东南大学计算机学院

计算机系统组成

主讲教师： 徐造林

第6章 指令系统

- 计算机指令
  - 指令：就是要计算机执行某种操作的命令；有微指令、机器指令和宏指令之分。
- 计算机能执行的机器指令全体称为该机的指令系统
  - 指令系统是软件编程的出发点和硬件设计的依据，它衡量机器硬件的功能，反映硬件对软件支持的程度。

6.1 指令系统概述

6.1.1 指令系统简介

1. 指令系统组成

- 指令系统主要是为计算机应用、编译程序和操作系统提供支持。
- 指令系统中指令的设计，需要从性能提高和带来的成本增加两个方面考虑。

- 两种类型的指令

（1）非特权指令：这类指令主要供用户使用，又可分为功能性指令和非功能性指令两种。

（2）特权指令：主要供系统程序员使用，一般不允许用户使用。其中包括I/O指令、停机等待指令、存储管理及保护指令、控制系统状态指令、诊断指令等。

2. 对指令系统性能的要求

- 完备性：指令系统应功能齐全，给用户带来方便。
- 规整性：指令系统的正交性、均匀性、对称性。
- 兼容性：不同机种之间具有相同的基本结构和共同的基本指令集，目的是给软件资源的重复利用带来方便。
- 可扩充性：指令系统中要保留一定的指令字空间，以便在需要时进行指令系统的功能扩充。

6.1.2 指令的格式

- 指令一般的格式如下：

操作码OP	地址码 A

1. 操作码

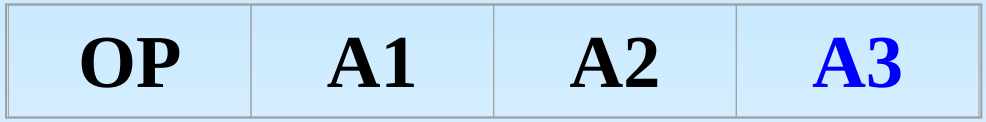
- 操作码指出指令应该执行什么性质的操作和具有何种功能；n位操作码字段的指令系统最多能够表示2^n条指令。

2. 地址码

- 地址码指出指令中操作数所在的存储器地址或寄存器地址。

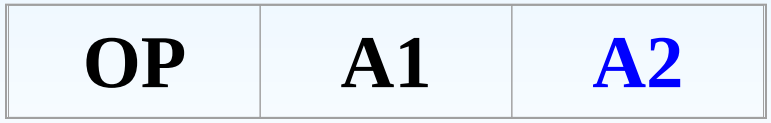
- 按指令包含的地址的个数可分：

1、三地址指令



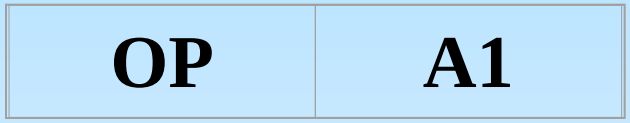
执行（A1）OP（A2）→ A3

2、二地址指令



执行（A1）OP（A2）→ A2

3. 单地址指令



执行（A1）OP （AC）→ AC

4、零地址指令

- 有两种可能：一是无需任何操作数，如空操作指令、停机指令等。二是所需的操作数地址是默认的。

- 指令字长度
  - 计算机中CPU能直接处理的二进制的位数称为机器字长；
  - 指令字长度：一个指令字包含的所有二进制代码的位数。有等长指令字结构和变长指令字结构。

- 指令字长度选取原则

1. 指令长度应为存储器基本字长的整数倍
2. 指令字长应尽量短

  - 指令短，可减少所需存储量和加快运行速度，但不能为了使指令短而影响指令系统的完备性和规整性。

- 指令助记符

表6.1 常用指令助记符

指令类型	指令助记符	二进制操作码
加法	ADD	001
减法	SUB	010
传送	MOV	011
跳转	JMP	100
存储	ST	101
读数	LD	110

6.1.3 指令系统设计概论

1. 指令系统设计的基本思路

- 任务是确定所有机器指令的格式、类型、操作以及对操作数的访问方式。出发点是提高指令系统的性能/价格比。
- 基本设计思想：

（1）确定计算机系统中的基本操作（包括操作系统和高级语言的）是由硬件实现还是由软件实现；

（2）按照尽量缩短平均码长、方便译码与执行的原则，设计指令字格式。

- 功能设计和指令格式设计

  - 基本功能设计：确定指令系统包含哪些基本操作；
  - 优化功能设计：从对目标程序、操作系统、高级语言的支持角度，确定哪些常用的、相对复杂的操作（指令串）可作为指令系统包含的操作（指令）；
  - 指令格式设计：设计出平均码长较短、便于译码和执行的指令字；
  - 规整和优化设计：形成长度规整的、信息冗余较小的指令字。

2. 衡量指令系统性能的指标

- 指令所占存储空间是否尽可能小；表现在指令中代码密度是否高、信息冗余量是否少；
- 指令代码对应用需求的效率是否高，表现在非特权指令中功能性指令所占比例是否高、指令中操作数的访问范围满足应用需求的概率是否大、对操作系统和编译程序的支持程度是否高；
- 指令的译码速度、执行速度是否快。

6. 2 操作数类型及存储方式

6.2.1 操作数类型

- 操作数的4种类型
  - 数值型数据；字符；地址；逻辑数据。
- 操作数类型在指令中的表示
  - 指令中增加一些二进制位标识；可选择存储时标识或处理时标识。

- 标识方法

OP	DF1	A1	DF2	A2

(a) 每个操作数标识

OP	DF	A1	A2

(b) 所有操作数共一个标识

图6.1 二地址指令数据类型表示方法

- 通常将操作码与数据类型标识合并成新的操作码。

6.2.2 操作数存储方式

- 操作数可以存储在指令、寄存器、堆栈和存储器中；
- 数据存储方式有大端（Big-Endian）和小端（Little-Endian）两种。

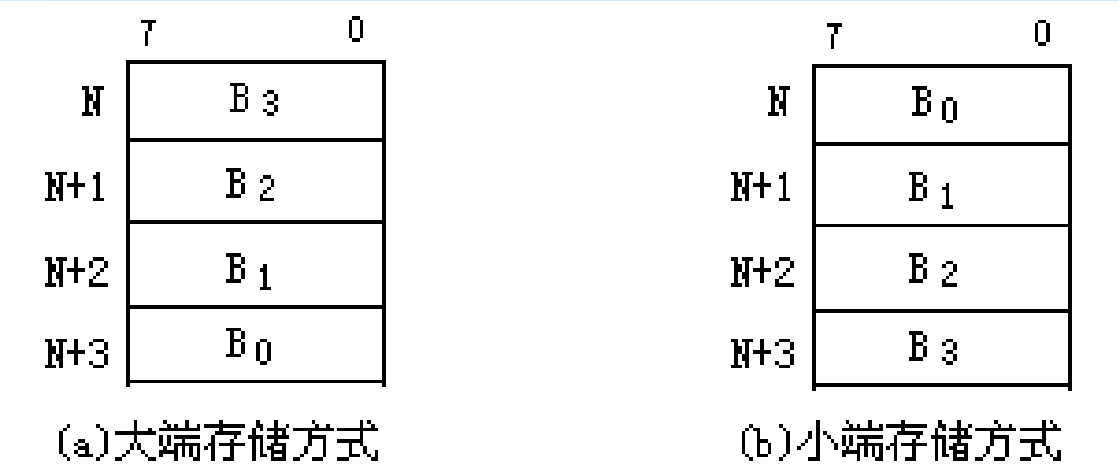


图6.2 数据在存储器中的两种存放方式

- 数据存储采用小端存储方式的处理机有Intel 80x86/Pentium，DEC VAX，DEC Alpha等。

6.2.3 数据对齐方式

- 操作数应存储在存储器空间阵列的同一行；

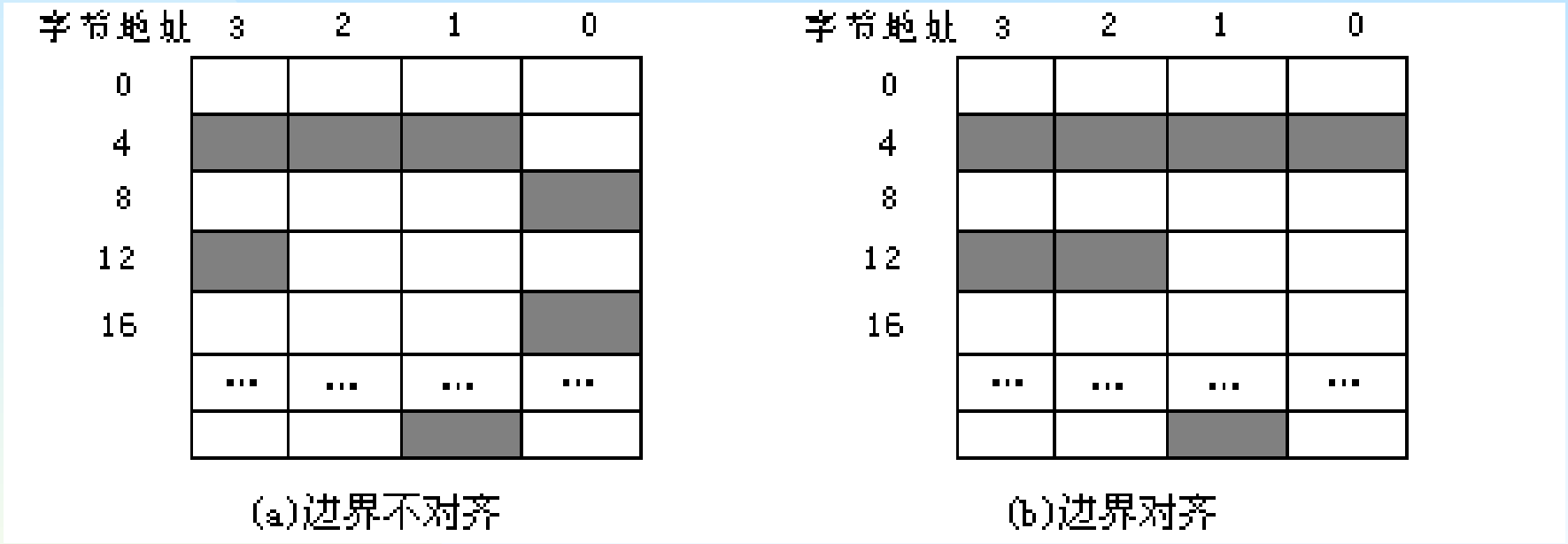


图6.3 数据的边界对齐

- 信息按整数边界存储原则：数据长度为2^n个字节，则该数据在存储器中最小存储地址的最低n位应为0。

6.2.4 堆栈存取方式

- 堆栈是一种按特定顺序访问的存储区；其特点是后进先出(LIFO)或先进后出(FILO)。
- 堆栈存取方式
  - 堆栈最底部存放数据的位置是固定不变的，该位置称为栈底；
  - 堆栈中存放的最上面数据的位置是不停变化的，该位置称为栈顶；
  - 存取数据只能在栈顶进行，不可中间插入或者从中间将数据取出。

