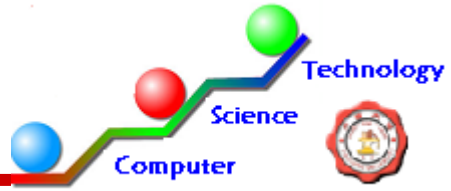


计算机组成原理

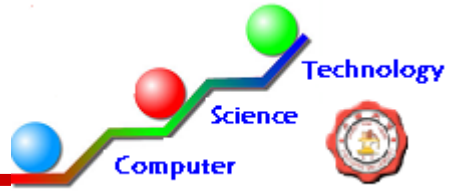
Computer Organization

2023 . 秋

西安交通大学 计算机科学与技术系

计算机组成原理课程组

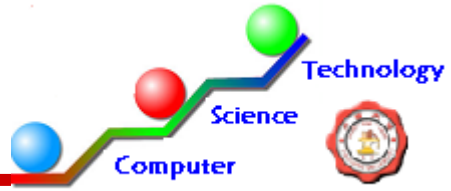
http://corg.xjtu.edu.cn



计算机组成原理

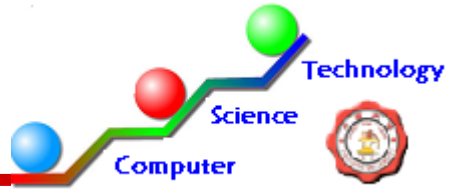
第二章 指令系统

目录



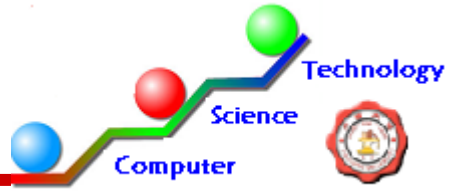
第二章 指令系统

指令系统概述



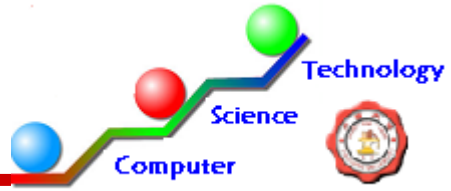
- 指令系统是指一台计算机所具有的全部机器指令的集合，它反映了该机所拥有的基本功能。
- 指令系统是计算机硬件的语言系统，也被称为机器语言。
- 指令系统是软件和硬件的主要交界面，也是计算机软件设计者和硬件设计者之间沟通的桥梁。
- 指令系统决定了机器硬件所具有的能力，也决定指令的格式和机器的硬件结构。
- 指令系统也直接影响到软件的结构、复杂度和性能。
- 指令系统的设计由体系结构设计者完成；指令系统的逻辑实现 是计算机组成的研究范畴。

目录



第二章 指令系统

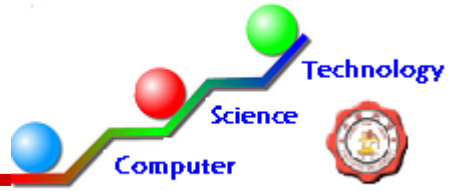
指令系统发展历程



指令系统经历了从简单到复杂，然后又从复杂到简单的**螺旋式**演变过程。

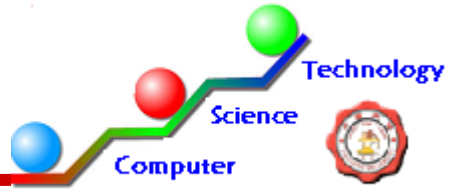
- 从简单到复杂
  - **50~60年代：电子管或 晶体管计算机**
    - 硬件结构比较简单
    - 仅有十几至几十条基本指令，且寻址方式简单
  - **60年代中期：集成电路计算机**
    - 硬件功耗、体积、价格下降，功能增强
    - 指令数达100～200条，寻址方式多样化

指令系统发展历程（续）



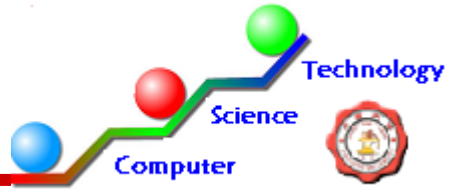
- **60年代后期到70年代中期：半导体存储器出现**
  - 系列计算机诞生
  - 新推出的机种指令系统包含旧机种的全部指令，旧机种上运行的各种软件可以不加修改便可在新机种上运行，即软件向后兼容
- **70年代末期：超大规模集成电路计算机**
  - 硬件成本下降，软件成本提高
  - 指令系统更加复杂和完备，指令数目可达300～500条，寻址方式也多样化

指令系统发展历程（续）



- 从复杂到简单
  - 庞大的指令系统不但使计算机的研制周期变长，而且增加了调试和维护的难度，其结果还可能降低计算机系统的性能。
  - 1979年，美国加州大学伯克利分校Patterson教授领导的研究组，首次提出了RISC（Reduced Instruction Set Computer，精简指令系统计算机）的思想。

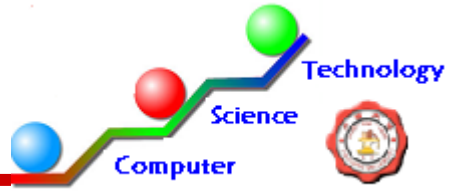
复杂指令系统计算机—CISC



- CISC指令系统的特征
  - 软件硬化：用一条功能复杂的新指令来取代原先需一串指令完成的功能
  - 支持高级语言程序：增加新的复杂指令以及复杂的寻址方式
  - 软件兼容：系列机软件要求向上兼容和向后兼容，指令系统不断扩大
- CISC计算机存在的主要问题
  - 仅有约20%的指令使用频度比较高，这些指令占据了80%的CPU时间

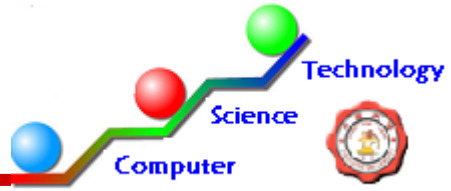
大量不经常使用的指令导致计算机硬件非常复杂，使得计算机研制周期变长，难以调试、维护且可靠性差。

精简指令系统计算机—RISC



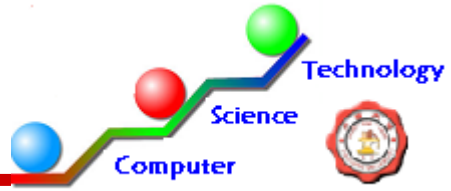
- RISC计算机的特点
  - 优先选取使用频率较高的简单指令
  - 指令长度固定，指令格式种类少，寻址方式种类少
  - 只有取数／存数指令访问存储器
  - CPU中通用寄存器数量相当多
  - CPU采用流水线结构，大部分指令可以在一个时钟周期内完成
  - 控制单元设计以硬布线控制逻辑为主
  - 采用编译优化技术，以减少程序执行时间

目录



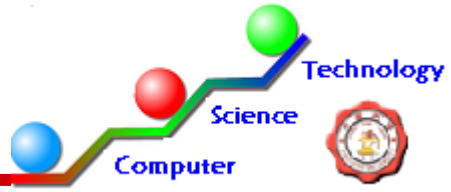
第二章 指令系统

指令系统设计原则



- 完备性——功能需求
  - CISC指令系统的主要特点
  - RISC指令系统不强调完整性
- 规整性——硬、软件设计需求
  - 对称性：寻址方式
  - 匀齐性：数据类型
  - 一致性：指令格式和数据格式
- 高效性——性能需求
  - CISC：完善指令系统功能，减小程序中指令的条数
  - RISC：降低每条指令的执行时间
- 兼容性——通用性要求

数据类型



- **数据类型**指面向应用或者软件系统所处理的各种数据结构
  - 基本数据类型：整数、实数、布尔数、字符等
  - 复杂数据类型：文件、图、表、树、阵列、队列、链表、栈、向量等

- **数据表示**指机器硬件能够直接识别、指令能够直接操作的数据结构。

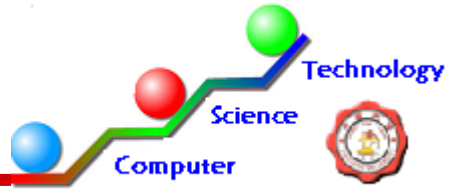
  例如定点数（整数）、逻辑数（布尔数）、浮点数（实数）、十进制数、字符、字符串等。

