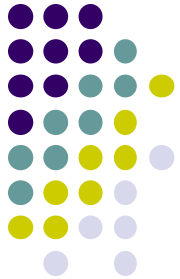


计算机组成与系统结构

第四章 指令系统

吕昕晨

lvxinchen@bupt.edu.cn

网络空间安全学院

第四章 指令系统

- 操作数类型
- 寻址方式
- 典型指令
- ARM汇编语言

一般的数据类型

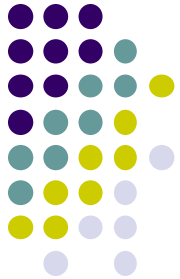
- 地址数据
  - 地址实际上也是一种形式的数据
  - 一般属于无符号整数，计算后确定内存有效地址（寻址）
  - ADD R0, [6]
- 数值数据（第二章）
  - 三种常用
    - 定点数、浮点数、压缩十进制（BCD码）
- 字符数据（第二章）
  - 文本数据或字符串，目前广泛使用ASCII码
- 逻辑数据
  - 一个单元中有几位二进制bit项组成，每个bit的值可以是1或0。当数据以这种方式看待时，称为逻辑性数据。

Pentium数据类型

- Pentium能处理8位、16位、32位、64位等数据类型
  - 数据类型变长
  - 充分利用存储器，地址不需要对齐
  - 32位数据总线

数据类型	说　明
常规	字节、字(16 位)、双字(32 位)和四字(64 位)，可位于任意存储位置上
整数	字节、字、双字、四字中的有符号二进制值，使用 2 的补码表示法
序数	字节、字、双字、四字中的无符号整数
未压缩的 BCD	范围 0～9 的 BCD 数字表示，每字节一个数字
压缩的 BCD	每字节表示两个 BCD 数字，值是 0～99
近指针	表示段内偏移的 32 位有效地址。用于不分段存储器中的所有指针和分段存储器中的段内访问
位串	一个连续的位序列，每位位置都认为是一个独立的单位。能以任何字节的任何位置开始一个位串，位串最长可有 $2^{32}-1$ 位
字符串	一个连续的字节、字或双字的序列，最长可有 $2^{32}-1$B
浮点数	单精度(32 位)、双精度(64 位)、扩展双精度(80 位)

Power PC 数据类型



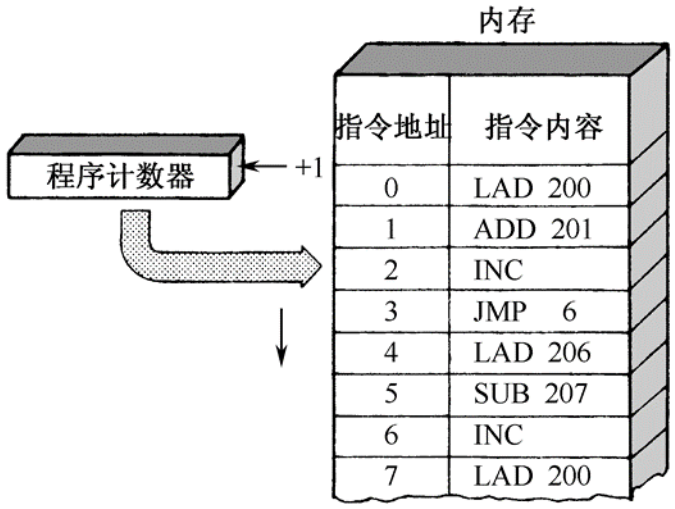
- Power PC：RISC架构
  - 1991年，APPLE、IBM、MOTOROLA提出
  - RISC：指令长度定长（32位）
  - 字长：32位
  - 数据长度变长：8位、16位、32位、64位等
- 数据类型
  - 无符号字节、无符号半字、无符号字：算数、地址
  - 有符号半字、有符号字：算数
  - 无符号双字（地址）
  - 字节串
  - 浮点数：IEEE 754标准

第四章 指令系统

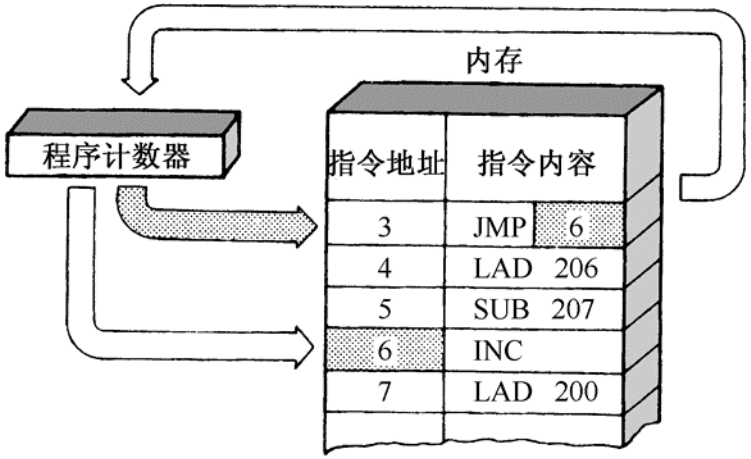
- 操作数类型
- 寻址方式
  - 指令寻址方式与操作数分类
  - 基本寻址方式
  - Pentium寻址方式
- 典型指令
- ARM汇编语言

指令的寻址方式

- 顺序方式
  - PC=PC+1
- 跳跃方式
  - 无条件、条件转移指令
  - 例如，JMP语句



(a) 指令的顺序寻址方式



(b) 指令的跳跃寻址方式

操作数来源分类

- 在指令执行过程中，操作数的来源一般有三个
  - 直接数：由指令中的地址码部分直接给出操作数
    - 虽然简便快捷，但是操作数是固定不变的
  - 寄存器：将操作数存放在CPU内的通用数据寄存器中
    - 很快获取操作数，但是可以存储的操作数的数量有限
  - 内存寻址：将操作数存放在内存的数据区中
    - 有效地址：在指令中直接给出操作数的实际访存地址
    - 形式地址：在指令执行时，将形式地址依据某种方式变换为有效地址再取操作数
- 寻址方式
  - 如何通过形式地址形成操作数有效地址

