



中山大學
SUN YAT-SEN UNIVERSITY

实验一 x86汇编语言程序设计

“Simplicity is prerequisite for reliability.”

——Edsger Dijkstra

数据科学与计算机学院

1. 8086的指令系统

- 8086是Intel x86系列处理器中最经典的16位处理器。即使到了Pentium处理器时代，系统中仍然保留着虚拟8086模式工作方式，以模拟16位处理器的工作环境
- 不管是32位还是64位，处理器的基本架构和编程理念始终没有发生质的变化，仍然是以8086的结构体系为基础，只不过计算速度更快，指令一次处理的字长更长，指令能完成的功能更多、更强大而已
- 掌握了8086的编程思想之后，学习8086以上的处理器指令就变得很容易了。

2、80286的指令系统

- CPU的地址线增加到了24位，同时引入了“实模式”和“保护模式”的概念，在实模式下，286与8086系统基本相同；在保护模式下，286的寻址空间可达16MB
- 在指令系统方面，放宽了一些指令的限制，同时也增加了一些新的指令。

- ◆ 1、数据传送指令：

- (1) 增加了PUSHA、POPA指令；
 - (2) PUSH指令允许操作数为立即数。

- ◆ 2、算术运算指令，扩充了IMUL指令的用法：

IMUL DST, IMM ; DST←DST×IMM

IMUL DST, SRC, IMM ; DST←SRC×IMM

- ◆ 3、逻辑操作指令，移位指令允许直接指定移位次数，而不要求通过CL指定。如“SHL AL, 2”是合法的。

- ◆ 4、字符串操作指令中，增加了INS、OUTS指令。

- ◆ 5、处理器控制指令：

- ◆ (1) 增加了BOUND、ENTER、LEAVE指令。

- ◆ (2) 增加了16条控制保护指令，包括LAR、LSL、LGDT、SGDT、LIDT、SIDT、LLDT、SLDT、LTR、STR、LMSW、SMSW、ARPL、CLTS、VERR、VERW等。这些指令使用频率较低，不作具体介绍。

- 80386是Intel x86系列中第1款32位的处理器
 - ◆ 80386的寄存器和数据总线都是32位的，地址总线也扩充到32位，可以直接寻址4GB的存储空间
 - ◆ 80386可以有3种工作模式：实地址模式、保护模式和虚拟8086模式。
 - ◆ 在内存管理方面，80386不仅支持内存分段管理，而且引入了内存分页管理的技术。
- 80386的存储器操作数有效地址可表达为
基址+变址*比例因子+位移量
- 80386系统放宽了对基址寄存器和变址寄存器的限制
 - ◆ 当地址偏移量为16位时，仍然只能用BP、BX和SI、DI分别作为基址、变址寄存器；
 - ◆ 当地址偏移量为32位时，基址寄存器可以是任何32位通用寄存器，变址寄存器可以是除ESP外的32位通用寄存器。
- 80386指令都支持32位操作数，如：
 - ◆ `MOV EAX,12345678H`