- **操作数**指机器指令中的数据，即硬件可以直接识别和处理的数据。

操作数类型

- 确定操作数类型的原则
  - 有利于缩短程序的运行时间
  - 有利于减少CPU与主存储器之间的通信量
  - 数据表示应具有通用性和较高利用率

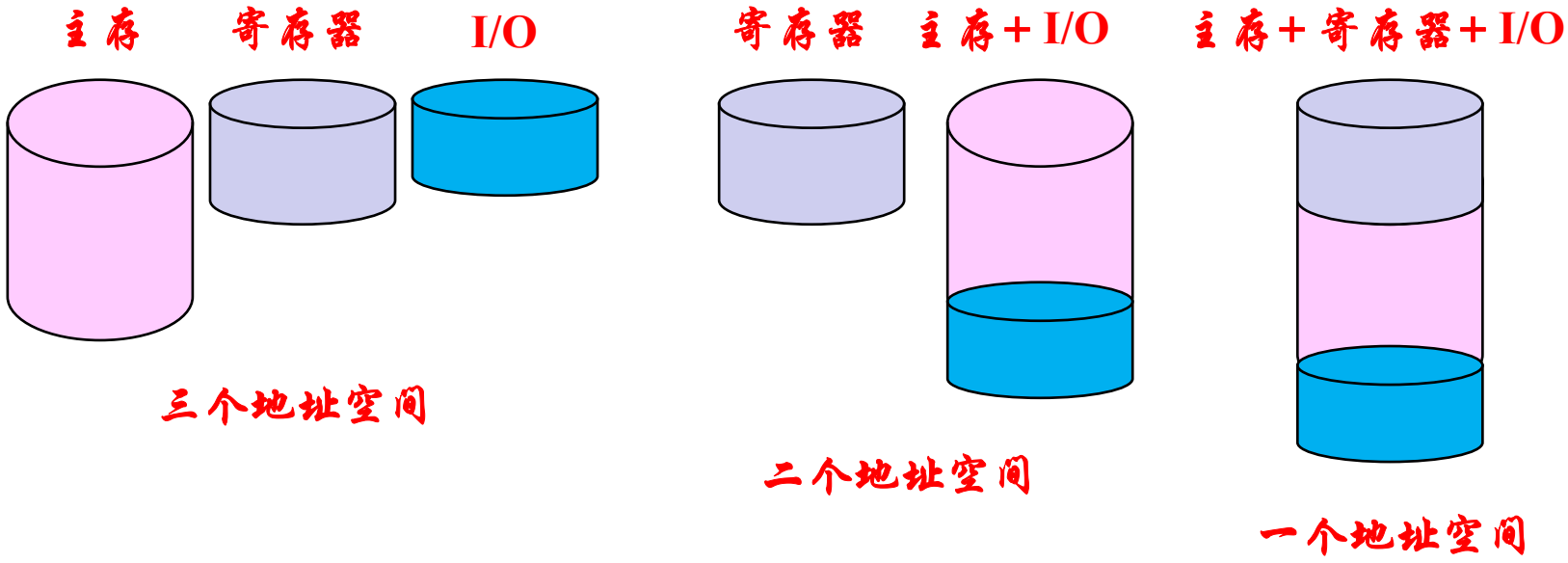
操作数类型（续）



- 地址
  - 操作数或指令被存放在数据存储设备的位置编码
  - 主要数据存储设备有通用寄存器、主存储器和I/O设备
  - 地址可以被认为是一个无符号整数
- 数字
  - 计算机处理的最基本操作数类型
  - 计算机中常用的数字类型有定点数、浮点数等
- 字符
  - 在非数值计算领域表示和处理文本信息
  - 将字符数字化表示，比如ASCII码
- 逻辑数
  - *n*位二进制数的组合，但各位之间可以没有任何关系
  - 用于逻辑运算

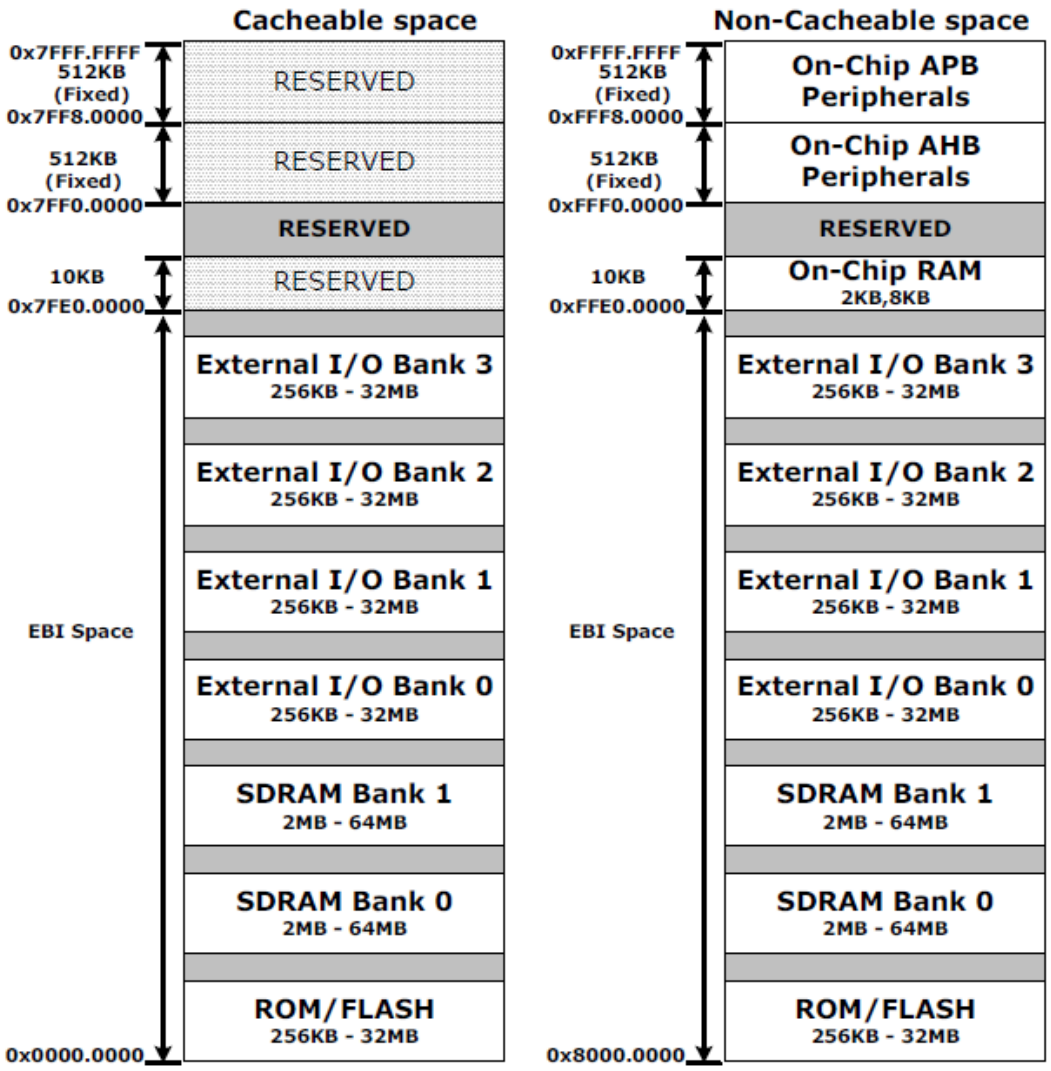
地址空间

计算机中主要的数据存储设备有通用寄存器、主存储器和 I/O 设备，它们各自都包含多个可编址的数据访问单元。对这些单元可以统一编址或者单独编址。

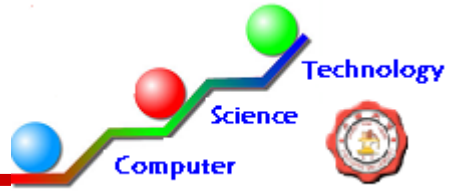


思考：不同地址空间对指令系统设计和硬件设计的影响？

ARM的二地址空间实例



主存编址方式



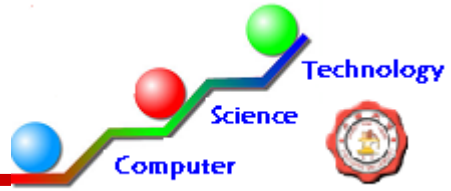
编址方式是指主存单元的地址编排方式。编址方式决定了主存最小访问单位。

- 按字编址方式
  - 主存的最小编址单位是一个存储字，通常，存储字长=机器字长
  - 对主存数据的访问以字为单位
  - 主存容量=存储字数×存储字长，单位为字（Word）或位（bit）

