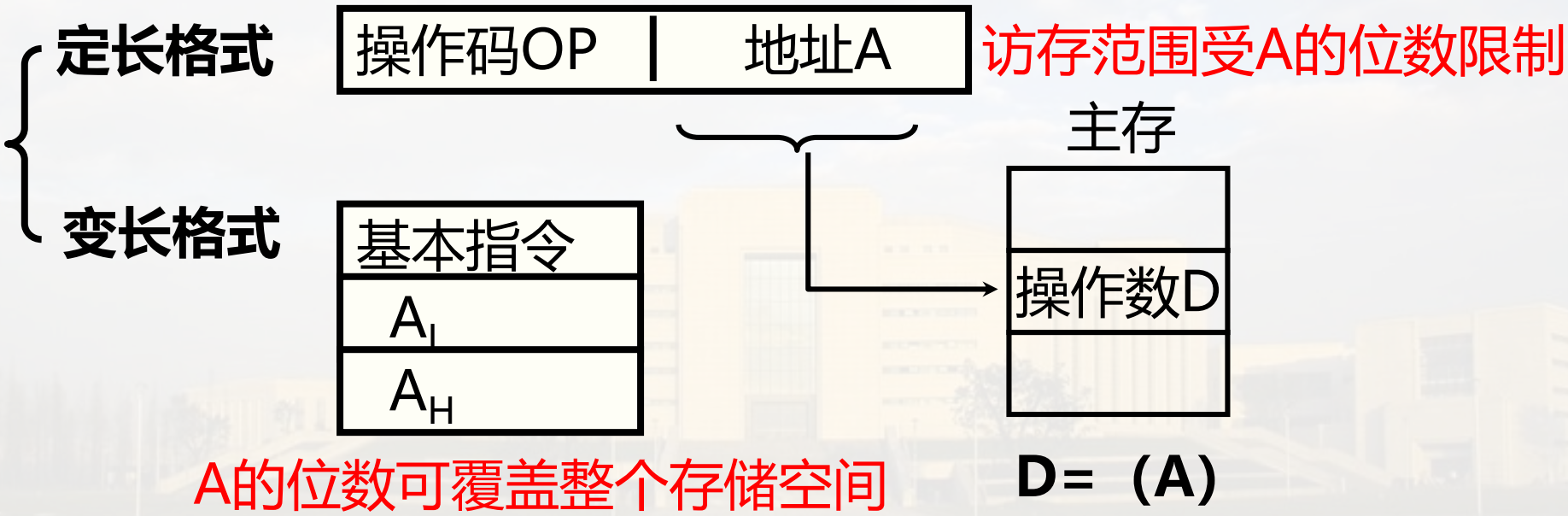
2、直接寻址方式（绝对地址）

由指令直接给出操作数地址，根据该地址可读取或写入操作数，这种方式称为直接寻址方式。

- 存储单元号 (数在M中)
- 寄存器号 (数在R中)

① 直接寻址（主存直接寻址）方式，助记符A

若指令中给出的地址码是主存的某个单元号，操作数存放在该指定的主存单元中,这种寻址方式称为直接寻址或主存直接寻址方式。



二、寻址方式

例：若主存储器数据区的地址与数据之间对应关系如下，指令给出地址码A=2000H，按直接寻址方式读取操作数。

地址	数据
1000H	1A00H
2000H	1B00H
3000H	1C00H

寻址过程：操作数地址 $\xrightarrow{M}$ 操作数

操作数S与地址码A的关系为：S=(A)

例如：MOV AX，[2000H]

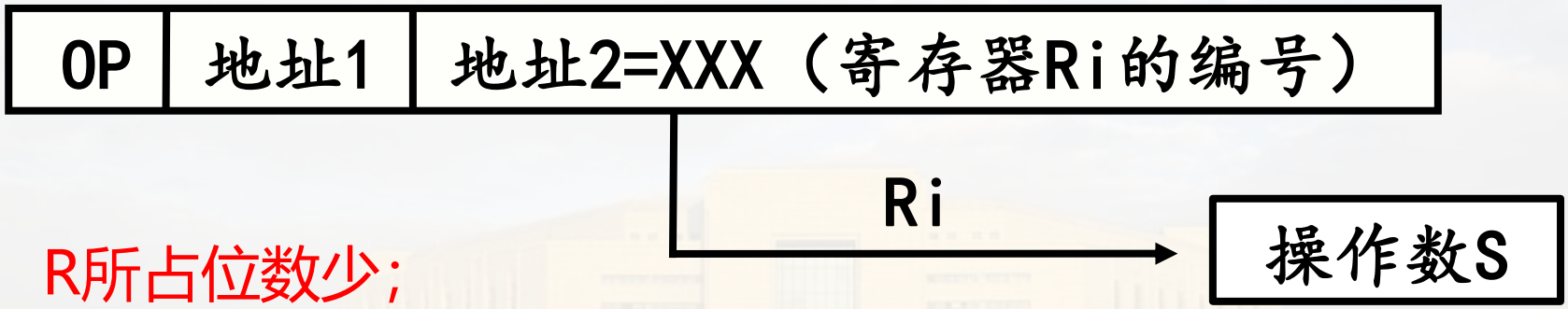
②寄存器寻址（寄存器直接寻址）方式，助记符R

若指令中给出的地址码是寄存器编号，操作数存放在该指定的寄存器中，这种寻址方式称为寄存器寻址或寄存器直接寻址方式。

在CPU中有若干寄存器，其中的一些是可编程访问的，称为可编址寄存器，模型机设计时为它们分配不同的寄存器编号，例如：R0＝000，R1＝001，R2＝010，R3＝011，SP=100，PSW=101，PC=111等。

②寄存器寻址（寄存器直接寻址）方式

指令中给出的寄存器号是Ri（按上述编码则代码为XXX），从寄存器Ri中可直接读取操作数S。



R所占位数少；
访问R比访问M快

②寄存器寻址（寄存器直接寻址）方式

例：若模型机CPU中寄存器内容如下，现指令中给出寄存器号为011，按寄存器寻址方式读取操作数。

R0	1000H
R1	2000H
R2	3A00H
R3	**3C00H**

寻址过程：寄存器号 $\xrightarrow{Ri}$ 操作数

操作数S与寄存器Ri的关系为：S=Ri

例如：MOV AX，BX

2、直接寻址方式（绝对地址）

直接寻址与寄存器寻址方式的比较：

a．直接寻址是访问一次主存才能读取所需操作数；寄存器寻址是从CPU的寄存器中读取操作数，不需访问主存，所需时间大约是从主存中读数时间的几分之一到几十分之一，因而寄存器寻址比直接寻址快得多。故在CPU中设置足够多的寄存器，以尽可能多地在寄存器之间进行运算操作，已成为提高工作速度的重要措施之一。

b．由于寄存器数远少于主存储器的单元数，所以指令中存放寄存器号的字段位数也就大大少于存放主存地址码所需位数。**采用寄存器寻址方式或其他以寄存器为基础的寻址方式（例如寄存器寻址、寄存器间址方式），可以大大减少指令中一个地址的位数，从而有效地缩短指令长度。这也使读取指令的时间减少，提高了工作速度。**

注意：减少指令中地址数目与减少一个地址的位数是两个不同的概念。

- 采用隐地址可以减少指令中地址的数目；
- 采用寄存器寻址方式、寄存器间址方式可以使指令中为给出一个地址所需的位数减少。

其实，均减少了指令长度。

3、间接寻址及其变形

几个术语

1. 若操作数存放在主存某个存储单元中，则该主存单元的地址被称为操作数地址。
2. 若操作数地址又存放在另一存储单元之中（不是由指令直接给出），则该主存单元被称为间址单元，间址单元本身的地址被称为操作数地址的地址，即操作数的间接地址。

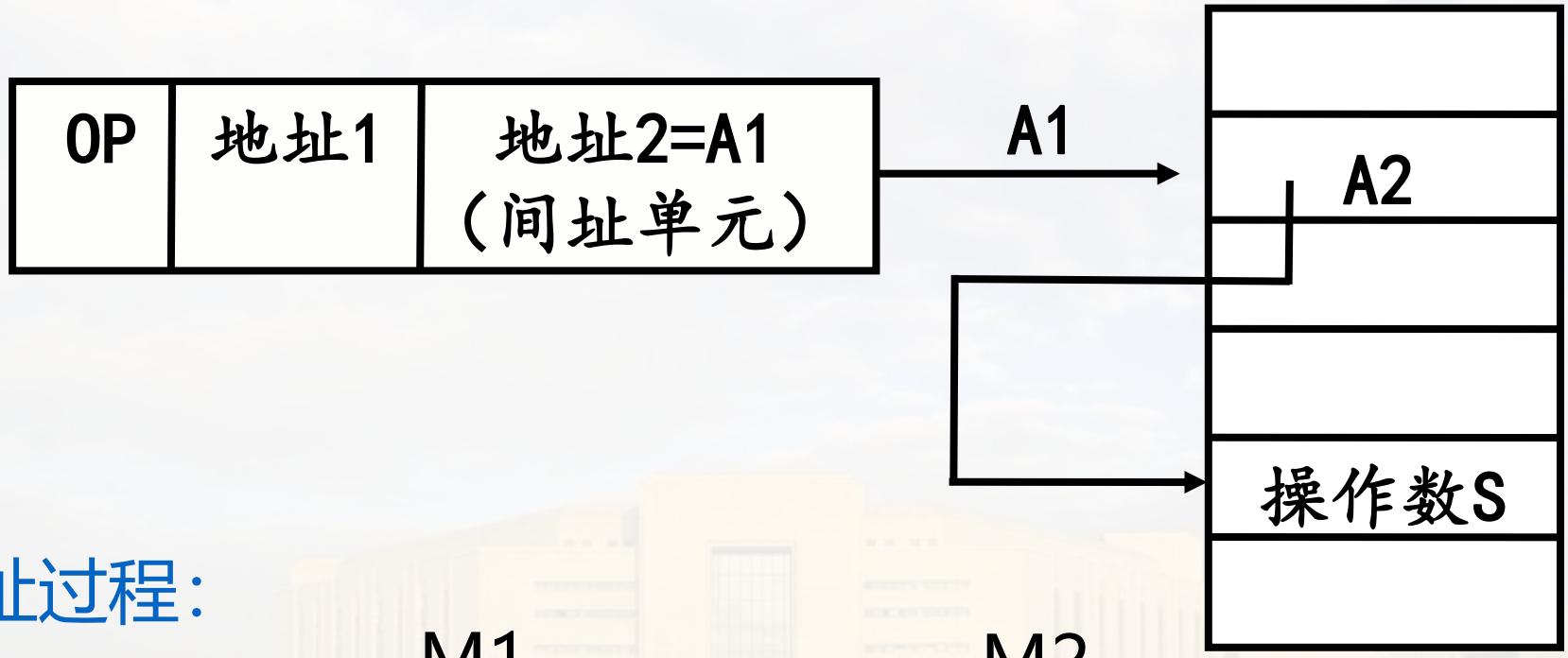
存储单元号 (数在M中)
寄存器号 (数在M中)

① 间接寻址（主存间接寻址）方式(助记符@A或((A)))

若指令中地址给出的是间址单元地址（即操作数地址的地址，而不是操作数地址，且在主存），需要先从中读取操作数地址，然后按照操作数地址再次访问主存，从相应单元中读写操作数，这种寻址方式称为间接寻址或主存间接寻址方式。

二、寻址方式

指令中给出地址A1，据此访问间址单元，从中读取地址A2，按A2再访问一次主存，读取操作数S。



寻址过程：

间址单元地址 —M1→ 操作数地址 —M2→ 操作数

操作数S与地址A1的关系为：S=@A1 或=((A1))

