第2章 IA-32处理器基本功能

2.1 IA-32处理器简介

2.2 通用寄存器及使用

2.3 标志寄存器及使用

2.4 段寄存器

2.5 寻址方式

2.6 指令指针寄存器和简单控制转移

2.7 堆栈和堆栈操作

2.1 IA-32处理器简介

2.1.1 IA-32系列处理器

2.1.2 保护方式和实地址方式

2.1.1 IA-32系列处理器

➢IA-32系列处理器

- ✓ 泛指：基于英特尔IA-32架构的32位微处理器

  Intel 80386/80486

  Intel Pentium（奔腾）

  Intel Xeon（至强）

  Intel Core（酷睿）

- ✓ 最大特点：保持与先前处理器的兼容

2.1.1 IA-32系列处理器

➢处理器的重要指标

- ✓ 处理数据的位数
  - 16位
  - 32位
  - 64位
- ✓ 主频
- ✓ 平行化程度
  - 流水线
  - 多核

2.1.1 IA-32系列处理器

➢早期的**16**位处理器

✓ 1978年，Intel率先推出16位微处理器8086

✓ 1979年，Intel推出准16位微处理器8088

✓ 1982年，Intel推出“超级”16位微处理器80286

2.1.1 IA-32系列处理器

➢第一款32位处理器

- ✓ 1985年，Intel推出32位微处理器80386
- ✓ 全面支持32位数据类型和32位操作
- ✓ 支持实地址方式和保护方式两种工作方式
- ✓ 保护方式下，可寻址的物理地址空间高达4G
- ✓ 保护方式下，提供了完善的保护机制
- ✓ 为进入32位时代做好了充分准备

2.1.1 IA-32系列处理器

➢不断推陈出新

- ✓ 1989年，推出80486
- ✓ 1993年，推出Pentium（奔腾）
- ✓ 1995年，推出Pentium Pro（高能奔腾）
- ✓ 2000年，Pentium 4系列处理器面世
- ✓ 2001年，Xeon（至强）处理器系列被推出
- ✓ 2003年，Pentium M处理器被推出
- ✓ 最近几年，新的功能更强的处理器不断被推出

从普通程序员角度去看，这些处理器并没有显著差异

2.1.2 保护方式和实地址方式

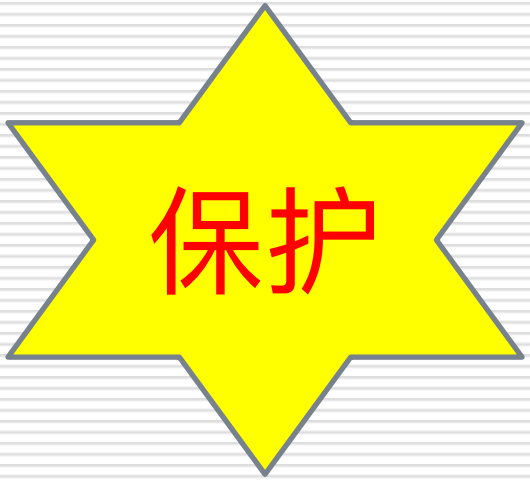
➢保护方式

- ✓ 现在保护方式（Protected mode）是IA-32系列处理器的常态工作方式
- ✓ 只有在保护方式下，IA-32系列处理器才能够发挥出其全部性能和特点
- ✓ Windows操作系统和基于IA-32处理器的Linux操作系统都运行于保护方式

2.1.2 保护方式和实地址方式

➢保护方式

- ✓ 全部32根地址线有效，可寻址高达4G字节的物理地址空间
- ✓ 支持存储器分段管理和可选的存储器分页管理机制
- ✓ 支持虚拟存储器的实现
- ✓ 提供完善的保护机制
- ✓ 支持操作系统实现多任务管理
- ✓ 支持虚拟8086方式（Virtual-8086 mode）



2.1.2 保护方式和实地址方式

➢实地址方式

✓ 实地址方式（Real-address mode）是最初的工作方式

- 实地址方式是处理器重新开始运行后的最初工作方式。
- 实地址方式是IA-32系列处理器中最初的处理器的工作方式。很久以前的 8086/8088处理器只具有所谓的实地址方式，没有保护方式。

