北京郵電大學
Beijing University of Posts and Telecommunications

Me rak
hacking securily

汇编语言与逆向工程
Assembly Language and Software Reverse Engineering

北京邮电大学
付俊松

2024年春季

第二章 汇编语言

- 1.CPU简介
- 2.寄存器
- 3.x86指令集
- 4.寻址方式

（1）CPU简介

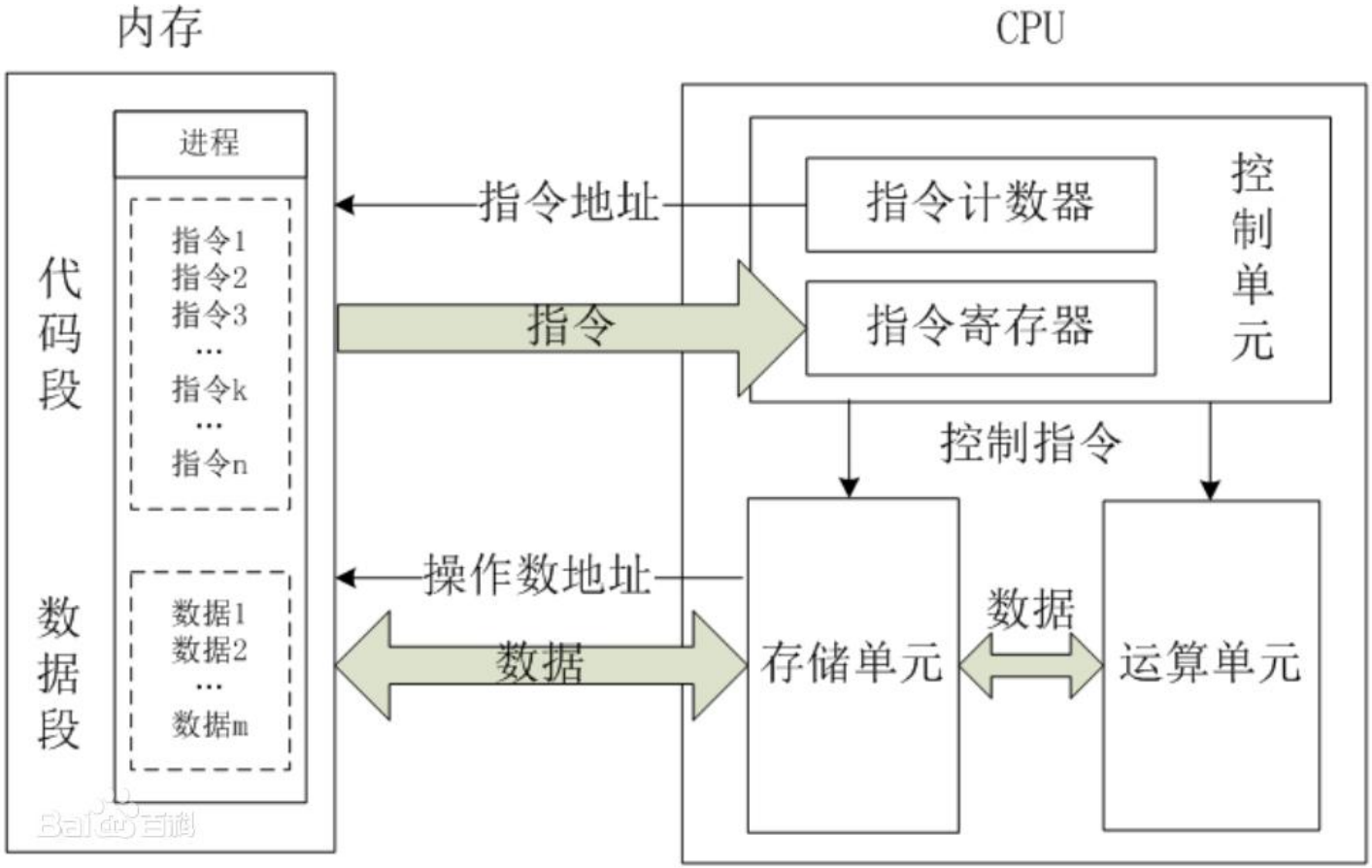
- 中央处理器（Central Processing Unit, CPU）
  - 结构密集
    - 金属氧化物半导体场效应晶体管（Metal Oxide Semiconductor Field-Effect Transistor, MOSFET）
    - i7处理器晶体管达到10亿级别（10亿-30亿）
  - 科技密集
    - CPU设计生产难度极大，生产每片CPU的工艺流程达到1000-3000个步骤
    - 目前，我国国产某型号CPU性能大约为主流Intel芯片的8%-10%
  - 经济密集
    - 一颗i9普通处理器售价大致3000-20000元，CPU成本占电脑整机成本20%左右

（1）CPU简介

- CPU的基本组成部件
  - 时钟（Clock）
  - 算数逻辑单元（Arithmetic and Logic Unit, ALU）
  - 控制单元（Control Unit, CU）
  - 存储单元（Memory Unit）

（1） CPU简介

CPU基本结构图



（1） CPU简介

- 时钟
  - 时钟是整个计算机运转的“节拍器”，机器指令的最小执行时间就是一个时钟周期
    - 注：精准的时钟是所有数字信号的基础，模拟信号向数字信号的转换过程，就是在时钟指导下的一个采样过程（Sampling），时钟的偏移将是数字信号失准
  - CPU的主频即为时钟频率，对CPU的整体性能具有直接影响
    - 注：CPU的位数也是影响CPU性能的主要因素之一

（1） CPU简介

- 算数逻辑单元
  - 算数逻辑单元，即ALU，顾名思义就是执行各种算数运算和逻辑运算
  - 随着各类运算任务的增加，出现了专用计算单元如浮点计算单元

（1） CPU简介

- 控制单元
  - CU是控制单元，是CPU的指挥中心，协调、控制各个指令执行的顺序，主要由下列部件组成：
    - 指令寄存器（Instruction Register）
    - 指令译码器（Instruction Decoder）
    - 指令控制器（Operation Controller）
    - 指令指针寄存器（Extended Instruction Pointer）

（1） CPU简介

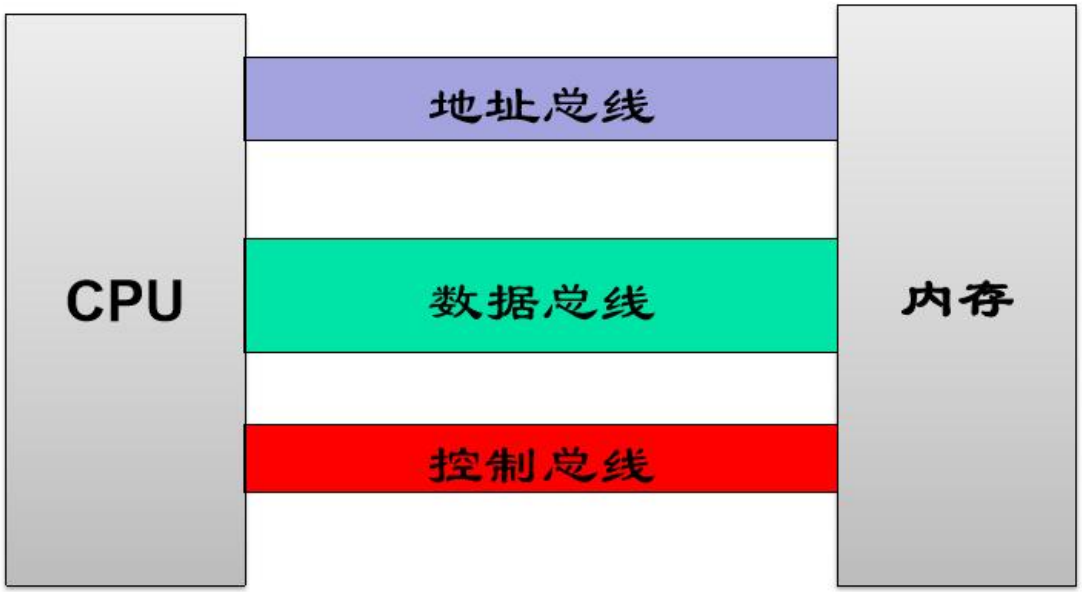
- 存储单元
  - 存储单元是CPU中暂时存放数据的地方，包括等待处理数据或数据处理结果等
    - 片上缓存或片内缓存，即Cache On-Chip，访问速度和存储容量均介于内存和寄存器之间的高速存储器件
    - 寄存器组，即Registers，最接近ALU的存储器件，访问速度最快，由于芯片面积和集成度限制，存储容量最小
      - 由于寄存器在芯片中的特殊地位，几乎所有汇编指令的执行均需要寄存器的协助，可以说寄存器是汇编语言的一部分

（1） CPU简介

- 寄存器
  - 寄存器的作用是临时存储数据和地址，供CPU操作，包括
    - 8个通用寄存器（EAX,EBX,ECX,EDX,EDI,ESI,ESP,EBP）
    - 1个指令指针寄存器（EIP），EIP始终指向下一条待执行指令的地址
    - 一个CPU状态寄存器（EFLAGS）
    - 6个段寄存器