例：若主存储器数据区的地址与单元内容之间对应关系如下，指令给出地址码A=2000H，按间接寻址方式读取操作数。

地址	存储内容
1000H	4000H
2000H	3000H
3000H	AC00H

指令给出间址单元地址A=2000H；

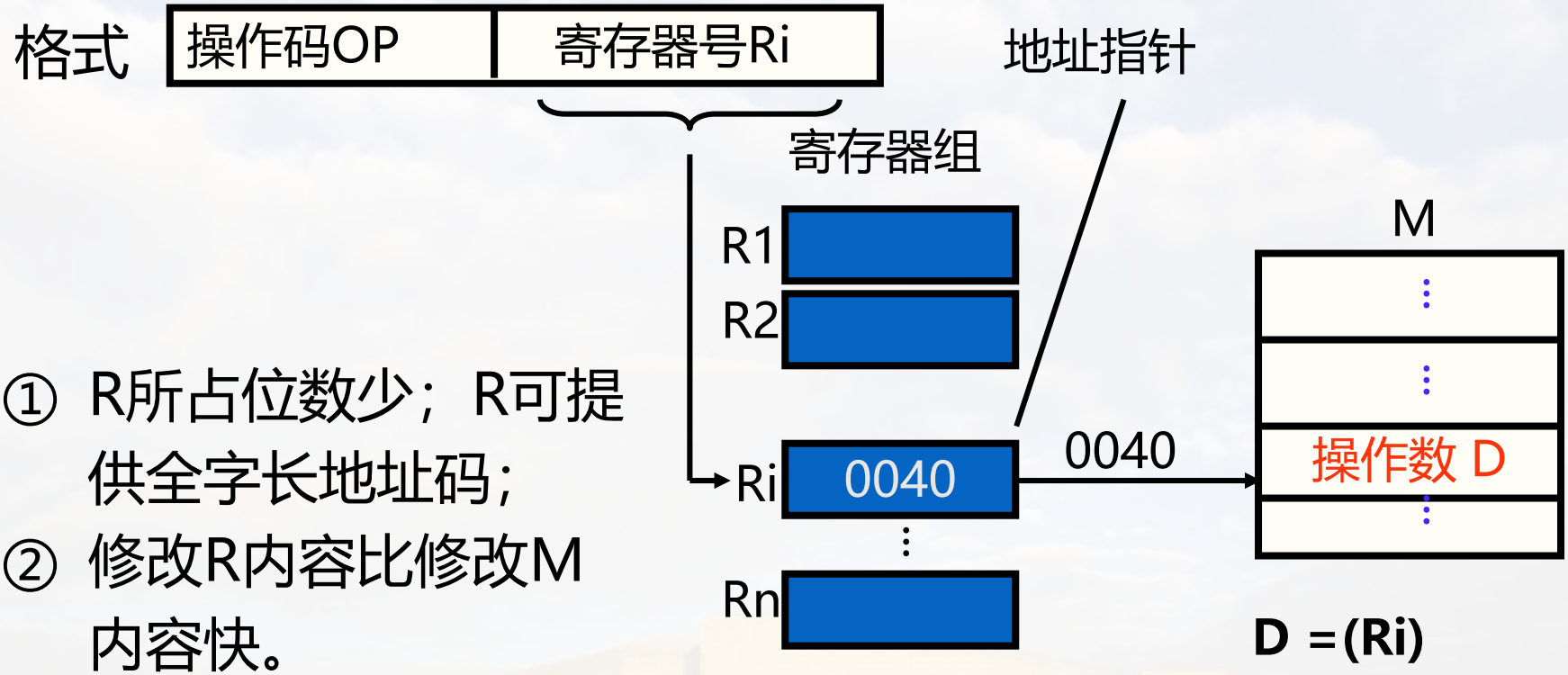
据此访问主存储器，则操作数地址(A)=3000H；

按此地址再次访问主存储器，则操作数S=((A))=AC00H。

② 寄存器间接寻址方式　助记符(R)

若指令中给出的地址码是寄存器编码，被指定的寄存器中存放的是操作数地址，按照该地址访问某主存单元，该单元的内容为操作数，这种寻址方式称为寄存器间接寻址。

二、寻址方式



① R所占位数少；R可提供全字长地址码；

② 修改R内容比修改M内容快。

指针不变(由指令指定)，指针内容可变，使同一指令可指向不同存储单元，以实现程序的循环、共享，并提供转移地址。

例：若模型机的指令中给出寄存器号为001，按寄存器间址方式读取操作数。

寄存器：			主存单元：	
	R0	1000H	1000H	3A00H
	R1	2000H	2000H	2C00H
	R2	3000H	3000H	3B00H

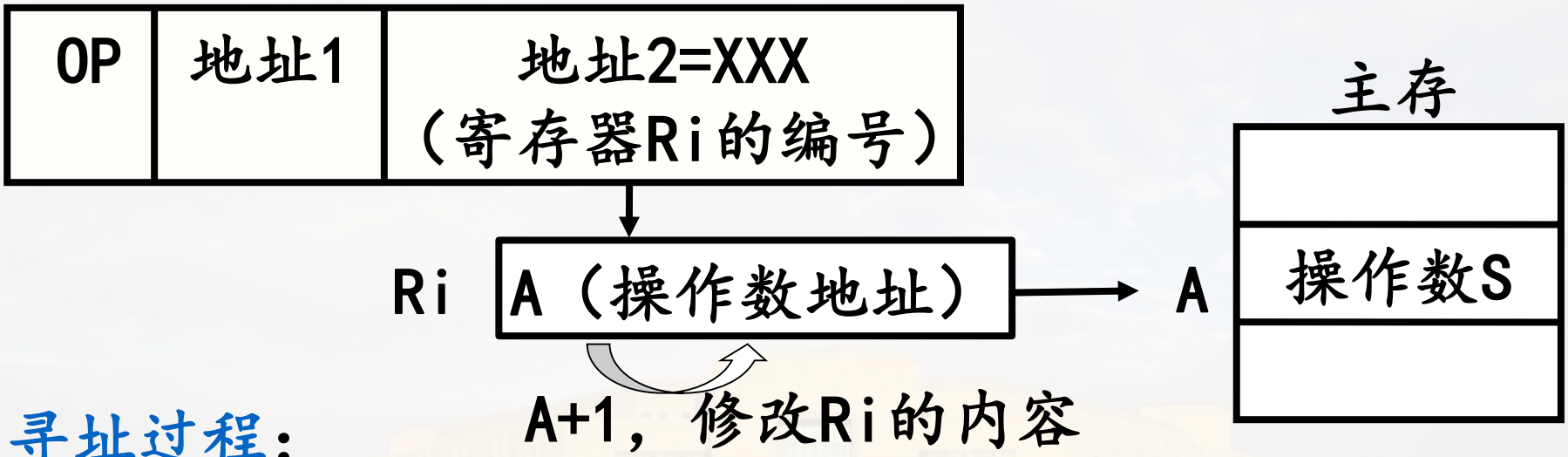
指令指定的寄存器为R1，则操作数地址R1=2000H，据此访问主存储器，则操作数为S= (R1)= 2C00H。

③ 自增型寄存器间址方式

寄存器间址的一种变型，若指令中给出存放操作数地址的寄存器号，从寄存器中读出操作数地址后，寄存器内容加1，这种寻址方式称为自增型寄存器间接寻址。自增型寄存器间址方式常用助记符(R)+表示。

二、寻址方式

指令中在地址段给出的是寄存器号Ri，从Ri中读出的是操作数地址A，按地址码A访问主存，从相应单元中读取操作数S，同时，对寄存器Ri 中的内容加1。



寻址过程：

寄存器号 —Ri→ 操作数地址 —M→ 操作数

Ri内容+1

操作数S与寄存器Ri的关系为：S=(Ri)

Ri=Ri+1

例：若模型机的指令中给出寄存器号为010，按自增型寄存器间址方式读取操作数，并修改指针。

寄存器：R0 1000H　　主存单元：3000H A300H
R1 2000H　　3001H BC00H
R2 3000H

指令指定的寄存器为R2，则操作数有效地址R2= 3000H；

按照该地址访问主存储器，则操作数为S= (R2)= A300H；

R2内容加1后，指针内容修改为R2=3001H。

照此继续，通过重复执行这同一条指令就可以沿着地址码增加的方向，访问从3000H单元开始的一段连续区间。

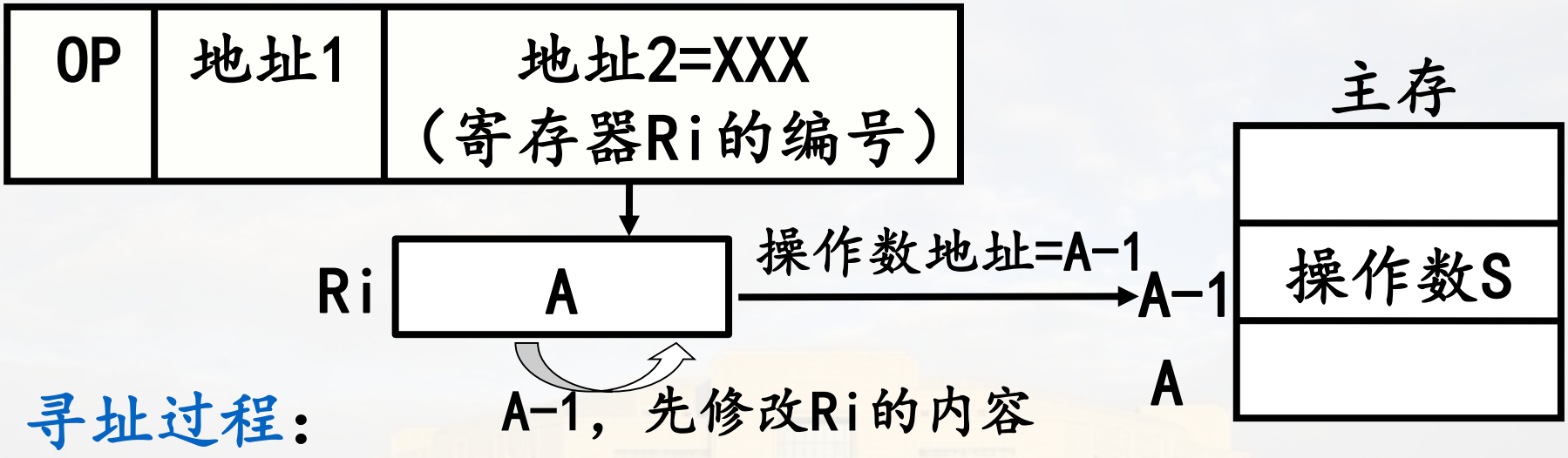
④ 自减型寄存器间址方式

寄存器间址的又一种变型。

若指令中给出寄存器号，被指定的寄存器内容减1后作为操作数地址，按照该地址访问主存储器，相应的主存单元内容为操作数，自减型寄存器间址方式常用助记符-(R)表示。

二、寻址方式

指令在地址段给出的是寄存器号Ri，将Ri中的内容减1作为操作数地址A，按地址码A访问主存，从相应单元中读取操作数S。



寻址过程：

寄存器号 —Ri→ 操作数地址=Ri-1 —M→ 操作数

操作数S与寄存器Ri的关系为：S=(Ri-1)

Ri=Ri-1

例：若指令中给出寄存器号为010，按自减型寄存器间址方式修改指针，并读取操作数。

寄存器：			主存单元：	
	R0	1000H	2FFEH	A300H
	R1	2000H	2FFFH	27FFH
	R2	3000H	3000H	BC00H

指令指定的寄存器为R2;

将R2的内容减1后作为操作数地址R2 - 1= 3000H - 1= 2FFFH;