4. 80486的指令系统

增加了如下新的指令

① 算术运算指令

◆相加并交换指令XADD、比较并交换指令CMPXCHG、字节交换指令BSWAP。

② 其他指令

◆如INVD、WBINVD、INVLPG等。

5. Pentium的指令系统

Pentium机在80486的基础上，又增加了如下新的指令：

① 算术运算指令中，增加了8字节的比较并交换指令 CMPXCHG8B

② 其他指令，如RDMSR、WDMSR、REM、RDTSC等。



- 1 8086微机系统结构简介
- 2 8086指令系统简介
- 3 8086汇编语言程序设计

1 8086微机系统结构简介

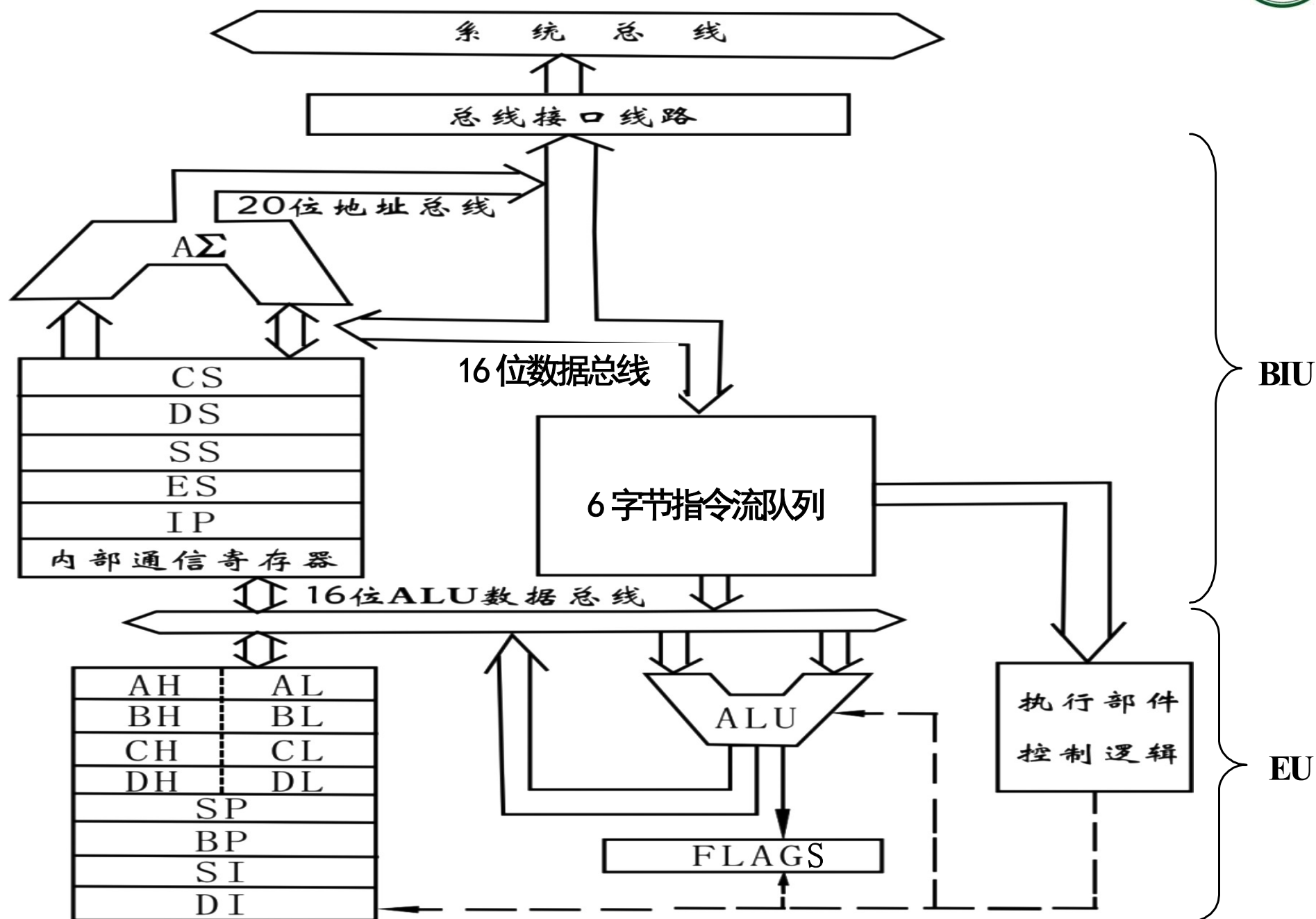


1.1 8086CPU及其寄存器

1.2 8086微机系统的主存储器与堆栈

1.3 8086CPU能直接处理的数据及其存放形式

1.1 8086CPU及其寄存器



8086CPU寄存器



A H	A L
B H	B L
C H	C L
D H	D L

A X 累 加 器

B X 基 址 寄 存 器

C X 计 数 寄 存 器

D X 数 据 寄 存 器

通 用 寄 存 器 组

S P
B P
S I
D I

堆 栈 指 针 寄 存 器

基 址 指 示 器

源 变 址 寄 存 器

目 的 变 址 寄 存 器

I P
F L A G S

指 令 指 针 寄 存 器

标 志 寄 存 器

C S
S S
D S
E S

代 码 段 寄 存 器

堆 栈 段 寄 存 器

数 据 段 寄 存 器

附 加 数 据 段 寄 存 器

段 寄 存 器 组

1) 通用寄存器组



- 寄存器中的信息可参与算逻运算，且结果可送入参加运算的任一寄存器中
- 包括AX、BX、CX、DX、SI、DI、BP、SP八个寄存器
- AX、BX、CX、DX既可作为16位寄存器使用，又可以分开作为两个8位寄存器
- 每个通用寄存器还有它们各自的专门用途

2) 段寄存器组



- 8086引入存储器**分段管理**机制，用户可以定义4种功能不同且相对独立的段，分别是**代码段、堆栈段、数据段和附加数据段**
- 8086CPU设置了4个16位的段寄存器：CS、SS、DS、ES，分别用来保存当前代码段、当前堆栈段、当前数据段和当前附加数据段的起始地址(即段基址)

3) 指令指示器(程序计数器)



□ 指令指示器**IP**用来存放程序代码段中指令的**偏移地址**