(1) CPU简介

- 总线
  - 总线在物理上是若干根用于连接其他芯片的导线。
  - 在逻辑上分为地址总线、数据总线、控制总线。



（1） CPU简介

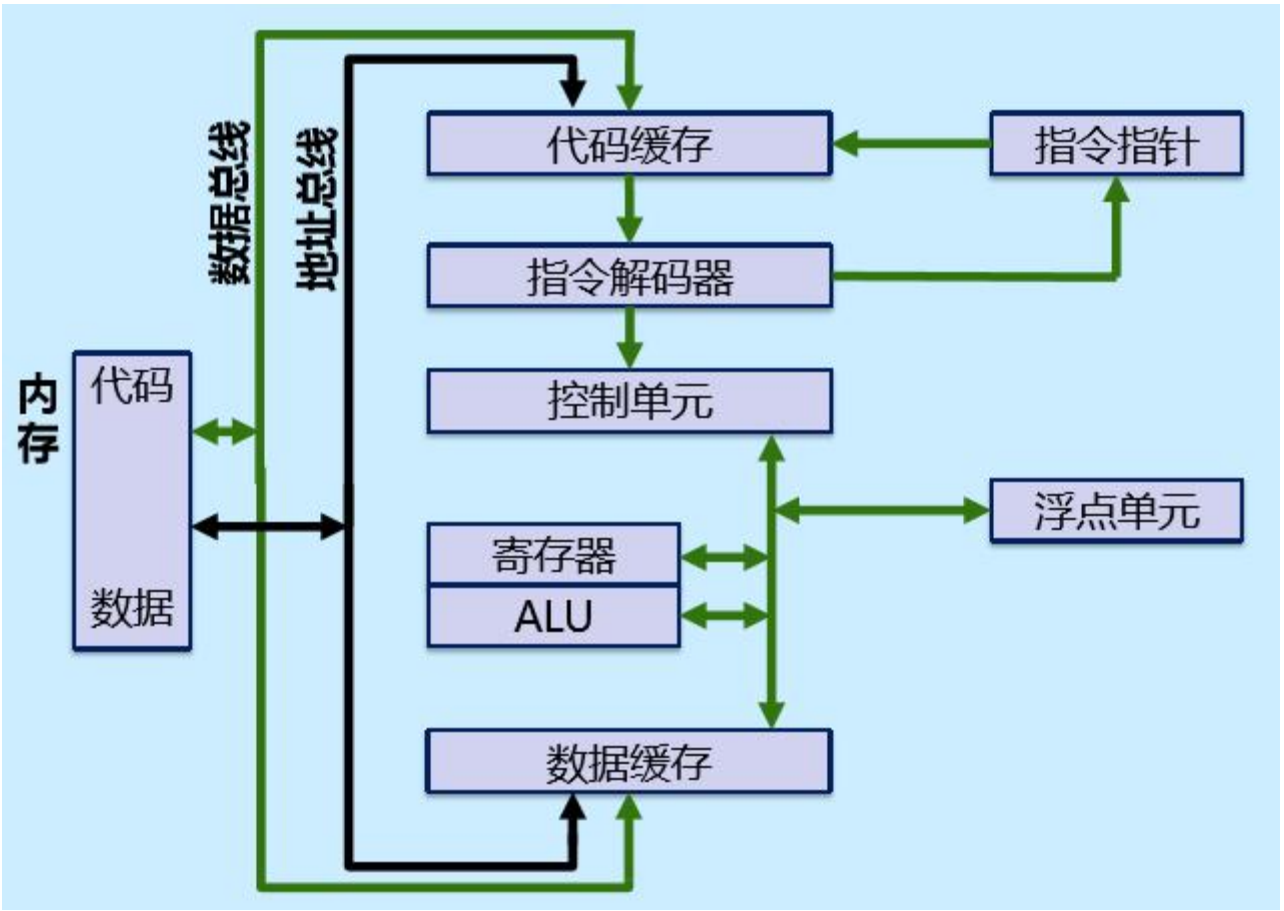
- 地址总线
  - CPU要将内存中的指令和数据取出，就必须访问内存中的地址，当CPU访问内存中的某个地址时，地址线上就会保持着那个地址值。
  - 地址总线能表示多少个不同的数值，这个CPU就能够访问多少内存地址
    - 宽度为N的地址线，它能够寻址的空间为0 ~ 2^N-1

（1） CPU简介

- 数据总线
  - 在CPU和内存中传输指令和数据
- 控制总线
  - 控制总线是不同控制线的集合，是一个总称
  - 有多少条控制总线，就意味着CPU能够对外界提供多少种控制
  - 控制总线的宽度决定了CPU控制外界设备的能力

（1） CPU简介

- CPU基本工作原理图



（1） CPU简介

- SEQ处理器（SEQuential Processor）结构
  - 取指令（Fetch）
  - 指令解码（Decode）
  - 指令执行（Execute）
  - 内存访问（Memory）
  - 结果回写（Write back）
  - 指令指针寄存器更新（PC update）
    - 注：现代处理器为了提高指令执行效率，通常采用流水线结构执行指令（Pipelined implementation）

第二章 汇编语言

- 1.CPU简介
- 2.寄存器
- 3.x86指令集
- 4.寻址方式

(2) 寄存器

- 什么是寄存器
- 寄存器分类
- 通用寄存器
- 段寄存器
- 程序状态与控制寄存器
- 指令指针寄存器

(2) 寄存器 — 什么是寄存器

- 什么是寄存器
  - 寄存器(Register)是中央处理器(CPU，Central Processing Unit)内部的组成部分
  - 寄存器是有限存储容量的高速存储部件，它们可用来暂存指令、数据和地址
  - 寄存器是计算机系统结构下存储层次的最顶端，也是系统中操作数据的最快速路径

(2) 寄存器

- 什么是寄存器
- 寄存器分类
- 通用寄存器
- 段寄存器
- 程序状态与控制寄存器
- 指令指针寄存器

（2）寄存器 — 寄存器分类

- IA-32架构提供了16个基本程序执行寄存器，用于系统和应用程序编程
  - IA-32是1985年的intel 80386到奔腾系列处理器所沿用的intel处理器架构
  - 16个寄存器可以被分成以下四类：
    - 8个通用寄存器(General-purpose registers)
    - 6个段寄存器(Segment registers)
    - 程序状态与控制寄存器(EFLAGS)
    - 指令指针寄存器(EIP register)

(2) 寄存器

- 什么是寄存器
- 寄存器分类
- 通用寄存器
- 段寄存器
- 程序状态与控制寄存器
- 指令指针寄存器

(2) 寄存器 — 通用寄存器

- 通用寄存器
  - 32位CPU通用寄存器共有8个：EAX，EBX，ECX，EDX，ESI，EDI，EBP，ESP
  - 它们可以用于传送和暂存以下数据
    - 逻辑和算术运算的操作数
    - 用于地址计算的操作数
    - 内存指针

（1）寄存器 — 通用寄存器

- 第1- 4个寄存器
  - EAX： 累加寄存器， 是操作数和结果数据的累加器
  - EBX： 基址寄存器， 指向DS段中数据的指针
  - ECX： 计数寄存器， 是字符串和循环操作的计数器
  - EDX： 数据寄存器， I/O指针
    - 上面4个寄存器主要用于算术运算（ADD/SUB/XOR/OR等）指令中，常用来保存常量与变量的值

(2) 寄存器 — 通用寄存器

- 变址寄存器（第5-6个）
  - ESI：(字符串操作源指针)源变址寄存器
  - EDI：(字符串操作目的指针)目的变址寄存器
    - ESI和EDI与特定的串操作指令（MOVS/LODS/STOS）一起使用，在字符串操作的时候用的比较多
    - 变址寄存器存放存储单元在段内的偏移量，用它们可实现多种存储器操作数的寻址方式，为通过多种方式访问存储单元提供便利

(2) 寄存器 — 通用寄存器

- 指针寄存器（第7-8个）
  - **ESP**：栈顶指针寄存器，用于存放当前堆栈的栈顶地址，专门用作堆栈指针，不可作为一般通用寄存器使用
  - **EBP**：栈底指针寄存器（基址指针寄存器），表示栈区域的基地址，永远指向当前函数栈的栈底位置，不可作为一般通用寄存器使用

(2) 寄存器 — 通用寄存器

- EAX、EBX、ECX、EDX四个通用寄存器
  - 为了实现与16位CPU(8086CPU)的兼容（即对低16位数据的存取），这些低 16位寄存器分别命名为AX、BX、CX、DX
  - 对低16位数据的存取，不会影响高16位的数据

General-Purpose Registers

31 16	15 8	7 0	16-bit	32-bit
	AH	AL	AX	EAX
	BH	BL	BX	EBX
	CH	CL	CX	ECX
	DH	DL	DX	EDX
	BP			EBP
	SI			ESI
	DI			EDI
	SP			ESP

(2) 寄存器 — 通用寄存器

- 为了兼容8086CPU的上一代CPU中8位寄存器，同样，8086CPU的AX、BX、CX、DX这四个16位寄存器又可分为两个独立使用的8位寄存器来用：
  - AX可分为AH和AL；
  - BX可分为BH和BL；
  - CX可分为CH和CL；
  - DX可分为DH和DL。

General-Purpose Registers

31 16	15 8	7 0	16-bit	32-bit
	AH	AL	AX	EAX
	BH	BL	BX	EBX
	CH	CL	CX	ECX
	DH	DL	DX	EDX
	BP			EBP
	SI			ESI
	DI			EDI
	SP			ESP

(2) 寄存器 — 通用寄存器

- EAX、AX、AH(AL)之间的关系
  - EAX为32位寄存器，AX为16位寄存器，AH(AL)为8位寄存器
  - 若想存储4个字节(DWORD，32位)的数据，就使用EAX
  - 若只想使用2个字节(WORD，16位)，使用EAX的低16位部分AX就可以
  - AX又可分为高8位的AH寄存器和低8位的AL寄存器
  - 每个寄存器都有自己的名称，可独立存取，程序员可利用数据寄存器的这种“可分可合”的特性，灵活地处理字/双字/字节的信息