从地址2FFFH单元中读得操作数S= (R2 - 1)= 27FFH。

照此继续，通过重复执行这同一条指令，就可以访问从2FFFH开始，沿地址码减小方向的一个连续数据区。

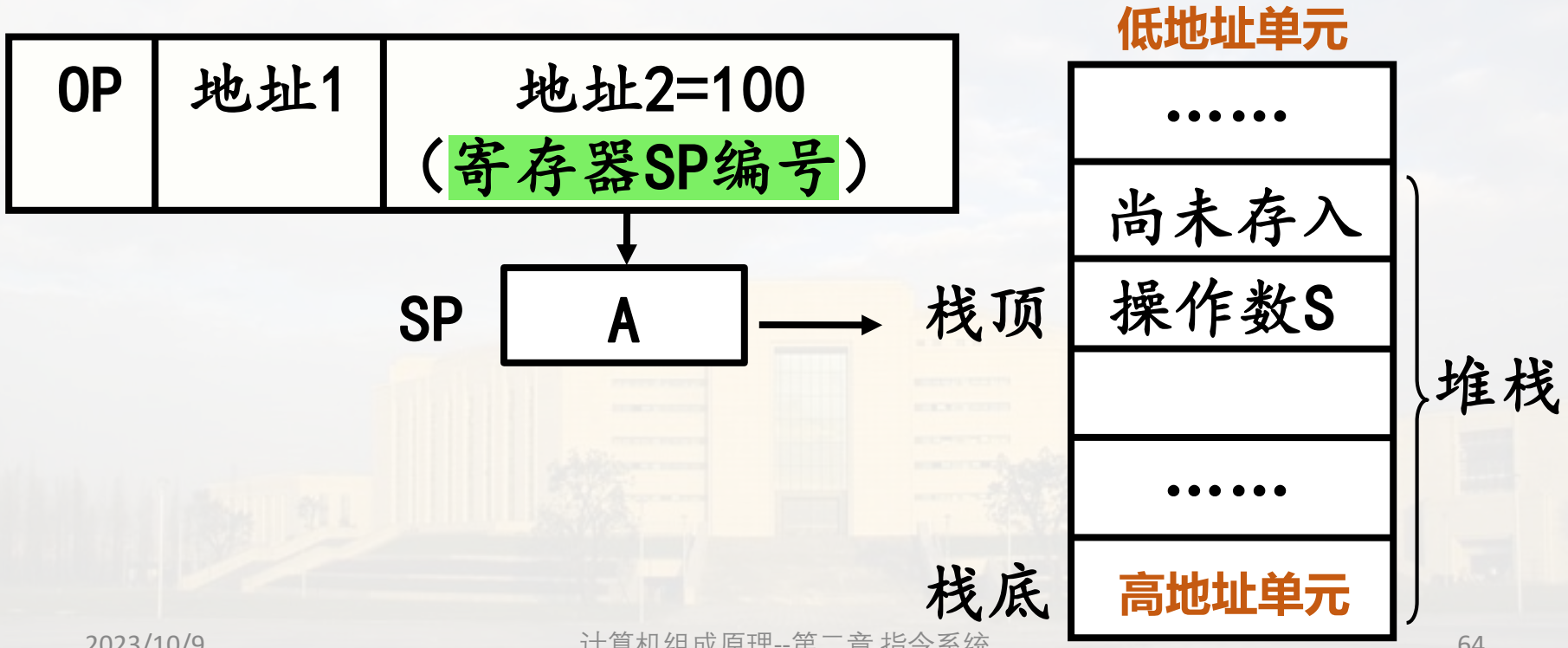
⑤ 堆栈寻址

堆栈寻址方式是指操作数在堆栈中，指令隐含约定由堆栈指针SP寄存器提供栈顶单元地址（SP也可以编码形式出现在指令中），进行读出或写入的一种寻址方式。

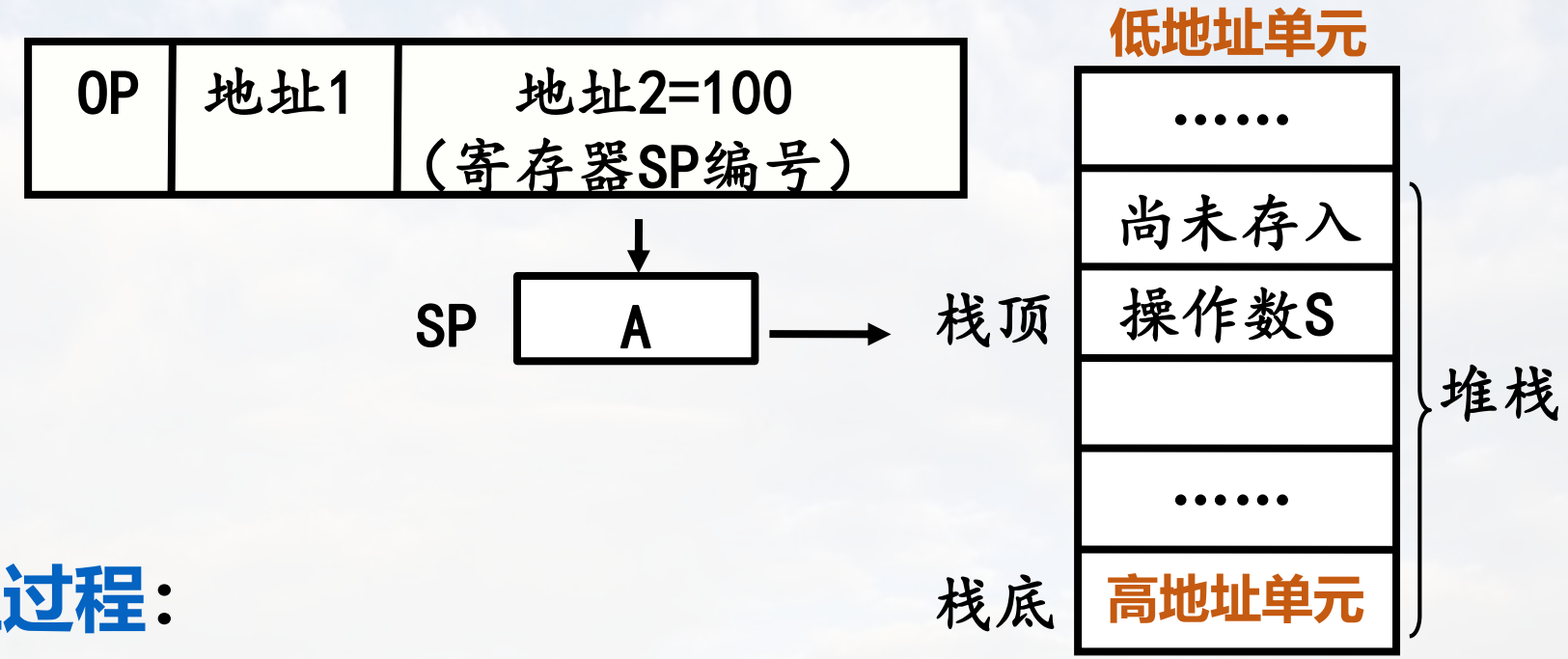
根据压入数据时栈顶单元的地址是减小还是增大或不变,可以将堆栈的工作方式大致分为向上生成方式、向下生成方式两种。后面以上生成方式进行讲解。

二、寻址方式

指令在地址段给出的是寄存器号SP，对SP中的内容进行相应操作（减1或不变，对应压栈或出栈），得到操作数地址A，按地址码A访问主存，从相应单元中读取操作数S。



二、寻址方式



寻址过程：

1）压栈：寄存器号 $\xrightarrow{SP}$ 操作数地址= SP−1 $\xrightarrow{M}$ 操作数

操作数S与寄存器SP的关系为：SP=SP−1　S=(SP−1)

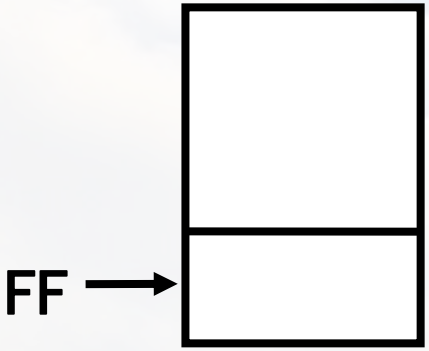
2）出栈：寄存器号 $\xrightarrow{SP}$ 操作数地址 $\xrightarrow{SP}$ 操作数

操作数S与寄存器SP的关系为：S=(SP)　SP=SP+1

例：对堆栈的连续压入与连续弹出（自底向上生长方式），SP内容为00FFH，压入第一个数据元素a，然后压入第二个数据元素b，最后弹出栈顶单元内容。

最基本的堆栈操作指令有两种：

1. 压入指令PUSH（进栈、压栈），将指定的操作数存入栈顶；
2. 弹出指令POP（出栈），将栈顶数据读出，送入指定目的地。

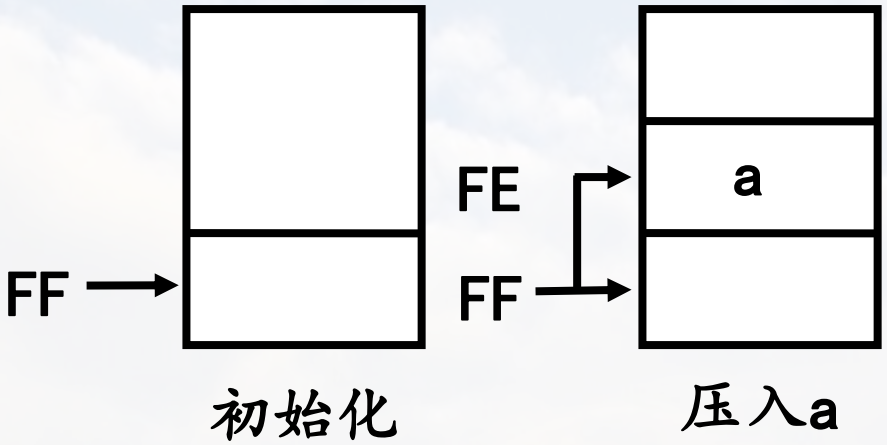


初始化

1）初始化

将栈底地址即初始值送入堆栈指针SP寄存器，本例中假定初始值为00FFH。

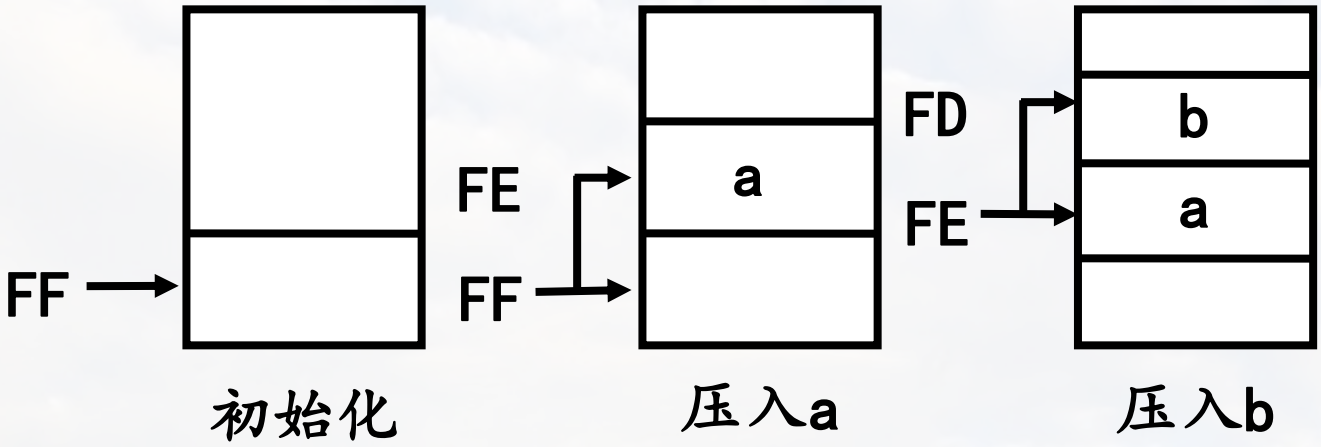
（在某些实际系统中，将压入数据的第一个堆栈单元称为栈底，SP则初始化为栈底地址+1）