- 堆栈操作只有入栈（push）和出栈（pop）两种。
- 堆栈和其它形式的存储部件之间的差别：

（1）堆栈在数据存取时不需要地址，而其它存储部件在数据存取时需要地址；

（2）堆栈只能按先进后出或后进先出方式存取数据，而其它存储部件可以根据地址随机存取数据；

（3）堆栈不可以在同一位置连续写入或取出数据，而其它存储部件可以。

堆栈的实现方法

1. 用移位寄存器实现堆栈

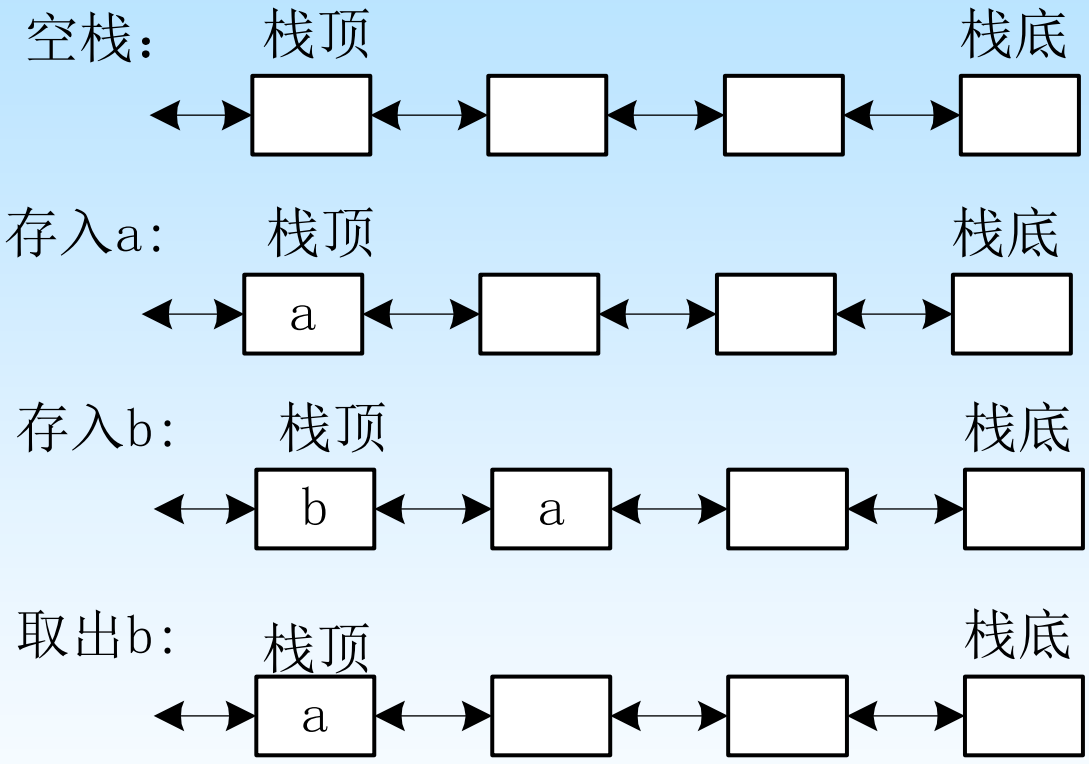


图6.4 栈顶固定方式堆栈及其存取

2. 内存中开辟堆栈区

- 选取固定的存储器单元为堆栈区，存储器堆栈的具体位置由程序员指定，空间大小由程序员分配；
- 建栈时设置堆栈指针SP，指示栈顶位置。

1）自底向上生成堆栈（满递减和空递减）：

- 建栈时堆栈指针SP指向栈底下面一个单元（栈底是堆栈中地址最大的单元）；
- 入栈操作（PUSH）步骤： i）SP-1→SP
  ii）存入数据 →（SP）
- 出栈操作（POP）步骤： i）（SP）内容读出
  ii）SP+1→SP

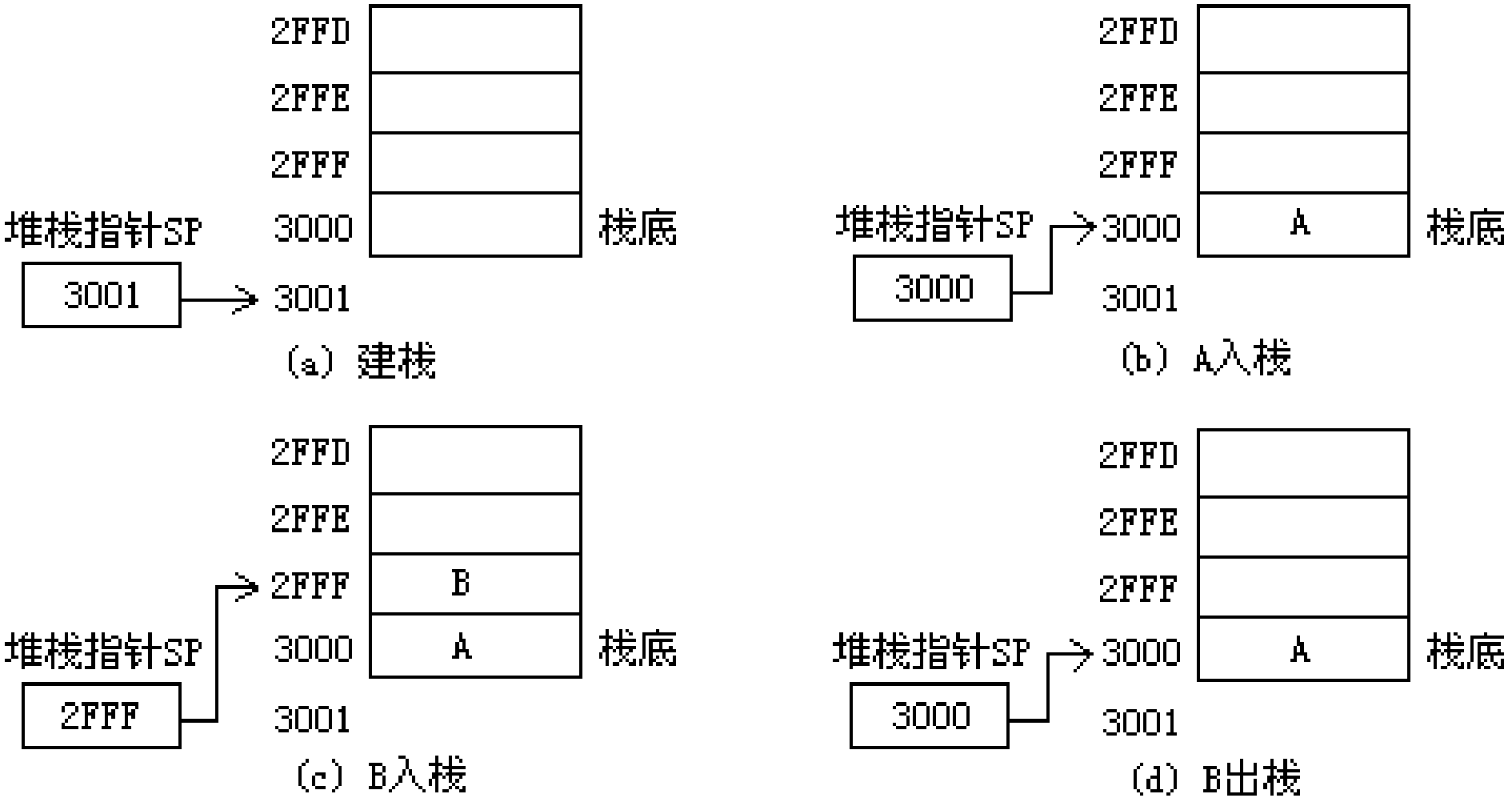


图6.5 自底向上堆栈操作示例

2）自顶向下生成堆栈（满递增和空递增）：

- 堆栈建栈时堆栈指针SP指向栈底上面一个单元（栈底是堆栈中地址最小的单元），

- 入栈操作（PUSH）步骤：　i) SP+1→SP
  ii) 存入数据→（SP）
- 出栈操作（POP）步骤：　i)（SP）内容读出
  ii) SP-1→SP

- 两者指针变化方向不同。

- 堆栈存取方式在指令中的应用

- 堆栈操作对临时保存和恢复某些数据极为简便。堆栈存取方式对应的操作有建栈、入栈和出栈三种，可对应指令系统中的三条指令。

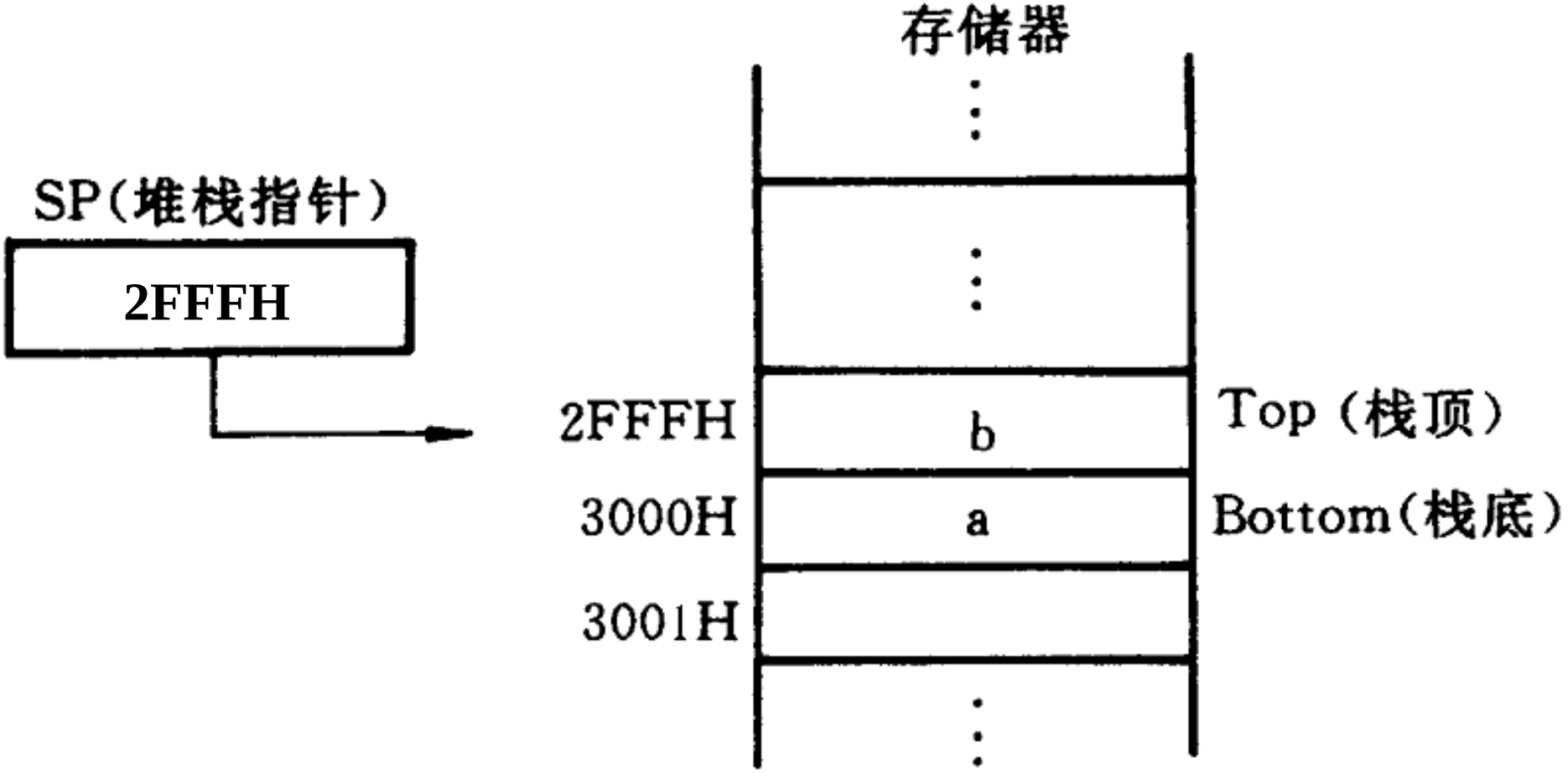
例2　某存储器堆栈，栈底地址 **Bottom=3000H**，栈中已压入两个数据**a**和**b**，**SP**为堆栈指针。