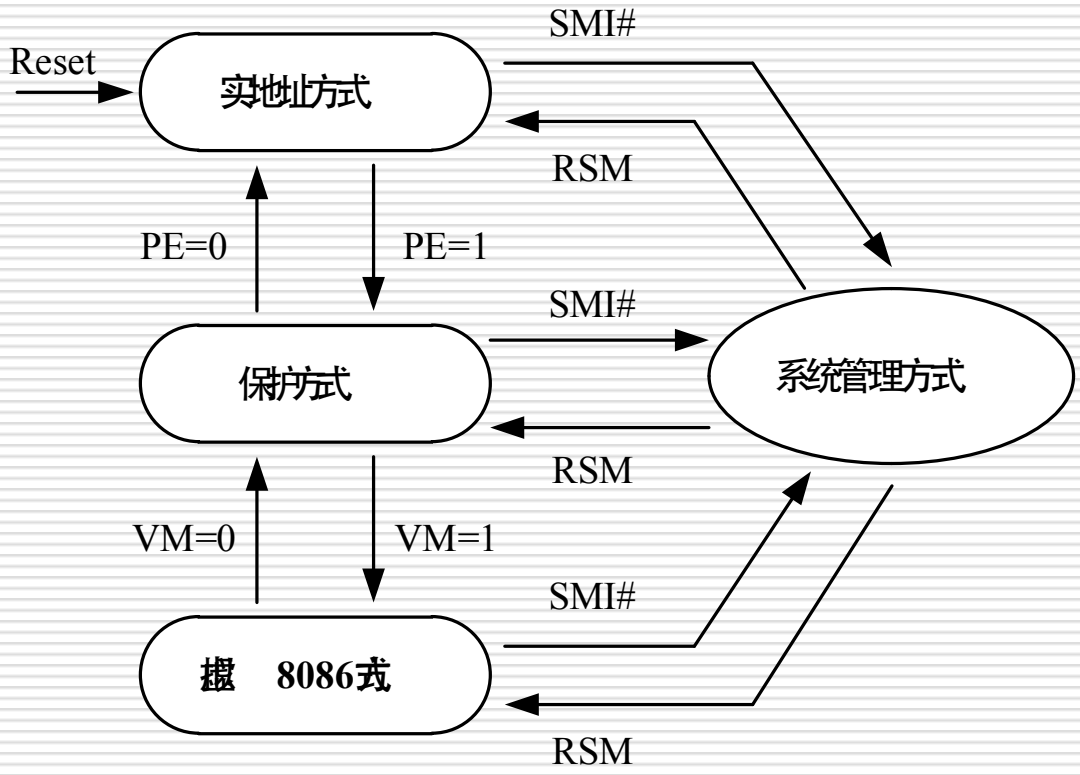
2.1.2 保护方式和实地址方式

➢实地址方式

- ✓ 只能访问最低端的1M字节的物理地址空间。地址空间的范围是00000H至FFFFFH
- ✓ 只支持存储器的分段管理，而且每个存储段的大小限于64K字节
- ✓ 实地址对应保护方式下的虚地址。这应该是实地址方式的名称由来。实地址方式常常被简称为**实方式**
- ✓ 在实方式下，IA-32系列处理器不能发挥其全部性能