1）压入第一个数据元素a

a. SP - 1→SP。先修改堆栈指针，指向待存入的新栈顶。SP内容00FFH减1后，修改为00FEH。

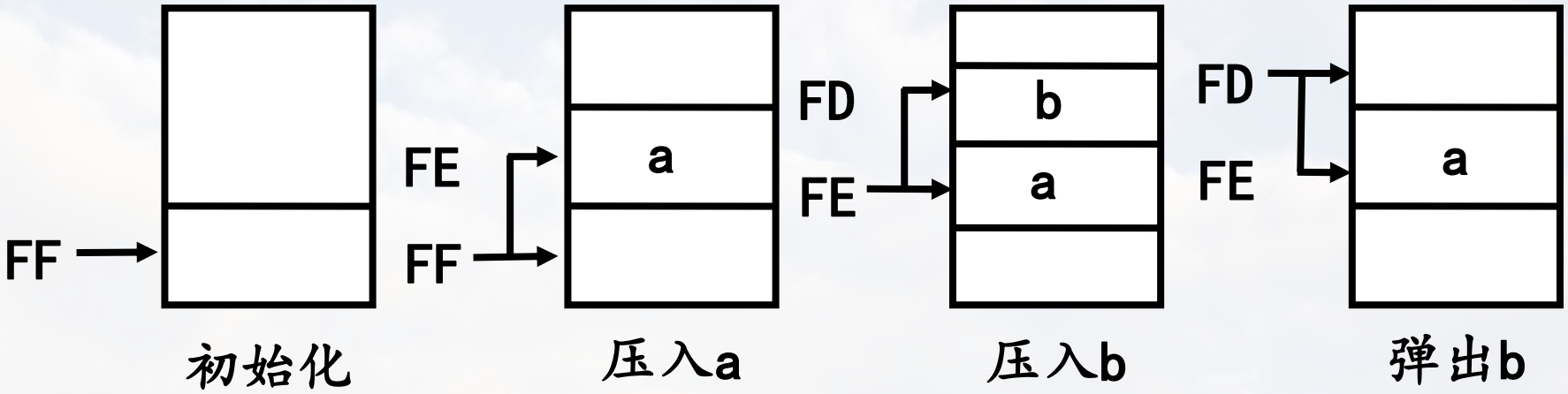
b. 压栈。将SP的内容00FEH送入地址寄存器(MAR)，将待存数据a送入00FEH单元,00FEH单元成为新栈顶。



3）压入第二个数据元素b

a. SP - 1→SP。SP内容由00FEH修改为00FDH。

b. 压栈。将待存数据b送入00FDH单元，00FDH单元成为新栈顶。



4）弹出

a. 将SP的内容00FDH送入主存地址寄存器MAR，从栈顶单元将最后压入的数据b读出，送入指定地方。

b. SP + 1→SP。弹出数据后再修改堆栈指针，让SP内容加1，由00FDH修改为00FEH，指向新栈顶。

例如：。。。。。。。。。。。。。。。。。。。。。。

```
STACK1  SEGMENT PARA STACK
    DW   100  DUP（0）           ；长度100（64H）
STACK1  ENDS
。。。。。。。。。。。。。。。。。。。。。。
PHSH  AX                         ；入栈
PUSH  DS
PUSH  DATA-WORD
PUSHF
。。。。。。。。。。。。。。。。。。。。。。
POPF                             ；出栈
POP  DATA-WORD
POP  DS
POP  AX
```

⑥多重间接寻址（主存多重间接寻址）方式(了解)

上述间址方式均只有一层间址。

有的机器允许多重间址,即根据指令找到间址单元，其中的内容还不是操作数地址,而是又一层间址单元的地址；根据该地址访问又一层间址单元,取出来的才是操作数地址(存放操作数的存储单元的地址码)。

⑥多重间接寻址（主存多重间接寻址）方式

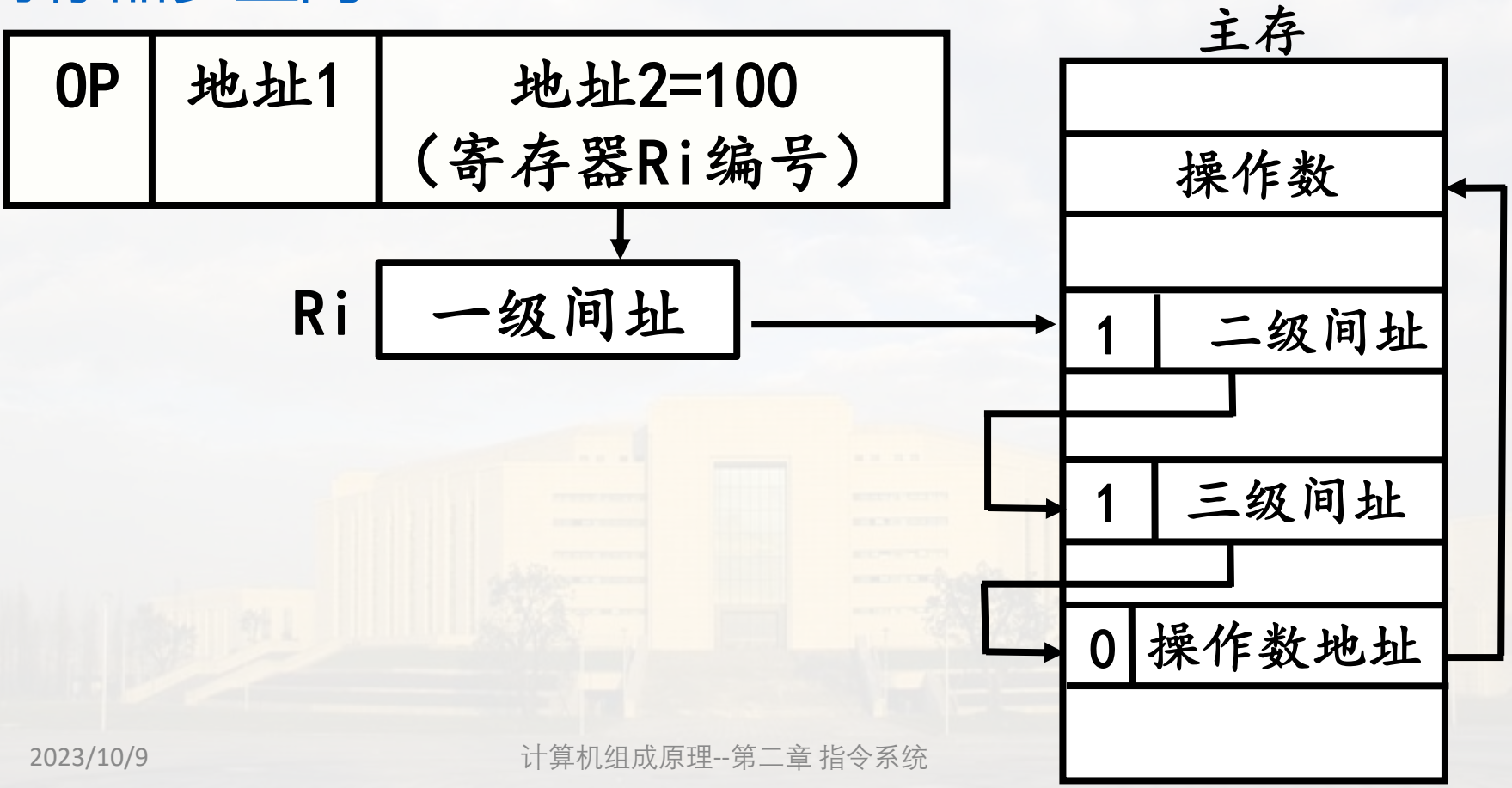
怎么知道从存储单元中读出的是有效操作数地址还是间接地址呢?

可在间址单元的存储内容中设置一位间址标志位,一般选取最高位。

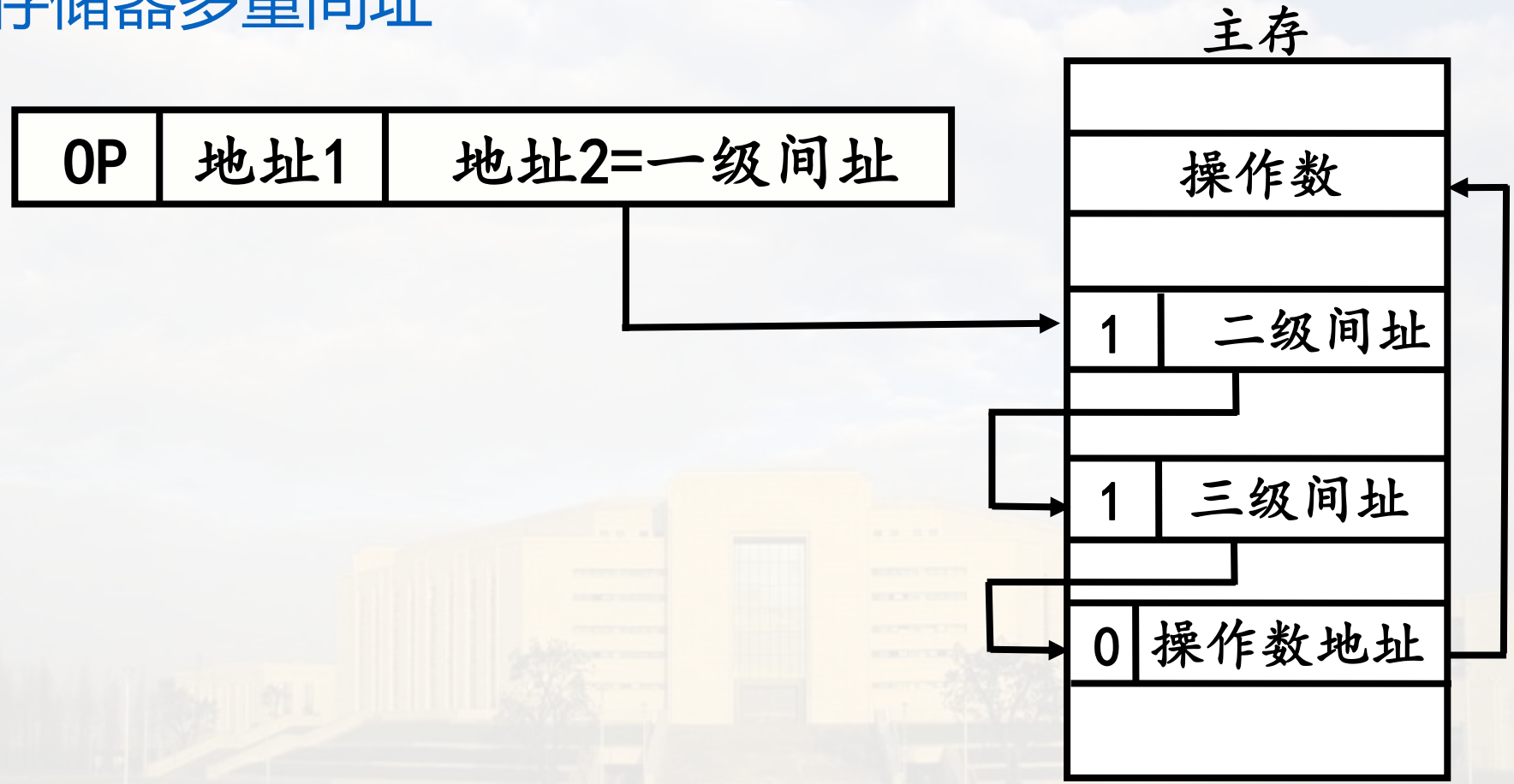
当该位为1时,表明所读出的是间接地址,还需再次间址；直到该位为0,表明这次取出的是操作数的有效地址，按这个地址访问主存，读出的是操作数(即间址过程结束)。