    eg. 128M ×32位

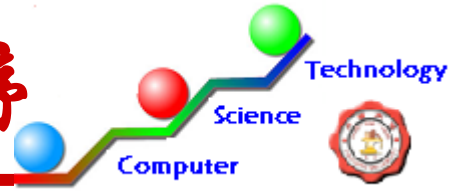
按字编址方式对应用来说不够灵活和方便，特别是在非数值计算应用领域。

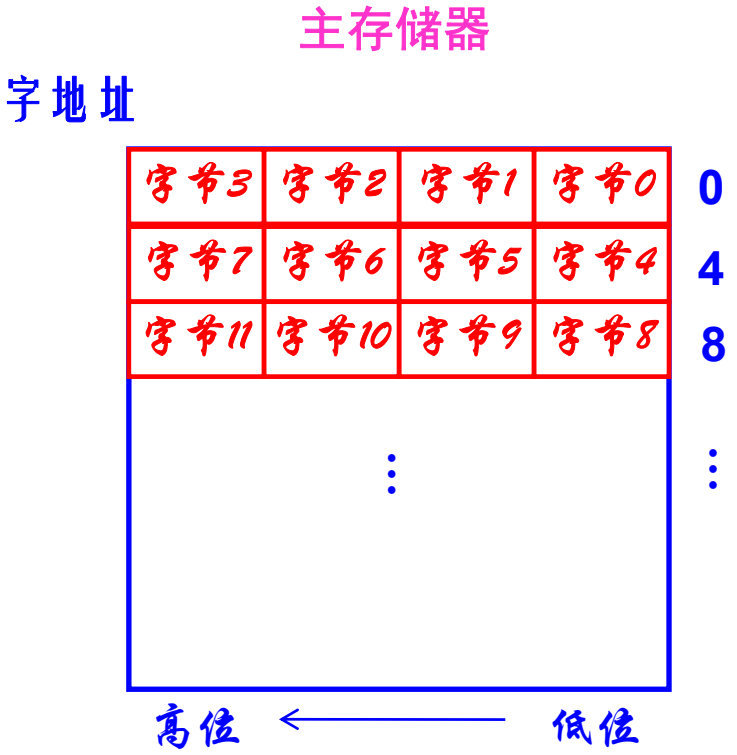
主存编址方式（续）



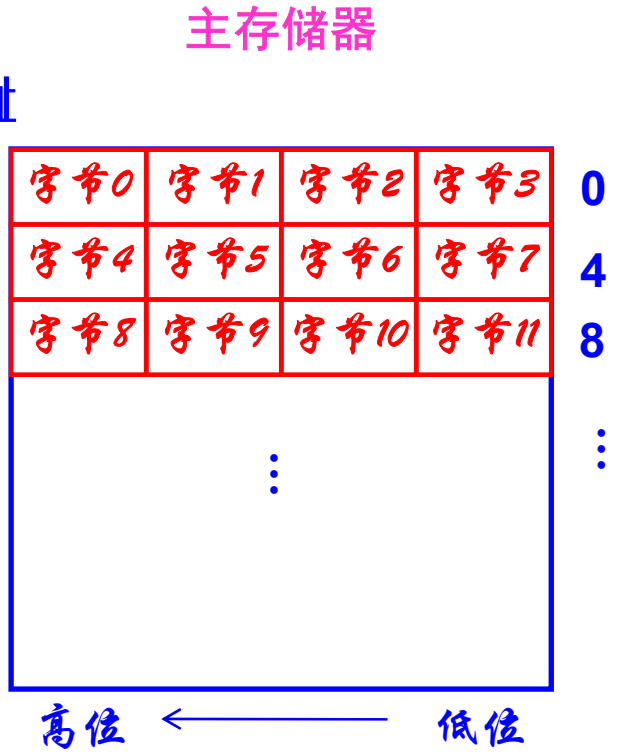
- 按字节编址方式
  - 主存的最小编址单位是一个字节，描述主存储容量时，以字节（Byte，B）为单位
  - 对主存数据既能以字节为单位访问，也能以字为单位访问
  - 当按字节访问主存时，使用字节地址；当按字访问主存时，使用字地址。
  - 通常，存储字长是字节整数倍，字节地址是连续的，字地址是不连续的
  - 多个字节数据存放在一个字单元，有两种编址顺序：低字节低地址（小端方式）、高字节低地址（大端方式）；也有存放边界问题：边界对齐、边界不对齐。

主存编址方式——字节编址顺序





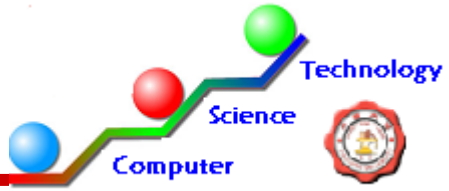
（a）低字节低地址
小端方式

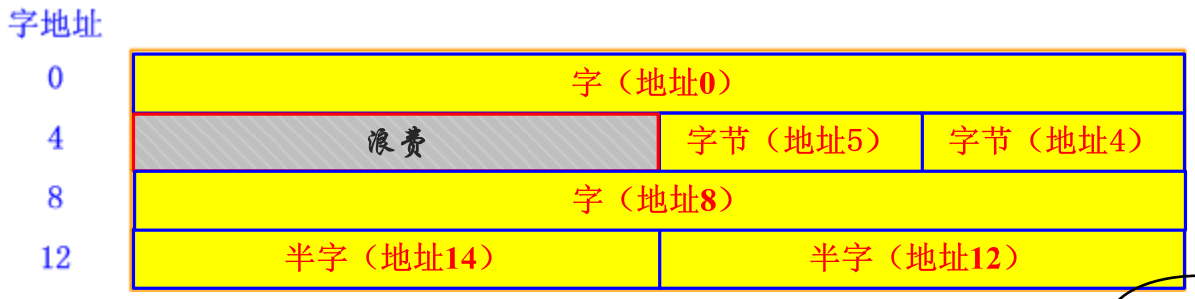


（b）高字节低地址
大端方式

思考：高级语言程序定义的不同类型数据，如何在主存中存储和访问？

主存编址方式——边界问题





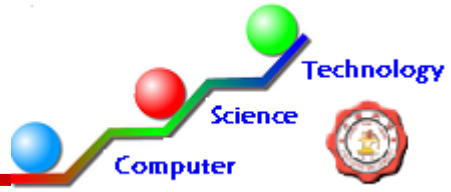
边界对齐方式

规定了各种数据类型存放的起始位置

边界不对齐方式

通常，按字节编址的机器硬件都支持边界不对齐方式，为了保证程序执行速度，软件可以选择采用对齐或不对齐方式。

操作类型



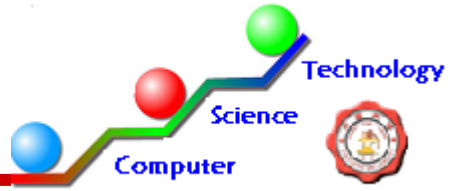
所谓操作类型就是把指令系统按功能进行分类。一般指令系统包含 5 大类指令。

- 数据传送指令
- 数据运算指令
  - 算术运算指令
  - 逻辑运算指令
  - 移位指令
  - 位操作指令

操作类型（续）

- 程序控制指令
  - 无条件转移指令
  - 条件转移指令
  - 调用与返回指令
  - 陷阱指令
- 输入/输出（I/O）指令
- 其它指令

数据传输指令



- 传输主体
  - 主存<->主存
  - 主存<->寄存器
  - 寄存器<->寄存器
- IA-32
  - MOV AL, BL
  - MOV AX, [BL]
- MIPS
  - LW $ s1,($ s2)
  - SW $ s1,($ s2)

算数运算指令

- 最基本指令
  - ADD
  - SUB
- IA-32
  - ADD AX, BX
  - SUB AX, BX
- MIPS
  - ADD $s1, $s2, $s3
  - SUB $s1, $s2, $s3

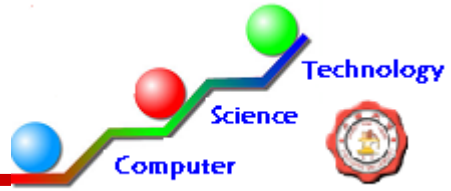
逻辑运算指令

- 最基本指令
  - AND 、NOT 或者
  - OR、NOT
- IA-32
  - AND AX, BX
  - OR AX, BX
- MIPS
  - AND $ s1, $ s2, $ s3
  - OR $ s1, $ s2, $ s3

移位指令

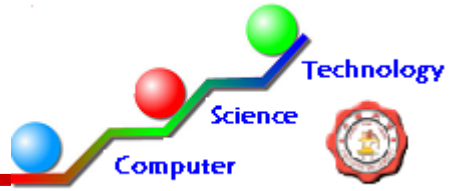
- 最基本指令
  - 左移和右移
- 参数
  - 方向，一般指令中给出
  - 位数，显式给出
- 举例
  - SHL AL，1
  - MOV DL，5
  - SHL AL，DL

程序控制指令



- 最基本指令
  - 条件转移
  - 无条件转移
  - 函数调用与返回
  - TRAP指令
- 举例
  - JZ
  - JMP
  - CALL
  - RET

目录



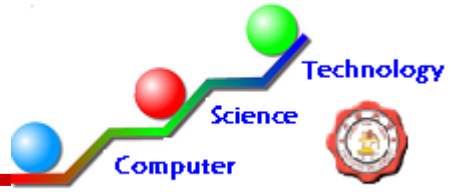
第二章 指令系统

指令格式

- 指令由操作码和地址码组成
- 操作码字段指出指令的操作性质，即指令要完成的功能
- 地址码字段指出操作数的地址，即指令操作对象所在的位置，或者下一条指令在主存储器中的地址。

指令格式：

指令字长

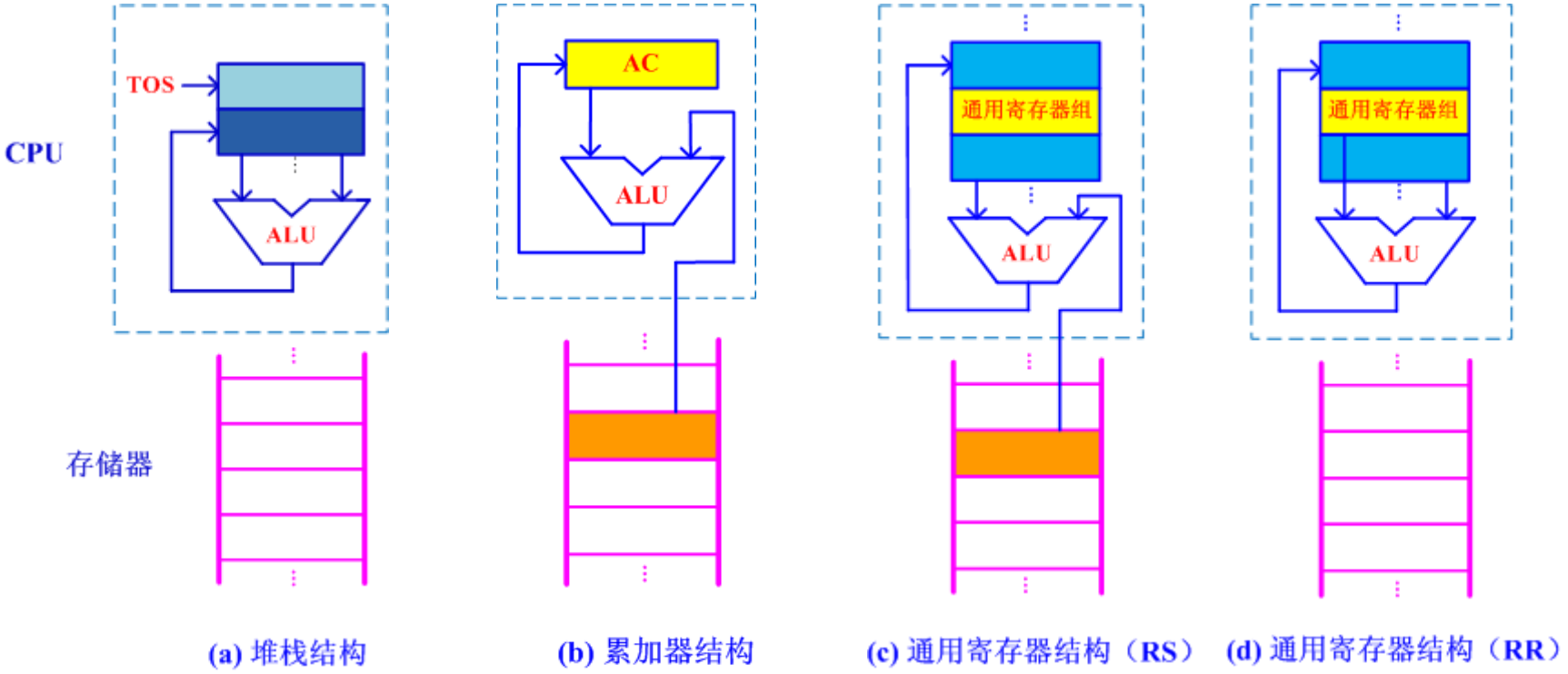


指令字长指一条指令中包含的二进制码位数。它取决于操作码的长度、地址码的长度和地址码的个数。

- 指令字长选取的基本原则
  - 指令字长尽可能短，以节省存储空间和提高处理速度
  - 指令中各信息位利用率尽可能高，以有效压缩指令字长
- 常见指令结构
  - 等长指令字结构：所有指令字长均相等，通常，指令字长=机器字长
  - 变长指令字结构：各种指令长度不等