2.1.2 保护方式和实地址方式

➢工作方式的切换



2.2 通用寄存器及使用

2.2.1 通用寄存器

2.2.2 简单传送指令

2.2.3 简单加减指令

2.2.4 VC嵌入汇编和实验

2.2.1 通用寄存器

➢寄存器

✓处理器内的特殊存储单元

✓处理器内有多种不同用途的寄存器

✓寄存器分别有各自的名称，以便表示及访问

2.2.1 通用寄存器

➢通用寄存器

✓IA-32系列CPU有8个32位的通用寄存器（General-Purpose Registers）

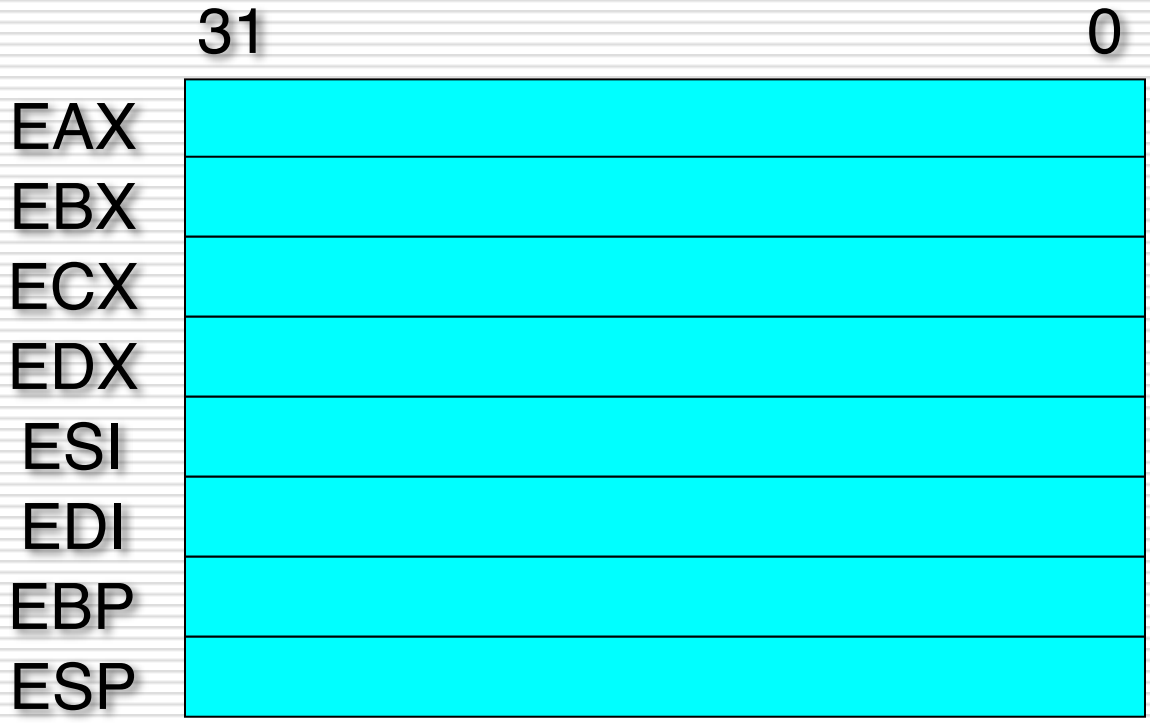
✓通用寄存器不仅能存储数据，而且能参与算术逻辑运算，还能给出存储单元的地址

✓名称分别是：

EAX、EBX、ECX、EDX、ESI、EDI、EBP、ESP

2.2.1 通用寄存器

➢通用寄存器



2.2.1 通用寄存器

➢通用寄存器

```
MOV    EAX, 12345678H      ;EAX=12345678H
MOV    ESI, 11223344H      ;ESI=11223344H
ADD    EAX, ESI            ;EAX=235689BCH
MOV    EBX, EAX            ;EBX=235689BCH
MOV    ECX, [ESI]          ;ESI作为指针给出存储单元地址
MOV    EDX, [EBX+8]        ;EBX作为计算存储单元地址的一部分
```