多重间址有分为寄存器多重间址与存储器多重间址。

寄存器多重间址



存储器多重间址



⑥多重间接寻址（主存多重间接寻址）方式

1)寄存器多重间址：

寄存器号 $\xrightarrow{R}$ 一级间址 $\xrightarrow{M}$ …… $\xrightarrow{M}$ 操作数

2)存储器多重间址：

一级间址 $\xrightarrow{M}$ 二级间址 $\xrightarrow{M}$ …… $\xrightarrow{M}$ 操作数

多重间接寻址产生地址的方法提供了编程的灵活性，但多重间址方式增加了访存次数，因而极大的减慢了计算机工作速度，所以现在已很少采用。

4、变址、基址寻址及其变化

通过地址计算使地址灵活可变。

① 变址寻址 助记符 X(R)

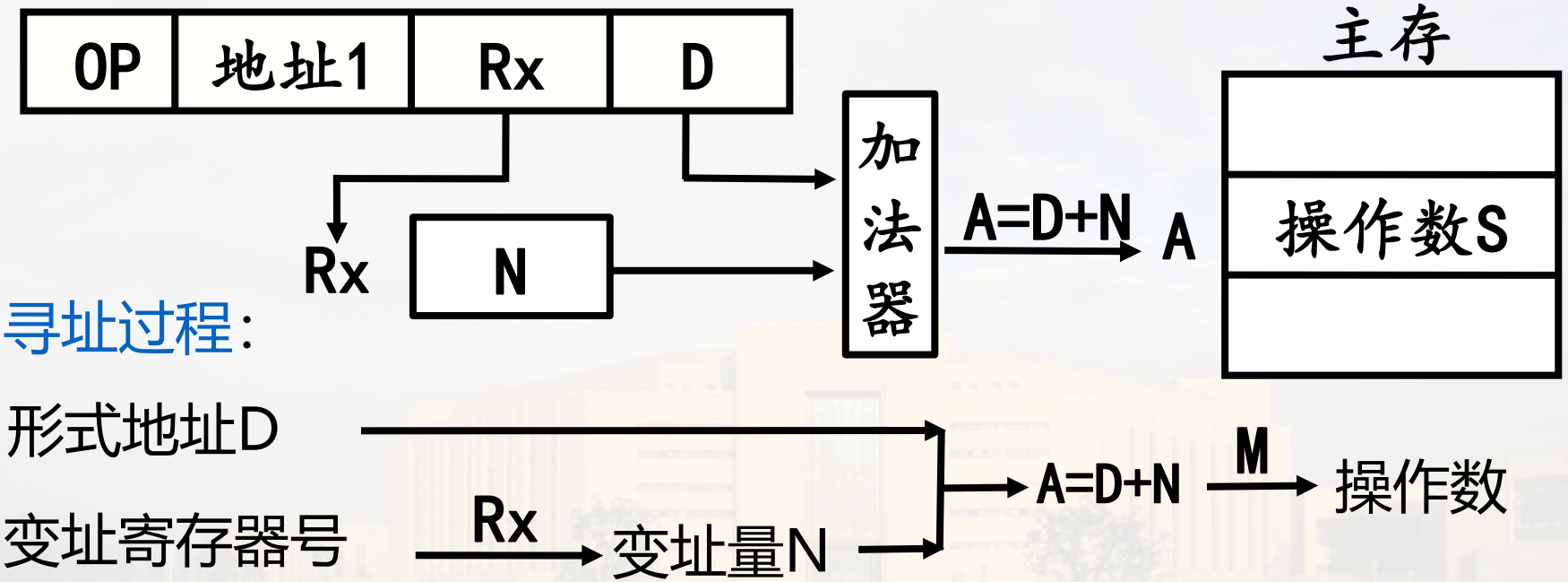
若指令中给出变址寄存器号和一个**形式地址**，变址寄存器的内容(称为变址量)与形式地址相加，得到操作数有效地址(即操作数实际地址)，按照有效地址访问某主存单元，该单元的内容即为操作数,这种寻址方式称为变址寻址方式。变址方式常用助记符X(R)表示。

在8086/8088中推荐使用SI（源变址寄存器）、DI（目的变址寄存器）（隐含使用DS段）

二、寻址方式

指令中为获得某个操作数地址给出两个信息：

形式地址D，变址寄存器Rx。有效地址A= D+Rx= D+N，根据A访问主存储器，读写操作数S。



操作数S与形式地址D 、变址寄存器Rx的关系为： S=(Rx+ D)

（4）变址、基址寻址及其变化

例：若指令中给出变址寄存器号为000，形式地址为1000H，按变址方式读取操作数。

寄存器：			主存单元：	
	R0	0030H	1000H	7A00H
	R1	1000H	102FH	1000H
	R2	2000H	1030H	2C00H

变址寄存器是R0，则变址量为N=R0＝0030H；

形式地址D＝1000H，则变址计算：

A= D+R0 =D+N=1000H+0030H=1030H；

据此访问主存储器，读得操作数S= (A)= 2C00H。

二、寻址方式

例：用变址方式访问首地址为A的一组连续区间内的数组元素。

A为存储区首址；

R_X为所访单元距离首址的长度；

R_X初值为0，每访问一个单元，$R_X+1 \rightarrow R_X$ 。

A的位数有限，若不能提供全字长地址码，会使访存空间受到限制。

OP	地址1	Rx	D

② 基址寻址

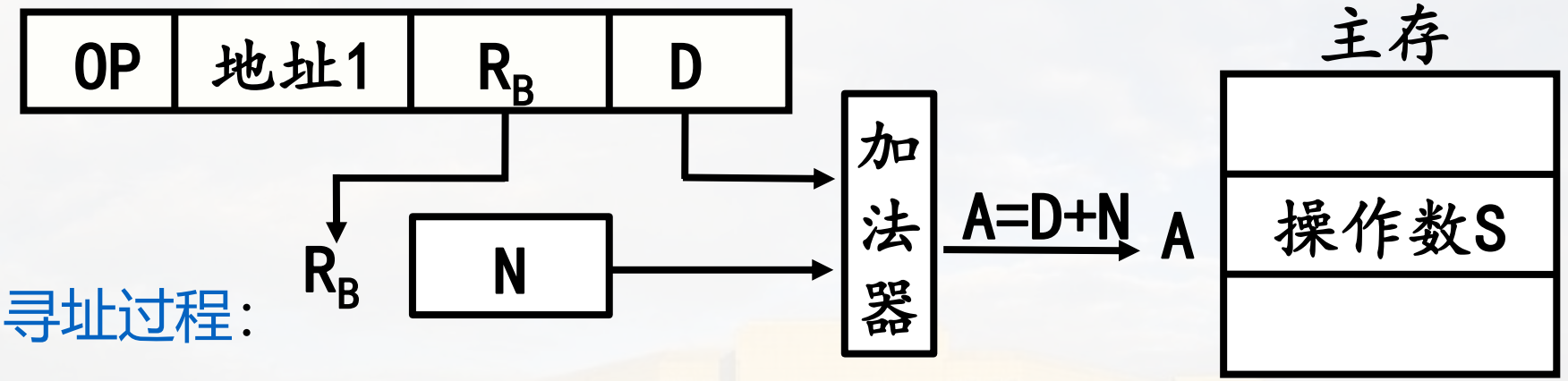
若指令中给出基址寄存器号和一个形式地址，基址寄存器内容（作为基准地址）与形式地址（作为位移量）相加，其和为操作数有效地址(即操作数实际地址)，按照该地址访问主存储器，该单元的内容即为操作数，这种寻址方式称为基址寻址。

在8086/8088中推荐使用BX（基址寄存器：隐含使用DS段）、BP（基址指针寄存器：隐含使用SS段）

二、寻址方式

指令中为获得某个操作数地址给出了两个信息：

形式地址D，基址寄存器R_B。有效地址A= D+R_B= D+N，根据A访问主存储器，读写操作数S。



寻址过程：

形式地址D

基址寄存器号 R_B 基准地址N

A=D+N M 操作数

操作数S与形式地址D 、变址寄存器R_B的关系为：S=(R_B+ D)

二、寻址方式

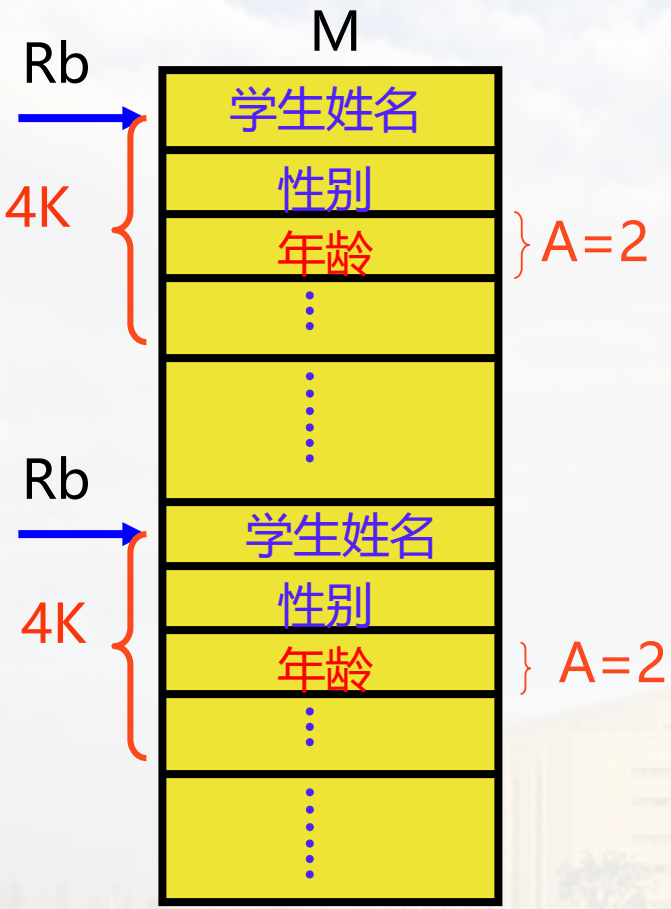
在CPU中有专用的基址寄存器，或者由程序指定某个通用寄存器担任基址寄存器。

例：若指令中给出基址寄存器号为001，形式地址（位移量）为002FH，按基址寻址方式读取操作数。

寄存器：			主存单元：	
	R0	0030H	1000H	7A00H
	R1	1000H	102FH	3A00H
	R2	2000H	1030H	2C00H

基址寄存器为R1，则基准地址为N= R1= 1000H;
位移量为D=002FH，则基址计算:
A=D+R1=D+N= 002FH+1000H=102FH;
据此访问主存储器，读得操作数S= (A)= 3A00H。

基址寻址（续）



A的位数只需覆盖一个较小的存储区间

改变Rb的内容，程序能访问存储空间中任何一个定长区间(4K)。

二、寻址方式

	变址寻址	基址寻址
相同点	有效地址计算方法相同； 在一些计算中，都是由相同硬件实现。	
不同点	变址寄存器提供修改量（可变的），而指令中提供基准量（固定的）	基址寄存器提供基准量（固定的），而指令中提供位移量（可变的）
	面向用户的，用于访问字符串，向量和数组等成批数据	面向系统，主要用于逻辑地址和物理地址的转换，用以解决程序在主存中的再定位和扩大寻址空间等问题

在某些大型机中，基址寄存器只能由特权指令来管理，用户指令无权操作和修改。在某些小，微型计算机中，基址和变址寻址实际上是合二为一的。

4、变址、基址寻址及其变化

在某些大型机中，基址寄存器只能由特权指令来管理，用户指令无权操作和修改。在某些小，微型计算机中，基址和变址寻址实际上是合二为一的。

例如：8086指令系统中，把变址寻址和基址寻址统一成了**寄存器相对寻址。**

例如：MOV DX，VAR[BP] ；

等价于MOV DX，SS：VAR+[BP]

③ 基址加变址方式

基址寻址方式的目的是扩大有限字长指令的寻址空间,变址寻址方式的目的是为了灵活修改地址以适应连续区间(程序循环)的操作。

如果在同一条指令中要兼有这两种功能，可以采取复合型的寻址方式，即基址加变址方式。

③ 基址加变址方式

指令给出两个寄存器号和一个地址量，寄存器内容与地址量之和为有效地址。

格式

变址寄存器号　　基址寄存器号　　位移量

D =RX+Rb+ A

便于处理两维数组。

二、寻址方式

例：某商场的销售金额汇总表如表所示，采用基址加变址方式查询某天的销售额。假定：存储首址为1000H（内容为1月1日销售金额），每天的销售金额存放在一个主存单元之中，为每个月份分配31个单元。