- 它指向下一条要执行指令的偏移地址

- CPU一旦取到这条指令就立即修改**IP**的内容，使它又指向下一条指令的偏移地址

□ 根据**CS**中存放的代码段段基址和**IP**的内容，可以确定指令在内存中的物理地址

一个简单的8086系统下的汇编语言程序



数据段	{	DATA SEGMENT	
		STRING DB	'HELLO WORLD!' , 0DH, 0AH, '\$'
		DATA ENDS	
代码段	{	CODE SEGMENT	
		ASSUME	CS:CODE, DS:DATA
		BEGIN:MOV	AX, DATA
		MOV	DS, AX ; 初始化数据段的段地址
		MOV	AH, 09H
		LEA	DX, STRING; 输出字符串
		INT	21H
		MOV	AH, 4CH
		INT	21H; 调用4CH号系统功能返回DOS
		CODE ENDS	
	END	BEGIN	

□ 辅助进/借位标志AF

- 字节操作时，低半字节向高半字节有进位/借位
- 主要用于十进制算术运算的调整

高半字节	低半字节
0001	0111B
+	0011
<hr/>	
0101	0000B

低半字节向高半字节
有进位，AF=1

高字节	低字节
01011001	01011011B
+	00011000
<hr/>	
01110001	10101110B

低半字节向高半字节
没有进位，AF=0



□进/借位标志CF(Carry Flag)

■对于算术运算，若结果的最高位有进/借位，则
CF=1，否则CF=0

■CF标志除了受指令执行结果的影响外，
8086CPU还专门设置三条指令改变CF的状态：

◆CLC: $CF \leftarrow 0$

◆STC: $CF \leftarrow 1$

◆CMC: CF取反

- 若CF=1，则 $CF \leftarrow 0$ ，否则 $CF \leftarrow 1$



□ 溢出标志OF (Overflow Flag)

■ 在进行有符号数的算术运算时，若运算结果发生溢出，则OF=1，否则OF=0

◆ 字节有符号数的表示范围：-128~+127

◆ 字有符号数的表示范围：-32768~+32767



□符号标志SF (Sign Flag)

- 符号标志位永远与运算结果的最高位保持一致，即结果的最高位(字节为D7，字为D15)为1，则SF=1，否则SF=0
- 因为有符号数用最高位表示数据的符号，故最高位的值就是符号位状态
- 实例：