（1）画出此时堆栈示意图。

（2）现将数据 **c**，**d** 和 **e** 按顺序压入堆栈，且用累加器**AC** 进行数据交换，写出数据入栈步骤，画出数据入栈后的堆栈情况。

（3）写出数据 **e** 出栈的操作步骤。

解：（1）堆栈情况如下图所示



（2）数据入栈操作

AC ← c；SP ← SP-1；(SP) ← AC

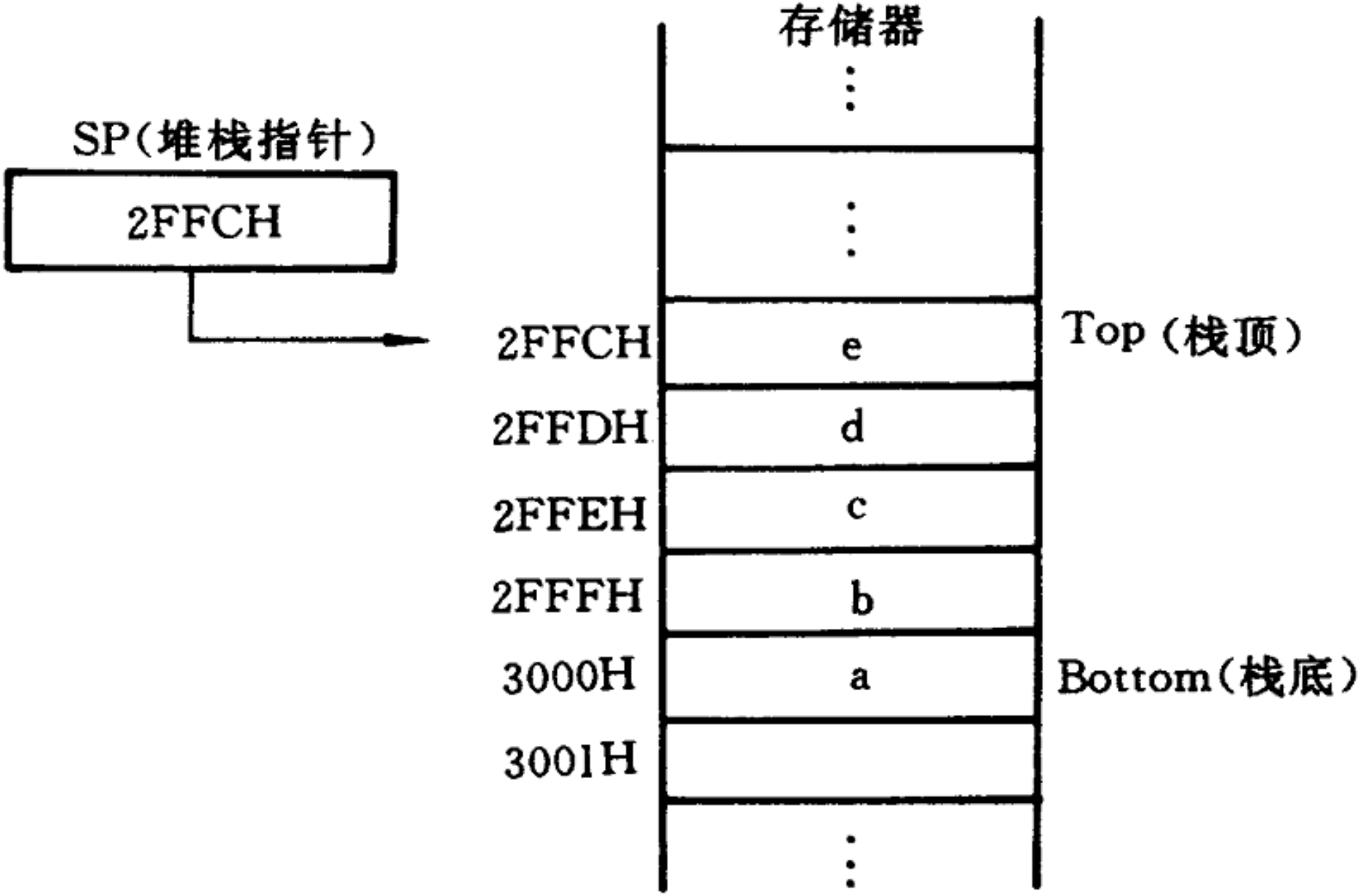
AC ← d；SP ← SP-1；(SP) ← AC

AC ← e；SP ← SP-1；(SP) ← AC

（3）数据 e 的出栈操作

AC ← (SP)

SP ← SP+1


存储器
SP(堆栈指针)
2FFCH
2FFCH
e
Top(栈顶)
2FFDH
d
2FFEH
c
2FFFH
b
3000H
a
Bottom(栈底)
3001H

6.3 指令系统功能设计

- 指令集中包含的所有操作功能的集合称为指令系统的功能集

- 操作码是指令系统功能集中各种功能（操作）的编码。

6.3.1 功能类型分类

1. 数据传送类指令

① 传送（MOV）指令：实现寄存器与寄存器间、常数操作数与寄存器间的数据传送；

② 取数（LOAD或LD）指令：实现存储器到寄存器的数据传送；

③ 存数（STORE或ST）指令：实现常数操作数或寄存器到存储器的数据传送；

④ 数据交换（XCHG）指令：实现两个数据之间的交换，可以看成是双向传送；

⑤ 入栈（PUSH）指令：实现寄存器或存储器到堆栈的数据传送；

⑥ 出栈（POP）指令：实现堆栈到寄存器或存储器的数据传送。

2. 算术运算类指令

- **指令主要包括加（ADD）、减（SUB）、乘（MUL）、除（DIV）指令，求反（NOT）、求补（NEG）指令，算术移位（SLA、SRA）、算术比较（COMP）指令等。**

- 为区分数据类型和运算规则，形成扩展运算，如十进制运算、带进位运算、双精度运算等指令。

3. 逻辑运算类指令

- 指令主要包括逻辑与（AND）、逻辑或（OR）、逻辑异或（XOR）、逻辑非（NOT）、逻辑移位（SLL、SRL）、循环移位（ROL、ROR、ROLC、RORC）指令。

4. 程序控制类指令

- 指令主要包括无条件转移（JMP）、条件转移（Jcc）、跳步（SKIP）、转子（CALL）、返主（RET）、循环（LOOP）指令等。 。
- 当前指令地址码给出直接地址或相对于当前指令位置的偏移地址，

5. 输入输出类指令

- 实现CPU与外部设备间的数据交换、传送控制命令及取得设备状态等功能。
- 这类指令有输入（IN）和输出（OUT）两种指令。

6. 字符串类指令

- 指令包含字符串转换、字符串传送、字符串比较、字符串查找、字符串抽取、字符串替换等指令。

7. 系统控制类指令

- 能够改变系统的工作状态、实现操作系统所需要的特殊功能。大多数为特权指令。
- 包括停机（HALT）、开中断（STI）、关中断（CLI）、自陷（Trap，即软中断INTn）、系统管理、存储管理等指令。

8. 其它指令

- 特定功能的专用指令；包含状态寄存器置位（STC、CLD等）、暂停（WAIT）、测试（TEST）、空操作（NOP）、中断返回（IRET）等指令。 。

6.3.2 指令系统功能集设计

- 指令系统是计算机软、硬件主要交界面，直接反映了计算机的性能/价格。

▲ 复杂指令集计算机CISC(Complex Instruction Set Computer)的提出。

- 计算机的硬件成本不断下降，软件成本不断上升；
- 强化指令功能，实现软件功能向硬件功能转移；
- 指令系统增加了越来越多功能强大的复杂命令，以便使机器指令的功能接近高级语言语句的功能。

▲ 指令系统越来越复杂的出发点：

① 使目标程序得到优化：把原来要用一段程序才能完成的功能，只用一条指令来实现；

② 给高级语言提供更好的支持：改进指令系统，设置一些在语义上接近高级语言语句的指令，就可以减轻编译的负担，提高编译效率；

③ 提供对操作系统的支持：操作系统日益发展，其功能也日趋复杂，要求指令系统提供越来越复杂的功能。

▲ 复杂的指令系统带来的问题

- 计算机的结构也越来越复杂，不仅增加了计算机的研制周期和成本，而且难以保证其正确性，有时还可能降低系统的性能；
- 庞大的指令系统中，只有算术逻辑运算、数据传送、转移、子程序调用等几十条基本指令才是常使用的，在程序中出现的概率占到80%以上；
- 需要大量硬件支持的复杂指令的利用率却很低，造成了硬件资源的大量浪费。

▲ 精简指令集计算机RISC（Reduced Instruction Set Computer）的提出。

- 各种高级语言的语义之间有很大差别；不可能设计出一种能对所有高级语言都能提供很好支持的指令系统。
- 指令系统越复杂，包含的指令越多，编译时生成目标程序的方法也越多，对最终优化编译造成困难。

▲ 精简指令系统计算机特点

- 通过简化指令使计算机的结构更加简单合理，从而提高机器的性能。

① 指令数目较少，一般都选用使用频度最高的一些简单指令；

② 指令长度固定，指令格式种类少，寻址方式种类少；

③ 大多数指令可在一个机器周期内完成；

④ 通用寄存器数量多，只有存数/取数指令访问存储器，而其余指令均在寄存器之间进行操作。

▲ 采用RISC技术

- 指令系统可以采用速度较快的硬连线逻辑来实现，且更适合于采用指令流水技术，可使指令的执行速度进一步提高；
- 指令数量少，固然使编译工作量加大，但由于指令系统中的指令都是精选的，编译时间少，反过来对编译程序的优化又是有利的；
- 结构更适合VLSI、并行处理，更能够提高计算机的性能；
- CISC和RISC技术都在发展，两者都各有自己的优点和缺点。

▲ CISC与RISC之争论

- 70年代中期，IBM公司、斯坦福大学、加州大学伯克利分校等机构分别先后开始对CISC技术进行研究，其成果分别用于IBM、SUN、MIPS等公司的产品中；
- 八十年代中期，RISC技术蓬勃发展，先后出现了PowerPC、MIPSR4400、MC88000、Super Spare、Intel0860等高性能RISC芯片以及相应的计算机；
- RISC也随着速度、芯片密度的不断提高，使RISC系统日趋复杂；
- CISC机采用了部分RISC先进技术(强调指令流水线、分级Cache 和多设通用寄存器)，其性能更加提高。

6.4 寻址方式

- 指令如何指定操作数或操作数地址称为寻址方式。
- 操作数的寻址方式主要解决的是操作数存放在指令、寄存器和存储器中的寻址问题。
- 确定指令系统的寻址方式时，须考虑以下几点：
  - 希望指令内所含地址尽可能短；
  - 希望能访问尽可能大的存储空间；
  - 寻址方法尽可能简单；

- 在不改变指令的情况下，仅改变地址的实际值，从而能方便地访问数组、串、表格等较复杂数据。

- 设指令格式：

OP	寻址特征MOD	形式地址D

6.4.1 常用的寻址方式

1. 立即寻址

- **操作数在指令中；Data=A。**

指令

OP	F1	A

存储器地址	存储器内容
n	操作码
n+1	8位立即数
n+2	下条指令

(a) 8位立即数

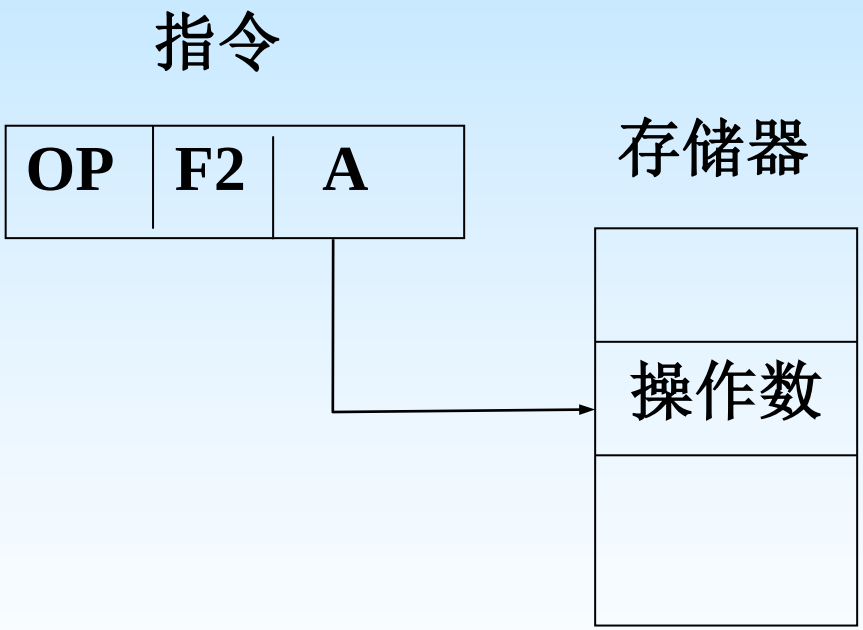
存储器地址	存储器内容
n	操作码
n+1	立即数低8位
n+2	立即数高8位
n+3	下条指令