指令地址形式

- 寻址方式
  - 把操作数的形式地址变换为操作数的有效地址
- 单地址指令格式
  - 操作码OP、变址X、间址I、形式地址A

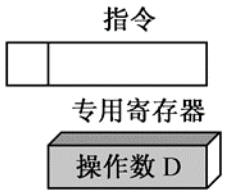
操作码 OP	变址 X	间址 I	形式地址 A

  - 形式地址A：偏移量
  - 变址X、间址I：寻址方式特征位
    - 若无变址、间址要求，形式地址=有效地址
    - 否则需要进行变换（寻址）

第四章 指令系统

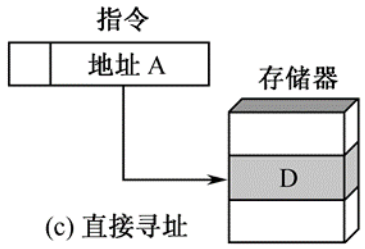
- 操作数类型
- 寻址方式
  - 指令寻址方式与操作数分类
  - 基本寻址方式
  - Pentium寻址方式
- 典型指令
- ARM汇编语言

基本寻址方式

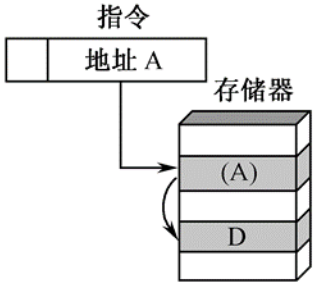


(a) 隐含寻址

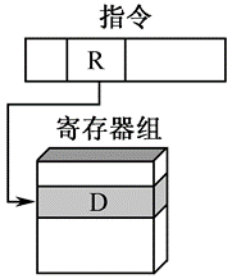
(b) 立即寻址



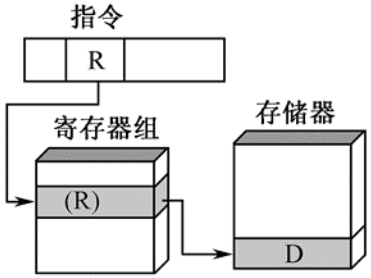
(c) 直接寻址



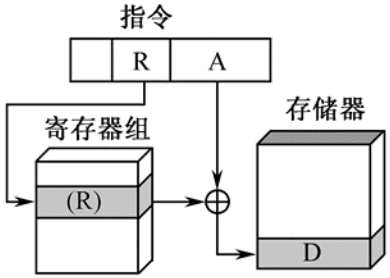
(d) 间接寻址



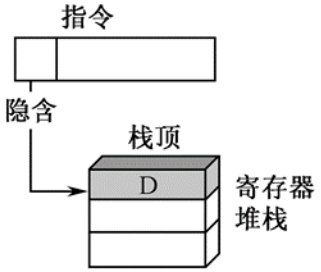
(e) 寄存器寻址



(f) 寄存器间接寻址



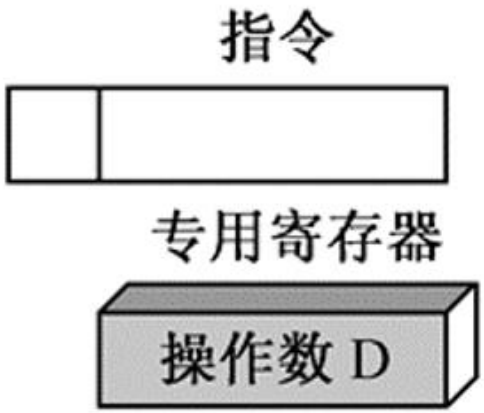
(g) 偏移寻址



(h) 堆栈寻址

隐含寻址

- 指令中隐含着操作数的地址
- 如某些运算，隐含了累加器AC作为源和目的寄存器
  - 如8086汇编中的STC指令
    - 功能：清除进位标志，设置标志寄存器的C为1