地址码字段（续）

❒ 四种 CPU 结构



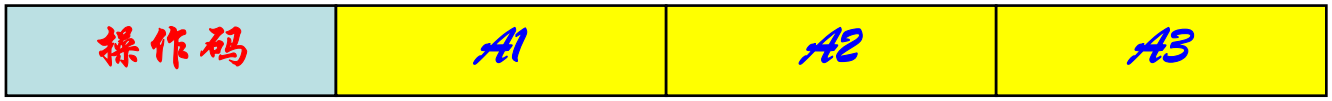
(a) 堆栈结构　(b) 累加器结构　(c) 通用寄存器结构（RS）　(d) 通用寄存器结构（RR）

通用寄存器结构分为：寄存器—寄存器型（RR型）、寄存器—存储器型（RS型）

地址码字段（续）

❒ 地址码个数

❍三地址指令



操作：A3 ←（A1）OP （A2）

❍二地址指令

操作：A1 ←（A1）OP （A2）

地址码字段（续）

❍一地址指令

操作： AC ←（AC） OP（A）

❍零地址指令

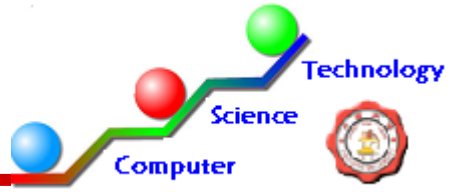
操作数来自（送往）堆栈栈顶

操作码字段

为了表示不同功能的指令，每一条指令安排一个唯一的操作码。操作码字段位数选取原则是能够表示指令系统中的全部指令。

- 操作码长度
  - 定长操作码：所有指令的操作码位数相同，并将操作码集中安排在指令字的一个固定的字段中
  - 变长操作码：各种指令的操作码位数不一致，并且操作码可以分散在指令字的不同字段中

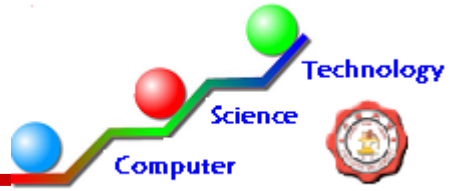
操作码扩展技术



当采用定长指令字格式，且多种地址码结构混合使用时，可利用地址码个数较少的指令空出的地址码字段，来增加操作码的位数。

- 操作码扩展方法
  - 等长扩展法
    - 4-8-12扩展法
    - 3-6-9扩展法
  - 不等长扩展法

操作码扩展举例



指令格式及操作码编码				说明
OP_Code	A1	A2	A3	4位操作码的三地址指令 15条
0000 0001 … 1110	A1	A2	A3	
OP_Code		A1	A2	8位操作码的二地址指令 15条
1111 0000 1111 0001 … … 1111 1110		A1	A2	
OP_Code			A	12位操作码的一地址指令 15条
1111 1111 0000 1111 1111 0001 … … … 1111 1111 1110			A	
OP_Code				16位操作码的零地址指令 16条
1111 1111 1111 0000 1111 1111 1111 0001 … … … … 1111 1111 1111 1111				

指令格式举例（续）

❒ MIPS 32 指令格式

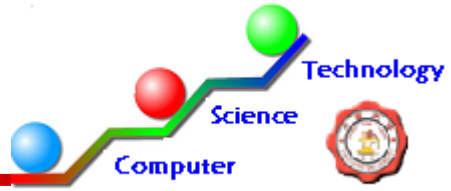
32位定长指令格式，操作码字段也是固定长度，没有专门的寻址方式字段，由指令格式确定寻址方式。

(a) R-型指令

(b) I-型指令

(c) J-型指令

目录



第二章 指令系统

寻址方式

计算源操作数、目的操作数以及下一条指令所在存储设备具体位置的方法称为寻址方式。

❒ 与地址相关的概念

- ❍ 形式地址：也称符号地址，通常指由指令中显式给出的地址。
- ❍ 有效地址：也称逻辑地址，它通常可以根据形式地址通过某种变换得到，变换规则由具体的寻址方式来确定。

有效地址EA由寻址方式和形式地址共同来确定的。通过形式地址求有效地址所采用的算法就是寻址方式。

寻址方式—指令寻址

寻址方式分为指令寻址方式和数据寻址方式两类。

❒ 指令寻址方式

确定下一条将要执行的指令所在主存单元地址的方法。其目的是为 PC设置新值。按照程序运行轨迹可分为：

❍ 顺序指令寻址

下一条指令地址由当前指令地址自增产生，即

$$PC = (PC) + 1$$

❍ 跳跃指令寻址

由转移类指令给出下一条指令的地址信息。**即PC内容按转移地址重新设置，而不是由PC顺序计数提供。**

寻址方式—数据寻址

❒ 数据寻址方式

- ❍确定当前指令所涉及的操作数在数据存储设备中的地址的方法。
- ❍现代计算机中，寻址方式较多，为了正确、有效地获得操作数，通常在指令中安排几位标志位表示所用的寻址方式，称为“寻址方式码”或“寻址特征码”。
- ❍指令格式如下：

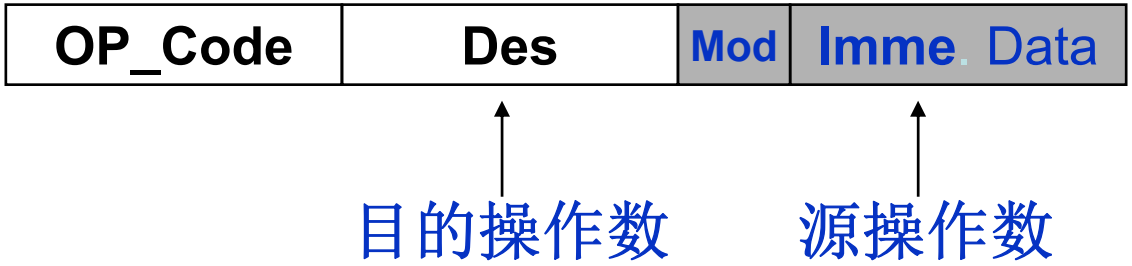
操作码	寻址特征1	A1	寻址特征2	A2

- ❍数据寻址方式可分为基本寻址方式和复合寻址方式。
- ❍基本寻址方式根据数据所在的位置不同可分为四大类：立即寻址、寄存器寻址、存储器寻址和堆栈寻址。

寻址方式—立即寻址

❒ 立即寻址方式

❍ 所需的操作数在指令的地址码部分直接给出



❖ 说明

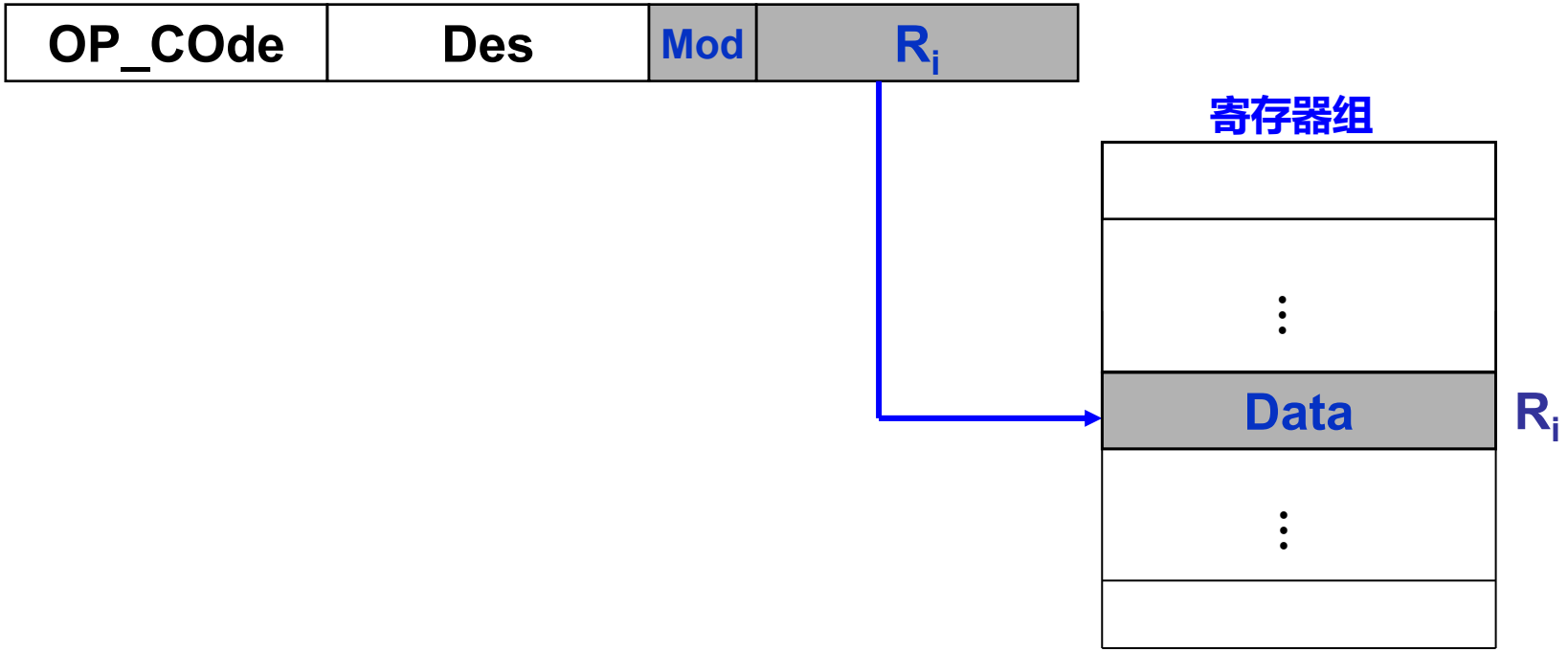
- 立即寻址通常只适用于源操作数
- **Operand = Imme. Data**
- 例如，**IA-32**中，**mov eax,100**