2.2.1 通用寄存器

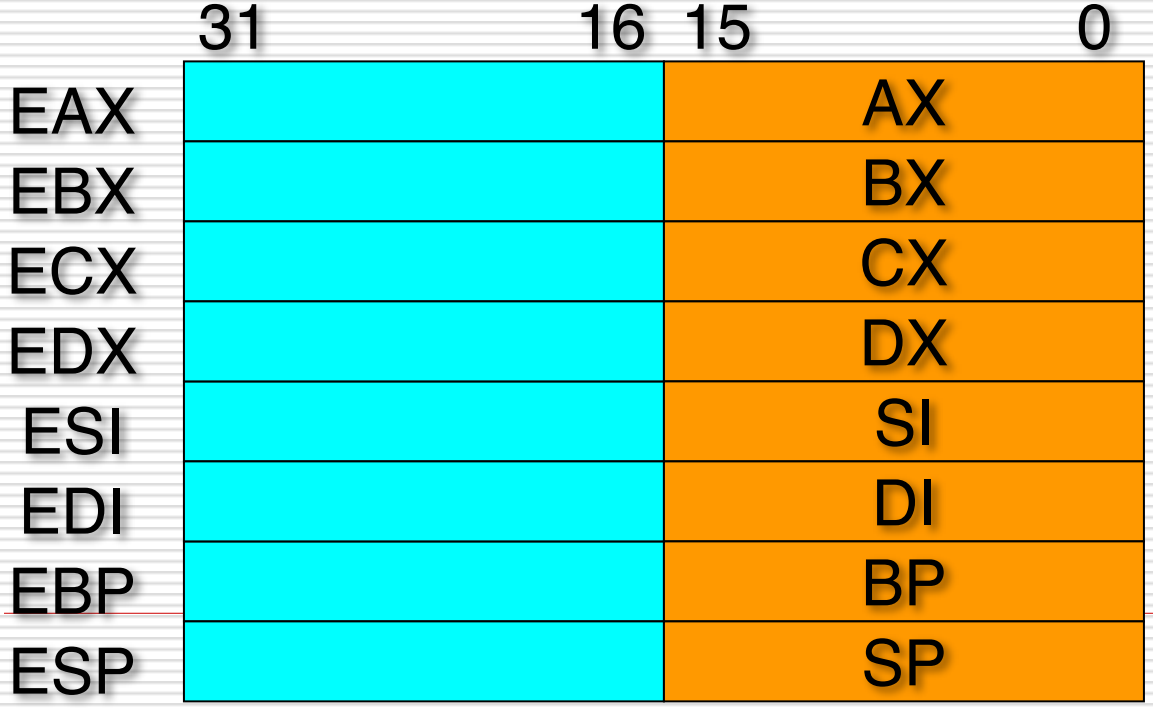
➢通用寄存器

✓可以单独直接访问这些通用寄存器的低16位

✓它们是8个16位的通用寄存器

✓名称分别是 AX、BX、CX、DX、SI、DI、BP、SP

✓对应16位处理器Intel 8086的8个通用寄存器



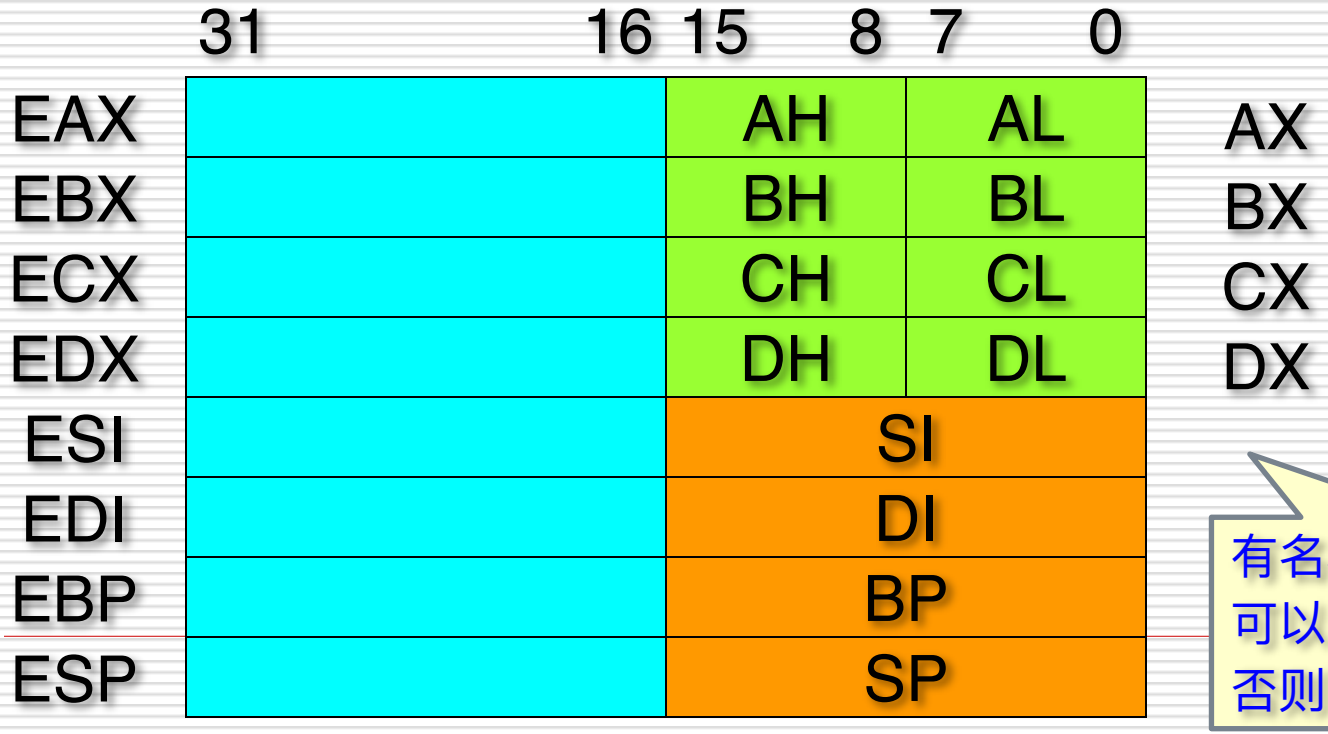
2.2.1 通用寄存器

➢通用寄存器

✓可单独直接访问AX、BX、CX和DX的高8位和低8位

✓它们是8个8位的通用寄存器

✓名称分别是 AH、AL、BH、BL、CH、CL、DH、DL



有名称的通用寄存器，
可以独立访问；
否则，不行

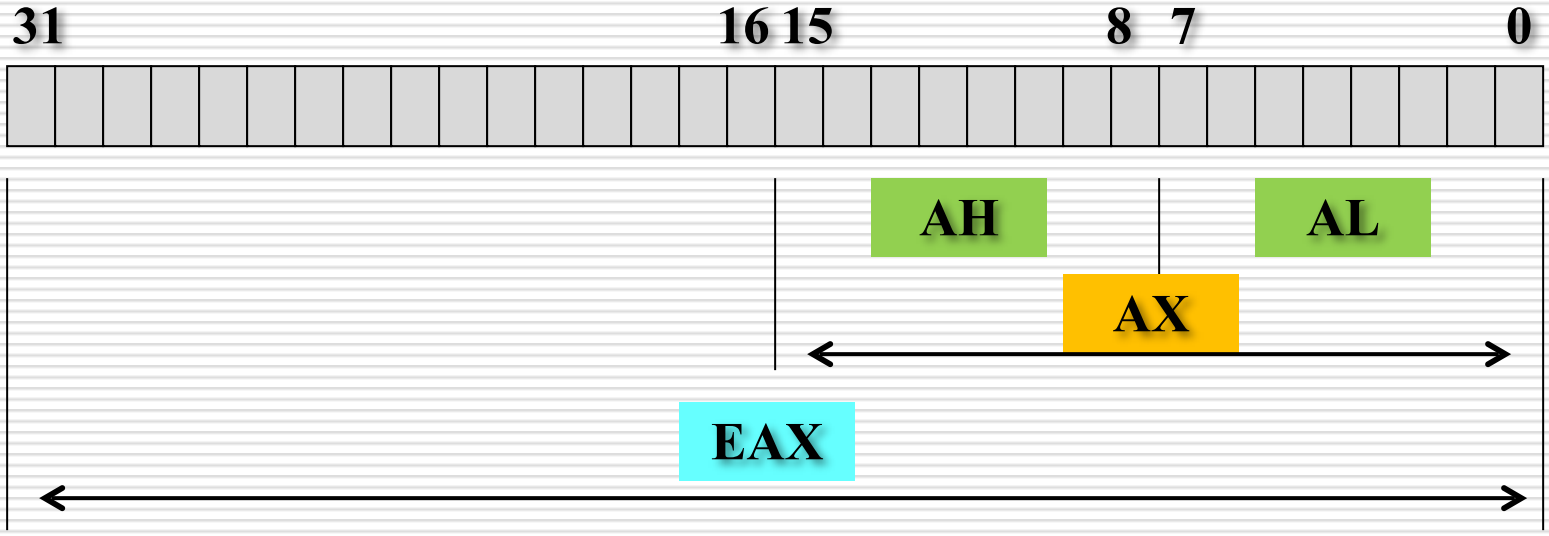
2.2.1 通用寄存器

➢通用寄存器

✓32位通用寄存器的名称是在对应16位寄存器名称前加字母E

✓AH是AX的高（High）字节；AL是AX的低（Low）字节

✓EAX是AX的扩展



2.2.1 通用寄存器

➢通用寄存器

对32位寄存器低16位独立操作，
不影响高16位

```
MOV    EAX, 11112222H        ;EAX=11112222H
MOV    AX, 9999H             ;EAX=11119999H
MOV    EDX, EAX              ;EDX=11119999H
MOV    DX, 8765H             ;EDX=11118765H
ADD    AX, DX                ;EAX=111120FEH
```