(2) 寄存器 — 通用寄存器

- 指针寄存器EBP、ESP和变址寄存器ESI、EDI
  - 有类似的低16位寄存器BP、SP、SI、DI
  - 但是它们不可分割成8位寄存器

General-Purpose Registers

31 16	15 8	7 0	16-bit	32-bit
	AH	AL	AX	EAX
	BH	BL	BX	EBX
	CH	CL	CX	ECX
	DH	DL	DX	EDX
	BP			EBP
	SI			ESI
	DI			EDI
	SP			ESP

(2) 寄存器

- 什么是寄存器
- 寄存器分类
- 通用寄存器
- 段寄存器
- 程序状态与控制寄存器
- 指令指针寄存器

(2) 寄存器 — 段寄存器

- 段寄存器产生的历史背景（可忽略）
  - 8086CPU有20位地址总线，可以传送20位地址，最大寻址空间为1MB()
  - 而8086CPU是16位结构的CPU
    - 也就是说在8086内部，能够一次性处理、传输、暂时存储的信息的最大长度是16位，表现出的寻址能力只有64KB()
    - 为了克服CPU寻址能力不足的缺点，Intel公司最终决定在CPU内部采用两个16位地址合成的方式，形成一个20位的物理地址

(2) 寄存器 — 段寄存器

- 物理地址=段基址(段地址*10H）+偏移地址
  - 8086的物理地址是20位的，而段寄存器只有16位，在合成物理地址时需要先将段寄存器中16位的段地址左移四位得到一个20位的段地址，也就是在段地址低位补四个0，相当于乘了16进制的10，这就是段地址*10H
  - 段基地址不能随意乱取，通常都是“小段的起始地址”
    - “小段”即是在物理地址中从00000H开始，每16个字节而划分的，那么整个物理地址空间就可以划分为64K个小段，且首地址的最后四位均为0（用二进制表示时），所以是16的倍数。

(2) 寄存器 — 段寄存器

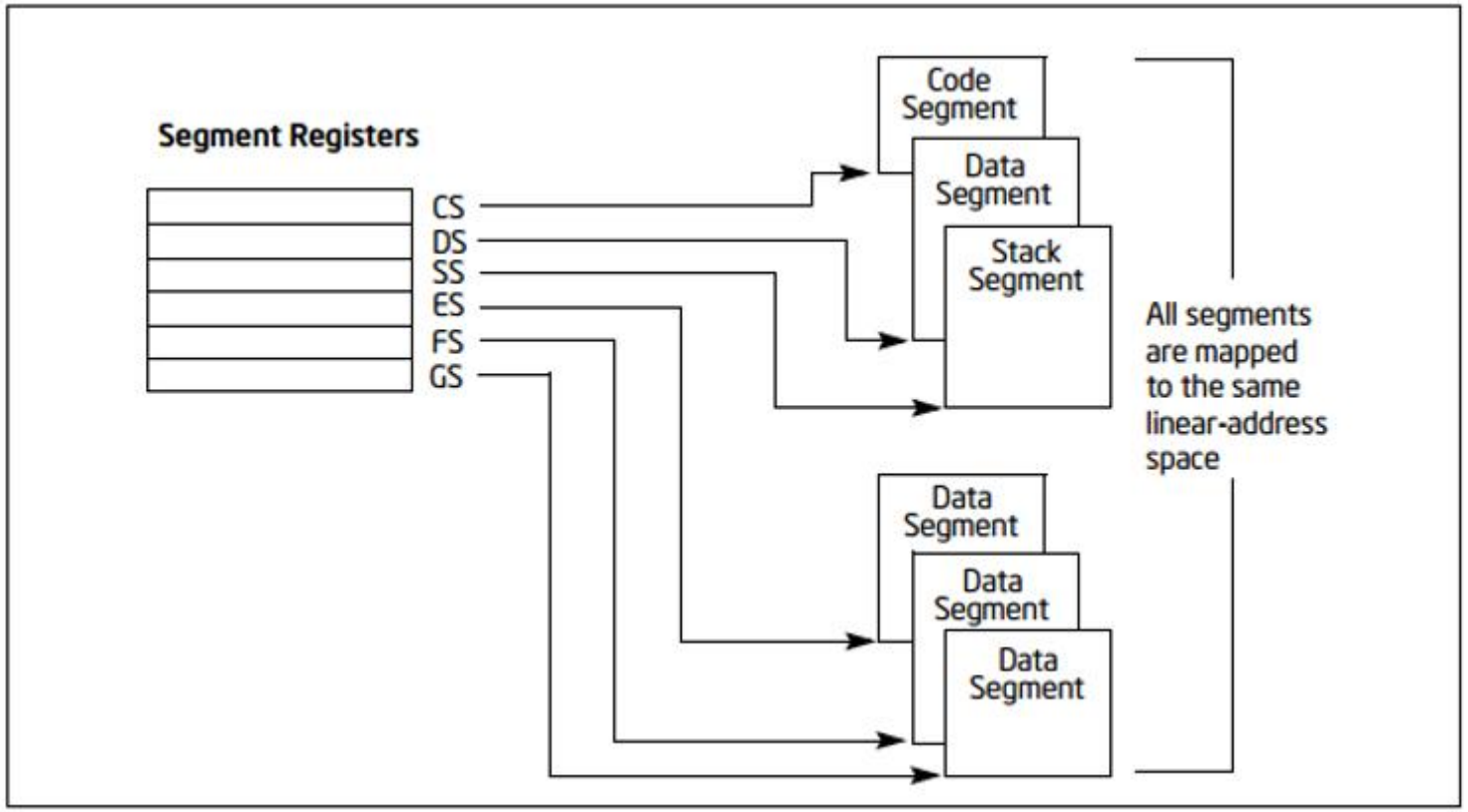
- 8086CPU使用以上方式给出内存单元的物理地址，可以用分段的方式管理内存
  - 将内存分为很多段，每一段都有一个段基址，这个段基址自然是一个20位的内存地址
  - 分段的起源和段寄存器的诞生
    - 由于8086CPU只有16位，因此当时设计了四个段寄存器(CS、DS、ES和SS)分别用于指令、数据、其它和堆栈
    - 段寄存器来存储段基址的高16位，即存储段地址，段地址左移4位()后，再加上16位的偏移地址，得到最后的物理地址。

(2) 寄存器 — 段寄存器

- 从80386CPU开始
  - 80x86家族的32位微处理器始于80386
  - 80386的数据总线和地址总线都达到了32根
  - 最大物理寻址空间为4GB
  - 由于寻址空间的增加，80x86为段部分提供了6个段寄存器：CS、DS、ES、SS、FS和GS

(2) 寄存器 — 段寄存器

- 目前80x86系统一共有6个段寄存器



(2) 寄存器 — 段寄存器

- CS：Code Segment，代码段寄存器
  - 存放应用程序代码所在段的段基址
- DS：Data Segment，数据段寄存器
  - 用于存放数据段的段基址
- SS：Stack Segment，堆栈段寄存器
  - 用于存放栈段的段基址
- ES、FS、GS，附加数据段寄存器
  - 用于存放程序使用的附加数据段的段基址