(a) 隐含寻址

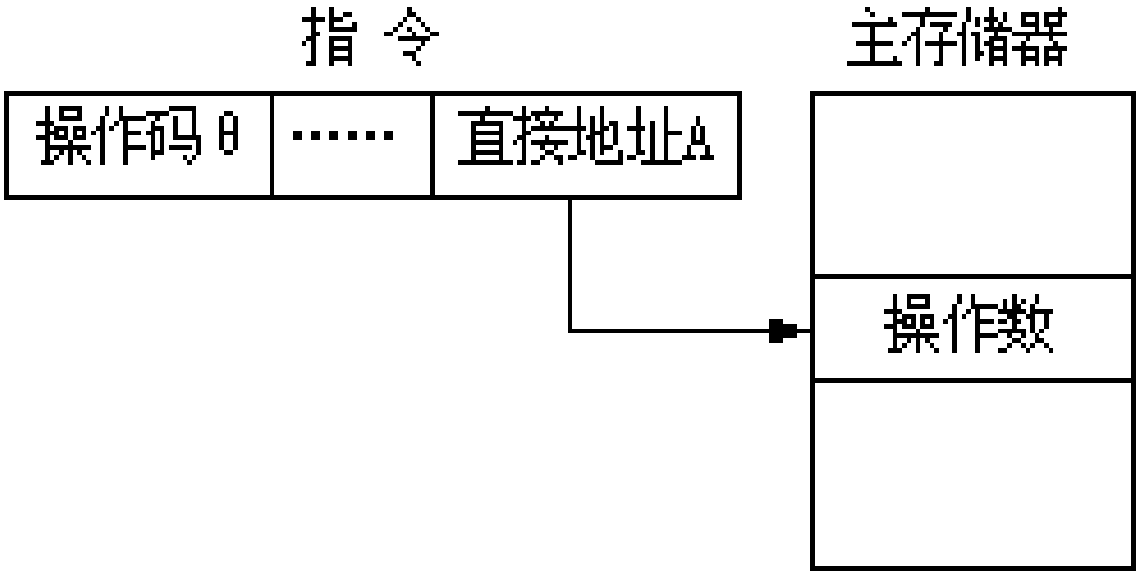
立即寻址

- 指令中在操作码字段后面的部分不是通常意义上的操作数地址，而是操作数本身
- 数据就包含在指令中，只要取出指令，就取出了立即使用的操作数
- 操作数被称为立即数。
- 指令格式：操作码OP 操作数D

(b) 立即寻址

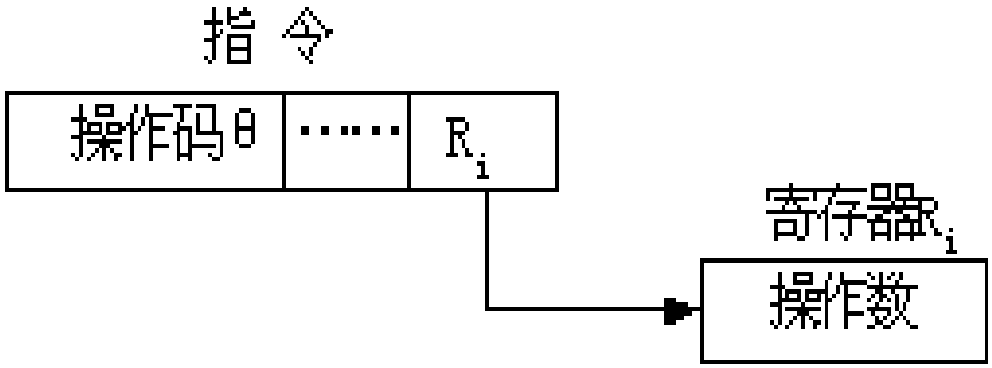
直接寻址

- 指令中地址码字段给出的地址A就是操作数的有效地址EA(Effective Address)，即EA = A
  - 优点：寻址简单
  - 缺点：寻址范围受限



寄存器寻址

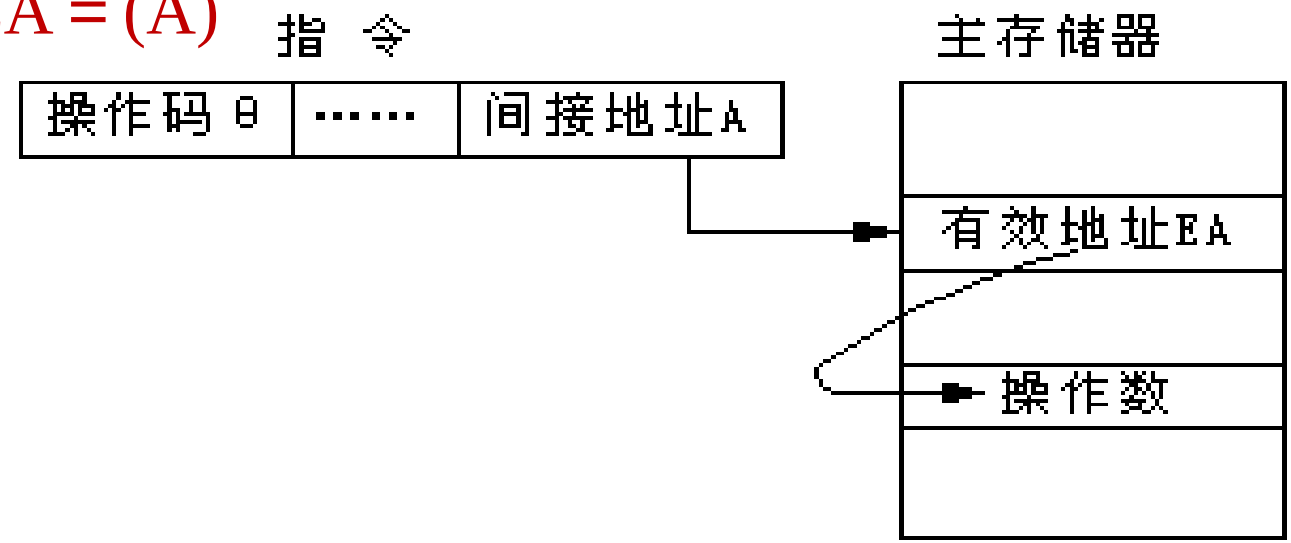
- 在指令的地址码部分给出CPU内某一通用寄存器的编号，指令的操作数存放在相应的寄存器中，即EA=Ri



- 优点
  - 由于寄存器在CPU的内部，指令在执行时从寄存器中取操作数比访问主存要快得多
  - 由于寄存器的数量较少，因此寄存器编号所占位数也较少，从而可以有效减少指令的地址码字段的长度
- 缺点：寄存器数量少，不够灵活