(b) 16位立即数

图6.6 按字节编址机器中的立即寻址指令

2. 直接寻址

- 指令直接给出操作数（有效）地址；即EA=A。



存储器地址	存储器内容
n	操作码
n+1	操作数地址低8位
n+2	操作数地址高8位
n+3	下条指令

图6.7 直接寻址

3. 存储器间接寻址

- 操作数地址在内存中；即EA =（A）。

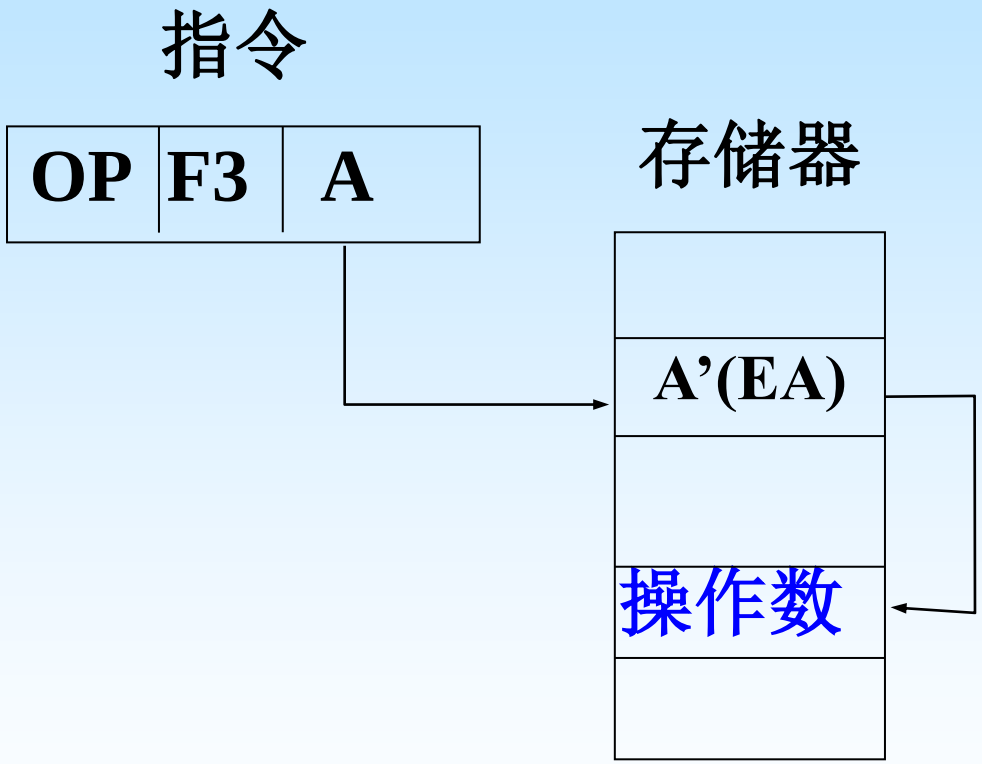


图6.8 间接寻址

4. 寄存器（直接）寻址

- 指令地址码字段给出存放操作数的寄存器编号；

  即data=（R）。

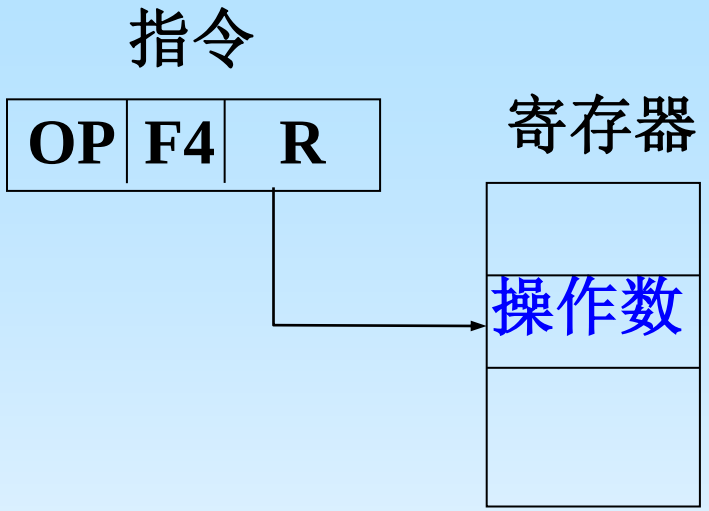


图6.9 寄存器寻址

- 寄存器寻址有以下优点：

① CPU寄存器数量远小于内存单元，所以寄存器号比内存地址短，因而寄存器寻址方式指令短；

② 不用访存，指令执行速度快。

5. 寄存器间接寻址

- 操作数地址在指令指定的CPU某个寄存器中；
  EA=（R）；如8086指令　MOV AL，B[SI]

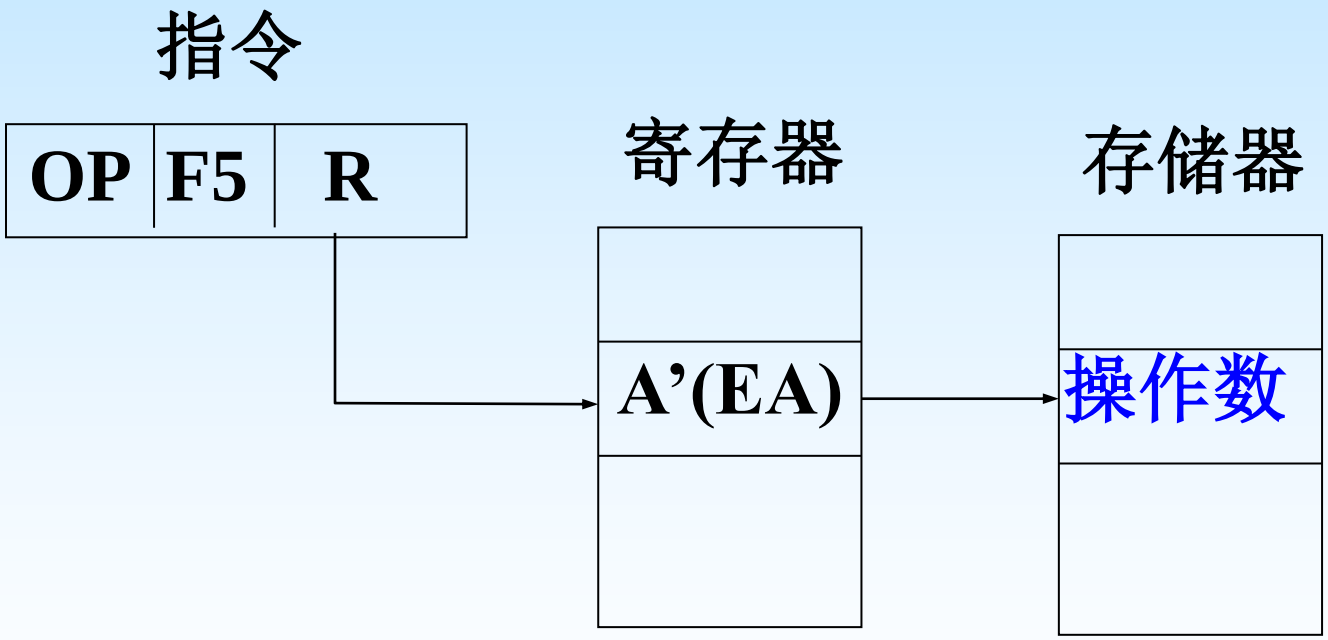


图6.10 寄存器间接寻址

6. 相对寻址

- 操作数地址为程序计数器PC中的内容与位移量A之和，即EA=（PC）+ A。

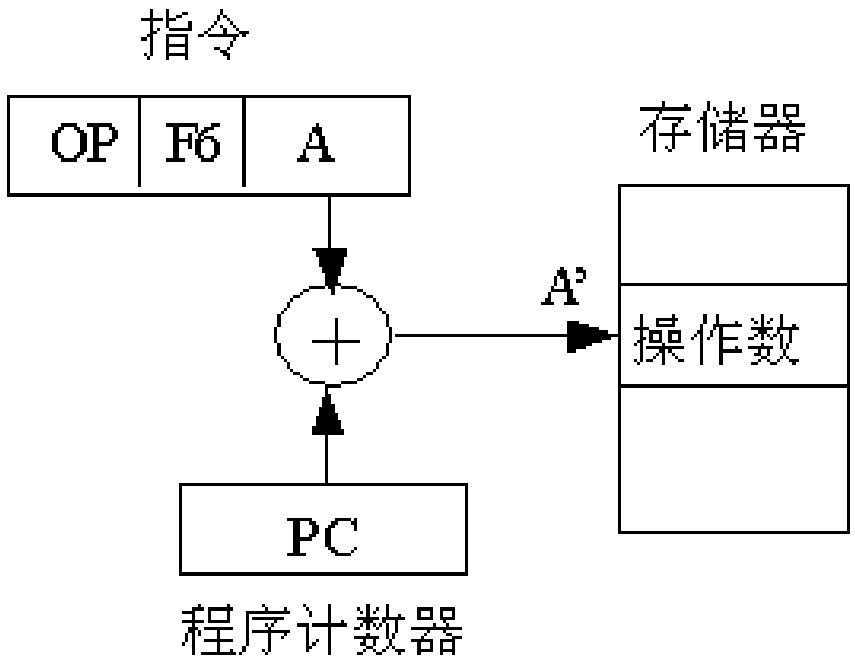
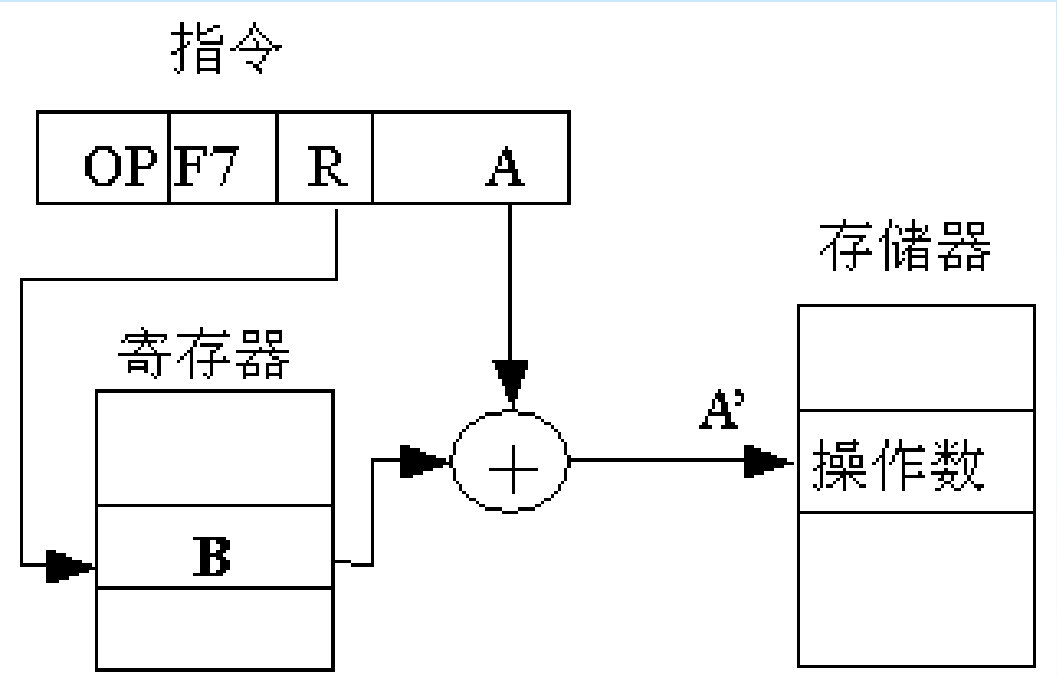


图6.11 相对寻址

7. 基址寻址

- 把由指令中给出的地址（位移量）与CPU 中的某个基址寄存器相加而得到实际的操作数地址。



$EA=(R)_{基址}+A$

图6.12　基址寻址

8. 寄存器变址寻址

- 操作数地址为变址寄存器中的内容与位移量之和；即 EA=（R）$_{变址}$+A。

OP	F8	R	A

8086指令

MOV AL，[SI+1000H]