  MIPS 32 中，**addi $s1,$s2,100**

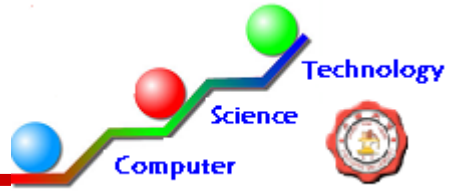
寻址方式—寄存器寻址

❒ 寄存器寻址方式

- ❍操作数在寄存器中，指令执行速度快。
- ❍地址码字段的形式地址部分给出通用寄存器编号 Ri，地址码字段短。



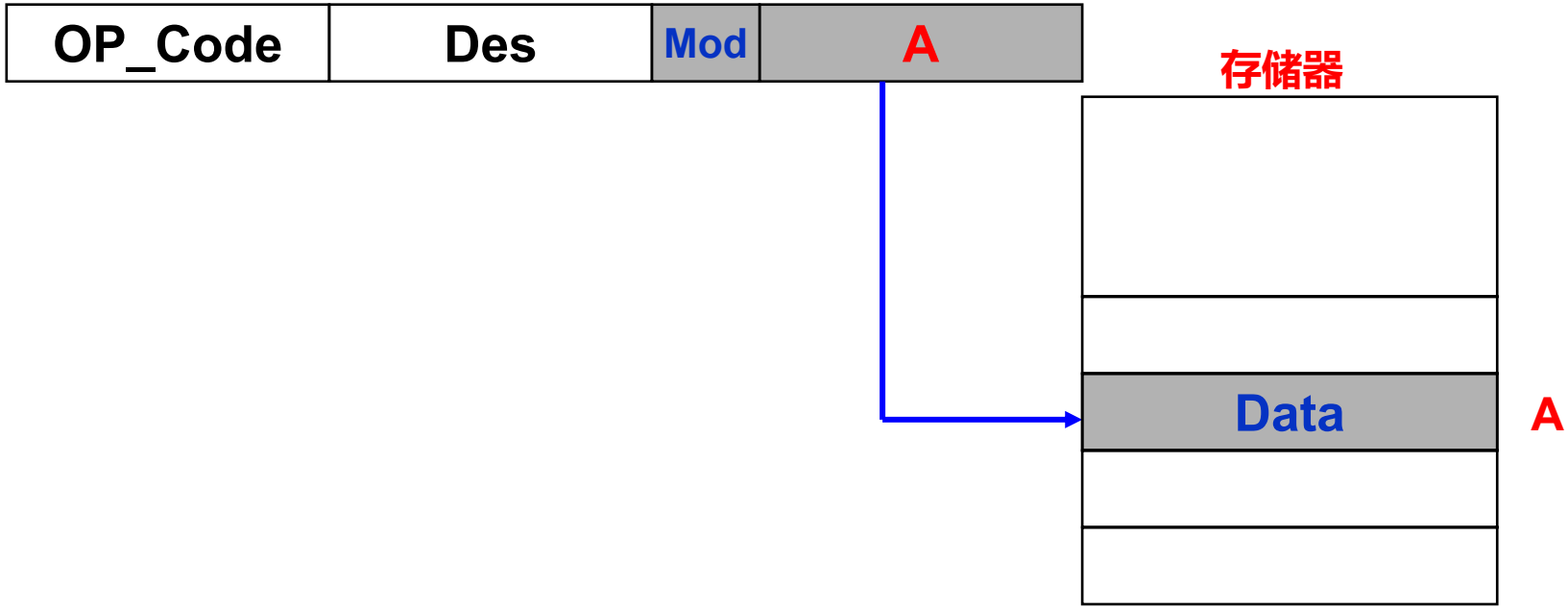
寻址方式—存储器寻址



- 存储器寻址方式
  - 根据有效地址的不同形成方法，又分为
    - 直接寻址
    - 存储器间接寻址
    - 寄存器间接寻址
    - 偏移寻址以及段寻址
  - 这些寻址方式也可以用于跳跃执行的指令寻址

寻址方式—直接寻址

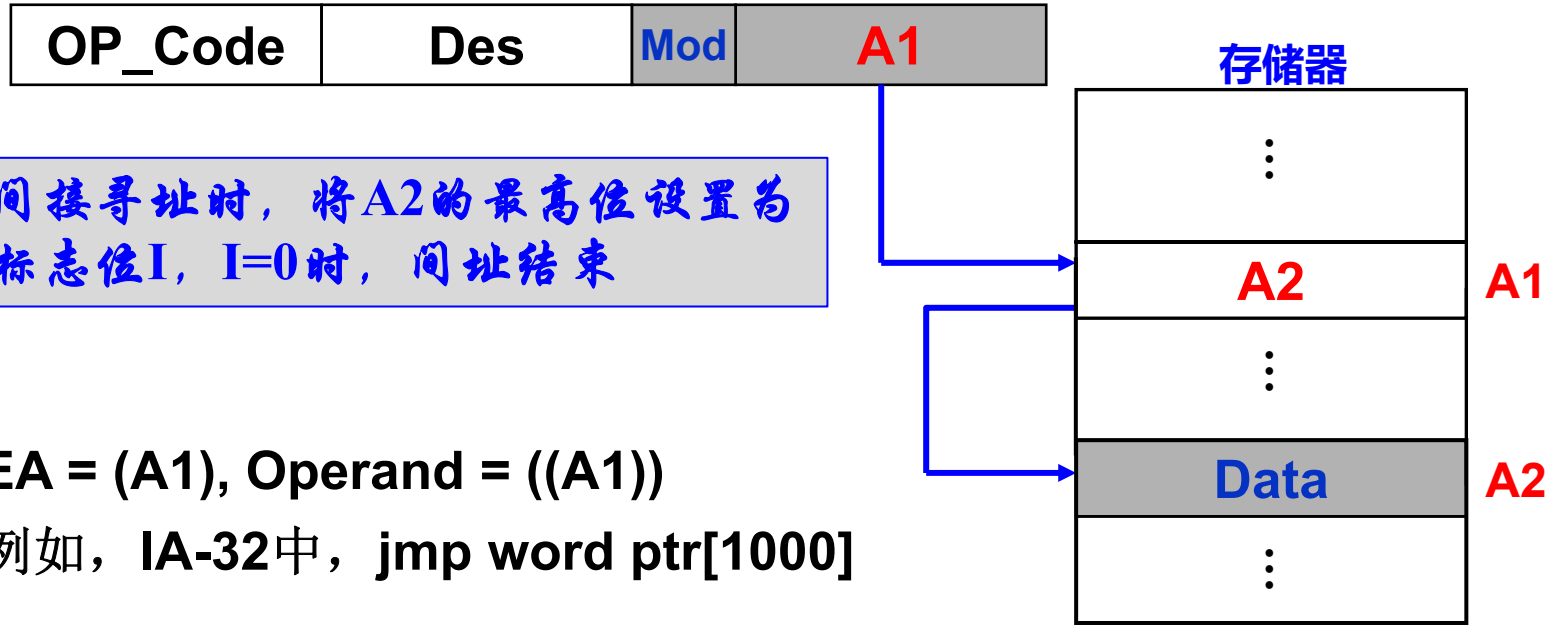
- 直接寻址方式
  - 指令地址字段直接给出操作数在存储器中的地址
  - 寻址速度快，但寻址范围小。



寻址方式—间接寻址

❍间接寻址方式

- ✧地址码字段给出的内容既不是操作数，也不是操作数的地址，而是操作数地址的地址。
- ✧分为一次间接和多次间接寻址。
- ✧可扩大指令寻址范围，但指令执行速度慢。