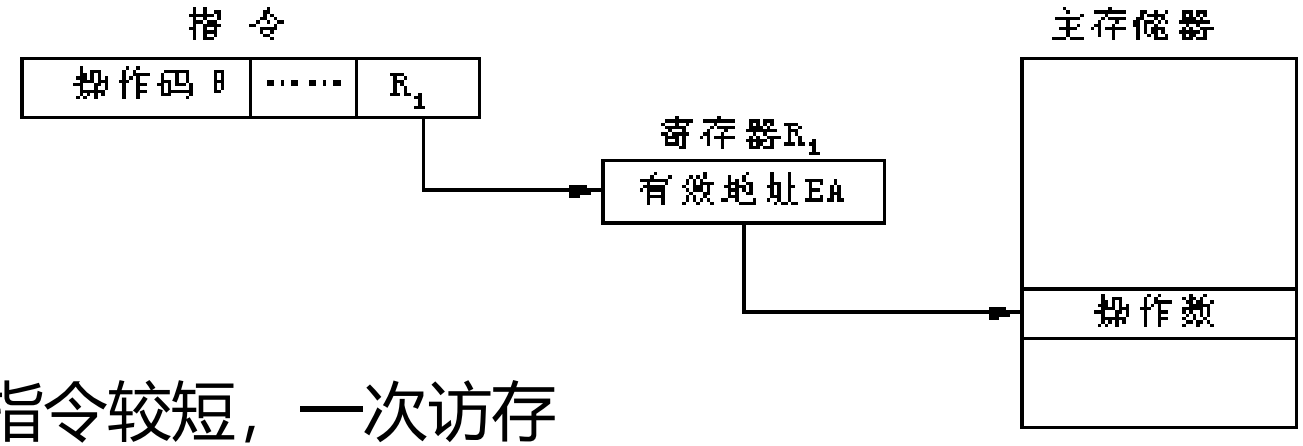
间接寻址

- 间接寻址是相对与直接寻址的方式
- 指令形式地址A不是操作数D的地址，而是给出存放操作数D地址的内存地址
- 即操作数地址的地址
- 缺点：两次访存，影响执行速度
- 记作：EA = (A)



寄存器间接寻址

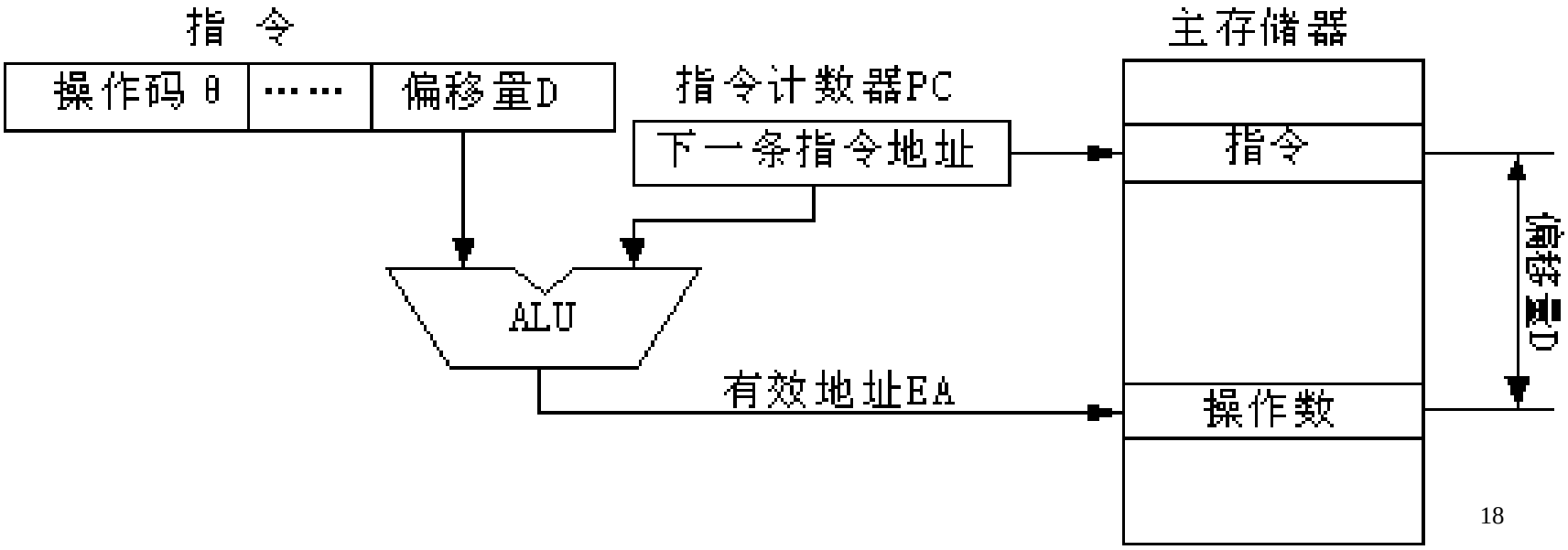
- 寄存器间接寻址：克服间接寻址中多次访存的缺点
- 将操作数放在主存储器中，而操作数的地址放在某一通用寄存器中
- 指令的地址码部分给出该通用寄存器的编号
- 记作：EA=(Ri)