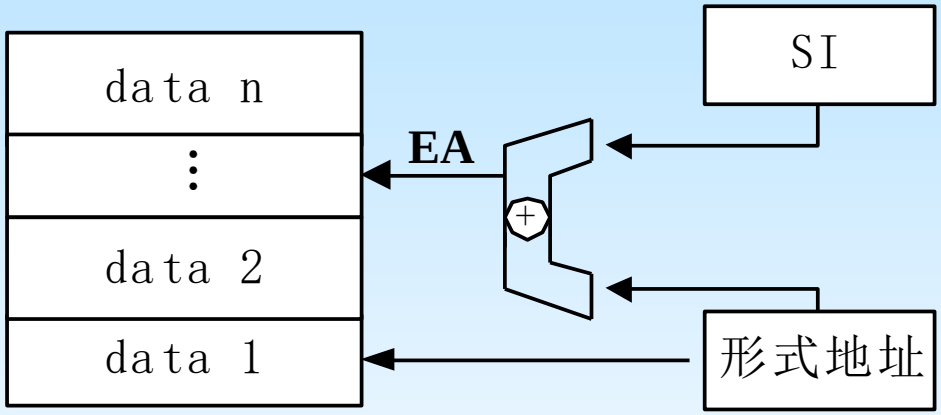


图6.13 变址寻址选择数组数据

- 变址寻址主要解决程序内部的循环问题；基址寻址则要求基址寄存器的内容能提供整个主存范围的寻址能力；在多道程序运行环境下，实现程序的再定位。

9．隐含寻址方式

- 指令没有明显地给出操作数地址，而在操作码中隐含着操作数地址。如操作数隐含在累加器，堆栈内。

10．其它寻址方式

- 有的计算机指令系统中还有更复杂的寻址方式，如基址变址寻址、位寻址、块寻址、串寻址等等。
- 在使用机器时，不仅要了解该机总体上有哪些寻址方式，还应了解各指令具体有哪些寻址方式。

例. 某计算机有变址寻址、间接寻址和相对寻址等寻址方式，设当前指令的地址码部分为001AH，正在执行的指令所在的地址为1F05H，变址寄存器中的内容为23A0H。请填充：

（1）当执行取数指令时，如为变址寻址方式，则取出的数为_____。

（2）如为间接寻址方式，则取出的数为_____。

（3）当执行转移指令时，转移地址为_____。

地址	内容
001AH	23A0H
1F05H	241AH
IFIFH	2500H
23A0H	2600H
23BAH	1748H

- 已知存储器的部分地址及相应内容：

地址	内容
001AH	23A0H
1F05H	241AH
IFIFH	2500H
23A0H	2600H
23BAH	1748H

解：（1）变址寻址

- 当前指令的地址码为001AH，变址寄存器中的内容为23A0H；
- 操作数地址为：（23A0H + 001AH）=23BAH；
- 则取出的数为1748H。

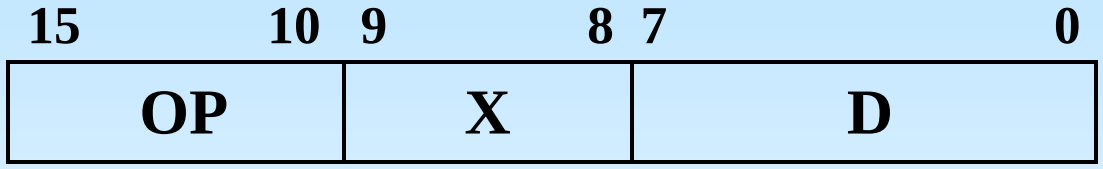
（2）间接寻址：地址码为操作数地址；

- 则取出的数为2600H 。

（3）正在执行的指令所在的地址为1F05H；

- 当执行转移指令时，转移地址为：（IF05H+2+001AH）=1F21H。

例. 某计算机指令格式如下：



图中X为寻址特征位，且X=0时不变址； X=1时用变址寄存器X_1进行变址；X=2时用变址寄存器X_2进行变址；X=3时相对寻址。设（PC）=1234H，（X_1）=0037H，（X_2）=1122H，请确定下列指令的有效地址。

（1）4420H（2）2244H（3）1322H（4）3521H（5）6723H

1122H+0044H

6.4.2 寻址方式设计

- 寻址方式设计的主要内容：

- 指令系统的寻址方式集，是指令系统支持的寻址方式的集合；
- 指令系统中每条指令的寻址方式子集，即每条指令支持的寻址方式的集合；
- 寻址方式集中每种寻址方式的性能参数。

1. 指令系统寻址方式集设计

- 指令系统寻址方式集分为常用的和必须的两种类型。

（1）寻址方式集的常用寻址方式设计

- 根据指令系统风格（**CISC和RISC**）和各种寻址方式的使用频率，选择使用频率较高的寻址方式作为指令系统寻址方式集的常用寻址方式。

▲ 频带分析法：

- 对大量应用程序中指令的寻址方式进行分析，特别是对复杂数据结构寻址方式的分解，将所有的寻址方式进行分类；
- 对分解后的各种寻址方式进行频率统计，包括每种寻址方式中性能参数（如立即数、偏移量范围等）频率分布的统计；
- 根据指令系统的风格及计算机的性能要求，将使用频率较高的寻址方式作为指令系统寻址方式集的常用寻址方式。

（2）寻址方式集的必须寻址方式设计

- 解决常用寻址方式设计中对寄存器、存储器寻址的遗漏问题。
- 选择使用频率最高的对寄存器或存储器寻址（常用寻址方式中所缺的）的一种寻址方式作为必须寻址方式中的寻址方式。

2. 指令寻址方式子集设计

- 每条指令的寻址方式子集是那些该指令常用的寻址方式，而不是全部寻址方式；
- 设计目标是为指令系统中每条指令确定它所支持的寻址方式。

▲ 指令寻址方式子集与指令系统寻址方式集的设计区别

（1）寻址方式子集设计不存在必须的寻址方式问题，

（2）指令系统寻址方式集的设计是针对所有指令进行的，而指令寻址方式子集设计是针对某条具体的指令进行的。

3. 寻址方式性能参数设计

- 性能参数是指该寻址方式满足应用需求所需要的操作数或操作数地址码位数。

▲ 寄存器号编码长度设计

- 寄存器号编码长度为$\log_2 N$位，其中N为指令系统可用寄存器个数。

- 寻址方式中指定专用的寄存器，寻址方式对应地址码中应省略该寄存器号编码。

▲ 存储器地址编码长度设计

- 存储器地址编码长度为$\log_2 M$位，其中M为指令系统可用存储器空间。
- 指令中存储器地址用相对于某地址的形式地址表示时，形式地址编码长度不受存储器地址长度限制。

▲ 寻址方式中立即数长度设计

- 根据应用需求对寻址方式中立即数值域的要求，确定立即数值范围及位数范围；
- 采用频带分析法，在频率分布中确定对应的立即数长度（值域）范围。

6.5 指令字格式设计

- 指令格式设计所要解决的问题：

（1）指令系统由哪些指令构成？

（2）每条指令中操作码表示什么？为什么这样编码？

（3）当一条指令的操作数有多种表示形式时，如何区分不同的寻址方式？

（4）指令字格式如何确定？如何提高指令格式的性能/价格比？

6.5.1 指令系统指令数目设计

- 指令系统中的指令必须能够实现指令系统功能集中支持的所有操作；
- 指令系统功能集中包含多少个操作，指令集中就必须对应多少条指令；
- 指令系统中指令数为功能集中操作的数量加上同一操作对应多种数据类型所增加出来的指令的数量。

6.5.2 指令字操作码编码设计

- 计算机硬件识别和执行指令系统的指令是通过二进制编码实现的；
- 不同操作码对应不同的操作和对不同数据类型的相同操作。

1. 定长操作码编码

- 定长操作码编码中所有操作码的长度固定；
- 如果指令系统功能集共支持N种操作，应满足关系式：$2^{n-1} \leq N \leq 2^n$。所有操作码的平均码长为n位；
- 定长操作码主要用于具有CISC风格指令系统的计算机和指令字长较长的计算机。

2. 变长操作码编码

- 变长操作码编码中使用频率较高的操作码长度较短，使用频率较低的操作码长度较长；
- 变长操作码的平均码长为 $l_{avg} = \sum_{i=1}^{N} P_i l_i$
- 定长操作码平均码长为：$\log_2 N$ 。

6.5.3 指令字寻址方式表示设计

- 设计的目标是确定操作数地址的寻址方式用什么方法表示及如何表示。

1. 寻址方式的表示方法

1）将寻址方式标志编码于操作码中；

2）在地址码字段为每个操作数设置一个地址描述符，由该地址描述符表示该操作数的寻址方式。

(a) 在操作码中表示寻址方式　　(b) 在地址码中表示寻址方式

图6.14　寻址方式的两种表示形式

2. 寻址方式的编码方法

▲ 寻址方式在地址码中表示

- 这种寻址方式的应用范围较均匀；
- 该寻址方式编码必须能够对任何指令适用，一般采用等长二进制编码方式。

▲ 寻址方式在操作码中表示

- 这种寻址方式的应用范围不均匀，某种类型的指令只支持1~2种寻址方式；
- 该寻址方式就不必进行统一编码，只要对具体指令类型进行编码即可。

6.5.4 指令字格式设计

- 设计任务是确定指令系统中各指令的具体组成格式，同时使指令字格式具有较好的性能/价格比。

▲ 指令字格式设计的四个条件：

1）操作码编码设计产生的各指令的操作码；

2）功能集设计产生的每条指令的操作数个数；

3）寻址方式表示设计产生的寻址方式表示方法及编码；

4）寻址方式设计产生的各寻址方式性能参数。

1. 指令字格式设计

▲变长编码格式

- 该编码格式的指令字有多种长度；
- 可以有效地减少指令系统中指令字的平均长度，降低目标代码的长度；
- 但会使各指令字长短不一，增加了译码器的实现难度和译码时间；
- 各指令执行时间悬殊较大，不利于流水和并行处理技术的应用。

▲ 定长编码格式

- 指令字长度均相同；
- 当指令数量和寻址方式较少时，可以有效地减少指令译码的复杂性和提高译码速度；
- 指令寻址方式少，执行速度较快，很适合流水和并行处理技术的应用；
- 但会使各指令字空间的利用率不够高，增加了目标代码的长度；
- 指令操作码采用变长编码格式，寻址方式在操作码中表示；指令系统指令数量和寻址方式种类较少，多用于RISC计算机。

▲混合编码格式

- 指令字长度只有有限的几种；
- 通过提供几种指令字长度，期望兼顾目标代码长度和降低译码复杂性这两个目标；
- 对流水和并行处理技术的应用方便性一般；
- 指令操作码和寻址方式表示与变长编码方式基本一致，指令数量和寻址方式种类适中。
- **IBM 360/370和Intel 80x86均采用这种编码方式。**

2. 指令字格式优化设计

- 优化设计：在不增加指令字长的前提下，尽可能使指令功能增强，冗余空间最少提高代码密度。

（1）对操作码进行优化

- 对操作码的优化：当操作码采用变长编码方式时，为提高操作码的规整性，减少译码器译码和实现难度而进行的优化；
- 对定长操作码：可将相同类型的操作码集中在一起，减少译码成本和提高译码速度，或利用指令空闲位作为操作码扩展。

（2）对地址码进行优化

- 指令字中地址码字段性能直接影响整个指令字的功能和性能；
- 优化思想：对高频率的指令，尽量缩短其指令长度以提高性能；对低频率的指令，主要考虑扩展指令功能以提高性能/价格比。
- 指令字地址码优化设计的两点原则：