3AH+7CH=B6H，最高位D7=1，故有SF=1



□ 奇偶标志位PF (Parity Flag)

■ 若运算结果低8位中二进制位“1”的个数为偶数，
则PF=1，否则PF=0

■ 注意：不论进行8/16位的操作，PF标志仅反映
结果低8位中“1”的个数是偶数还是奇数

■ 实例：

$3AH + 7CH = B6H = 10110110B$ ，因结果中有5个
1，是奇数，故：PF=0



□ 零标志ZF (Zero Flag)

■ 若运算结果为0，则ZF=1，否则ZF=0

■ 实例：

$84H + 7CH = (1)00H$ ，故ZF=1，CF=1



例5：字节59H与6CH相加后，FLAGS的六个状态标志位分别被设置成什么？

$$\begin{array}{r} 01011001\text{B} \\ +) 01101100\text{B} \\ \hline 11000101\text{B} \end{array}$$

AF=1, CF=0

OF=1, SF=1

PF=1, ZF=0

示例：902FH + 8761H



```
FOX DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Pro...
1000:001C 0000      ADD     [BX+SI],AL
1000:001E 0000      ADD     [BX+SI],AL
1000:0020 0000      ADD     [BX+SI],AL
1000:0022 0000      ADD     [BX+SI],AL
-t
AX=902F  BX=8761  CX=004F  DX=0000  SP=0400  BP=0000  SI=0000  DI=0000
DS=0760  ES=0760  SS=0775  CS=1000  IP=0006  OV UP EI PL NZ NA PO NC
1000:0006 01DB      ADD     AX,BX
-t
AX=1790  BX=8761  CX=004F  DX=0000  SP=0400  BP=0000  SI=0000  DI=0000
DS=0760  ES=0760  SS=0775  CS=1000  IP=0008  OV UP EI PL NZ AC PE CY
1000:0008 0000      ADD     [BX+SI],AL
-r ov
br Error
-r of
DS:8761=00
```

OF <----> OV(1) ; NV(0)

IF <----> EI(1) ; DI(0)

ZF <----> ZR(1) ; NZ(0)

PF <----> PE(1) ; PO(0)

DF <----> DN(1) ; UP(0)

SF <----> NG(1) ; PL(0)

AF <----> AC(1) ; NA(0)

CF <----> CY(1) ; NC(0)

br Error



□方向标志DF (Direction Flag)

■用于串操作指令，**控制串的处理方向**。每次串操作后，依据DF的值增/减源、目的地址：

◆DF=0时，从低地址到高地址处理串，即地址增加

◆DF=1时，从高地址到低地址处理串，即地址减少

■可以用指令改变DF标志的值

◆CLD, $DF \leftarrow 0$

◆STD, $DF \leftarrow 1$

□ 中断允许标志IF(Interrrrupt enable Flag)

■ 用于控制CPU是否响应外部可屏蔽的中断请求

◆ IF=1时，允许CPU接受外部的可屏蔽中断请求；否则不接受

■ 有两条指令可以改变IF的值

◆ CLI, $IF \leftarrow 0$

◆ STI, $IF \leftarrow 1$



□ 自陷标志TF (Trap Flag)

■ 用于控制CPU是单步还是连续执行指令

- ◆ TF=1, CPU进入单步执行方式

- ◆ TF=0, CPU连续执行程序

■ 没有专门的指令修改该标志位, 但可以通过堆栈间接地实现

➤ LAHF

■将标志寄存器的低8位传送到AH，即

$AH \leftarrow \text{FLAGSL8}$

➤ SAHF

■将AH的内容送到标志寄存器的低8位，即

$\text{FLAGSL8} \leftarrow AH$

1.2 8086微机系统的主存储器与堆栈



- 以**字节**为存储单元进行**编址**
- 8086有**20**条地址线，故其直接寻址能力可达1MB (**2^{20} 字节 = 1MB**)，即从**00000H到FFFFFFH**
- 物理地址与存储单元是一一对应的