表 商场销售金额统计示意表

月 \ 日 / 金额（万）	1	2	……	17	……	30	31
1	100	100		80		60	80
2	50	60		100			
6	60	80		90		80	
12	100	100		100		90	100

运用基址加变址的寻址方式查找6月17日的销售金额。

③ 基址加变址方式

若：指定R0为基址寄存器，其中存放表格的首址1000H，作为基准地址；

指定R1为变址寄存器，其中存放的变址量为：从表的首址到六月份这一行的起始单元之间的距离，即

$(5\times31)_{10}=(155)_{10}=(10011011)_2=009BH$；

位移量为从该行起点到17日这一列的距离，即0010H。则有：

操作数有效地址=1000H+009BH+0010H=10ABH

③ 基址加变址方式

通过修改基址,还可以实现程序段在存储空间中的重新定位。

现在,变址加基址的寻址方式广泛应用于许多计算机中，包括微型计算机。

例如：MOV AX，10[BX][SI] ；

等价于MOV AX，DS：10+[BX]+[SI]

MOV DX，VAR[BP][SI] ；

等价于MOV DX，SS：VAR+[BP]+[SI]

④ 相对寻址（浮动编址）

变址寻址的变形，助记符为X(PC)

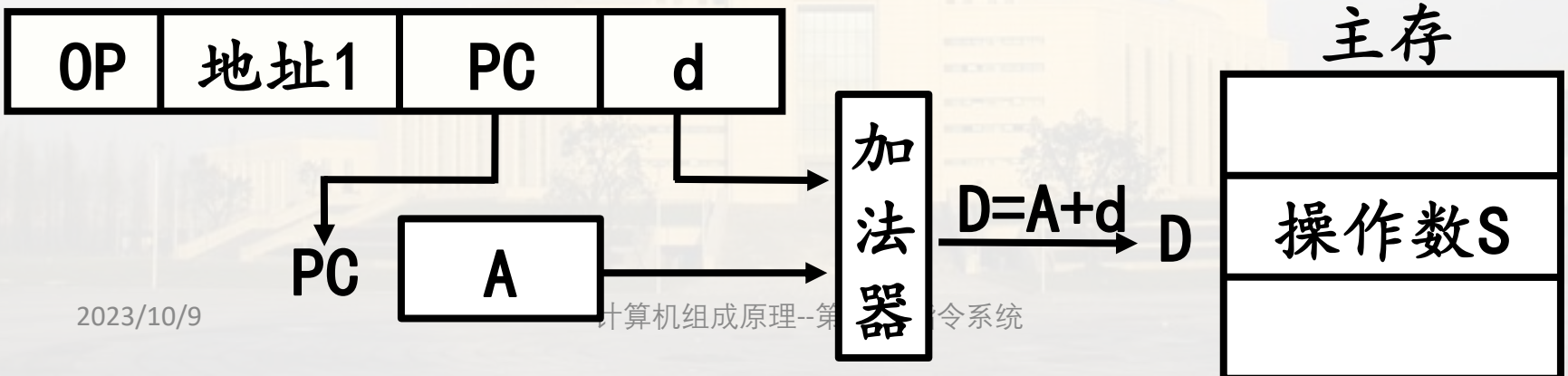
以程序计数器PC当前的内容作为基准地址，或是隐含地指定PC，指令中给出的形式地址作为位移量（可正、可负），二者相加后形成操作数的有效地址。这种寻址方式实际上就是以当前PC内容为基准，相对它进行位移定位（往前或往后），所以称为相对寻址。

二、寻址方式

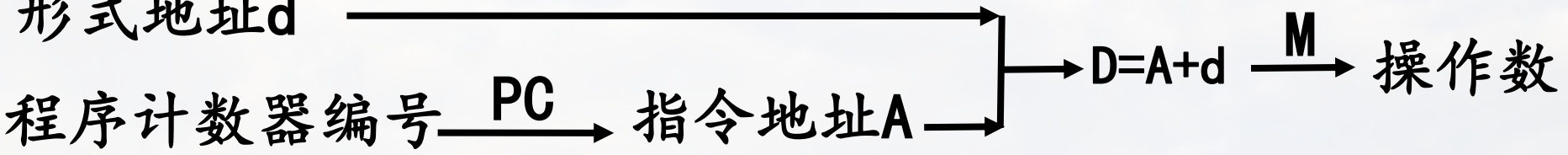
程序计数器PC的内容为A（注意：寻址时，当前PC的内容和当前指令执行的地址可能不一致！）；

指令中形式地址段给出位移量d，它是从现行PC寄存器所指向的存储单元到操作数S所在单元之间的距离（单元数）；

操作数有效地址D=A+d，据此访问主存储器，从D单元中读取操作数。



寻址过程：



操作数S与形式地址d 、变址寄存器PC的关系为：

$$S=(PC+d)$$

PC作为基准地址来看待。

例：若读取出一条指令，该指令采用相对寻址方式读取操作数，形式地址(位移量)为0003H。

寄存器：		主存单元：	
PC	1000H	0FFDH	BC00H
R0	2000H	1003H	AF00H
R1	3000H		

现行指令存放在PC= 1000H单元中，则基准地址为1000H；指令给出位移量为d=0003H，则操作数有效地址=PC+d=1000H+0003H=1003H，据此访问主存，操作数为S=(PC+ d)= (1003H)=AF00H。

例：若读取出一条指令，该指令采用相对寻址方式读取操作数，形式地址(位移量)为-0003H。

寄存器：PC 1000H
R0 2000H
R1 3000H

主存单元：0FFDH BC00H
1003H AF00H

现行指令存放在(PC)= 1000H单元中，则基准地址为1000H；指令给出位移量为d=-0003H，则操作数有效地址=PC+d=1000H+(-0003H)=0FFDH，据此访问主存，操作数为S=(PC+ d)= (0FFDH)=BC00H。

4、变址、基址寻址及其变化

⑤ 页面寻址

若不是将PC内容与位移量进行算术加，而是将PC内容的高位段与位移量相拼接，相对寻址就演变成页面寻址。

计算机中通常都采用了页式存储器管理技术，即将主存储器分为若干相同容量的页面，主存单元的地址就可映射 成“页号+页内地址”。

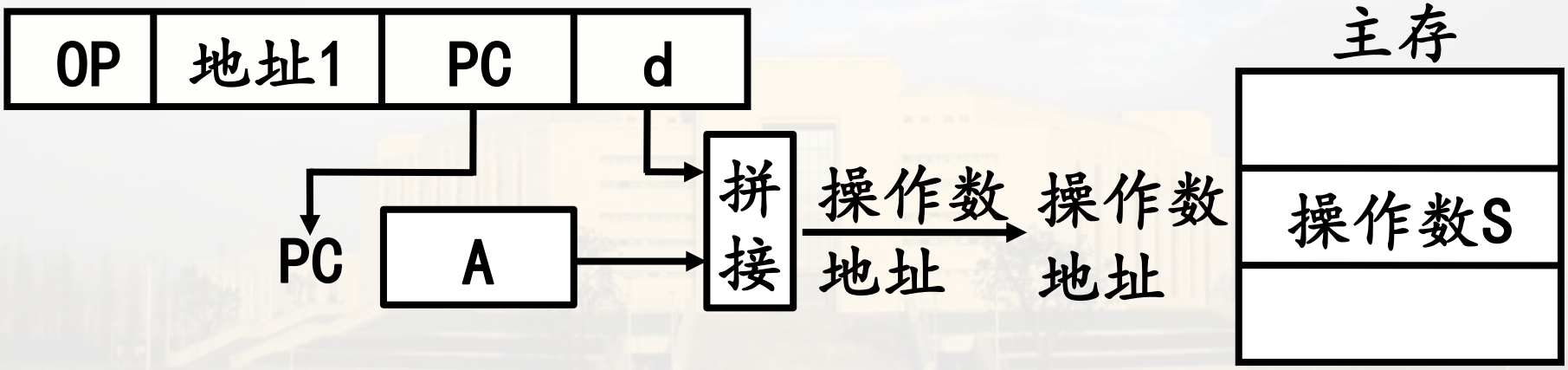
⑤ 页面寻址

程序计数器PC的内容为A，指令中形式地址段给出位移量d。

页号　　页内地址

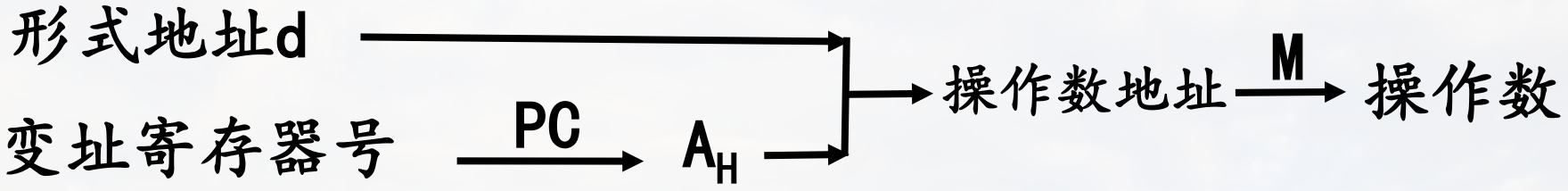
按页面寻址方式，操作数有效地址=(PC_H，d)；

据此访问主存储器，从单元中读取操作数。



⑤ 页面寻址

寻址过程：



操作数S与形式地址d 、变址寄存器PC的关系为：

$$S=(PC_H，d)$$

⑤ 页面寻址

例：若从1030H单元中取出一条指令，该指令采用页面寻址方式读取操作数，形式地址为FFH。

寄存器：			主存单元：	
	PC	1030H	10FFH	AC00H
	R0	2000H	1100H	7FC0H
	R1	3000H		

PC的内容为1030H，其高8位为10（低8位为30），与形式地址FFH相拼接，得到操作数有效地址10FFH，从主存储器中得到操作数AC00H。

⑤ 页面寻址

页面寻址方式适合于采取页面管理的存储组织。

例如:某机主存容量1MB，分为 1K页，每页1KB;

则取PC内容的高10位作为页面号(它也指明了现在程序运行在哪个页面)；

指令中提供10位的位移量(相对于页的起点)，也就是页内单元地址;

二者拼接为 20位有效地址。

5、寻址方式总结

以上四类十余种寻址方式,重点在“数在哪里”(在指令中、在CPU寄存器中、在主存中)。

① 如果操作数在主存中，指令直接给出有效地址还是通过“多次读取”间接获得有效地址(通过寄存器间址、通过存储单元间址)?

② 如何通过计算使地址量可变 (与变址寄存器内容加、与基址寄存器内容加、与程序计数器内容加或拼接)?