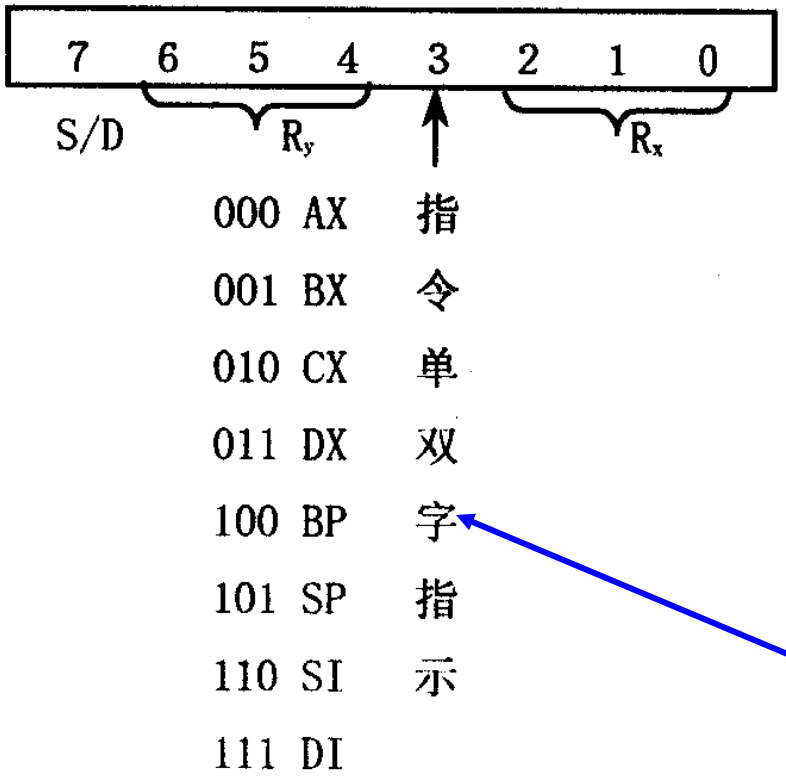
1）当地址码字段长度富裕时，可增加寻址方式或地址字，以增加指令的功能；

2）当地址码字段长度紧张或不够时，可采用特定的寻址方式，提高指令的性能/价格比；或增加指令字长，扩展指令的功能。

例：已知微机中有AX、BX、CX、DX、BP、SP、SI、DI八个寄存器，BX、BP为基址寄存器，SI、DI为变址寄存器。双操作数指令有12条，单操作数指令有46条，无操作数指令有6条。规定双操作数指令必须有一个操作数来自寄存器。请设计该指令系统。

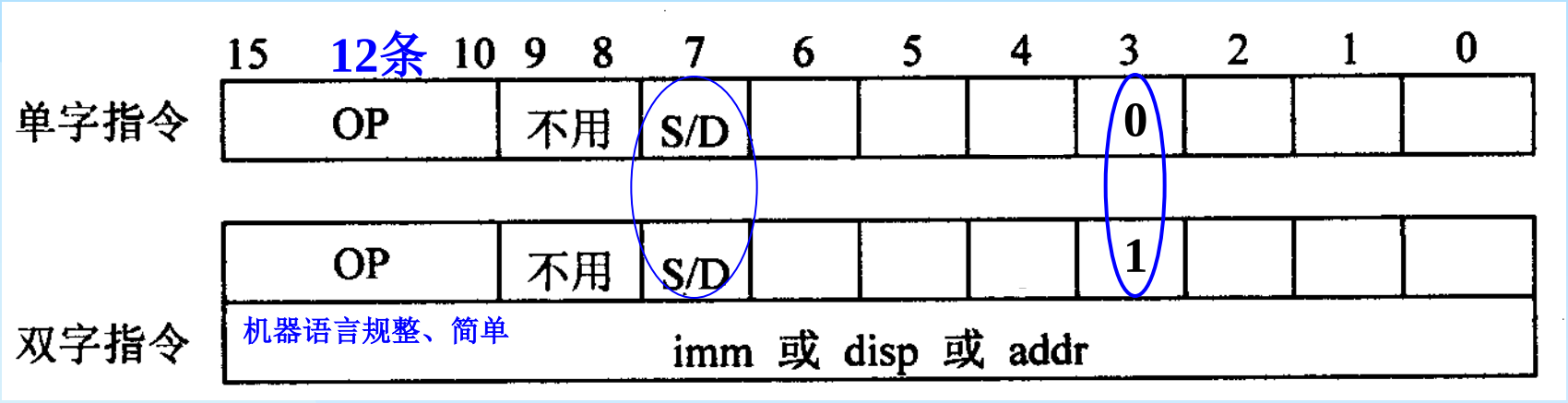
1）寻址方式设计

双操作数须有寄存器
S/D=0，R_y为源操作数

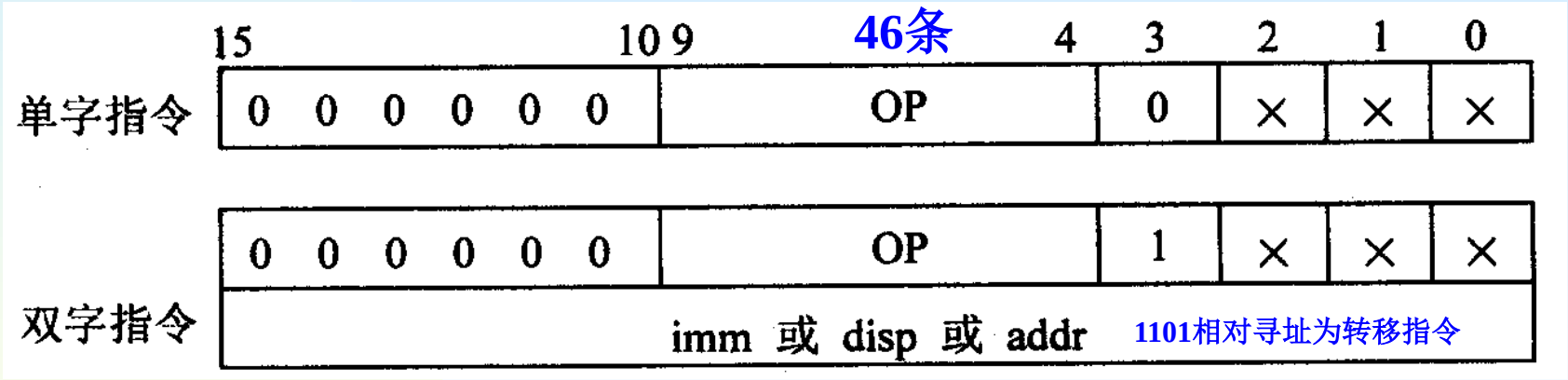


寻址字段（位3、2、1、0）	寻址方式及所用寄存器
0 0 0 0	AX
0 0 0 1	BX
0 0 1 0	CX
0 0 1 1	DX
	寄存器寻址
0 1 0 0	[BP]
0 1 0 1	[BX]
0 1 1 0	[SI]
0 1 1 1	[DI]
	寄存器间接寻址
1 0 0 0	imm 立即寻址
1 0 0 1	[BX+disp] 基址寻址
1 0 1 0	addr 直接寻址
1 0 1 1	[Addr] 间接寻址
1 1 0 0	[BP+disp] 基址寻址
1 1 0 1	[PC+disp] 相对寻址
1 1 1 0	[SI+Addr] 源变址寻址
1 1 1 1	[DI+Addr] 目的变址寻址

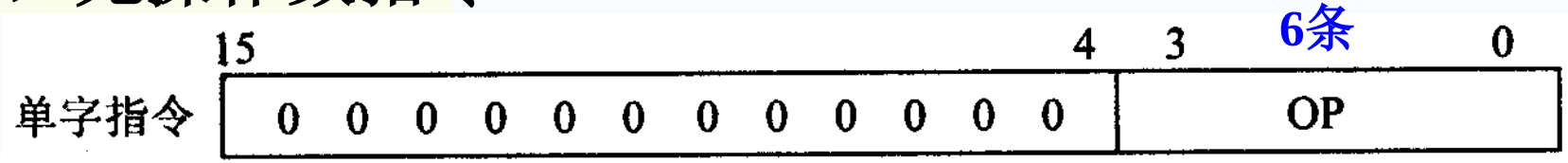
2）双操作数指令



3）单操作数指令



4）无操作数指令



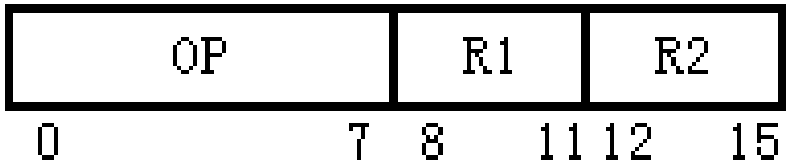
6. 6 指令系统举例

6.6.1 IBM 370系列机指令格式

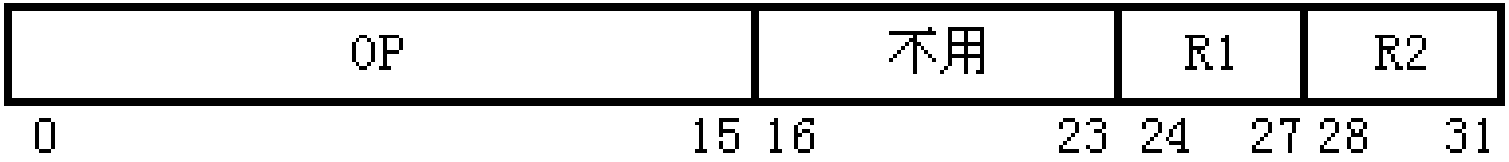
- RRE、S、SSE型指令的操作码为16位，其余指令的操作码均为8位。
- 操作码的第0位和第1位组合：
  - 00–RR型指令，01–RX型指令，10–RRE型、RS型、S型及SI型指令，11–SS型和SSE型指令。

▲ RR和RRE型指令都是寄存器-寄存器型指令

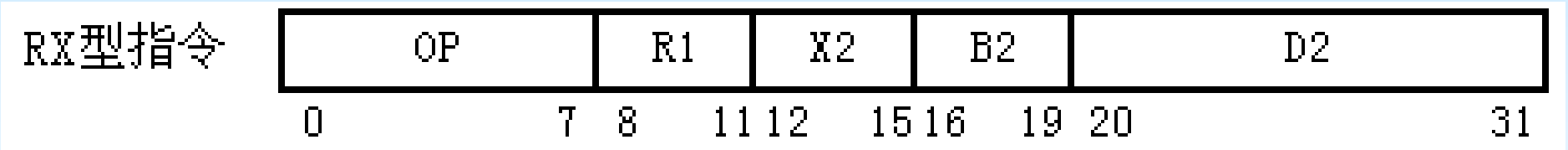
RR型指令



RRE型指令



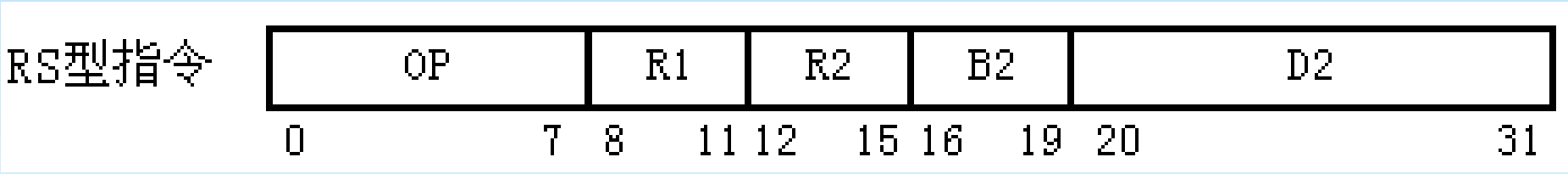
- **RX和RS型指令都是寄存器-存储器型指令，第一个操作数和结果放在R1中，另一个操作数在主存中。**



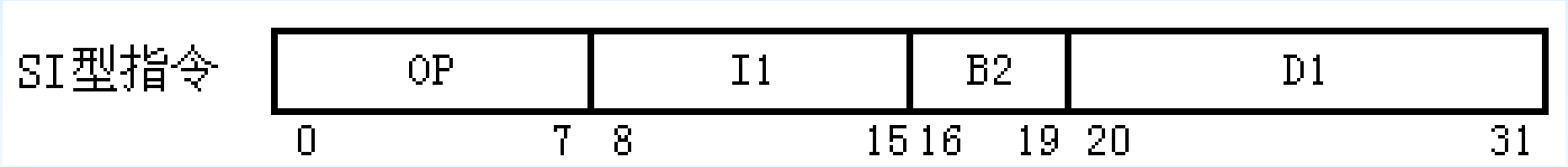
- **采用变址寻址方式，有效地址=（X2）+（B2）+D2，B2为基址寄存器，D2为位移量，x为变址寄存器号。**