(2) 寄存器

- 什么是寄存器
- 寄存器分类
- 通用寄存器
- 段寄存器
- 程序状态与控制寄存器
- 指令指针寄存器

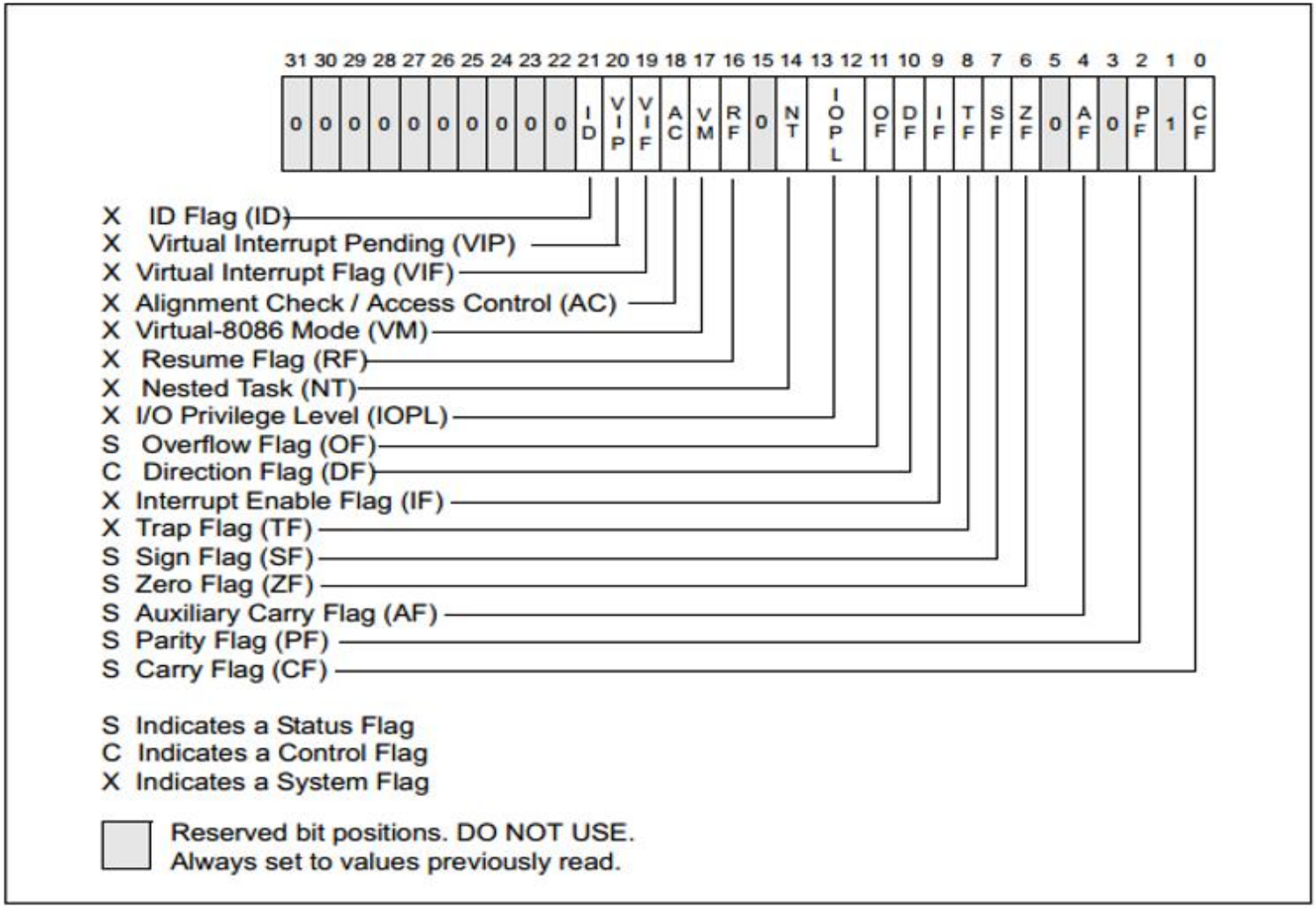
(2) 寄存器 — 程序状态与控制寄存器

- 程序状态与控制寄存器（也称为标志寄存器）
  - 主要有3种作用：
    - 用来存储相关指令的某些执行结果
    - 用来为CPU执行相关指令提供行为依据
    - 用来控制CPU的相关工作方式

(2) 寄存器 — 程序状态与控制寄存器

- 在IA-32中标志寄存器的名称为EFLAGS
  - 大小为4个字节， 共有32个位元，每一位都有专门的含义，记录特定的信息，每位的值为1或0，代表On/Off或True/False
  - 包含一组状态标志、一个控制标志和一组系统标志
  - 一些标志可以使用专用指令直接修改，但是没有指令可以将整个寄存器进行检查或修改

(2) 寄存器 — 程序状态与控制寄存器



(2) 寄存器 — 程序状态与控制寄存器

- EFLAGS寄存器的32位标志可以分为4类：
  - 系统标志(位8，9，14，16，17，18，19，20，21)和IOPL(I/O Privilege Level)字段(位12，13)
    - 控制操作系统或执行操作，应用程序不能修改以上标志位
  - 方向标志(DF，位10)：控制串操作指令的处理方向
    - DF=0，从低地址到高地址
    - DF=1，从高地址到低地址
  - 状态标志(位0，2，4，6，7和11)：表示算数指令的运算结果，如ADD、SUB、MUL和DIV指令
    - 和应用程序运行状态密切相关，需掌握
  - 预留标志位（位1，3，5，15，22-31）
    - DO NOT USE

(2) 寄存器 — 程序状态与控制寄存器

- 状态标志
  - 掌握6个与程序调试相关的状态标志(位0，2，4，6，7，和11)
    - CF(位0)
      - 进位标志位，一般情况下，在进行无符号数运算的时候，它记录了运算结果的最高有效位向更高位的进位值，或从更高位的借位值。
      - 在加法运算中，若运算结果从字或字节的最高位产生了进位，则CF=1；否则CF=0
      - 在减法运算中，若被减数无借位，则CF=0；否则CF=1
    - OF(位11)
      - 溢出标志位，一般情况下，OF记录了有符号数运算的结果是否发生了溢出
      - 如果发生了溢出，OF=1；如果没有发生溢出，OF=0
      - 注：CF和OF所表示的进位和溢出，是分别针对无符号数和有符号数运算而言的，一定要分清楚CF和OF的发生条件

(2) 寄存器 — 程序状态与控制寄存器

- **AF(位4)**
  - 辅助进位标志位，在发生以下情况时，辅助进位标志AF的值被置为1，否则其值为0：
    - » 在字操作时，发生低字节向高字节进位或借位时
    - » 在字节操作时，发生低4位向高4位进位或借位时
- **PF(位2)**
  - 奇偶标志位，记录相关指令执行后，其结果的最低有效字节中1的个数是否为偶数
  - 如果1的个数为偶数，PF=1；如果1的个数为奇数，PF=0

（2）寄存器 — 程序状态与控制寄存器

- **SF(位7)**
  - 符号标志位，记录相关指令执行后，其结果是否为负
  - 当操作数为有符号数时，若结果为负数，SF=1；若结果为非负数，SF=0
- **ZF(位6)**
  - 零标志位，记录相关指令执行后，其结果是否为0；若运算结果为0，则其值为1，否则其值为0

(2) 寄存器

- 什么是寄存器
- 寄存器分类
- 通用寄存器
- 段寄存器
- 程序状态与控制寄存器
- 指令指针寄存器

(2) 寄存器 — 指令指针寄存器

- EIP： Extended Instruction Pointer，指令指针寄存器
  - 在16位系统中：
    - 保存着CPU下一条将要执行指令的偏移量(offset)，这个偏移量是相对于目前正在运行的代码段寄存器CS而言的
    - 偏移量加上当前代码段的基地址，就形成了下一条指令的地址
  - 在32位系统中：
    - 它的大小为32位，是由原来的16位IP寄存器扩展而来
    - 往往直接保存CPU下一条将要执行指令的地址

(2) 寄存器 — 指令指针寄存器

- 程序运行时，CPU会读取EIP中一条指令的地址，将指令传送到指令缓冲区后，EIP的值自动增加
- CPU每次执行完一条指令，就会通过EIP寄存器读取并执行下一条指令
- 不能直接修改EIP的值，只能通过其他指令间接修改
  - 这些特定指令包括JMP、JC、CALL、RET
  - 可以通过中断或异常来修改EIP的值

第二章 汇编语言

- 1.CPU简介
- 2.寄存器
- 3.x86指令集
- 4.寻址方式

（3）x86指令集

- 一条汇编指令的标准格式

BEGING:	MOV	AL,	34H	; 注释
标号	指令助记符	目的操作数	源操作数	注释部分

（3）x86指令集

- 汇编代码是由两部分组成：操作码+操作数
  - 操作码在相应的机器指令体系中有相关的表示
  - 根据指令的功能，可以将大部分汇编语句分成如下几类：
    - 数据传送指令
    - 算数运算指令/逻辑运算/移位指令/浮点数运算指令
    - 串操作指令
    - 控制转移指令
    - 处理器控制指令

（3）x86指令集

- 数据传送指令
- 算数运算指令
- 逻辑运算和移位指令
- 串操作指令
- 控制转移指令
- 处理器控制指令
- 浮点运算指令

（3）x86指令集 — 数据传送指令

- 数据传送指令
  - 数据传送指令可以将数据、地址或立即数传送到寄存器或存储单元中
  - 大部分这类指令不影响状态标志位，部分涉及标志寄存器FLAGS的指令（SAHF和POPF）例外

（3）x86指令集 — 数据传送指令

- 数据传送指令
  - 通用数据传送指令
    - 数据传送指令
    - 堆栈操作指令
    - 数据交换指令
  - 地址传送指令
  - 标志寄存器传送指令

（3）x86指令集 — 数据传送指令

- 通用数据传送指令
  - MOV：把源操作数（OPS）传送到目的操作数（OPD）
  - MOVSX：带符号扩展传送
  - MOVZX：带零扩展传送
  - 指令格式：MOV(MOVSX/MOVZX) DEST，SRC
    - MOV EAX，EDX；寄存器EDX->EAX的数据传送。
    - MOVSX EAX，BL；将80H扩展为FFFFFF80H后送EAX中
    - MOVZX AX，BL；将80H扩展为0080H后送AX中。

（3）x86指令集 — 数据传送指令

- MOV传送指令（操作数类型）：
  - OPS可以为：存储器、通用寄存器、段寄存器和立即数。
  - OPD可以为：存储器、通用寄存器和段寄存器（CS除外）
  - 注意：
    - 立即数不能送段寄存器，其余可以任意搭配；立即数送存储器的指令有时难以确定操作数的长度，需要在存储器操作数的前面加上类型说明BYTE PTR或WORD PTR。
    - 例如：MOV BYTE PTR[SI+10H]，30 ；8位立即数30送偏移地址为SI+10H的字节单元；MOV WORD PTR[BX+DI]，2 ；16位立即数2送偏移地址为BX+DI的字单元