存储器	物理地址
17H	00000H
ABH	00001H
00H	00002H
	FFFFCH
	FFFFDH
00H	FFFFEH
27H	FFFFFH

□ 物理地址 = 段基址 × 16 + 段内偏移地址

逻辑地址

□ 一个物理地址可有多多个逻辑地址与之对应

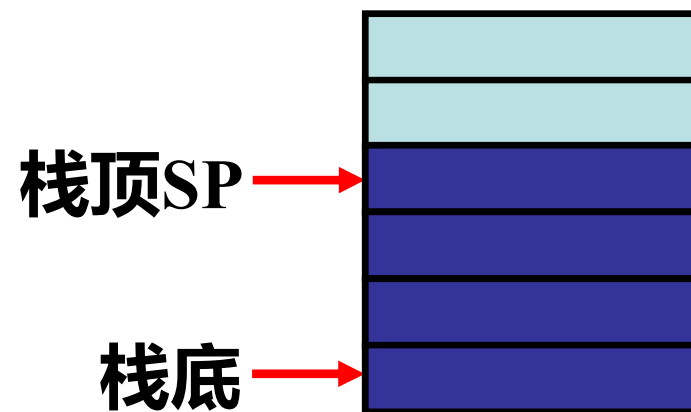
■ 逻辑地址 2000H:1000H

■ 逻辑地址 2100H:0000H

都对应物理地址
21000H

□什么是堆栈？

- 一端固定，一端活动，以后进先出(先进后出)方式组织起来的存储区域
- 活动端为栈顶，固定端为栈底。
栈顶和栈底重合时，表示栈空



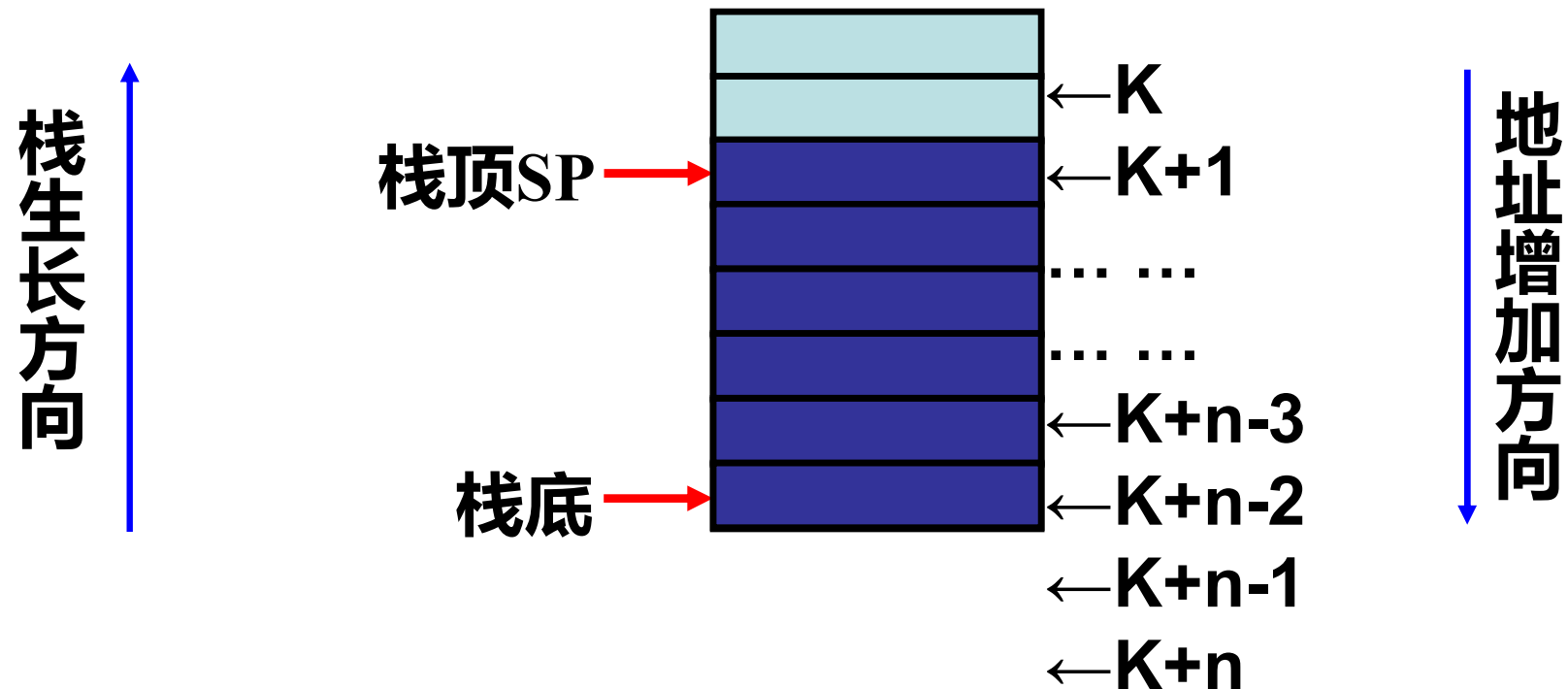
□堆栈的作用

- 解决多级中断、子程序嵌套和递归等程序设计中的实际问题
- 保存现场：寄存器内容、返回地址等
- 传递参数

□堆栈的实现

■8086采用在存储器中开辟的方式

■有的机器由寄存器堆积而成



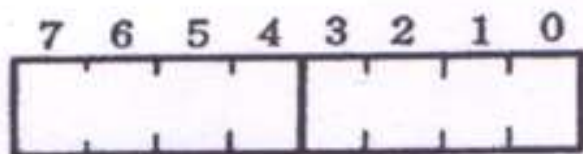
1.3 8086能直接处理的数据及存放形式



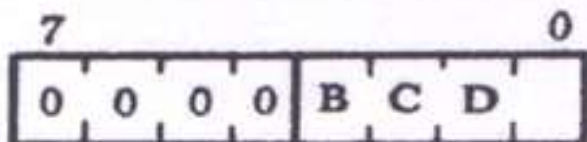
数据类型		位数	数的表示范围	说 明