▲ **RS型是三地址指令：R1存放结果，R2放一个源操作数，另一个源操作数在主存中；**

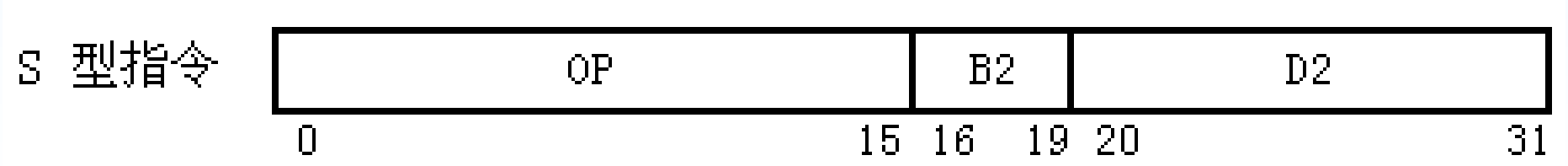
有效地址=（B2）+D2。



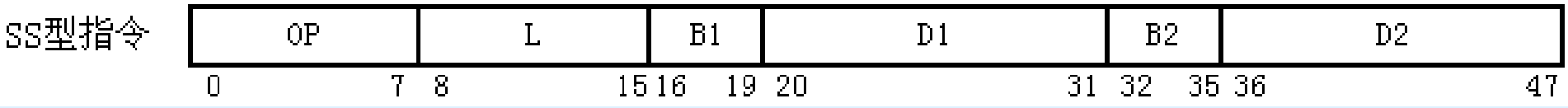
▲ **SI型是立即数指令**



▲ **S型是单操作数指令**



- **SS和SSE型指令是可变字长指令，用于字符串的运算和处理，L为串之长度。**



- **SSE指令与SS指令之差别是 SS指令中的L字段(8-15位)扩展成操作码。**

6.6.2 Pentium指令系统

1. 寻址方式

- **Pentium指令系统共支持9种寻址方式**

序号	寻址方式名称	线性地址LA算法	说明
1	立即寻址		操作数在指令中给出
2	寄存器寻址		操作数为指定寄存器的内容
3	偏移寻址	LA=(SR)+A	对应直接寻址
4	基址寻址	LA=(SR)+(B)	对应寄存器间接寻址
5	基址+偏移寻址	LA=(SR)+(B)+A	对应基址寻址
6	比例变址+偏移寻址	LA=(SR)+(I)×S+A	S为1时对应变址寻址
7	基址+变址+偏移寻址	LA=(SR)+(B)+(I)+A	对应基址+变址寻址
8	基址+比例变址+偏移寻址	LA=(SR)+(B)+(I)×S +A	
9	相对寻址	LA=(PC)+A	PC为程序计数器或指令指针

2. 指令格式

▲ **Pentium采用可变长指令格式，最短的指令只有一个字节，最长的指令可有十几个字节。**

▲ **前缀：位于指令操作码前**

前缀类型：	指令前缀	段前缀	操作数长度	地址长度
字节数：	0或1	0或1	0或1	0或1

- **前缀不是每条指令必须有的；大部分指令并无前缀，它们使用默认的条件或参数进行操作。如有的话，各种前缀也都是可选的。**

- 指定功能前缀：实现指定指令对存储器是独占访问或重复执行功能，本指令按此规则执行；
- 包括4种指令：LOCK、REP、REPE和REPNE。
- 段前缀字段：实现指定段寄存器的功能；缺省时，当前指令使用的段寄存器与上一条相同；
- 操作数长度前缀：实现指定操作数长度的功能；
- 在实模式下，操作数长度默认值是16位；在保护模式下，段描述符D=1时是32位，当D=0时是16位。

- 地址长度前缀字段：实现指定地址长度功能；可能是寄存器、指令中的存储器地址，非形式地址；
- Pentium的各种指令前缀编码必须能够和指令本身的操作码编码区分开来。

▲ 指令本身，各部分的长度和含义：

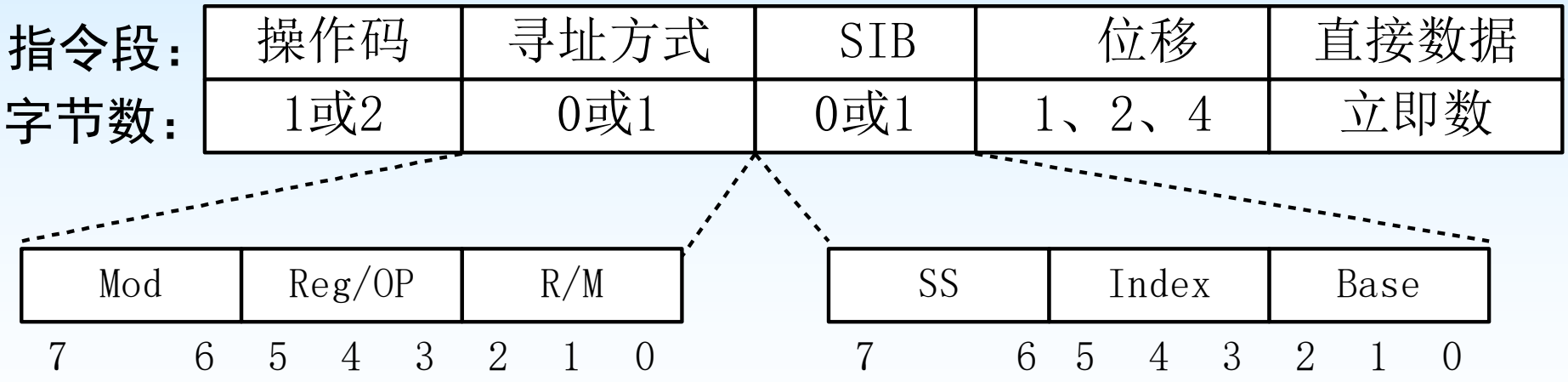
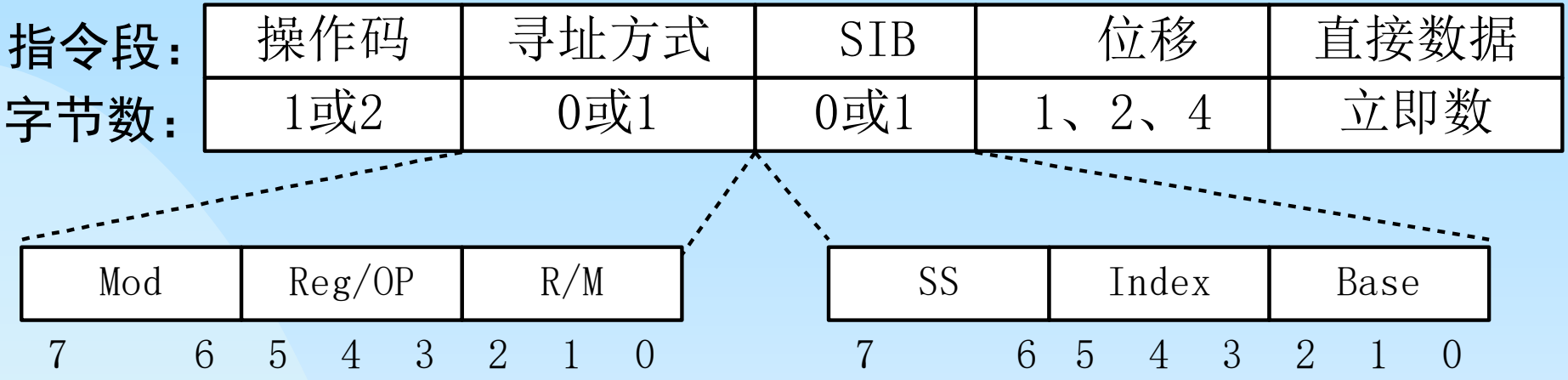


图6.15　Pentium指令格式



① 操作码字段：

- 指定指令的操作，还指明数据是字节还是全字长；指明REG字段指定的寄存器是源还是目标 。

② 寻址方式字段：

- 由MOD和R/M联合指定8种寄存器寻址和24种变址寻址方式，reg/OP指定某个寄存器为操作数或作为操作码的扩展用。

③基址变址参数（SIB）字段：

- SS指定比例系数（变址寻址方式时用）；Index指定变址寄存器；Base指定基址寄存器。

④偏移量参数（DISP）字段：

- 指定与存储器有关的寻址方式的操作数偏移量；非基址寻址方式时在指令中出现 。

⑤立即数参数（IMME）字段：

- 主要指定立即寻址方式中的立即操作数。

▲ Pentium的指令格式比较复杂

1）必须与8086指令系统兼容；

2）Pentium要实现对地址和数据的32位扩展，提高寻址方式灵活性。

3. Pentium物理地址的形成

- **Pentium的逻辑地址包括段和偏移量，段号经过段表直接得到该段的首地址，和有效地址相加形成一维的线性地址。**

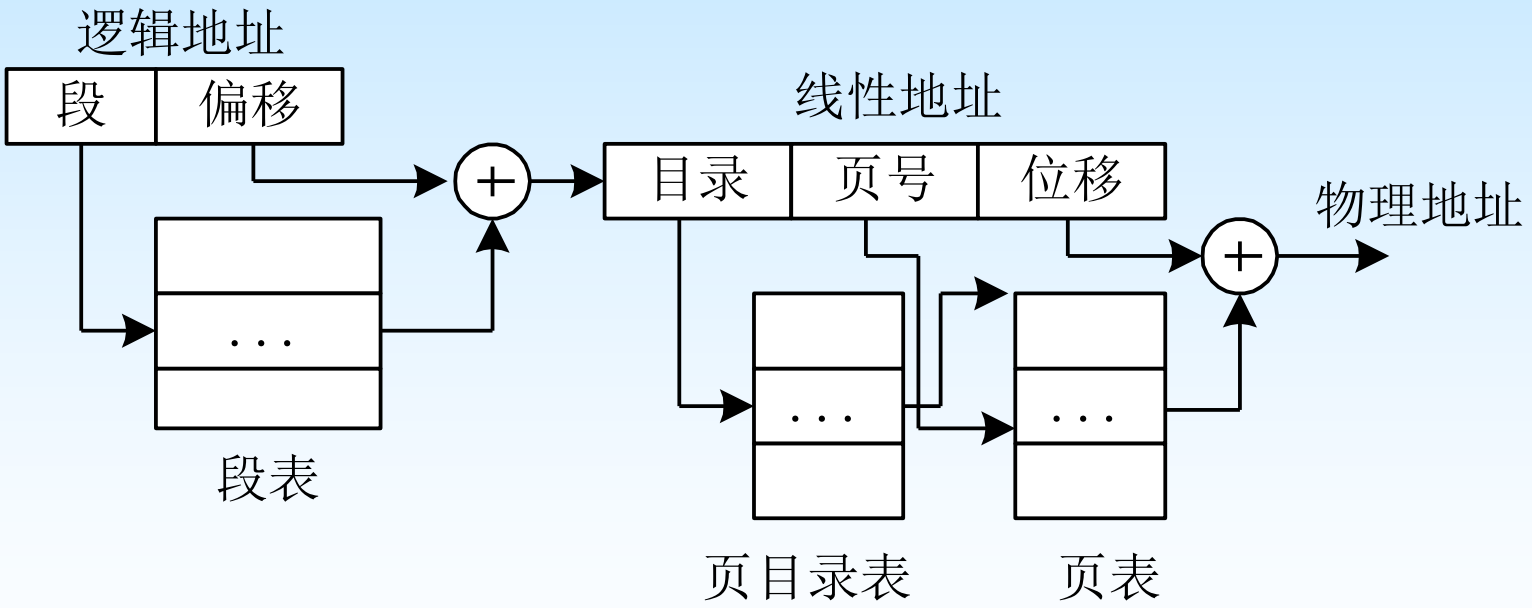


图6.16　Pentium物理地址的形成

6.6.3 Power PC指令系统

- Power PC处理器字长为32位，数据字长为32位；
- 具有32个32位通用寄存器，支持32位物理地址空间访问；
- 指令系统属于RISC结构，支持6种寻址方式，指令长度只有32位一种，指令格式有4种。