（3）x86指令集 — 数据传送指令

– MOV传送指令

➢注意：

– 两存储单元之间不能直接进行数据传送；两个段寄存器之间不能直接进行传送信息。但可以用CPU内部寄存器为桥梁来完成这样的传送

– 例如：MOV AL，AREA1　　例如：　MOV　AX，1000H

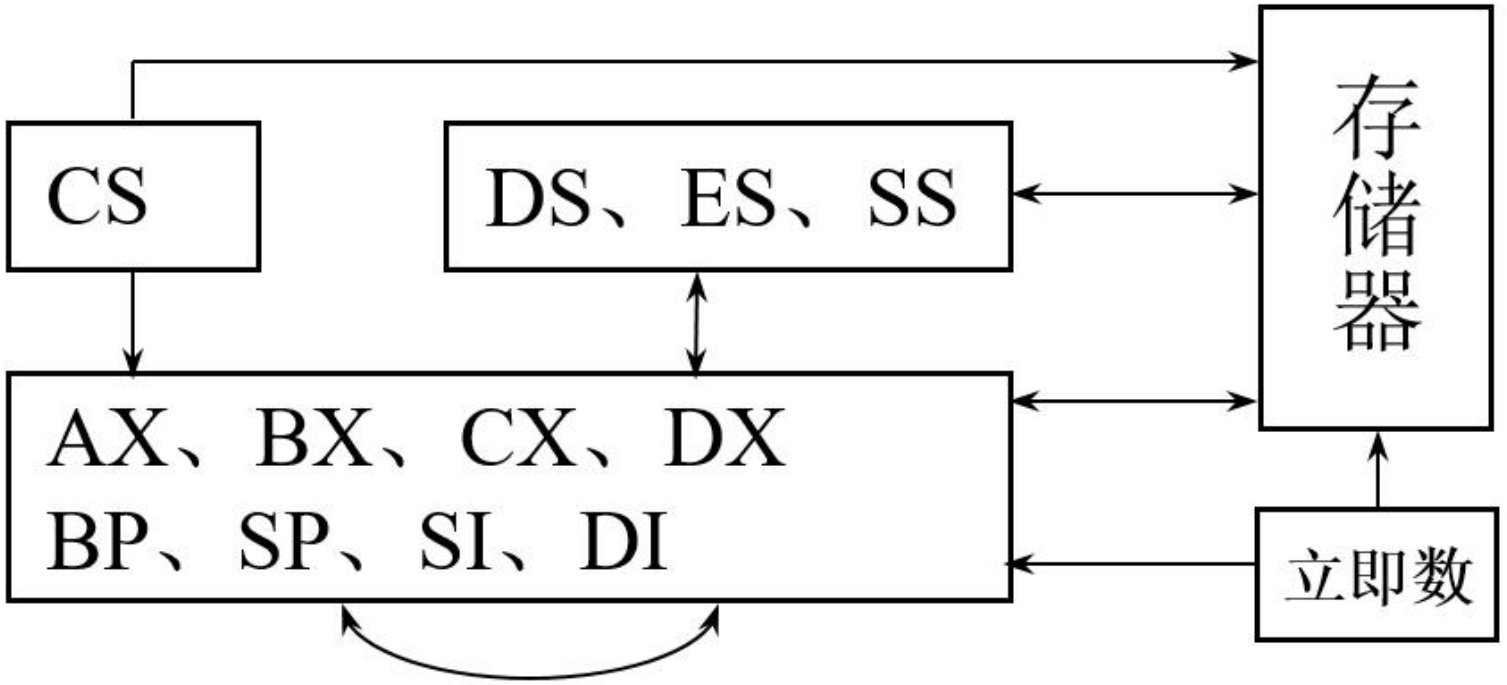
–　　MOV AREA2，AL　　MOV　DS，　AX

– 立即数、代码段寄存器CS只能作源操作数。

– IP寄存器不能作源操作数或目的操作数。

（3）x86指令集 — 数据传送指令

– MOV传送指令



（3）x86指令集 — 数据传送指令

– MOV传送指令

数据传送指令如下：

① 立即数送寄存器

```
MOV     AL，10H
MOV     BX，2100H
```

② 寄存器之间传送

```
MOV     DX，CX
MOV     AH，DL
MOV     DS，AX
MOV     DX，ES
```

③ 通用寄存器与存储器之间传送

```
MOV     AX，[1000H]
MOV     [BP]，DX
```

④ 段寄存器与存储器之间传送

```
MOV     [BX][DI]，ES
MOV     DS，10[BP+DI]
```

指出下列数据传送指令中的错误。

① MOV 10H，AX

立即数不能作为目的操作数

② MOV DS，2000

立即数不能送段寄存器

③ MOV CS，AX

CS不能作为目的操作数

④ MOV DS，ES

目的操作数和源操作数不能同时为段寄存器

⑤ MOV [DI]，[SI]

目的操作数和源操作数不能同时为存储器

⑥ MOV AL，BX

类型不匹配，AL为8位、BX为16位寄存器

⑦ MOV DL，300

类型不匹配，DL为8位寄存器，300超过1B

（3）x86指令集 — 数据传送指令

- 堆栈操作指令
  - PUSH：操作数进栈
  - PUSHA：把AX，CX，DX，BX，SP，BP，SI，DI依次压入堆栈。
  - PUSHAD：把EAX，ECX，EDX，EBX，ESP，EBP，ESI，EDI依次压入堆栈。
  - 指令格式：PUSH (PUSHA/PUSHAD) SRC

（3）x86指令集 — 数据传送指令

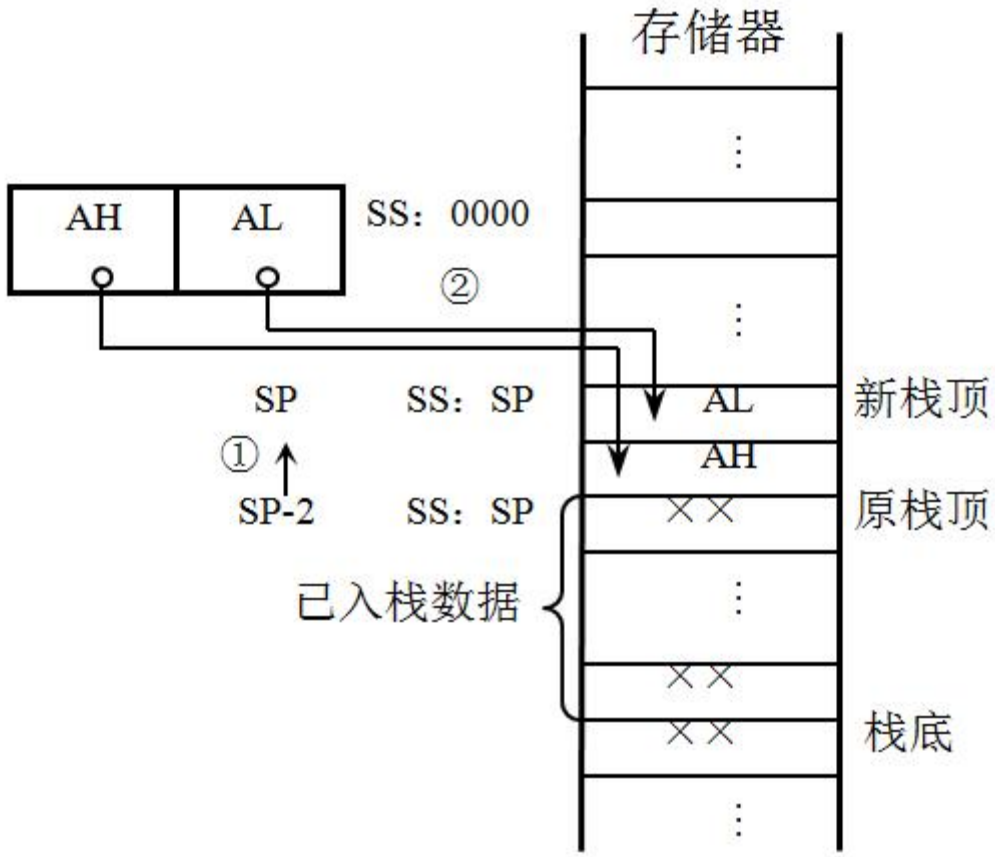
– **PUSH：** 操作数进栈

- 入栈指令
- 格式：**PUSH OPS**
- 操作：**SP ← SP-2，[SP+1][SP] ← OPS**
- 操作数类型： **OPS**可以是存储器、通用寄存器和段寄存器，但不能是立即数。
- 说明：**PUSH**指令先将**SP**的内容减**2**，然后再将操作数**OPS**的内容送入由**SP**指出的栈顶即偏移地址为**SP**和**SP+1**的两个连续字节中。

```
PUSH    AX          ；通用寄存器内容入栈
PUSH    CS          ；段寄存器内容入栈
PUSH    [SI]        ；字存储单元内容入栈
```

（3）x86指令集 — 数据传送指令

– PUSH：操作数进栈



（3）x86指令集 — 数据传送指令

- POP：出栈到目的操作数，把当前的SP所指向的堆栈顶部的一个字送到指定的目的操作数
- POPA：把DI，SI，BP，SP，BX，DX，CX，AX依次弹出堆栈。
- POPAD：把EDI，ESI，EBP，ESP，EBX，EDX，ECX，EAX依次弹出堆栈。
- 指令格式：POP (POPA/POPAD) DEST

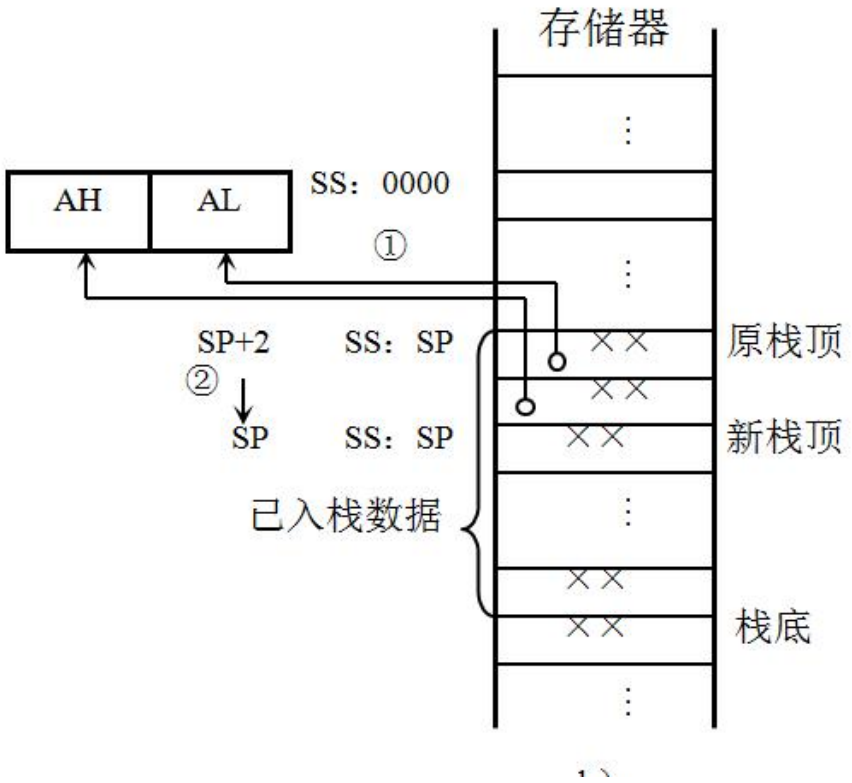
（3）x86指令集 — 数据传送指令

– **POP**：出栈到目的操作数，把当前的SP所指向的堆栈顶部的字送到指定的目的操作数

- 格式：**POP OPD**
- 操作：**OPD ← [SP+1][SP]，SP ← SP+2**
- 操作数类型：**OPD**可以是存储器、通用寄存器或段寄存器（但不能是**CS**），同样，不能是立即数。
- 说明：**POP**指令先将堆栈指针**SP**所指示的栈顶存储单元的值弹出到操作数**OPD**中，然后再将**SP**的内容加**2**。入栈和出栈操作如图所示。