- 优点：指令较短，一次访存
- 是目前在计算机中使用较为广泛的一种寻址方式

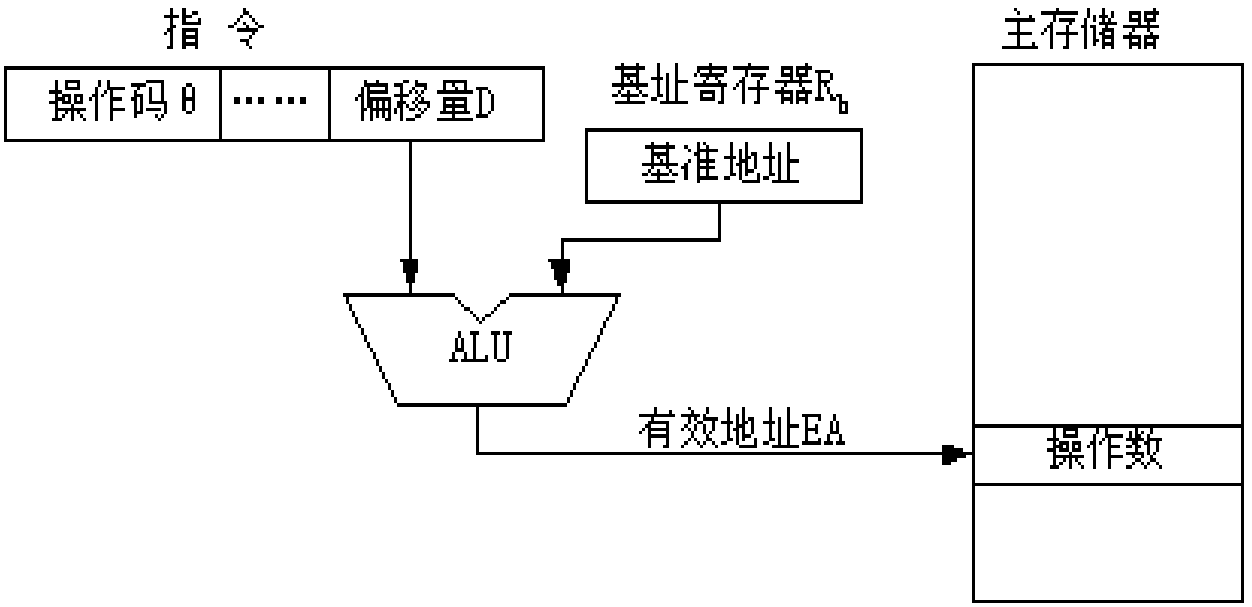
偏移寻址（1）

- 相对寻址方式
  - 由程序计数器PC提供基准地址，而指令的地址码部分给出相对的位移量D，相加后作为操作数的有效地址
  - 记作：EA = (PC) + D



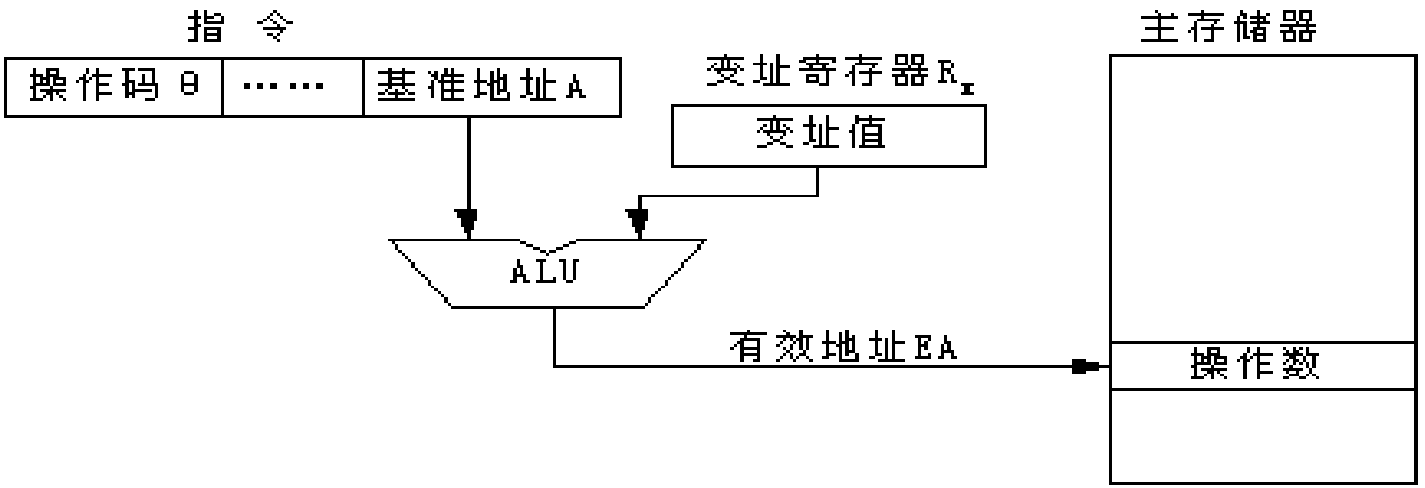
偏移寻址（2）

- 基址寻址方式
  - 参考值发生改变：基址寄存器
  - 优点：基址寄存器的位数可以设置得很长，从而可以在较大的存储空间中寻址



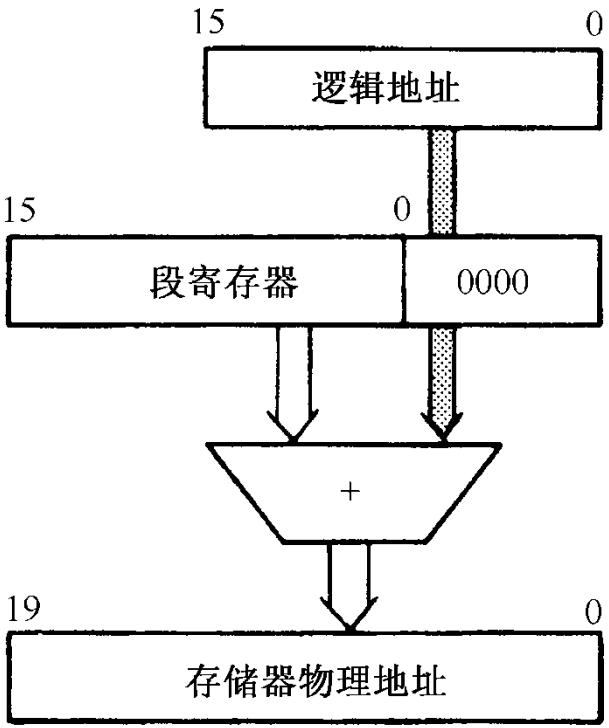
偏移寻址（3）

- 变址寻址
  - 参考值：基准地址A（与基址寻址相反）
  - 给出的基准地址A与CPU内某特定的变址寄存器Rx中的内容相加，以形成操作数的有效地址
  - 例如，对数组进行相加
    - EA=A+(R)，R=R+1