5、寻址方式总结

立即数寻址

指令

操作数D

存储器直接寻址

指令

地址A

主存

操作数D

寄存器直接寻址

指令

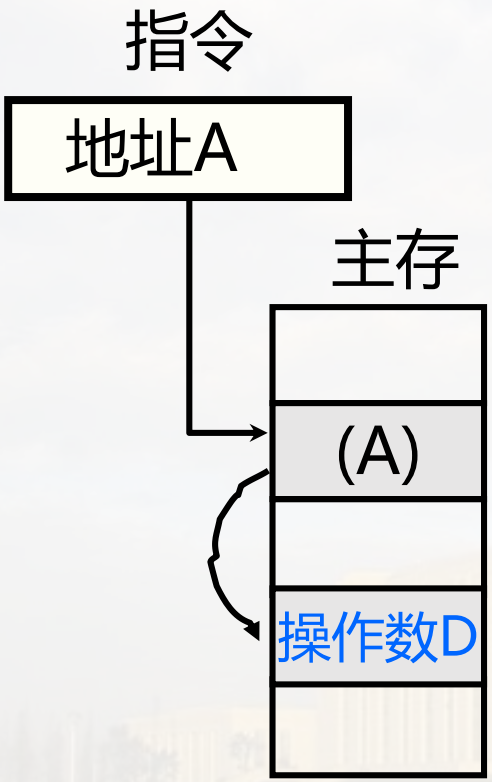
R

寄存器组

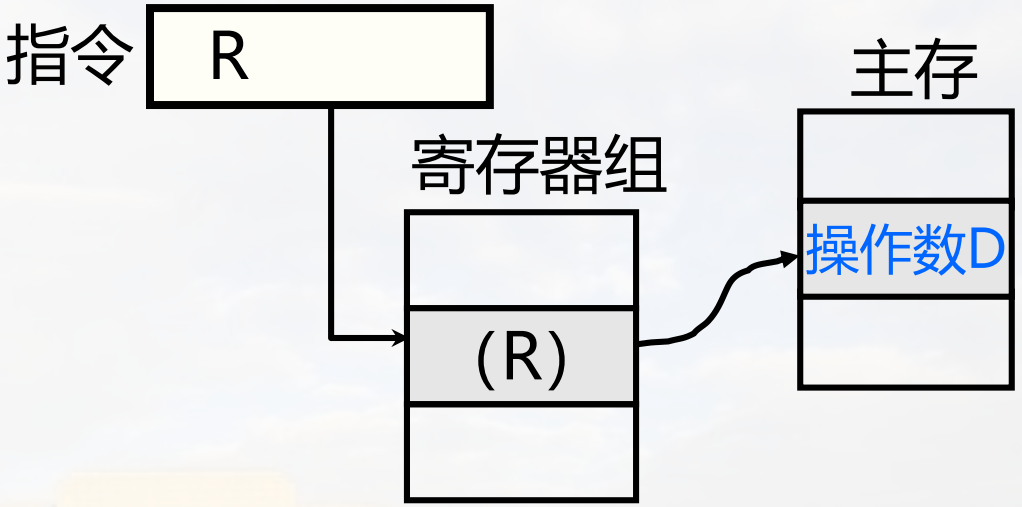
操作数D

5、寻址方式总结

存储器间接寻址

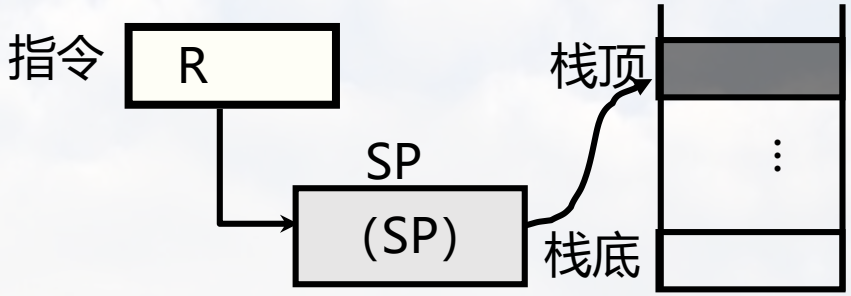


寄存器间接寻址

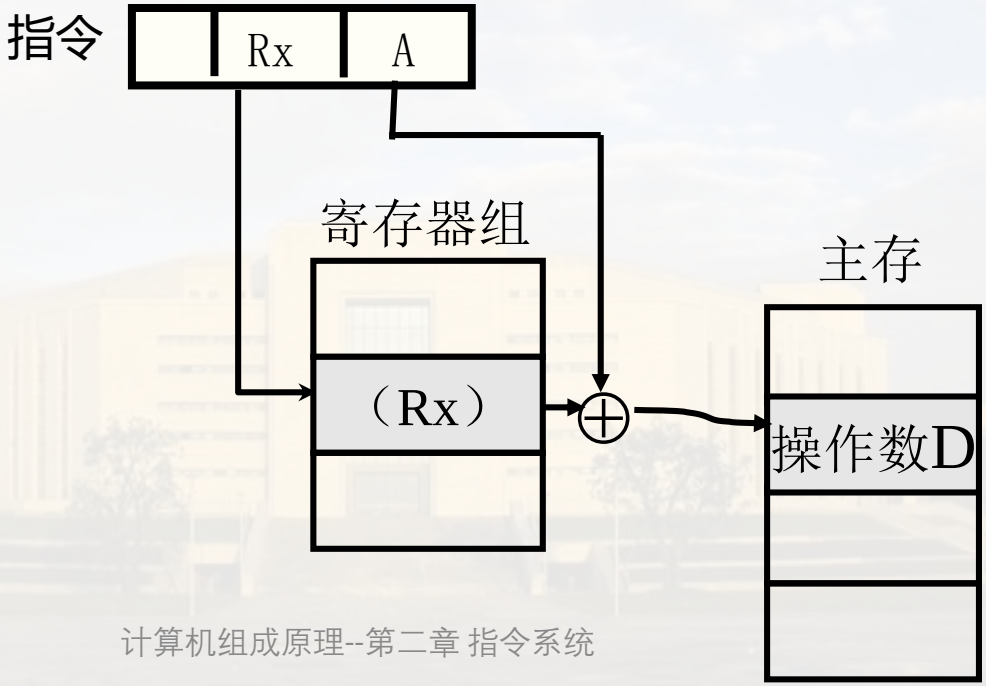


5、寻址方式总结

堆栈寻址



变址/基址寻址



6、对寻址方式的说明

（1）操作码隐含说明不同寻址方式

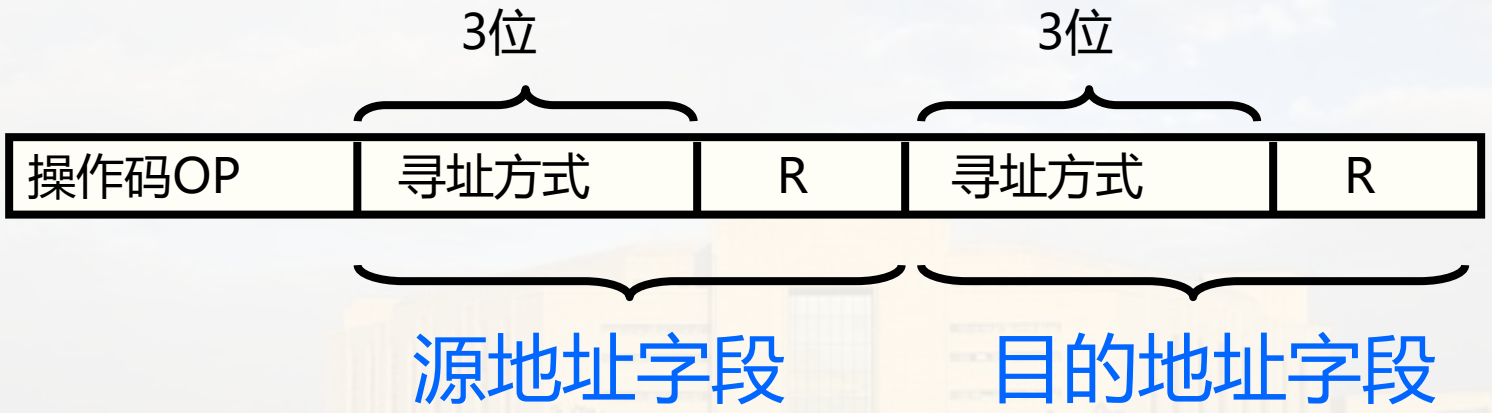
例.某机指令操作码最高两位

- 00：RR型指令，寄存器-寄存器寻址
- 01：RX型指令，寄存器-变址寻址
- 10：SI型指令，存储器-立即数寻址
- 11：SS型指令，存储器-存储器寻址

6、对寻址方式的说明

（2）指令中设置专门字段说明寻址方式

例.某机指令的每个地址字段中，各设置一个3位的寻址方式字段。



2.3 指令类型

- 01. 指令分类
- 02. 传送类指令
- 03. 输入/输出（I/O）指令

一、指令分类

RISC（Reduced Instruction Set Computer）
CISC（Complex Instruction Set Computer）

对指令的分类方法归纳起来大致有以下三类：

① 按指令格式分类

将指令格式分为双操作数指令、单操作数指令、程序转移指令等。

② 按操作数寻址方式分类

RR型（寄存器—寄存器型）　RX型（寄存器—变址存储器型）
RS型（寄存器—存储器型）　SI型（存储器—立即数型）
SS型（存储器—存储器型）

③ 按指令功能分类

现在的大部分微处理器，将指令分为：

传送指令、

输入/输出（I/O）指令、

算术运算指令、

逻辑运算指令、

程序控制类指令、

处理机控制类指令等。

1.传送指令

源地址 —数→ 目的地址

设置时需考虑：

（1）规定传送范围

例. DJS-100系列： R ⟷ M

ARM： R ⟷ M， R ⟷ R

IBM370： R ⟷ M， R ⟷ R， M ⟷ M

1.传送指令

（2）指明传送单位

一次传送的数据位数。

例. 用操作码说明(VAX-11)：

MOVB(8位)　　MOVW(16位)　　MOVL(32位)

用地址量说明(80X86)：

MOV AL, BL （8位）

MOV AX, BX （16位）

MOV EAX, EBX （32位）

1.传送指令

(3) 设置寻址方式

有的计算机为各种寻址方式分类编号，如本书模型机的0型、1型、2型、3型、4型、5型、6型等。

(4) 传送指令的特例

堆栈指令：包括出栈和入栈

```
PUSH AX    ；AX内容入栈
POP  AX    ；出栈后，源数据不再存在
```

交换指令：双向传送，即源操作数与目的操作数相互交换位置。

```
XCHL AX，  BX
```

2.输入/输出(I/O)指令

从广义的角度来看，I/O指令也是一种传送指令，只是传送范围的一方固定为输入/输出（I/O）设备，如键盘、鼠标、显示器、打印机等。主机对外围设备的访问一般就是对有关接口寄存器的访问。

2.输入/输出(I/O)指令

设置时需考虑：

（1）I/O指令的功能扩展

如何用通用I/O指令实现对各种具体设备的控制？

- I/O指令中留有扩展余地

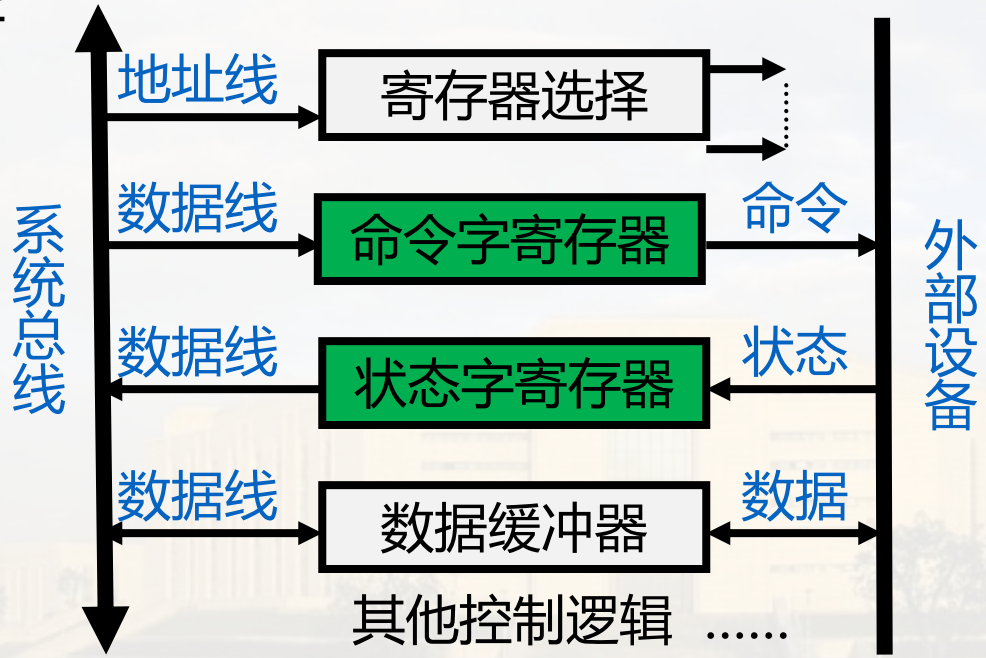
指令中某些字段编码事先不定义，需要时再约定其含义。早期采用，用于外设较少，且型号相对固定的场合，现在基本不使用。

- I/O接口中设置控制/状态寄存器

2.输入/输出(I/O)指令

主机用输出指令或传送指令将具体设备的控制命令按约定的代码格式送往接口中的控制寄存器，向外设发出命令。

外设的状态信息也以某种格式放在接口的状态寄存器中，主机用输入指令或传送指令从状态寄存器中取出有关信息进行查询、分析



如何设置控制/状态寄存器是接口设计的关键。

2.输入/输出(I/O)指令

(2) 主机对外设的寻址方式

寻找I/O接口中的寄存器的方式。

I/O端口

如何为I/O端口分配地址?