（3）x86指令集 — 数据传送指令

– **POP**：出栈到目的操作数，把当前的SP所指向的堆栈顶部的一个字送到指定的目的操作数



（3）x86指令集 — 数据传送指令

- 数据交换指令
  - XCHG：交换两操作数。允许通用寄存器之间，通用寄存器和存储器之间交换数据。
  - 指令格式：XCHG OPR1，OPR2
  - 例
    - XCHG EAX，EBX；通用寄存器之间交换数据。
    - XCHG EBX，[ESI]；通用寄存器和存储器之间交换数据
  - 注：两操作数不允许同时为存储器操作数，交换指令不影响标志位。

（3）x86指令集 — 数据传送指令

- 数据传送指令
  - 通用数据传送指令
  - 地址传送指令
  - 标志寄存器传送指令

（3）x86指令集 — 数据传送指令

- 地址传送指令
  - LEA（Load Effective Address）：将源操作数的有效地址传送到通用寄存器。
  - 指令格式：LEA REG，MEM
  - 例
    - LEA EAX，[EBP + var_cc]；将EBP + var_cc的值送入EAX
    - MOV EAX，[EBP + var_cc]；将存储在地址EBP + var_cc上的变量值送入EAX

（3）x86指令集 — 数据传送指令

- 标志寄存器传送指令
  - PUSHF：16位标志寄存器进栈
  - PUSHFD：32位标志寄存器进栈
  - POPF：16位标志寄存器出栈
  - POPFD：32位标志寄存器出栈

（3）x86指令集

- 数据传送指令
- 算数运算指令
- 逻辑运算和移位指令
- 串操作指令
- 控制转移指令
- 处理器控制指令
- 浮点运算指令

（3）x86指令集 — 算数运算指令

- 算数运算指令
  - 加法指令
  - 减法指令
  - 乘法指令
  - 除法指令

（3）x86指令集 — 算数运算指令

- 加法指令
  - ADD：将源操作数和目的操作数相加，结果送到目的操作数。
  - ADC：将源操作数与目的操作数以及CF值相加，结果传送到目的操作数。
  - 指令格式：ADD (ADC) DEST，SRC
    - 注：ADD，ADC指令影响的标志位是OF，SF，ZF，AF，PF，CF
  - INC：目的操作数加一，结果送入目的操作数
    - 目的操作数可以为通用寄存器或存储器操作数
  - 指令格式：INC DEST

（3）x86指令集 — 算数运算指令

- 减法指令
  - SUB：将目的操作数减去源操作数，结果送入目的操作数。
  - SBB：将目的操作数减去源操作数，再减去CF值，结果送入目的操作数。
  - 指令格式：SUB (SBB) DEST，SRC
    - 注：SUB，SBB指令影响的标志位是OF，SF，ZF，AF，PF，CF。
  - DEC：目的操作数减一，结果送入目的操作数
    - 目的操作数可以为通用寄存器或存储器操作数。
  - 指令格式：DEC DEST
    - 注：INC，DEC指令影响的标志位为OF，SF，ZF，AF，PF。

（3）x86指令集 — 算数运算指令

- 乘法指令
  - MUL：无符号数乘法指令，将源操作数和累加器中的目的操作数相乘，结果送入累加器中
  - IMUL：有符号数乘法指令，将源操作数和累加器中的目的操作数相乘，结果送入累加器中
  - 指令格式：MUL (IMUL) SRC
  - 说明：MUL，IMUL指令的源操作数为通用寄存器或存储器操作数，目的操作数缺省存放在ACC累加器（AL，AX，EAX）中。

（3）x86指令集 — 算数运算指令

- 除法指令
  - DIV：无符号除法指令
  - IDIV：有符号除法指令
  - 指令格式：DIV (IDIV) SRC
  - 说明：DIV、IDIV指令的源操作数作为除数，为通用寄存器或存储器操作数，目的操作数作为被除数，目的操作数缺省存放在ACC累加器（AL，AX，EAX）中

（3）x86指令集

- 数据传送指令
- 算数运算指令
- 逻辑运算和移位指令
- 串操作指令
- 控制转移指令
- 处理器控制指令
- 浮点运算指令

（3）x86指令集 — 逻辑运算和移位指令

- 逻辑运算和移位指令
  - 逻辑运算
  - 算术逻辑移位
  - 循环移位

（3）x86指令集 — 逻辑运算和移位指令

- 逻辑运算
  - **AND**：逻辑与，目的操作数和源操作数按位进行逻辑与运算，结果存目的操作数。
  - **OR**：逻辑或，目的操作数和源操作数按位进行逻辑或运算，结果存目的操作数。
  - 源操作数可以是通用寄存器、存储器操作数或立即数，目的操作数是通用寄存器或存储器操作数。
  - **XOR**：逻辑异或，目的操作数和源操作数按位进行逻辑异或运算，结果存目的操作数。
  - 指令格式：**AND (OR/XOR/XOR) DEST，SRC**

（3）x86指令集 — 逻辑运算和移位指令

- 逻辑运算
  - **NOT**：逻辑非，对目的操作数按位取反，结果存入目的操作数。
  - 指令格式：**NOT DEST**

（3）x86指令集 — 逻辑运算和移位指令

逻辑运算

- TEST：目的操作数与源操作数按位进行逻辑与操作，并修改标志位，结果不回送目的操作数。
- 指令格式：TEST DEST，SRC
- 注：TEST指令常用在检测某些条件是否满足，但又不希望改变原有操作数的情况下。紧跟在这条指令后面的往往是一条条件转移指令，根据测试结果产生分支，转向不同的处理程序
- 例
  - TEST ECX，ECX
  - JE Crackme.00401326；程序跳转到0x00401326处继续执行

（3）x86指令集 — 逻辑运算和移位指令

- 算术逻辑移位
  - SAR：算术右移
    - 按照操作数OPRD规定的移位位数，对目的操作数进行右移操作，每移一位，最低位移入标志位CF，最高位(符号位)保持不变。相当于对有符号数进行除2操作
  - SHR：逻辑右移
    - 按照操作数OPRD规定的移位位数，对目的操作数进行右移操作，每移一位，最低位移入标志位CF，最高位补0
  - 指令格式：SAR (SHR) DEST，OPRD

（3）x86指令集 — 逻辑运算和移位指令

- 算术逻辑移位
  - SAL：算术左移指令
  - SHL：逻辑左移指令
  - 指令格式：SAL (SHL) DEST，OPRD
    - 说明：
    - SAL、SHL指令功能完全相同，按照操作数OPRD的规定的移位位数对目的操作数进行左移操作，每移一位，最低位补0，最高位移入标志位CF中

（3）x86指令集 — 逻辑运算和移位指令

- 算术左移和算术右移
  - 主要用来进行有符号数的倍增、减半
- 逻辑左移和逻辑右移
  - 主要用来进行无符号数的倍增、减半

（3）x86指令集 — 逻辑运算和移位指令

- 循环移位
  - ROL：循环左移
  - ROR：循环右移
  - 指令格式：ROL(ROR) DEST， OPRD

（3）x86指令集

- 数据传送指令
- 算数运算指令
- 逻辑运算和移位指令
- 串操作指令
- 控制转移指令
- 处理器控制指令
- 浮点运算指令

（3）x86指令集 — 串操作指令

- 串指连续存放在存储器中的一些数据字节、字或双字。
- 串操作允许程序对连续存放的数据块进行操作。
- 串操作通常以DS：ESI来寻址源串，以ES：EDI来寻址目的串

（3）x86指令集 — 串操作指令

- 对于字符串的基本操作，80x86提供了五个指令

指令名称	字节/字操作	字节操作	字操作	双字操作
字符串传送	MOVS 目的串，源串	MOVSB	MOVSW	MOVSD
字符串比较	CMPS 目的串，源串	CMPSB	CMPSW	CMPSD
字符串扫描	SCAS 目的串	SCASB	SCASW	SCASD
字符串装入	LODS 源串	LODSB	LODSW	LODSD
字符串存储	STOS 目的串	STOSB	STOSW	STOSD

（3）x86指令集 — 串操作指令

- MOVS 目的串，源串
  - 指令功能：字符串传送指令，把由ESI作指针的源串的字节或字，传送到由EDI作指针的目的串中
- CMPS 目的串，源串
  - 指令功能：字符串比较指令，由DS：ESI规定的源串元素减去ES：EDI指出的目的串元素，结果不回送，仅影响标志位CF，AF，PF，OF，ZF和SF