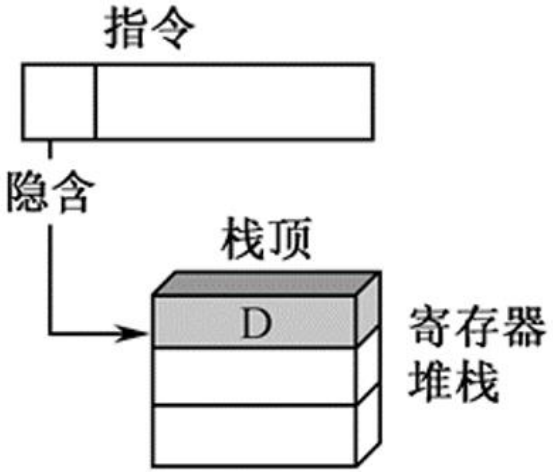
段寻址方式

- 本质：基址寻址
- 例如，8086-微机系统
  - 逻辑地址：16位
  - 物理地址：20位
  - 地址扩展：16位→20位
  - 方式：段寻址
    - 段寄存器：CS、DS等
    - 物理地址形成方式：
      - 段寄存器左移4位
      - 与逻辑地址相加

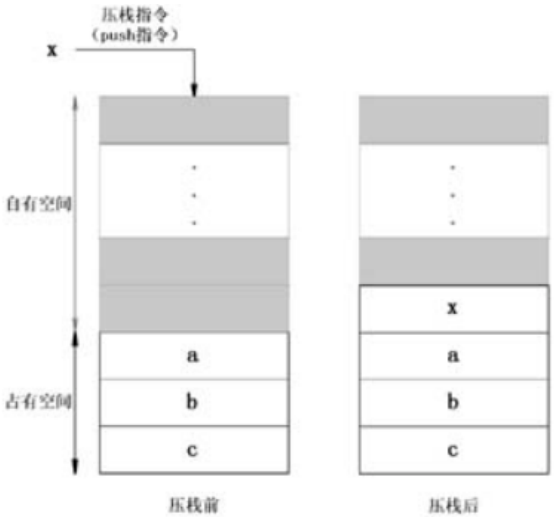


堆栈寻址

- 分类
  - 寄存器堆栈、存储器堆栈两种形式
- 存储原理
  - 先进后出方式
  - 保存现场，函数调用等
- 操作
  - 压栈：PUSH，堆栈指示器-1
  - 出栈：POP，对战指示器+1

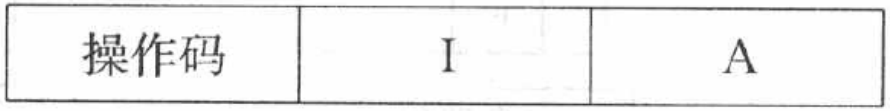


(h) 堆栈寻址



寻址方式组合

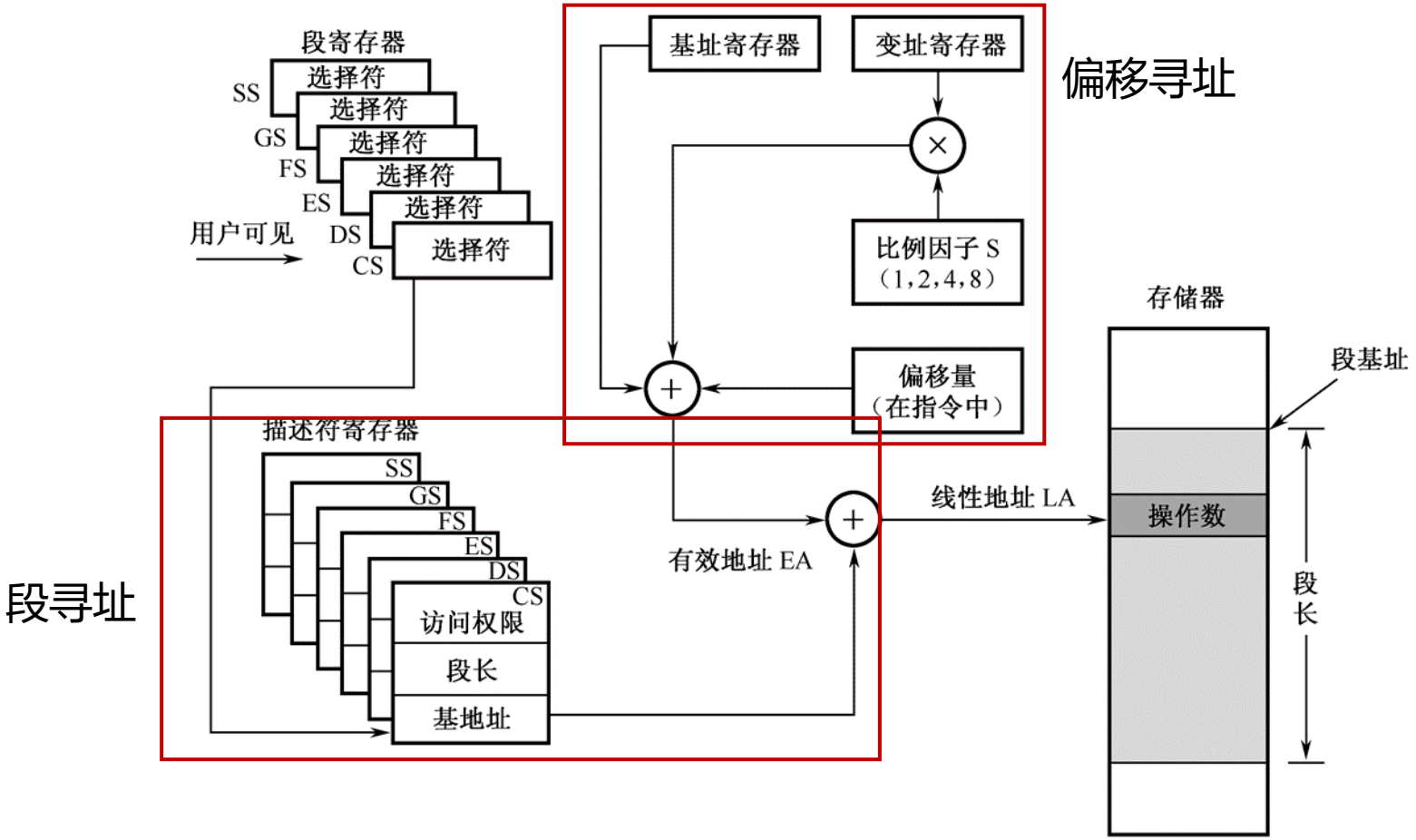
- 不同的指令系统采用不同的方式指定寻址方式
  - 有些指令固定使用某种寻址方式
  - 有些指令则允许使用多种寻址方式
    - 在指令中加入寻址方式字段指明
    - 对不同的寻址方式分配不同的操作码而把它们看作是不同的指令
    - 有些指令系统会把常见的寻址方式组合起来，构成更复杂的符合寻址方式
  - 例如，间接寻址方式
    - 特征位I=0（直接寻址）
    - 特征位I=1（间接寻址）