字节数据	无符号整数	8	0~255	
	无符号小数	8	$0\sim 1\cdot 2^{-8}$	小数点在最高（第7）位之前
	有符号整数	8	-128~+127	最高（第7）位是符号位
	有符号小数	8	$-1.0\sim 1\cdot 2^{-7}$	最高位是符号位，小数点在该位之后
	非压缩十进制数	8	0~9	一个字节表示一位十进制数
	压缩十进制数	8	0~99	一个字节表示两位十进制数
字数据	无符号整数	16	0~65535	除作为数据外，也可作为段和偏移地址使用
	无符号小数	16	$0\sim 1\cdot 2^{-16}$	小数点在最高（第15）位之前
	有符号整数	16	-32768~+32767	最高（第15）位为符号位
	有符号小数	16	$-1.0\sim 1\cdot 2^{-15}$	最高（第15）位为符号位，小数点在该位之后
双字	乘数、被乘数、积、被除数、短浮点数	32	无符号整数 $0\sim 2^{32}-1$	有符号整数为 $-2^{31}\sim +2^{31}-1$
	地址指针	32	0~1M	16位段基址，16位偏移地址

8086能直接处理的数据，是指其指令系统中设有处理相应数据的指令

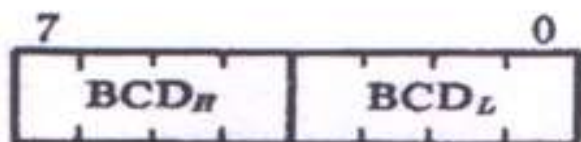
数据在存储器中的存放格式 (1/2)