（3）x86指令集 — 串操作指令

- **SCAS 目的串**
  - 指令功能：串扫描指令，比较EAX中内容与目的串中内容进行比较，由EAX的内容减去ES：EDI规定的目的串元素，结果不回送，仅影响标志位CF，AF，PF，SF，OF，ZF
- **LODS 源串**
  - 指令功能：串装入指令，将DS：ESI所指的源串元素装入累加器EAX中
- **STOS 目的串**
  - 指令功能：串存储指令，将累加器EAX中值存入ES：EDI所指的目的串存储单元中

（3）x86指令集

- 数据传送指令
- 算数运算指令
- 逻辑运算和移位指令
- 串操作指令
- 控制转移指令
- 处理器控制指令
- 浮点运算指令

（3）x86指令集 — 控制转移指令

- 控制转移指令
  - 无条件转移指令
  - 条件转移指令
  - 循环控制指令

（3）x86指令集 — 控制转移指令

- 无条件转移指令

指令	说明
JMP	无条件转移
CALL	过程调用
RET	过程返回

（3）x86指令集 — 控制转移指令

- 条件转移指令
  - 条件转移指令是根据上一条指令执行后，CPU设置的状态标志作为判别测试条件，来决定是否转移
    - 跳转常用标志主要包括ZF，SF等
  - 每一种条件转移指令都有它的测试条件
    - 当条件成立，便控制程序转向指令中给出的目的地址
    - 否则，程序仍按顺序执行

（3）x86指令集 — 控制转移指令

- 根据单个标志位的状态判断转移的指令

指令	转移条件	说明
JC DEST	CF = 1	有进位/借位时转移
JNC DEST	CF = 0	无进位/借位时转移
JE/JZ DEST	ZF = 1	相等/等于零时转移
JNE/JNZ DEST	ZF = 0	不相等/不等于零时转移
JS DEST	SF = 1	是负数时转移
JNS DEST	SF = 0	不是负数时转移
JO DEST	OF =1	有溢出时转移
JNO DEST	OF = 0	无溢出时转移
JP/JPE DEST	PF = 1	有偶数个“1”时转移
JNP/JPO DEST	PF = 0	有奇数个“1”时转移

（3）x86指令集 — 控制转移指令

- 循环控制指令
  - 这类指令用ECX计数器的内容控制循环次数，先将循环次数存放在ECX中，每循环一次ECX内容减1，直到ECX为0时循环结束
    - REP 当CX/ECX>0时重复
    - REPE/REPZ 当ZF=1或比较结果相等,且CX/ECX>0时重复
    - REPNE/REPNZ 当ZF=0或比较结果不相等,且CX/ECX>0时重复
    - REPC 当CF=1且CX/ECX>0时重复
    - REPNC 当CF=0且CX/ECX>0时重复.

（3）x86指令集

- 数据传送指令
- 算数运算指令
- 逻辑运算和移位指令
- 串操作指令
- 控制转移指令
- 处理器控制指令
- 浮点运算指令

（3）x86指令集 — 处理器控制指令

- 处理器控制指令
  - 空操作指令
    - NOP（机器码0x90）：空操作，除了使EIP增1外，不做任何操作
  - 中断指令：
    - INTn：软件中断指令，也称为软中断指令，其中n为终端类型号，其值必须在0~255的范围内
    - 可以在编程时安排在程序中的任何位置上，因此也被称为陷阱中断

（3）x86指令集

- 数据传送指令
- 算数运算指令
- 逻辑运算和移位指令
- 串操作指令
- 控制转移指令
- 处理器控制指令
- 浮点运算指令

（3）x86指令集 — 处理器控制指令

- 浮点运算专用指令
  - 如果一条汇编指令以字母F开头，则其大概率为浮点运算指令，CPU中浮点运算单元主要负责处理此类指令
  - 由于浮点运算过程相对封闭，与软件运行流程、内部逻辑之间的关系较弱，本课程不再讲述此类汇编指令

第二章 汇编语言

- 1.CPU简介
- 2.寄存器
- 3.x86指令集
- 4.寻址方式

(4) 寻址方式

- 寻址方式(Addressing Mode)
  - 处理器寻找指令和操作数的方式
    - 指令寻址
      - 确定下一条要执行指令地址的方法
    - 操作数寻址
      - 确定当前指令操作数地址方法

(4) 寻址方式

- 指令寻址
  - 顺序寻址
    - 当程序不发生跳转时，通过EIP自加进行寻址的方式
  - 跳转寻址
    - 当程序发生跳转时，下一条要执行的指令的具体地址由跳转指令给出

(4) 寻址方式

- 操作数寻址
  - 根据操作数的不同表示，其寻址方式不同
    - 立即寻址
    - 寄存器寻址
    - 直接寻址
    - 寄存器间接寻址
    - 寄存器相对寻址
    - 基址变址寻址
    - 相对基址变址寻址
    - 比例变址寻址
    - 基址比例变址寻址
    - 相对基址比例变址寻址

(4) 寻址方式

- 立即寻址
  - 操作数直接放在指令中，紧跟在操作码之后，它作为指令的一部分存放在代码段中，这种操作数叫做立即数
  - 立即数寻址常用来给寄存器赋初值，不用访问寄存器、存储器，指令执行速度快
  - 例：**MOV EAX，26H**；将一个立即数**26H**送到**EAX**寄存器中
    - 注：立即数只能作为源操作数，不能作为目的操作数，源操作数的长度应该和目的操作数保持一致

(4) 寻址方式

– 示例： mov ax,1234H

– AH = ？ H， AL= ？ H

(4) 寻址方式

- 示例： mov ax,1234H
- AH = 12H， AL=34H

(4) 寻址方式

- 寄存器寻址
  - 操作数存放在寄存器中，指令执行时会到指定寄存器中取出相应的操作数，源和目的操作数都可以是寄存器，这种寻址方式由于操作数就在寄存器中，不需要访问存储器来取得操作数
  - 例： MOV EAX，EBX；
  - 寄存器寻址只访问寄存器，不访问存储器，速度快
  - 指令中可以引用的寄存器及其符号名称如下：
    - 32位寄存器有：EAX、EBX、ECX、EDX、ESI、EDI、ESP和EBP等
    - 16位寄存器有：AX、BX、CX、DX、SI、DI、SP和BP
    - 8位寄存器有：AH、AL、BH、BL、CH、CL、DH和DL

（4）寻址方式

- 例： MOV EAX，EBX；
- 若执行前EAX = 3047H，EBX =2378H，执行后EAX = ?H。

（4）寻址方式

- 例： MOV EAX，EBX；
- 若执行前EAX = 3047H，EBX=2378H，执行后EAX= EBX= 2378H

(4) 寻址方式

- 以上立即寻址和寄存器寻址两种寻址方式是对寄存器的寻址，不涉及对内存中数据的寻址
- 下面介绍的寻址方式，都是针对在存储区中的数据，通过不同的寻址方式求得操作数地址，从而取得操作数

（4）寻址方式

- 在80x86系统中，内存单元的物理地址由段基址和偏移地址(又称为偏移量)组成
  - 段基址：在IA-32的保护模式下，段基址由16位的段选择符得到，这些段选择符存放在6个段寄存器(CS，SS，DS，ES，FS，GS)中。
  - 有效地址的计算包含以下四个基本部分：
    - ①基址寄存器
    - ②变址寄存器
    - ③比例因子
    - ④位移量

（4）寻址方式

- 将基址寄存器、变址寄存器、比例因子、位移量四部分，按某种计算方法组合形成的偏移地址，称为有效地址EA(Effective Address)
- 有效地址EA=基址+变址*比例因子+位移量
- 其中，基址、变址、位移量的值可正可负，比例因子只能为正。

(4) 寻址方式

- 当采用16位寻址方式时，有效地址四种成分的组成为：
  - 基址寄存器：BX，BP
  - 变址寄存器：SI，DI
  - 比例因子：1
  - 位移量：0，8，16位
- 当采用32位寻址方式时，有效地址四种成分的组成为：
  - 基址寄存器：所有的32位通用寄存器
  - 变址寄存器：除ESP以外的32位通用寄存器
  - 比例因子：1，2，4，8
  - 位移量：0，8，16，32位

(4) 寻址方式

- 各种访存类型下所对应的段的默认选择

访存类型	段寄存器	缺省选择规则
指令	CS	用于取指令
堆栈	SS	堆栈操作 任何使用ESP或EBP作为基址寄存器的访存
局部数据	DS	除堆栈和目的串操作之外的其他数据的访问
目的串	ES	串处理指令的目的串

(4) 寻址方式

- 段超越前缀
  - 当使用内存操作数时， 无论哪种内存操作数寻址都有默认的段寄存器， 然而至多一个内存操作数可不使用默认段寄存器时，允许在程序中自行选择段寄存器，就需要使用段超越前缀
  - 段超越前缀的格式为：

    段寄存器：指令操作数
  - 例 MOV EAX，ES：[EBP]

(4) 寻址方式

- 虽然段超越前缀，允许改变系统所指定的默认段，但是有些情况下不允许修改：
  - 串处理操作中目的串必须使用ES段，即默认为ES:EDI不可修改
  - 压栈(push)、弹栈(pop)必须使用SS段，即默认为SS:ESP不可修改
  - 指令必须存放在CS段中

(4) 寻址方式

- 直接寻址
  - 指令中直接包含有操作数的有效地址(偏移地址)
    - 注：操作数一般存放在数据段DS，所以操作数的有效地址由DS加上指令中直接给出的16位偏移地址得到
  - 指令示例：
  - MOV AX,[1234H]
  - 假设DS = 4567H，内存中[468A4H] = 0001H，那么有效地址 = [4567H] *16+ [1234H]=[468A4H]
  - 那么寄存器AX = ?H