2.2.1 通用寄存器

➢通用寄存器

对**16**位寄存器的**8**位独立操作，
不影响另外**8**位

```
MOV    EBX, 11112222H       ;EBX=11112222H
MOV    BH, 77H              ;EBX=11117722H
MOV    BL, 99H              ;EBX=11117799H
ADD    BL, 82H              ;EBX=1111771BH
```

2.2.2 简单传送指令

- 传送指令MOV
- 交换指令XCHG

2.2.2 简单传送指令

➢传送指令（MOV）

✓MOV指令的一般格式

MOV　DST，SRC

✓MOV指令的动作

DST ← SRC

- 把一个字节、一个字或者一个双字，从源SRC送到目标DST。

这是在程序中用得最多的指令。注意：
源和目标的尺寸必须一致；
不能同时是存储单元!

2.2.2 简单传送指令

➢传送指令（MOV）

✓使用举例

MOV EAX, 12345678H	EAX = 12345678H
MOV EBX, EAX	EBX = 12345678H
MOV ESI, 256	ESI = 00000100H
MOV ECX, -1	ECX = FFFFFFFFH
MOV BX, ‘b’	EBX = 12340062H
MOV AH, AL	EAX = 12347878H
MOV CX, AX	ECX = FFFF7878H
MOV AX, SI	EAX = 12340100H
MOV SI, BX	ESI = 00000062H
MOV AL, BH	EAX = 12340100H

2.2.2 简单传送指令

➢交换指令（XCHG）

✓XCHG指令的一般格式

XCHG　*OPRD1*，*OPRD2*

✓XCHG指令的动作

OPRD1 ←→ OPRD2

- 操作数OPRD1的内容与操作数OPRD2的内容交换。

源和目标的尺寸必须一致。
不能同时是存储单元。

2.2.2 简单传送指令

➢交换指令（XCHG）

✓使用举例

```
XCHG    AL, AH          ;8位交换
XCHG    SI, BX          ;16位交换
XCHG    EAX, EBX        ;32位交换

XCHG    AL, [EBX]       ;AL与由EBX指定的字节存储单元交换
XCHG    [ESI], BX       ;BX与由ESI指定的字存储单元交换
XCHG    EDX, [EDI]      ;EDX与由EDI指定的双字存储单元交换
```

2.2.3 简单加减指令

- 加法指令ADD
- 减法指令SUB
- 加1指令INC
- 减1指令DEC
- 取补指令NEG

2.2.3 简单加减指令

➢加法指令（ADD）

✓ADD指令的一般格式

```
ADD    DST，SRC
```

✓ADD指令的动作

```
DST ← DST + SRC
```

- 把目标DST和源SRC相加，结果送到目标DST。

这是在程序中运用得很普遍的指令。
实现8位相加、16位相加或者32位相加。
注意：源和目标的尺寸必须一致。

2.2.3 简单加减指令

➢加法指令（ADD）

✓使用举例

MOV	EAX, 12345678H	EAX = 12345678H
MOV	ECX, 01010101H	ECX = 01010101H
ADD	EAX, ECX	EAX = 13355779H
ADD	CX, AX	ECX = 0101587AH
ADD	AL, CH	EAX = 133557D1H
ADD	AL, 3	EAX = 133557D4H
ADD	CX, 2000H	ECX = 0101787AH

2.2.3 简单加减指令

➢加法指令（ADD）

✓使用举例

MOV	EAX, 00001298H	EAX = 00001298 （十六进制）
ADD	AL, 81H	EAX = 00001219
MOV	EAX, 00001298H	EAX = 00001298 （十六进制）
ADD	AX, 81H	EAX = 00001319
MOV	EAX, 00009876H	EAX = 00009876 （十六进制）
MOV	EBX, 00008765H	EBX = 00008765
ADD	BX, AX	EBX = 00001FDB
MOV	EAX, 00009876H	EAX = 00009876 （十六进制）
MOV	EBX, 00008756H	EBX = 00008765
ADD	EAX, EBX	EBX = 00011FDB

2.2.3 简单加减指令

➢减法指令（SUB）

✓SUB指令的一般格式

SUB　*DST*，*SRC*

✓SUB指令的动作

DST ← DST - SRC

- 把目标DST减去源SRC，结果送到目标DST。

实现8位相减、16位相减或32位相减。
注意：源和目标的尺寸必须一致。

2.2.3 简单加减指令

➢减法指令（SUB）

✓使用举例

```
SUB    EDX, 1000       ;使EDX减去1000
SUB    ESI, EBX        ;使ESI减去EBX值
SUB    DI, 20          ;使DI减去20
SUB    DH, CL          ;使DH减去CL值
SUB    AL, 7           ;使AL减去7
SUB    ECX, [EDI]      ;使ECX减去由EDI指定的双字存储单元值
```