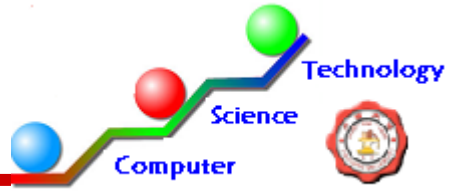


多次间接寻址时，将A2的最高位设置为间址标志位I，I=0时，间址结束

❖ 说明

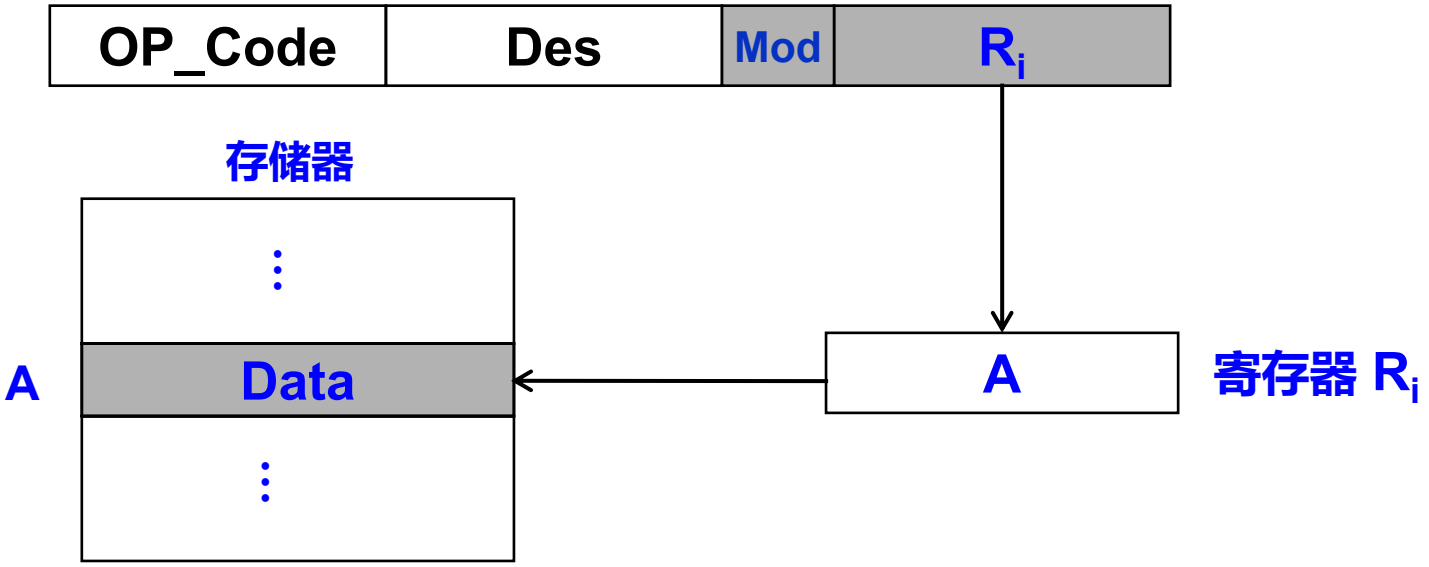
- ➢EA = (A1), Operand = ((A1))
- ➢例如，IA-32中，jmp word ptr[1000]

寻址方式—寄存器间接寻址



❍寄存器间接寻址方式

- ✧地址码字段给出某一通用寄存器的编号
- ✧该寄存器中存放的是操作数在主存单元的地址。

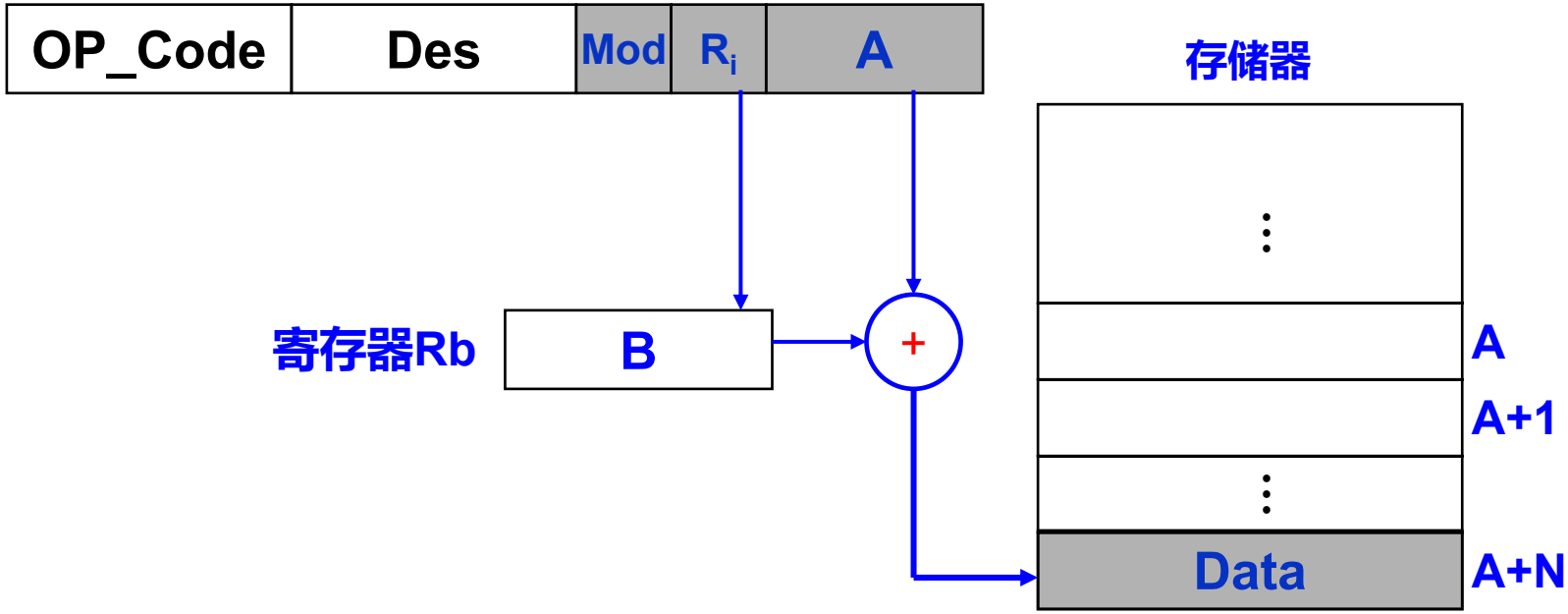


- ➢EA = (Ri), Operand = ((Ri))
- ➢例如，IA-32中，mov eax, [ebx]

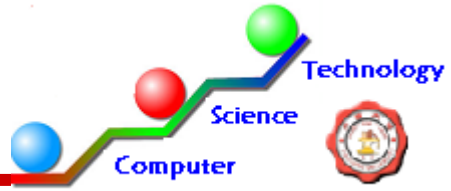
寻址方式—偏移寻址

❍偏移寻址方式

- ✧形式上可认为是直接寻址和寄存器间接寻址的结合
- ✧地址码字段既要给出形式地址，也要指出引用哪一个寄存器 Ri内容实现偏移。
- ✧ EA=（Ri）+A



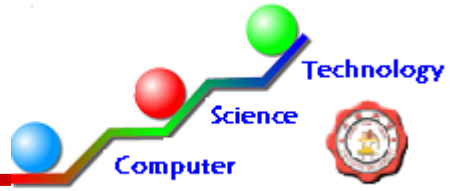
寻址方式—偏移寻址



- 常用的偏移寻址方式
  - 相对寻址
  - 变址寻址
  - 基址寻址

- 相对寻址
  - 引用专门的程序计数器 PC，即 EA =（PC）+A
  - 指令中只需要给出偏移量 A
  - 有利于程序在内存中浮动

寻址方式—偏移寻址



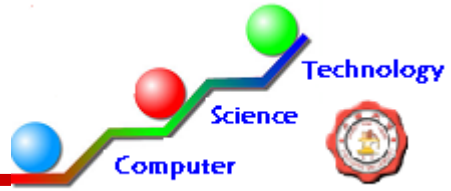
✧变址寻址

- 引用一个变址寄存器 Rx，EA =（Rx）+ A
- 变址寄存器 Rx 可以是专用寄存器或通用寄存器
- 若变址寄存器 Rx 是专用寄存器，地址字段中就不需要指出该寄存器（默认）
- 若变址寄存器 Rx是通用寄存器，地址字段要给出寄存器的编号 Ri

✧基址寻址

- 引用一个基址寄存器 Rb，EA =（Rb）+ A
- 基址寄存器 Rb 可以是专用寄存器或通用寄存器
- 若基址寄存器 Rb 是专用寄存器，地址字段中就不需要指出该寄存器（默认）
- 若基址寄存器 Rb 是通用寄存器，地址字段要给出寄存器的编号 Ri