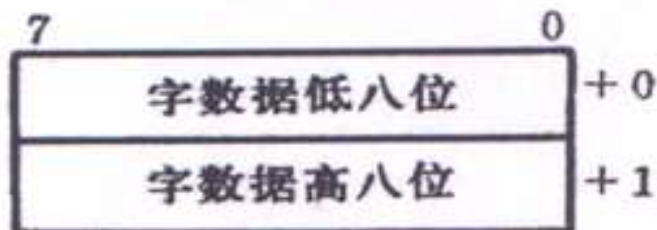
二进制字节数据



非压缩十进制数

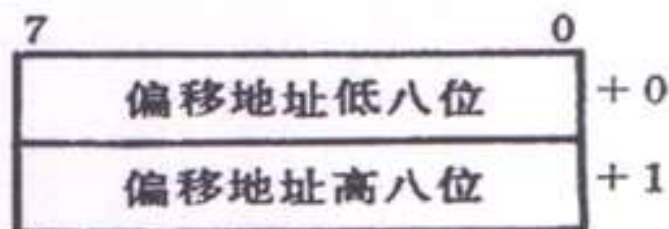


压缩十进制数

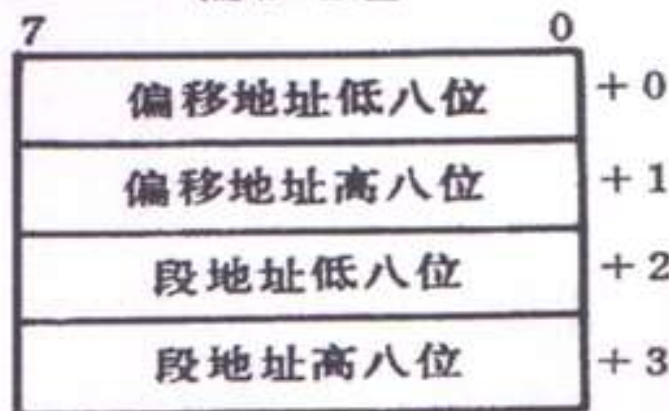


字数据

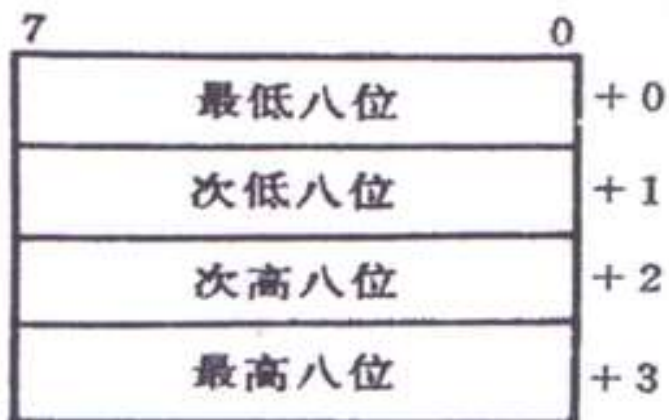
地址增加方向



偏移地址



地址指针



双字被除数

地址增加方向

□ 一个物理地址既可看成**字节**存储单元地址，又可看作**字**存储单元地址，还可看作**双字**存储单元地址，究竟当作何种地址要依据所存取的数据类型而定

例：物理地址为50000H，对应的**字节数据是1AH**，**字数据是191AH**，**双字数据是1718191AH**。

注意：对于多字节的数据，高位存放在高地址中，低位存放在低地址中。 **为什么？**

**X86采用小端
存储方式**

1AH	50000H
19H	50001H
18H	50002H
17H	50003H



- 1 8086微机系统结构简介
- 2 8086指令系统简介
- 3 8086汇编语言程序设计

2 8086指令系统简介



2.1 指令系统概述

2.2 8086指令格式

2.3 8086寻址方式

2.2 8086指令格式



□ 机器指令一般由**操作码**和**地址码**两部分组成。

■ 操作码：什么操作

■ 地址码：操作对象、操作结果存放何处

■ 一般格式：

操作码字段	地址码字段
-------	-------

■ 一条指令中，操作码不能少于一个，地址码可是零个、一个或多个

□ 设计指令格式时，要考虑**指令长度**、**操作码结构**和**地址码**结构三个方面

□8086机器指令格式

■属于CISC指令类型，其指令种类繁多，代码长度也各不相同

■一般形式：

0/1字节 0/1字节 1/2字节 0/1字节 0/1/2字节 0/1/2字节

指令前缀	段超越	操作码	寻址方式	位移量	立即数
------	-----	-----	------	-----	-----

8086机器指令格式

□ 符号指令格式

- 用**助忆符**表示**操作码**，用**符号或符号地址**表示**操作数或操作数地址**
- 它与**机器指令**具有一一对应关系
- 一般形式：
[标号] <指令助忆符> [[操作数1] [,操作数2][,操作数3]] [;注释]
- **符号**
 - ◆ 由字母和数字构成的字符串
 - ◆ **首字符是字母字符**
 - ◆ 字母不区分大、小写

□ 符号指令格式