2.2.3 简单加减指令

➢加1指令（INC）

✓INC指令的一般格式

```
INC    DST
```

✓INC指令的动作

```
DST ← DST + 1
```

- 对操作数DST加1，然后把结果送回DST。

操作数DST可以是通用寄存器，也可以是存储单元。
操作数DST可以是8位、16位或32位。

2.2.3 简单加减指令

➢加1指令（INC）

✓使用举例

```
INC    ESI          ;使寄存器ESI值加1
INC    DI           ;使寄存器DI值加1
INC    CL           ;使寄存器CL值加1
```

2.2.3 简单加减指令

➢减1指令（DEC）

✓DEC指令的一般格式

DEC　*DST*

✓DEC指令的动作

DST ← DST - 1

- 对操作数DST减1，然后把结果送回DST。

操作数DST可以是通用寄存器，也可以是存储单元。
操作数DST可以是8位、16位或32位。

2.2.3 简单加减指令

➢减1指令（DEC）

✓使用举例

```
DEC    EDI          ;使寄存器EDI值减1
DEC    SI           ;使寄存器SI值减1
DEC    AL           ;使寄存器AL值减1
```

2.2.3 简单加减指令

➢取补指令（NEG）

✓NEG指令的一般格式

NEG　*OPRD*

✓NEG指令的动作

OPRD ← 0 - OPRD

- 取得操作数的负数，结果送回OPRD。
- 操作数是以补码表示的。

操作数**OPRD**可以是通用寄存器，也可以是存储单元。

2.2.3 简单加减指令

➢取补指令（NEG）

✓使用举例

```
MOV   AL, 3          ;AL=03H
NEG   AL             ;AL=FDH（-3）
MOV   EDX, -5        ;EAX=FFFFFFFBH
NEG   EDX            ;EAX=00000005H
```