- **单独编址**

编址到寄存器：为每个寄存器(I/O端口)分配独立的端口地址；

I/O指令中给出端口地址。

I/O地址空间不占主存空间，可与主存空间重叠。

需设置标志区分访问对象，如 $M/\overline{IO}$ { =1 访问存储器；=0 访问I/O端口

2.输入/输出(I/O)指令

- **统一编址**

编址到寄存器

为每个寄存器(I/O端口)分配总线地址；

访问外设时，指令中给出总线地址。

I/O端口占据部分主存空间。

常将存储空间的低端分配给主存单元，高端分配给I/O端口，以示区分。

单独编址与统一编址的比较

	单独编址方式	统一编址方式
优点	I/O指令和传送指令容易区分，外设地址线少，译码简单，主存空间不会闲置	可用传送指令代替专用I/O指令，通过地址总线访问外设接口中的寄存器（如同通过地址总线访问主存单元一样）
缺点	控制类总线中增加了I/O Read 和 I/O Write 信号线	接口中的寄存器占用主存一部分地址，减少了主存的可用空间

(3) I/O指令的设置方法

通常有三类常见的I/O指令设置方法，一台计算机可以选取其中的一种或数种。

① 设置专用的I/O指令：IN，OUT

② 采用通用的数据传送指令实现I/O操作

③ 通过I/O处理器（或I/O处理机）控制I/O操作

三、输入/输出(I/O)指令

① 设置专用的I/O指令：IN，OUT

- 设置专用I/O指令——— 显式I/O指令

针对单独编址，用I/O指令访问I/O端口。
指令中说明输入/输出操作，并给出端口地址。

例. 80X86 I/O指令设置

输入：IN AL，n；　(n) → AL → (直接端口寻址)

端口地址

IN AL，DX；　(DX) → AL → (间接端口寻址)

间接端口地址

输出：OUT n，AL；　AL → n → (直接端口寻址)

OUT DX，AL；　AL → (DX) → (间接端口寻址)

三、输入/输出(I/O)指令

② 采用通用的数据传送指令实现I/O操作　　隐式I/O指令

针对统一编址，用传送指令访问I/O端口。
不设专用I/O指令。

例. 某机I/O接口中设置

控制/状态寄存器CSR,其总线地址为177550
数据缓冲寄存器DBR，其总线地址为177552

控制/状态字格式：

15	14		12		7	6		2	1	0
出错	故障		忙		完成	允许中断		维护	校验	启动

主机调用输入机：

启动：　主机 —控制字(启动位为1)/传送指令→ 177550

查询：　主机 ←状态字/传送指令— 177550

取数：　主机 ←数据/传送指令— 177552

③ 通过I/O处理器（或I/O处理机）控制I/O操作

主机在与外设交互信息时，为了减轻CPU在I/O方面的工作负担，现代计算机中常设置一种专门用于管理I/O操作的协处理器，即IOP（I/O Processor，输入/输出处理机），在大规模计算机系统中甚至设置专门的外围处理机。相应地，设置的专用I/O指令可以分为两级：

- 一级是CPU管理IOP的I/O指令，负责启动及停止IOP等操作，这类指令的操作类型较少，功能比较简单；
- 另一级是IOP执行的指令（如通道程序等），负责控制外围设备具体的I/O操作，这一级的指令功能相对丰富一些。

3.算术逻辑运算指令

（1）算术运算指令

设置时需考虑操作数类型、符号、进制等；

运算结束后设置相应状态标志。

（2）逻辑运算指令

实现对代码位的设置、测试、清除、修改等。

或　　与　　异或

4. 程序控制指令

(1) 转移指令

无条件转移 ： 操作码 转移地址

条件转移 ： 操作码 转移地址 转移条件

循环 ： 转移条件为循环计数值

(2) 转子指令与返回指令

转子：操作码 子程序入口

返回：操作码

返回地址的存取：用堆栈存放返回地址。

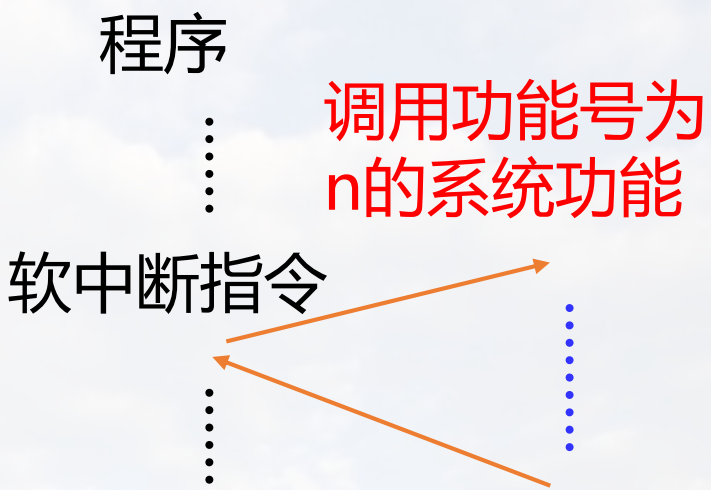
(3) 软中断指令

早期主要用于程序调试。

现在常用于系统功能调用。

以INT n; 的形式出现在程序中。

n 表示不同的功能调用



(4) 控制指令

CLC 清除进位标志

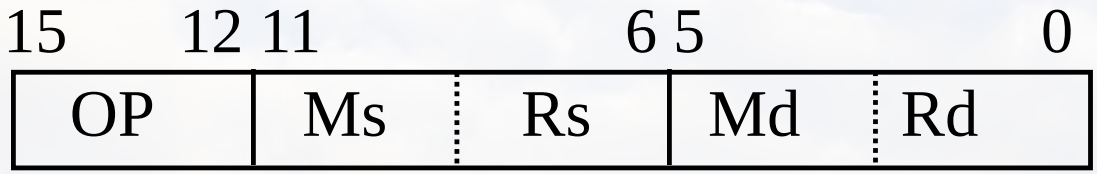
NOP 空操作指令

HLT 暂停指令

……

四、其他类型指令

例: 某计算机字长为16位, 主存存储单元的容量为128KB, 按字编址。采用单字长指令格式, 指令各字段定义如下:



转移指令采用相对寻址, 相对偏移量用补码表示。寻址方式定义如下:

Ms/Md	寻址方式	助记符	含义
000B	寄存器直接	Rn	操作数=(Rn)
001B	寄存器间接	(Rn)	操作数=((Rn))
010B	寄存器间接、自增	(Rn)+	操作数=((Rn)), (Rn)+1→ Rn
011B	相对	D(Rn)	转移目标地址=(PC)+(Rn)

四、其他类型指令

请回答以下问：

(1) 该指令系统最多可有多少条指令，该计算机最多有多少个通用寄存器，存储器地址寄存器(MAR)和存储器数据寄存器至少各需要多少位(MDR)？

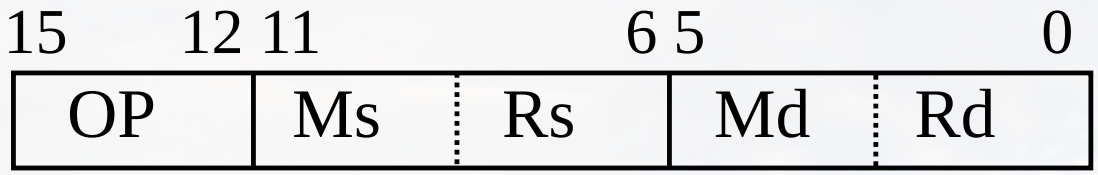
(2) 转移的目标地址范围是多少？

(3) 若0010B表示加法操作(助记符add)，寄存器R4和R5的编号分别为100B和101B，R4的内容为1234H，R5的内容为5678H，地址1234H中的内容为5678H，地址5678H中的内容为1234H，则汇编指令“add (R4), (R5)+”(逗号前为源操作数，逗号后为目标操作数)对应的机器码是什么(用十六进制表示)？该指令执行后，哪些寄存器和存储单元的内容会改变？改变后的内容是什么？

四、其他类型指令

解答:

(1) 该指令系统最多可有多少条指令, 该计算机最多有多少个通用寄存器, 存储器地址寄存器(MAR)和存储器数据寄存器至少各需要多少位(MDR)?



因为操作码为4位, 所以最多有16条指令。因为用来表示寄存器的位数是3位, 所以,最多有8个通用寄存器。

存储空间按字编址, 因此存储单元数量为128KB/2=64K字。需要16位地址, 所以地址寄存器MAR为16位。因为字长16位, 所以数据寄存器MDR为16位。

(2) 转移的目标地址范围是多少?

因为地址位数和字长都是16位, 所以PC和通用寄存器的位数均为16位, 所以转移目标地址位数为16位, 因此转移目标地址范围0000H~FFFF。

四、其他类型指令

(3) 若0010B表示加法操作, 寄存器R4和R5的编号分别为100B和101B, R4的内容为1234H, R5的内容为5678H, 地址1234H中的内容为5678H, 地址5678H中的内容为1234H, 则汇编指令“add (R4), (R5)+”(逗号前为源操作数, 逗号后为目标操作数)对应的机器码是什么(用十六进制表示)? 该指令执行后, 哪些寄存器和存储单元的内容会改变? 改变后的内容是什么?

指令“add (R4), (R5)+”操作码为0010, 源操作数采用间接寻址, 编码001, 目标操作数采用间接自增寻址, 编码010, R4的编码为100, R5的编码为101, 所以对应的机器码为

0010	001	100	010	101	(2315H)
↓	↓	↓	↓	↓	
操作码	源间接寻址	R4编码	目标间接自增寻址	R5编码	

指令执行后, 结果存入5678H单元,因此由原来的1234H变为68ACH, R5的内容加1后变为5679H。

小结

1、I/O指令的功能扩展(目的、方法)，
外设编址方式和指令设置方式。

2、基本概念：扩展操作码(扩展方法)、
地址结构(简化方法)、隐地址、显地址；

3、基本寻址方式(立即、直接、间址、变址)
的含义与应用场合。

二、部件与与数据通路

重点掌握： 模型机的寻址方式，如下题：

某存储器部分单元的地址码和内容对应关系如下：

地址码	存储内容
2000H	0A57H
2001H	0C03FH
2002H	1200H
2003H	0F49H
2004H	0D04H

（1） 若采用寄存器间址方式读取操作数，指定寄存器R0的内容为2001H，则操作数是（　　）。

（2） 若采用自减型寄存器间接寻址-(R1)读取操作数，R1内容为2001H，则操作数是（　　）。

（3）若采用变址寻址方式X(R2)读取操作数，指令给出形式地址d=3H，变址寄存器R2内容为2001H，则操作数是(　　）。

作业：

书232：

第2、3、4大题

谢谢观看

计算机基础

2023/10/9

信息与软件工程学院
School of Information and Software Engineering