寻址方式—偏移寻址



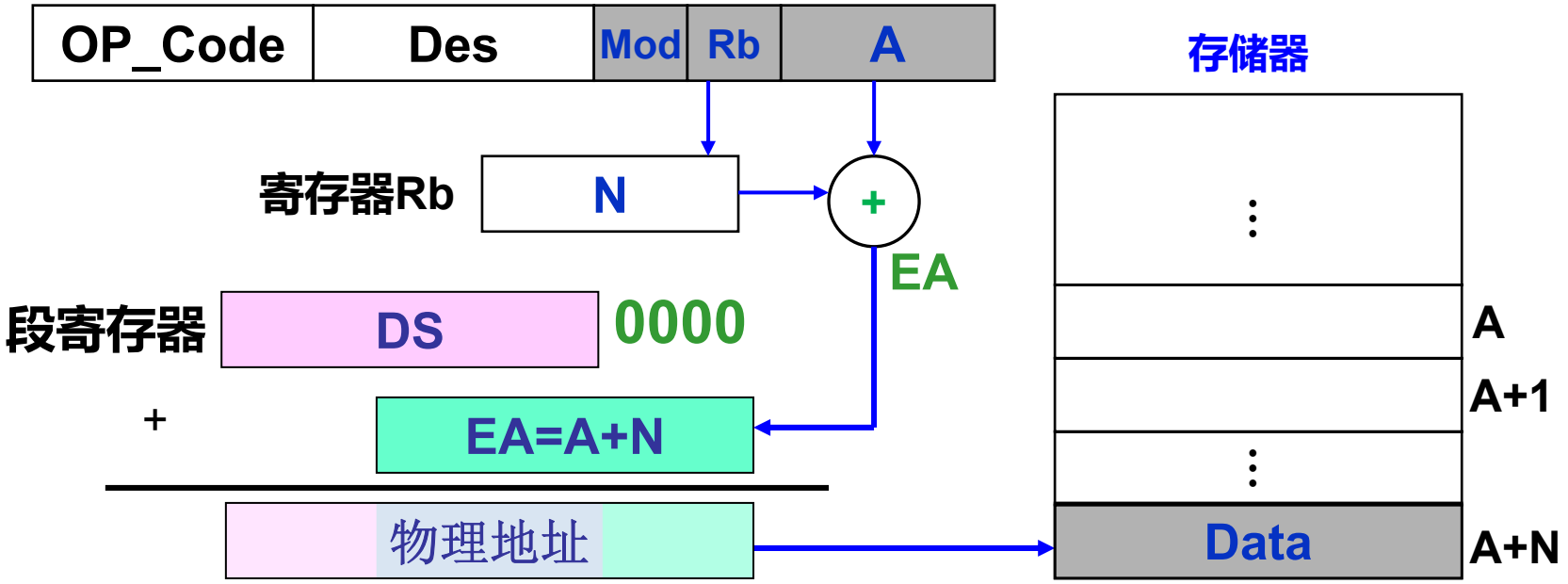
变址寻址与基址寻址的区别：

- 基址寄存器中存放基地址，一旦由系统设定一般用户不能改变，程序中指令或数据的改变由不同的位移量完成；
- 变址寄存器存放地址修改量（变址值），而形式地址给出基本地址值，操作数地址的变化由变址值增、减量完成（典型的变址寻址其变址值的增、减量由硬件自动完成）；
- 基址寄存器的内容通常由系统程序设定，而变址寄存器的内容通常由用户设定；
- 基址寻址常用来实现对用户程序的动态定位，而变址寻址常用于数组处理及串操作。

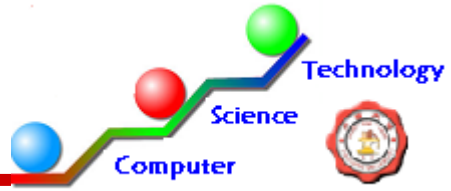
寻址方式—段寻址

❒ 段寻址方式

- ❍ 用于地址长度超过机器字长的场合
- ❍ 将与机器字长相等的段地址和段内位移量错位相加，以获得更长的主存地址。



寻址方式—堆栈寻址

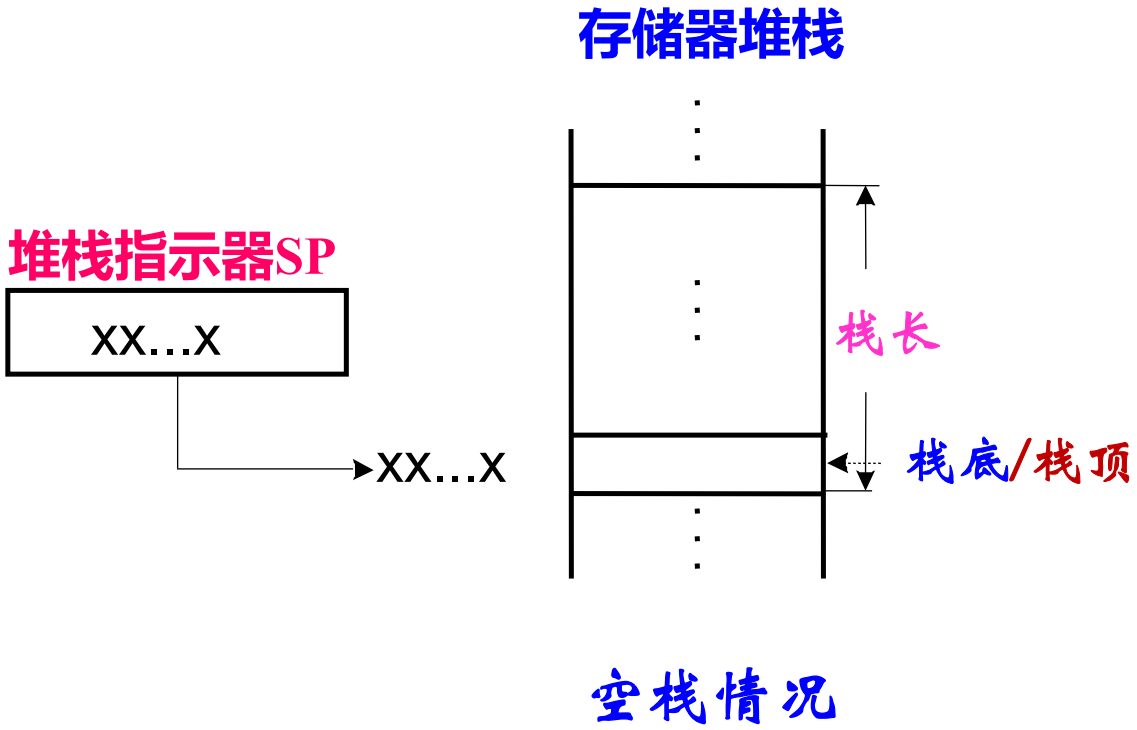


❒ 堆栈

- ❍堆栈是一种后进先出（LIFO）的存储装置
- ❍有寄存器堆栈（硬堆栈）和存储器堆栈（软堆栈）
- ❍存储器堆栈是在主存中开辟一块区域，该区域一端固定，称为栈底；另一端是浮动的，称为栈顶
- ❍栈顶是数据唯一的出入口。堆栈指针（SP）始终指向栈顶。
- ❍栈底与栈顶的地址设定方法
  - ✧栈底设在栈区的低址端，栈顶设在高址端，堆栈向上生长
  - ✧栈底设在栈区的高址端，栈顶设在低址端，堆栈向下生长