2.3 标志寄存器及使用

2.3.1 标志寄存器

2.3.2 状态标志

2.3.3 状态标志操作指令

2.3.4 带进位加减指令

2.3.1 标志寄存器

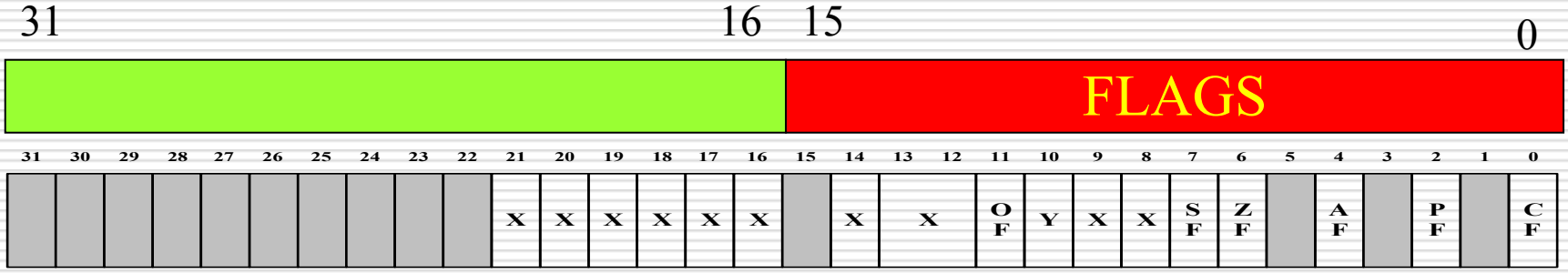
➢标志寄存器（EFLAGS register）

✓一个32位的寄存器

✓用于反映处理器的状态和运算结果的某些特征

✓可以暂时认为主要是，状态标志和控制标志

✓低端16位对应8086的FLAGS寄存器



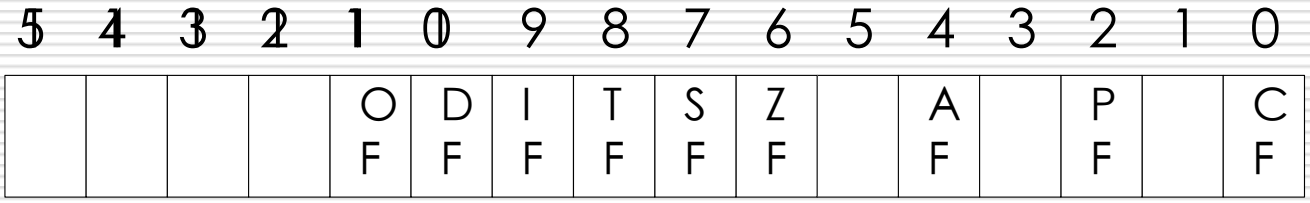
2.3.1 标志寄存器

➢标志寄存器的低16位（FLAGS register）

✓一组状态标志

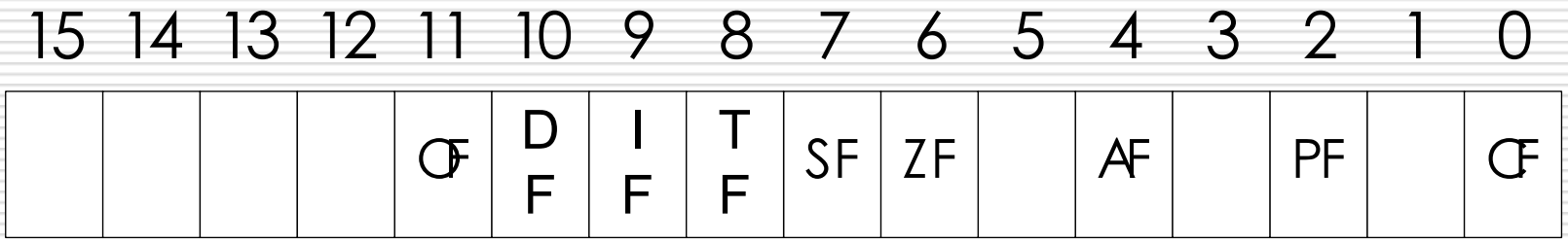
✓一组系统标志

✓一个控制标志



2.3.2 状态标志

➢状态标志



➢CF：进位标志（Carry Flag）

➢ZF：零标志（Zero Flag）

➢SF：符号标志（Sign Flag）

➢OF：溢出标志（Overflow Flag）

➢PF：奇偶标志（Parity Flag）

➢AF：辅助进位标志（Auxiliary Carry Flag）

2.3.2 状态标志

➢CF：进位标志（Carry Flag）

✓当算术运算产生进位或者借位时，置标志；否则清标志。

✓作为无符号数运算产生溢出的条件。

✓MOV 无符号数运算产生溢出的条件。

```
MOV   EAX, 77009966H
MOV   EDX, 55440000H
ADD   EAX, EDX
ADD   EDX, EAX
ADD   AX, AX
ADD   AL, 6
ADD   AL, 52H
ADD   AX, 0CDDCH
ADD   EAX, 33BC0000H
```

```
EAX = 77009966    （十六进制）
EDX = 55440000
EAX = CC449966    CF=0
EDX = 21889966    CF=1
EAX = CC4432CC    CF=1
EAX = CC4432D2    CF=0
EAX = CC443224    CF=1
EAX = CC440000    CF=1
EAX = 00000000    CF=1
```

2.3.2 状态标志

➢**ZF：零标志（Zero Flag）**

✓当运算结果为0时，置标志；否则清标志。

MOV	EAX, 77009966H	EAX = 77009966	（十六进制）
MOV	EDX, 55440000H	EDX = 55440000	
ADD	EAX, EDX	EAX = CC449966	ZF=0
ADD	EDX, EAX	EDX = 21889966	ZF=0
ADD	AX, AX	EAX = CC4432CC	ZF=0
ADD	AL, 6	EAX = CC4432D2	ZF=0
ADD	AL, 52H	EAX = CC443224	ZF=0
ADD	AX, 0CDDCH	EAX = CC440000	ZF=1
ADD	EAX, 33BC0000H	EAX = 00000000	ZF=1

2.3.2 状态标志

➢SF：符号标志（Sign Flag）

✓反映运算结果的符号位；（符号位为1，置标志；否则清）

✓与运算结果的最高位相同。

MOV	EAX, 77009966H	EAX = 77009966 （十六进制）
MOV	EDX, 55440000H	EDX = 55440000
ADD	EAX, EDX	EAX = CC449966 SF=1
ADD	EDX, EAX	EDX = 21889966 SF=0
ADD	AX, AX	EAX = CC4432CC SF=0
ADD	AL, 6	EAX = CC4432D2 SF=1
ADD	AL, 52H	EAX = CC443224 SF=0
ADD	AX, 0CDDCH	EAX = CC440000 SF=0
ADD	EAX, 33BC0000H	EAX = 00000000 SF=0

2.3.2 状态标志

➢OF：溢出标志（Overflow Flag）

✓反映有符号数的加减运算是否引起溢出；

✓如果溢出，置标志；否则清标志。

MOV	EAX, 77009966H	EAX = 77009966	（十六进制）
MOV	EDX, 55440000H	EDX = 55440000	
ADD	EAX, EDX	EAX = CC449966	OF=1
ADD	EDX, EAX	EDX = 21889966	OF=0
ADD	AX, AX	EAX = CC4432CC	OF=1
ADD	AL, 6	EAX = CC4432D2	OF=0
ADD	AL, 52H	EAX = CC443224	OF=0
ADD	AX, 0CDDCH	EAX = CC440000	OF=0
ADD	EAX, 33BC0000H	EAX = 00000000	OF=0

2.3.3 状态标志操作指令

➢进位标志操作指令

```
CLC                ; 清CF
STC                ; 置CF
CMC                ; CF取反
```

➢使用举例

```
MOV    AX, 8899H
ADD    AL, AH
CLC
STC
CMC
CMC
```

AX = 8899H

AX = 8821H CF=1

CF=0

CF=1

CF=0

CF=1

2.3.3 状态标志操作指令

➢获取状态标志操作指令（LAHF）

LAHF

- 把标志寄存器的低8位，送到通用寄存器AH中

2.3.3 状态标志操作指令

➢设置状态标志操作指令（SAHF）

SAHF

- 使得状态标志SF、ZF、AF、PF和CF分别成为来自寄存器AH中对应位的值

2.3.3 状态标志操作指令

➢演示程序**dp25**

```
#include  <stdio.h>
int  main( )
{  //定义3个无符号字节变量
    unsigned  char  flag1, flag2, flag3;
     //嵌入汇编
    _asm  {
        .. .. ..
    }
    printf("flag1=%02XH\n", flag1);  //显示为flag1=02H
    printf("flag2=%02XH\n", flag2);  //显示为flag1=13H
    printf("flag3=%02XH\n", flag3);  //显示为flag1=86H
    return  0;
}
```