1. 寻址方式

序号	寻址方式名称	线性地址LA算法	应用指令类型	说明
1	立即寻址		定点指令	操作数在指令中给出
2	寄存器寻址		定点、浮点指令	操作数为指定寄存器的内容
3	绝对寻址	LA=(SR)+A	转移指令	对应直接寻址
4	间接寻址	LA=(SR)+(B)+A	装入、转移指令	对应基址寻址
5	间接变址寻址	LA=(SR)+(B)+(I)	装入指令	对应基址变址寻址
6	相对寻址	LA=(PC)+A	转移指令	PC为程序计数器

2. 指令格式

	6位	5位	5位	5位	5位	4~5位	0~2位
立即数型	OPER	RD/OPTION	RS/CR	DISP			R/F
寄存器型	OPER	RD/RS	RS/RD	RS/SHAMT	FUNC/MOVL		R
	OPER	RD	RS	RS	RS	FUNC	R
转 移 型	OPER	DISP					F

OPER—操作码，RD—目标寄存器号，RS—源寄存器号，DISP—偏移量或立即数
FUNC—功能码或操作码扩展，R—状态寄存器记录条件，F—寻址方式或操作码扩展
OPTION—转移方法，CR—转移条件，SHAMT—移位量

6.7 MMX技术

- MMX是Intel公司为提高PC机处理多媒体和通信能力而推出的新一代处理器技术，增加8个64位寄存器和57条新指令来实现。

1. MMX的由来与特点

▲ 多媒体应用中的图形、图像、视频、音频的操作中存在大量共同特征的操作：

- 短整数类型的并行操作(如8位图形象素和16位音频信号)；
- 频繁的乘法累加(如FIR滤波，矩阵运算)；

- 短数据的高度循环运算(如快速傅里叶变换FFT、离散余弦变换DCT);
- 计算密集型算法(如三维图形、视频压缩);
- 高度并行操作(如图像处理)。

▲ MMX技术与以前的Intel CPU(简称IA-Intel Architecture)结构相比，增加的功能。

(1) 引进了新的数据类型和通用寄存器

- MMX技术的主要数据类型为定点紧缩整数，它定义了4种新的64位数据类型。

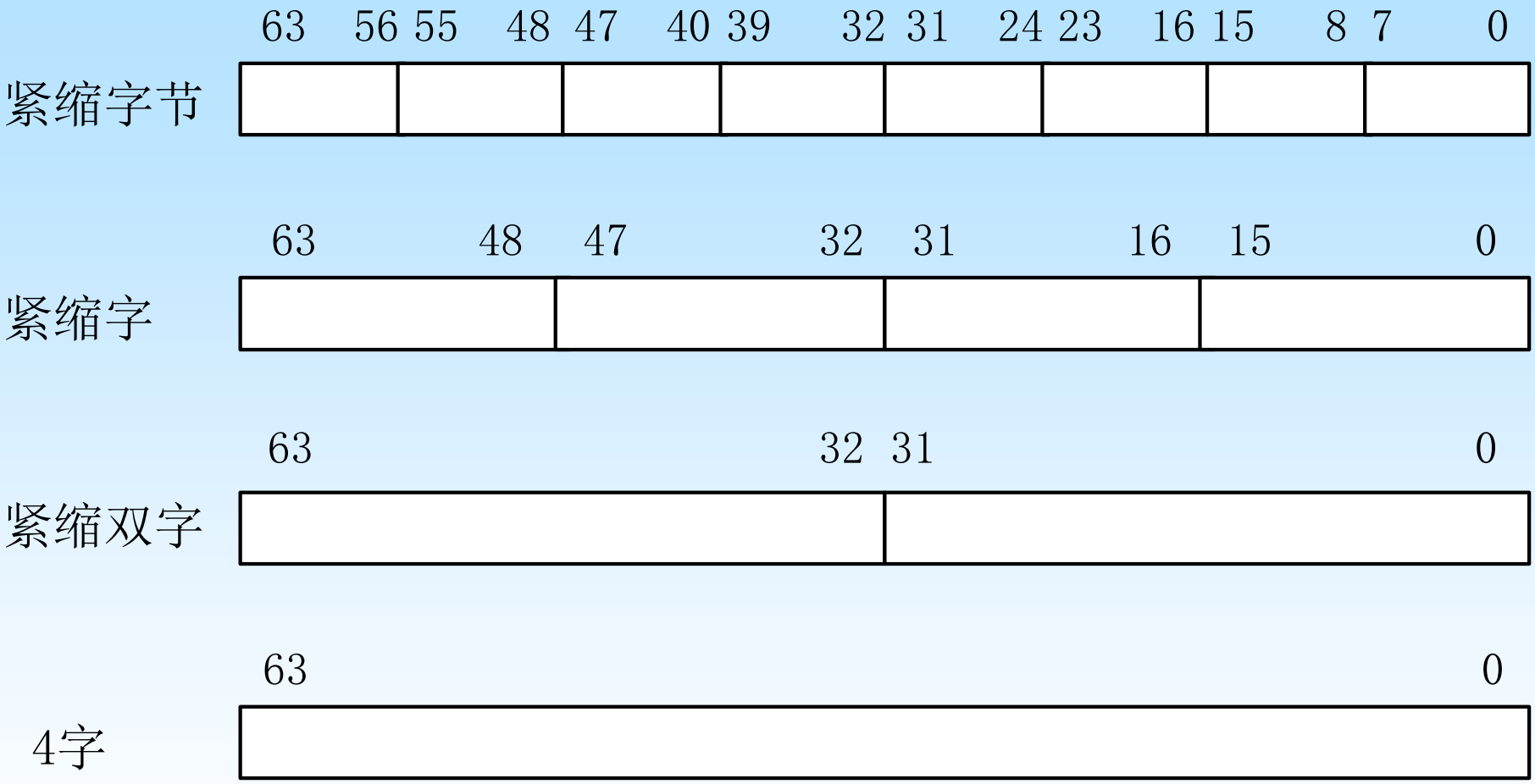


图6.17 MMX技术引入的数据类型

(2) 采用SIMD(Single Instruction Multi Data技术

- 单条指令同时并行地处理多个数据元素，提高运算速度。例如，一条指令可以完成图形/图像中8个象素(每象素8位)的并行操作。

(3) 饱和(Situration)运算

- 环绕运算或称非饱和运算，上溢或下溢的结果被截取，返回低有效位值，F3H+1DH=10H。
- 饱和运算：上溢与下溢结果被截取至各类数据值域的最大值或最小值。如，F3H+1DH=FFH 。

表6.2 MMX饱和运算范围

		下限		上限	
符号	字节	16 进制	十进制	16 进制	10 进制
		80H	-128	7FH	127
	字	8000H	-32768	7FFFH	32767
符号	字节	00H	0	FFH	255
	字	0000H	0	FFFFH	65535

- 饱和运算的应用：设a点亮度值为F3H，b点亮度值为1DH，采用非饱和运算模式，其线性插值的结果为10H/2=08H。该结果的亮度值比b点还低。
- 引入饱和运算后，a与b占亮度插值应为：

$$\frac{F3H + 1DH}{2} = FFH/2 = 7FH$$

(4) 兼容性

- MMX技术与现有的IA(Intel Architecture)处理器和OS保持向下兼容。

2. MMX指令系统

- 早期MMX结构中引入了57条新的指令，P4 MMX指令增为144条，SIMD得到了强化，MMX寄存器也扩充为128位。

- 57条指令分成7大类：算术运算指令、逻辑运算指令、比较指令、转换指令、移位指令、数据传送指令、置空MMX状态指令。

(1) 指令句法

·数据类型：紧缩字节、紧缩字、紧缩双字、64位字；有符号数与无符号数；环绕和饱和运算。

·句法： 词头：P(Packed)表示紧缩型指令

指令： ADD(加)、CMP(比较)、XOR(异或)...

词尾： -US 表示无符号饱和运算

-S 表示有符号饱和运算

-B、W、D、Q分别表示数据类型为字节、字、双字、4字

- 例如 PADDUSW 表示紧缩无符号字饱和相加指令。

(2) 指令类型

1) 算术运算类指令

- 有紧缩加(ADD)、减(SUB)、乘法(MUL)、乘加(MADD)等指令。

- PADD(Packed Add) 环绕字节加法，无数据类型时为字节运算；

PADDS(Packed Add With Saturation) 饱和字节加法。

2) 比较指令

- PCMPEQ(相等比较)、PCMPGT(是否大于)，这类指令用于条件转移操作。

3) 转换指令

- 实现各类紧缩数据之间的转换。

4) 逻辑指令

- 在64位数据上进行按位“与”、“与非” 或“与非”和“异或”操作。

5) 移位指令

- 64位数据的逻辑左/右移和算术右移。

6) 数据传送指令

- 在主存储器和MMX寄存器之间或MMX寄存器本身之间64位数据传送；或整型寄存器与MMX寄存器之间32位数据传送。

7) EMMX(Empty MMX State)

- 置空MMX状态。在MMX例程结束时，清除MMX状态。

(3) 操作数

- MMX指令中除数据传送指令外，源操作数都驻留在存储器或MMX寄存器中，而目标操作数驻留在MMX寄存器中；

- DEST(左操作数) ← DEST(左操作数) OP SRC(右操作数)
  结果 目标数据 源数据。

- 操作数的书写方法：

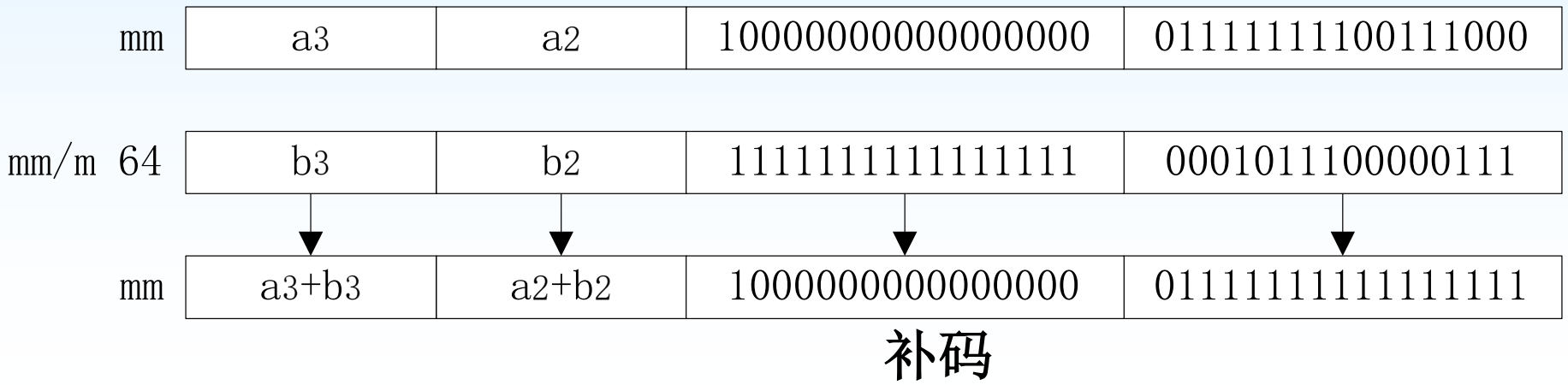
· imm：一个字节有符号立即数；
· r/m 32：一个32位双字寄存器或存储器操作数；
· mm/m32：一个MMX寄存器的低32位或32位存储器数；
· mm/m64：一个64位的MMX寄存器或64位存储器数。

如：

① **MOVD mm，r/m 32** 从整数寄存器或存储器传送32位数据到MMX寄存器。

② **PADDB mm，mm/m 64** 环绕字节运算。MMX寄存器内容或64位存储器数与MMX寄存器内容环绕相加，结果写入MMX寄存器。

③ **PADDSW mm，mm/64** 带符号紧缩字饱和相加。



作业一：**P164— 4、5、6**

作业二：**P164～165— 9、10、11**