第四章 指令系统

- 操作数类型
- 寻址方式
  - 指令寻址方式与操作数分类
  - 基本寻址方式
  - Pentium寻址方式
- 典型指令
- ARM汇编语言

Pentium寻址方式

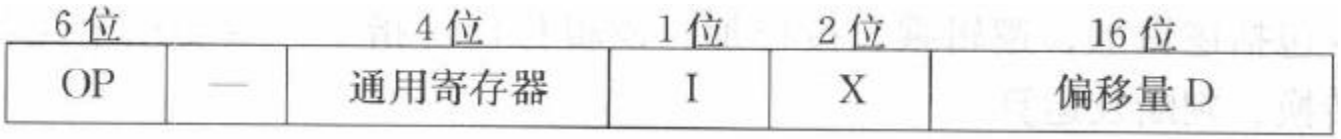


Pentium寻址方式

方式	算法
立即	操作数=A
寄存器	LA=R
偏移量	LA=(SR)+A
基址	LA=(SR)+(B)
基址带偏移量	LA=(SR)+(B)+A
比例变址带偏移量	LA=(SR)+(I)×S+A
基址带变址和偏移量	LA=(SR)+(B)+(I)+A
基址带比例变址和偏移量	LA=(SR)+(B)+(I)×S+A
相对	LA=(PC)+A

寻址方式例题

[例4] 一种二地址RS型指令的结构如下：

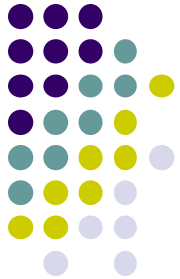


其中I为间接寻址标志位，X为寻址模式字段，D为偏移量字段。通过I，X，D的组合，可构成如下寻址方式：

1）直接寻址
2）相对寻址
3）变址寻址
4）寄存器间接寻址
5）间接寻址
6）基址寻址

寻址方式	I	X	有效地址E算法	说明
(1)	0	00	E=D	
(2)	0	01	E=(PC) ± D	PC为程序计数器
(3)	0	10	E=(R_2) ± D	R_2为变址寄存器
(4)	1	11	E=(R_3)	
(5)	1	00	E=(D)	
(6)	0	11	E=(R_1) ± D	R_1为基址寄存器

第四章 指令系统

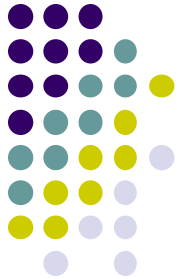


- 操作数类型
- 寻址方式
- 典型指令
- ARM汇编语言

指令的分类

- 数据传送类指令
  - 一般传送指令：　MOV　AX，BX
  - 数据交换指令：　XCHG
  - 堆栈操作指令：　PUSH，POP
- 运算类指令
  - 算术运算指令：加、减、乘、除以及加1、减1、比较
  - 逻辑运算指令：
  - 移位指令
- 程序控制类指令
  - 程序控制类指令用于控制程序的执行方向，并使程序具有测试、分析与判断的能力。
- 输入和输出指令
- 字符串处理指令、特权指令（多任务系统）、其他指令

复杂指令系统（CISC）



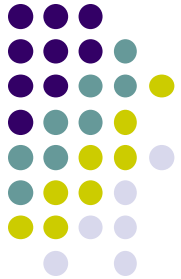
- 2-8定律
  - CISC中大约有20%的指令使用频率高，占据了80%的处理机时间，而有80%的不常用指令只占用处理机的20%时间。
- VLSI技术发展引起的问题
  - VLSI工艺要求规整性，而大量复杂指令控制逻辑极其不规整，给VLSI工艺造成了很大的困难
  - 现在用微程序实现复杂指令与用简单指令组成的子程序相比，没有多大的区别
- CISC中，通过增强指令系统的功能，简化了软件，但增加了硬件的复杂程度，在计算机体系结构设计中，软硬件的功能分配必须恰当