2.3.3 状态标志操作指令

➢演示程序**dp25**

```
_asm {
    MOV   AH, 0
    SAHF              //SF=0，ZF=0，PF=0，AF=0，CF=0
    LAHF              //把标志寄存器低8位(02H)又回送到AH
    MOV   flag1, AH   //把AH的值保存到变量flag1
    ;
    MOV   DX, 7799H   //DX=7799H
    ADD   DL, DH      //DX=7710H，AF=1，CF=1
    LAHF              //把标志寄存器低8位(13H)送到AH寄存器
    MOV   flag2, AH   //把AH的值保存到变量flag2
    ;
    SUB   DH, 84H     //DH=F310H，SF=1，CF=1
    CLC               //CF=0
    LAHF              //把标志寄存器低8位(86H)送到AH
    MOV   flag3, AH   //把AH的值保存到变量flag3
}
```

2.3.4 带进位加减指令

- 带进位加法指令ADC
- 带借位减法指令SBB

2.3.4 带进位加减指令

➢带进位加指令（ADC）

✓ADC指令的一般格式（ADD with Carry）

```
ADC    DST，SRC
```

✓ADC指令的动作

```
DST ← DST + SRC+ CF
```

- 把目标DST、源SRC和进位标志CF相加，结果送到目标DST。

该指令实现带进位的加操作。
注意：源和目标的尺寸必须一致。

2.3.4 带进位加减指令

➢带进位加指令（ADC）

✓使用举例

```
SUB    EAX, EAX          ;EAX=0, CF=0
ADC    EAX, 2            ;EAX=2, CF=0
STC                      ;CF=1
ADC    EAX, 2            ;EAX=5, CF=0
```

2.3.4 带进位加减指令

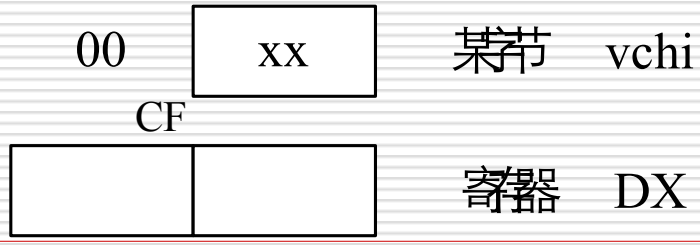
➢演示程序**dp26**

```
#include  <stdio.h>
int  main( )
{
    unsigned  char vch1=188,vch2=172,vch3=233;//定义3个字节变量
    unsigned  int   sum=0;                    //无符号整型变量
    //嵌入汇编
    _asm {
     .. .. ..
    }
    printf("sum=%u\n", sum);     //显示为sum=593
    return  0;
}
```

2.3.4 带进位加减指令

➢演示程序**dp26**

```
_asm {
   SUB   EDX, EDX        //使EDX为0，用DX存放累加和
   ADD   DL, vch1        //加第1个字节
   ADC   DH, 0           //高8位相加（保持形式一致）
   ADD   DL, vch2        //加第2个字节
   ADC   DH, 0           //高8位相加（考虑可能出现的进位）
   ADD   DL, vch3        //加第3个字节
   ADC   DH, 0           //高8位相加（考虑可能出现的进位）
   MOV   sum, EDX        //把结果送到变量sum
   }
```



2.3.4 带进位加减指令

➢带借位减指令（SBB）

✓**SBB指令的一般格式**（ Integer Subtraction with borrow ）

SBB　*DST*，*SRC*

✓**SBB指令的动作**

DST ← DST –（SRC+ CF）

- 把目标DST减去源SRC和借位标志CF，结果送到目标DST

该指令实现带借位减操作。
注意：源和目标的尺寸必须一致。

2.3.4 带进位加减指令

➢带借位减指令（SBB）

✓使用举例

```
MOV    AX, 620H          ;AX=0620H
SUB    AL, 21H           ;AL=FFH, CF=1, AX=06FFH
SBB    AH, 2             ;AH=03H, CF=0, AX=03FFH
SBB    AH, 2             ;AH=01H, CF=0
```

*指令对标志位的影响

✓指令对标志的影响情况，根据每条指令的功能而定。

✓算术运算指令根据运算结果，影响6个状态标志。

MOV EAX, 12345678H	EAX = 12345678 （十六进制）
MOV EDX, 20000000H	EDX = 20000000
SUB EAX, EDX	EAX = F2345678 CF=1，ZF=0，SF=1，OF=0，PF=1， AF=0
ADC DL, 0	EDX = 20000001
ADD AL, AH	CF=0，ZF=0，SF=0，OF=0，PF=0， AF=0
STC	EAX = F23456CE CF=0，ZF=0，SF=1，OF=1，PF=0， AF=0 CF = 1

*指令对标志位的影响

@续前页

```
SBB     EAX, EDX
ADC     AX, 39H
SBB     DH, DL
SUB     DL, DL
```

EAX = F23456CE （十六进制）

EDX = 20000001 CF=1

EAX = D23456CC

CF=0，ZF=0，SF=1，OF=0，PF=1，AF=0

EAX = D2345705

CF =0，ZF=0，SF=0，OF=0，PF=1，AF=1

EDX = 2000FF01

CF=1，ZF=0，SF=1，OF=0，PF=1，AF=1

EDX = 2000FF00

CF=0，ZF=1，SF=0，OF=0，PF=1，