寻址方式—堆栈寻址

- 堆栈的实现
  - SP指向栈顶空单元

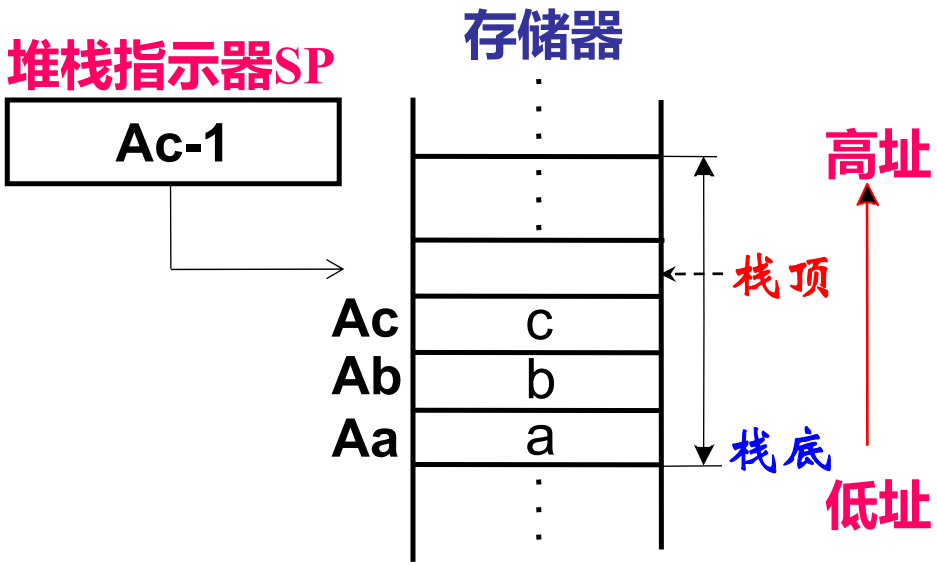


空栈情况

寻址方式—堆栈寻址

❒ 堆栈的实现

❍SP指向栈顶空单元



非空栈情况

堆栈操作

压栈操作（PUSH）：

(SP) ← 数据

SP ← (SP) +1

出栈操作（POP）：

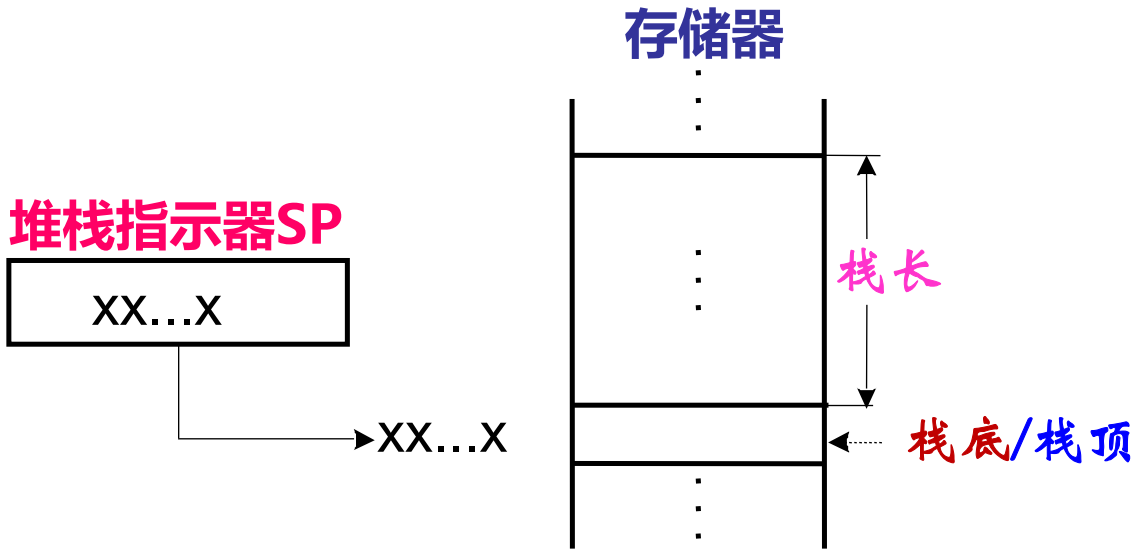
SP ← (SP) - 1

[(SP)] 出栈

寻址方式—堆栈寻址

❒ 堆栈的实现

❍SP指向栈顶非空单元

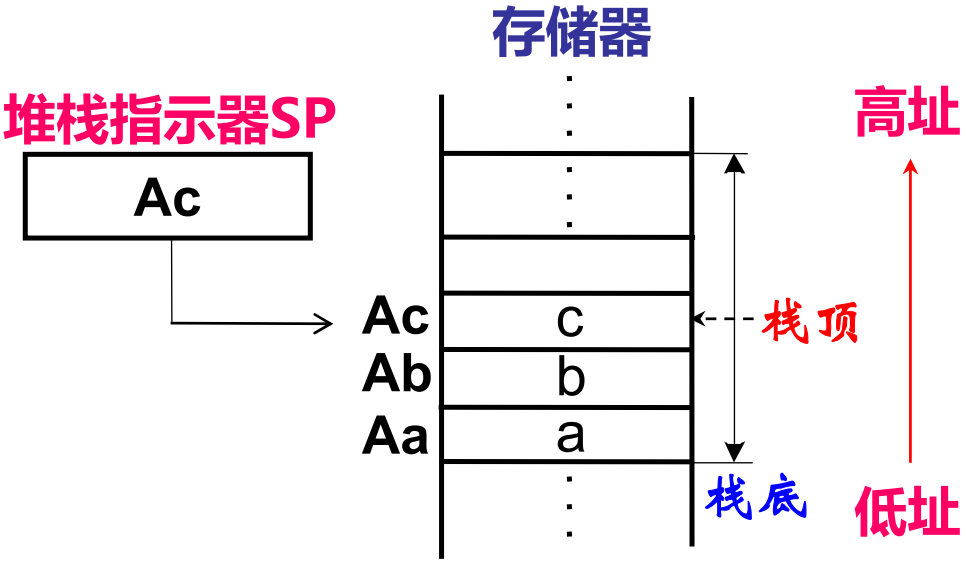


空栈情况

寻址方式—堆栈寻址

❒ 堆栈的实现

❍SP指向栈顶非空单元



非空栈情况

堆栈操作

压栈操作（PUSH）：

SP ← (SP) + 1

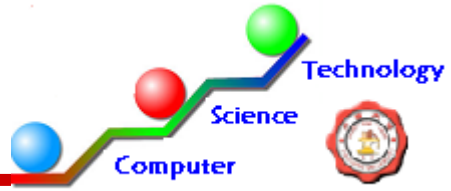
(SP) ← 数据

出栈操作（POP）：

[(SP)] 出栈

SP ← (SP) - 1

寻址方式—复合寻址



❒ 复合寻址方式

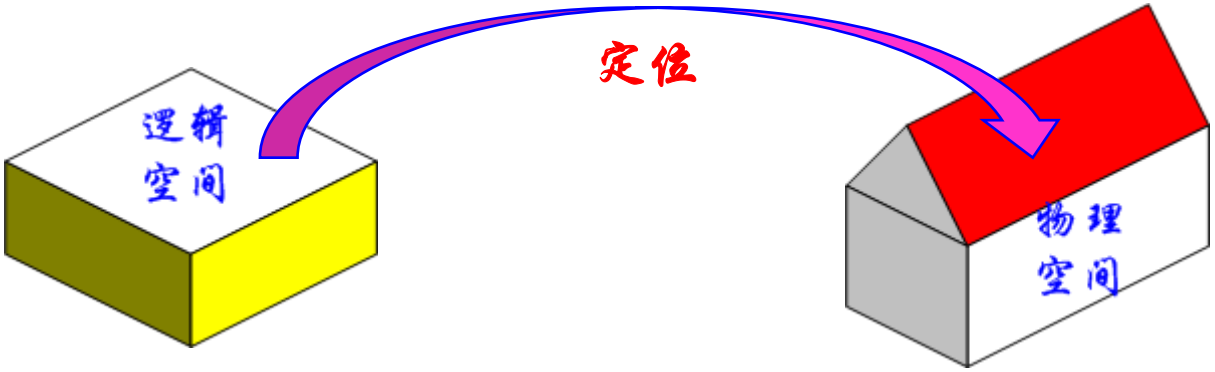
- ❍ 两种以上寻址方式联合使用，称为复合寻址
- ❍ 关键是地址计算的顺序，习惯从名称上加以反映
- ❍ 变址间接寻址：先变址，后间接。即

  $EA=[(Rx)+A]$，　$Rx \leftarrow (Rx)+\Delta$
- ❍ 间接变址寻址：先间接，后变址。即

  $EA=(Rx)+(A)$，　$Rx \leftarrow (Rx)+\Delta$

以上基本寻址方式的介绍是原理性的，不涉及具体机器。这些寻址方式在各种实际机器中实现时有许多变通的命名和规则，在使用时应遵循实际机器汇编语言的具体规定。

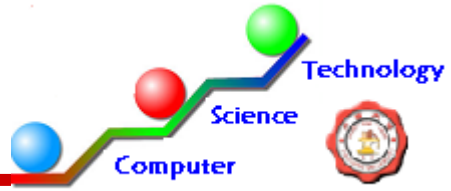
程序定位方式



程序空间的转换

定位方式是指程序中指令和数据的逻辑地址到物理地址的变换时间和实现方式。

程序定位方式（续）

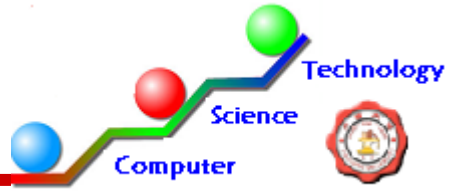


- 程序定位方式
  - 直接定位方式
    - 直接使用主存物理地址来编写或编译程序，即无需地址变换。
    - 将主存物理空间划分为若干个固定且大小相同的分区，为每个任务分配相应的分区。
    - 若程序比较大超出了分配给它的主存物理空间，把它分割成若干个程序段，在程序运行过程中逐段调入主存物理空间，称为“覆盖”。

程序定位方式（续）

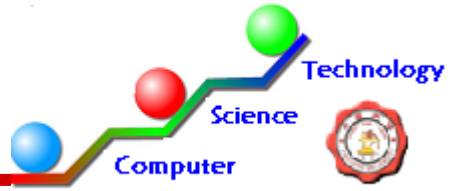
- 静态定位方式
  - 在程序加载到主存时，一次性为指令和数据分配主存物理地址。
  - 由加载程序（操作系统程序）完成定位功能。
  - 若程序比较大超出了分配给它的主存物理空间，采用“覆盖”技术。
  - 程序多次运行时可以装入到不同的主存物理空间，一旦装入，执行期间不能在主存中移动。

程序定位方式（续）



- 动态定位方式
  - 在程序执行过程中，进行逻辑地址到物理地址的转换。
  - 需要硬件支持，采用基址寻址方式实现。
  - 一个程序可以被分配在多个不连续的主存物理空间内。
  - 多个程序可以共享存放在主存中的同一个程序段
  - 支持虚拟存储器，为用户提供一个比实际主存储器的物理空间大得多的逻辑地址空间。

本章作业（总第2次作业）



2.7

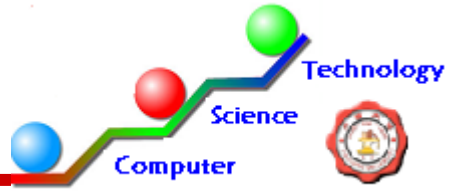
2.13

2.15 (3)改成变址寻址

2.17

注意：下周一提交本次作业！

考研题(I)



- （2/150分）某机器字长16位，主存按字节编址，转移指令采用相对寻址，由两个字节组成，第一字节为操作码字段，第二字节为相对位移量字段。假设取指令时，每取一个字节PC自动加1。若某转移指令所在主存地址为2000H，相对位移量字段的内容为06H，则该转移指令成功转移以后的目标地址是：：

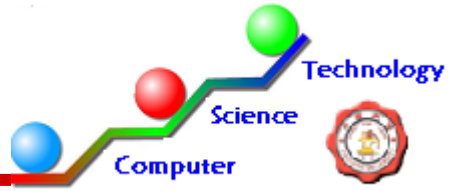
  A. 2006H

  B. 2007H

  C. 2008H

  D. 2009H

考研题(2)



- （2/150分）下列关于RISC的叙述中，错误的是：

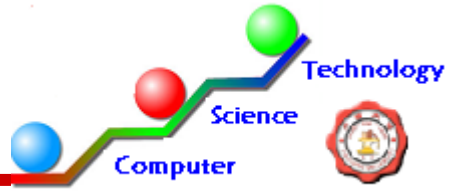
  A. RISC普遍采用微程序控制器

  B. RISC大多数指令在一个时钟周期内完成

  C. RISC的内部通用寄存器的数量比CISC多

  D. RICS的指令数、寻址方式和指令格式种类相对CISC少

考研题(3)

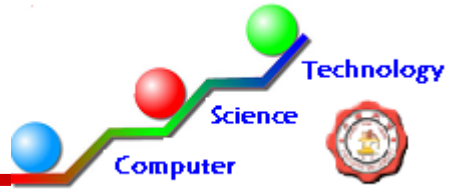


- （2/150分）偏移寻址通过将某个寄存器内容与一个形式地址相加而生成有效地址。下列寻址方式中，不属于偏移寻址方式的是

A．间接寻址　　B．基址寻址

C．相对寻址　　D．变址寻址

考研题(4)



- （2/150分）某机器有一个标志寄存器，其中有进位/借位标态 CF 、零标志 ZF、符号标志 SF 和溢出标志 OF ，条件转移指令 bgt（无符号整数比较大于时转移）的转移条件是

A. $CF+OF=1$　B. $\overline{SF}+ZF=1$　C. $\overline{CF+ZF}=1$　D. $\overline{CF+SF}=1$