精简指令系统（RISC）

- 特点（采用流水线技术）
  - 简单而统一格式的指令译码
  - 大部分指令可以单周期执行
  - 只有LOAD/STORE可以访问存储器
  - 简单的寻址方式

指令类型	操作名称		说　明
数据传送	MOV	传送	由源向目标传送字，源和目标是寄存器
	STO	存数	由 CPU 向存储器传送字
	LAD	取数	由存储器向 CPU 传送字
	EXC	交换	源和目标交换内容
	CLA	清零	传送全 0 字到目标
	SET	置 1	传送全 1 字到目标
	PUS	进栈	由源向堆栈顶传送字
	POP	退栈	由堆栈顶向目标传送字
算术运算	ADD	加法	计算两个操作数的和
	SUB	减法	计算两个操作数的差
	MUL	乘法	计算两个操作数的积
	DIV	除法	计算两个操作数的商
	ABS	绝对值	以其绝对值替代操作数
	NEG	变负	改变操作数的符号
	INC	增量	操作数加 1
	DEC	减量	操作数减 1

第四章 指令系统



- 操作数类型
- 寻址方式
- 典型指令
- ARM汇编语言

ARM汇编语言

- 汇编语言是计算机机器语言（二进制指令代码）进行符号化的一种表示方法，每一个基本汇编语句对应一条机器指令。
- 表4.11列出了嵌入式处理机ARM的汇编语言。其中操作数使用16个寄存器

指令类别	指令	示例	含义	说明
算术运算	加	ADD r1,r2,r3	r1=r2+r3	三寄存器操作数
	减	SUB r1,r2,r3	r1=r2−r3	三寄存器操作数
数据传送	取数(字)至寄存器	LDR r1,[r2,#20]	r1=存储单元[r2+20]	内存单元至寄存器字传送
	自寄存器存数(字)	STR r1,[r2,#20]	存储单元[r2+20]=R1	寄存器至内存单元字传送
	取半字数至寄存器	LDRH r1,[r2,#20]	r1=存储单元[r2+20]	内存单元至寄存器半字传送
	取半字带符号数至寄存器	LDRHS r1,[r2,#20]	r1=存储单元[r2+20]	内存单元至寄存器半字带符号数传送
	自寄存器存半字数	STRH r1,[r2,#20]	存储单元[r2+20]=R1	寄存器至内存单元半字传送
	取字节数至寄存器	LDRB r1,[r2,#20]	r1=存储单元[r2+20]	内存单元至寄存器字节传送
	取字节带符号数至寄存器	LDRBS r1,[r2,#20]	r1=存储单元[r2+20]	内存单元至寄存器字节带符号数传送
	自寄存器存字节数	STRB r1,[r2,#20]	存储单元[r2+20]=R1	寄存器至内存单元字节传送
	交换	SWP r1,[r2,#20]	R1=存储单元[r2+20], 存储单元[r2+20]=r1	自动交换存储单元和寄存器
	传送	MOV r1,r2	r1=r2	寄存器间拷贝

汇编语言特点

- 在进行汇编语言程序设计时，可直接使用英文单词或其缩写表示指令，使用标识表示数据或地址，从而有效地避免了记忆二进制的指令代码
- 不用由程序设计人员对指令和数据分配内存地址，直接调用操作系统的某些程序段完成输入输出
- 用编辑程序建立好的汇编语言源程序，需要经过系统软件中的“汇编器”翻译为机器语言程序之后，才能交付给计算机硬件系统去执行

ARM汇编语言例题

例4：将ARM汇编语言翻译成机器语言。已知5条ARM指令格式译码如下表所示：

指令名称	cond	F	I	opcode	S	Rn	Rd	operand 2
ADD（加）	14	0	0	4	0	reg	reg	reg
SUB（减）	14	0	0	2	0	reg	reg	reg
ADD（立即数加）	14	0	1	4	0	reg	reg	constant（12 位）
LDR（取字）	14	1	—	24	—	reg	reg	address（12 位）
STR（存字）	14	1	—	25	—	reg	reg	address（12 位）

设r3寄存器中保存数组A的基值，h放在寄存器r2中。C语言程序语句 A[30]=h+A[30] 可编译成如下3条汇编语句指令：

```
LDR r5 , [r3, #120]        ;寄存器r5中获得A[30]
ADD r5 , r2 , r5 ,         ;寄存器r5中获得h+A[30]
STR r5 , [r3, #120]        ;将h+A[30]存入到A[30]
```

请问这3条汇编语言指令的机器语言是什么？

ARM汇编语言例题

汇编语言：**LDR r5 , [r3, #120]；ADD r5 , r2 , r5；STR r5 , [r3, #120]；**
先对照写十进制，再写二进制

指令名称	cond	F	I	opcode	S	Rn	Rd	operand 2
ADD（加）	14	0	0	4	0	reg	reg	reg
SUB（减）	14	0	0	2	0	reg	reg	reg
ADD（立即数加）	14	0	1	4	0	reg	reg	constant（12 位）
LDR（取字）	14	1	—	24	—	reg	reg	address（12 位）
STR（存字）	14	1	—	25	—	reg	reg	address（12 位）

	cond	F	opcode			Rn	Rd	offset
			I	opcode	S			operand
十进制	14	1	24			3	5	120
	14	0	0	4	0	2	5	5
	14	1	25			3	5	120
二进制	1110	1	11000			0011	0101	0000 1111 0000
	1110	0	0	100	0	0010	0101	0000 0000 0101
	1110	1	11001			0011	0101	0000 1111 0000

第四章作业

- 4-3，4-4，4-5
- 4-6，4-8，4-12