(4) 寻址方式

- 直接寻址
  - 指令中直接包含有操作数的有效地址(偏移地址)
    - 注：操作数一般存放在数据段DS，所以操作数的有效地址由DS加上指令中直接给出的16位偏移地址得到
  - 指令示例：
  - MOV AX，[1234H]
  - 假设DS = 4567H，内存中[468A4H] = 0001H，那么物理地址 = [4567H] *16+ [1234H]=[468A4H]
  - 那么寄存器AX = 0001H

(4) 寻址方式

- 例题
  - MOV AX,[8054H]
  - 如(DS) = 2000H,
  - 28054H里的内容为3050H
  - 则执行结果为(AX) = ?H

(4) 寻址方式

- 例题
  - MOV AX,[8054H]
  - 如(DS) = 2000H
  - 28054H里的内容为3050H
  - (物理地址=20000H+8054H=28054H)
  - 则执行结果为(AX) = 3050H

（4）寻址方式

- 直接寻址（段超越前缀）
  - 因为默认的是DS寄存器，其实也可以指定前缀寄存器，即段超越前缀
  - **MOV AX, SS:[1234H]**
  - 把SS数据段中偏移地址为1234H 的字复制到寄存器AX

(4) 寻址方式

- 寄存器间接寻址
  - 操作数的有效地址在寄存器中，这种寻址方式为寄存器间接寻址
  - 如果操作数的有效地址在EAX，EBX，ECX，EDX，ESI和EDI中，以上寄存器默认使用DS作为段寄存器，即DS段寄存器为段基值。
  - 如果操作数的有效地址在ESP，EBP，这两个寄存器默认使用SS作为段寄存器，即SS段寄存器为段基值。

（4）寻址方式

- 例：MOV EAX，[EBP]
- 把SS段中EBP指向的单元复制到EAX。
  - (EAX)= (SS)*16+ (EBP)
- 例：MOV EAX，[EDX]
- 把DS段中EDX指向的字节复制到EAX。
  - (EAX)= (DS)*16+ (EDX)
- 例：MOV [EDX]，EBX
- 把EBX的值复制到DS段中EDX指向的单元。
  - 地址(DS)*16+ (EDX)中的值为EBX

(4) 寻址方式

- 例题
  - MOV AX,[SI]
  - 如果(DS) = 5000H, (SI) = 1234H
  - 51234H地址中的内容为:6789H
  - 执行该指令后,(AX) = ?H

(4) 寻址方式

- 例题
  - MOV AX,[SI]
  - 如果(DS) = 5000H, (SI) = 1234H
  - 则物理地址 = 50000 + 1234 = 51234H
  - 51234H地址中的内容为:6789H
  - 执行该指令后,(AX) = 6789H

(4) 寻址方式

- 寄存器相对寻址
  - 操作数的有效地址EA为基址寄存器或变址寄存器的内容和指令中的位移量之和
    - EAX，EBX，ECX，EDX，ESI和EDI，以上寄存器默认使用DS作为段寄存器。
    - ESP，EBP，这两个寄存器默认使用SS作为段寄存器。
  - 例：MOV ECX，[EAX+24H]，也可以写成 MOV ECX，24H [EAX]；
    - 由DS段中EAX指向的内容加上位移量24,最终组成操作数的有效地址。
    - (ECX) = (DS*16+EAX+24H)

(4) 寻址方式

- 例题
  - MOV AX,[DI+1223H]
  - 假设，(DS) = 5000H，(DI) = 3678H
  - 则物理地址 = ？ H
  - 5589BH地址中的内容:568AH
  - 5489BH地址中的内容:55AAH
  - 5491BH地址中的内容:5B87H

  - 执行该指令后AX = ？ H

(4) 寻址方式

- 例题
  - MOV AX,[DI+1223H]
  - 假设，(DS) = 5000H，(DI) = 3678H
  - 则物理地址 = 50000H + 3678H + 1223H = 5489BH
  - 5489BH地址中的内容:55AAH
  - 执行该指令后AX = 55AAH

(4) 寻址方式

- 基址变址寻址
  - 操作数的有效地址由基址寄存器的内容与变址寄存器的内容之和获得
  - 通常将指令中的第二操作数的第一个寄存器作为基址寄存器，第二个寄存器作为变址寄存器
  - 其中，ESP不能作为变址寄存器

（4）寻址方式

- 例：MOV EAX，[EBX][ESI]，也可写成：MOV EAX，[EBX+ESI]
  - EBX为基址寄存器，ESI为变址寄存器，该指令将DS段中地址为EBX+ESI的存储单元的4字节数据送到EAX。
- 例：MOV EAX，[EBP+ESI]
  - 将SS段中地址为EBP+ESI的存储单元的4字节数据送到EAX。
- 例：MOV EAX，ES：[EBX+ESI]
  - 将ES段中地址为EBX+ESI的存储单元的4字节数据送到EAX。

（4）寻址方式

- 例题：
  - MOV AX,[BX][DI]
  - 如:(DS)=2100H，(BX)=0158H, (DI)=10A5H
  - 如：
    - 221FDH地址中的内容:2234H
    - 212FDH地址中的内容:1224H
    - 222FDH地址中的内容:1232H
  - 执行该指令后AX = ?H

（4）寻址方式

- 例题：
  - MOV AX,[BX][DI]
  - 如:(DS)=2100H, (BX)=0158H, (DI)=10A5H
  - 则有效地址EA=0158H + 10A5H = 11FDH
  - 物理地址=21000H + 11FD H= 221FDH
  - 221FDH地址中的内容:2234H
  - 执行该指令后AX = 2234H

（4）寻址方式

– 下面指令中，目的操作数采用基址加变址寻址

– **MOV DS:[BP+SI],AL**

(4) 寻址方式

- 相对基址变址寻址
  - 操作数的地址为基址寄存器的内容、变址寄存器的内容和指令中的位移量之和。
    - 常用于对二维数组的寻址
  - 例：MOV EAX 10H [EBX][ESI]
  - 或MOV EAX，[EBX+ESI+10H]
  - 或MOV EAX，10H [EBX+ESI]；
  - 将DS段中地址为EBX + ESI + 10H的存储单元的4字节数据送到EAX

(4) 寻址方式

- 例题:
  - MOV AX,[BX+DI-2H]
  - 假设，(DS) = 5000H, (BX) = 1223H, DI = 54H, (51274) = 43H (51275) = 54H, (51276) = 76H，
  - 物理地址=？ H
  - 执行该指令后 (AX) = ？ H

(4) 寻址方式

- 例题:
  - MOV AX,[BX+DI-2H]
  - 假设，(DS) = 5000H, (BX) = 1223H, DI = 54H, (51275) = 54H, (51276) = 76H
  - 物理地址= 50000 + 1223 + 0054 + FFFE(-2 各位取反末位加一) = 51275H
  - 执行该指令后 (AX) = 7654H

（4）寻址方式

- 比例变址寻址
  - 由指令中的变址寄存器的内容乘以比例因子再加上位移量得到操作的有效地址
  - EA＝变址寄存器×比例因子＋位移量
  - 此寻址只有32位寻址一种情况
  - 例：MOV EAX，1000H[ESI＊4]
  - DS段中地址为ESI所指向的内容乘4再加上1000H的内容形成有效地址
  - 例： MOV [EDI*2+100H]， ECX；
  - 把ECX的内容存储到由EDI*2+100H寻址的DS段存储单元中

(4) 寻址方式

- 基址比例变址寻址
  - 由指令中的变址寄存器的内容乘以比例因子，再加上基址寄存器的内容，得到操作的有效地址
  - EA＝变址寄存器×比例因子＋[基址寄存器]
    - 注1. 此寻址方式只有32位寻址一种情况
    - 注2.此寻址方式主要用于数组元素大小为2、4、8字节的二维数组操作。
  - 例： MOV EAX，[EBX+ECX*4]，也可写成MOV EAX，[EBX][ECX*4]；
  - 把由EBX+4*ECX之和寻址的DS段存储单元的4字节内容装入EAX

（4）寻址方式

- 相对基址比例变址寻址
  - 操作数的有效地址EA是变址寄存器的内容乘指令中的比例因子，加上基址寄存器的内容，再加上位移量之和。
  - EA＝变址寄存器×比例因子＋[基址寄存器]+位移量
    - 注1. 此寻址方式只有32位寻址一种情况。
    - 注2.此寻址方式主要用于数组元素大小为2、4、8字节，且数组起始地址不为0的二维数组操作。
  - 例：MOV EAX，[EBP+EDI*2+2]；
  - 把由EBP+2+EDI*2寻址的SS段存储单元的4字节内容装入EAX。
  - MOV ECX,10H[EDX*8][EAX]
  - MOV AX,10H[EBX*4][ESI]

(4) 寻址方式

- 格式总结
  - 1、立即寻址： MOV AL，05H
  - 2、寄存器寻址： MOV AL，BL
  - 3、直接寻址： MOV AL，[2000H]
  - 4、寄存器间接寻址： MOV AL，[SI]
  - 5、基址寻址： MOV AL，[BX+3]
  - 6、变址寻址： MOV AL，[SI +3]
  - 7、基址加变址寻址： MOV AL，[BX+SI]
  - 8、带位移的基址加变址寻址： MOV AL，[BX+SI+3]
  - 9、比例变址寻址： MOV EAX，50[ESI＊4]
  - 10、基址加比例变址寻址： MOV EAX，[ESI＊4][ECX]
  - 11、带位移的基址加比例变址寻址： MOV EAX，[ESI＊4][ECX＋10]

只适合32位寻址

谢　谢！