- 用助忆符表示**操作码**，用符号或符号地址表示**操作数**或**操作数地址**
- 它与机器指令具有一一对应关系
- 一般形式：
[标号] <指令助忆符> [[操作数1] [,操作数2][,操作数3]] [;注释]
- **标号**
 - ◆ 用符号表示指令在内存中的首地址
 - ◆ 书写方法：符号后加1个冒号
 - ◆ 标号通常作为程序控制类指令的操作数

1) 汇编有关的定义约定(1/4)



□ 常量

- 表示固定值的量

- 字面常量

- ◆ 值的形式表示：如46AH、123D、123Q、11111100B等。

- ◆ 当十六进制数据的第1个数字为字母时，在其前面加“0”以便与符号区别，如0FFH

- 符号常量

- ◆ 用符号表示的常量

- ◆ 符号常量需通过定义后才有效，它不对应存储空间，如定义COUNT为20

1) 汇编有关的定义约定(2/4)



□ 变量

■ 变量是**存储器数据**的符号表示

◆ 可为一个数据或一个数据区（多个数据）

◆ 当表示数据区时，变量仅指向该数据区的第一个数据项

■ 变量的构成法同符号，需定义后使用

1) 汇编有关的定义约定(3/4)



□对于8086，有效地址特指偏移地址

□段基址由如下规则确定：

■有段超越时，由段超越决定段基址

■无段超越而有变量名时，由变量所在段决定段基址

■既无段超越也无变量名时，由访问存储器的方式决定段基址

1) 汇编有关的定义约定(4/4)



由访问存储器的方式决定段基址？？
？

- 取指令时，段基址为CS
- 进行堆栈操作时，段基址为SS
- 串操作的访存：
 - ◆串操作的源操作数（用SI访问），段基址为DS
 - ◆串操作的目的操作数（用DI访问），段基址为ES
- BP作基址的寻址方式，段基址为SS
- 其它方式，段基址均为DS

2 8086指令系统简介



2.1 指令系统概述

2.2 8086指令格式

2.3 8086寻址方式

□ 为什么要设置多种寻址方式?

- 缩短指令长度，扩大寻址空间
- 对数据访问的灵活性和有效性提供支持

□ 两个重要概念

■ 形式地址(Formal Address)

- ◆ (机器)指令中给出的操作数地址

■ 有效地址(Effective Address)

- ◆ 形式地址按一定的规则变换后，得到能直接访问操作数的地址

□ 寻址方式：从形式地址产生有效地址的方法

- 对于具有多个操作数的指令，每个操作数都可以有各自的寻址方式

□ 寻址方式的字节编码：

MOD

REG

R/M

MOD≠11时，有效地址EA的计算				MOD = 11，寄存器寻址		
R/M	MOD = 00	MOD = 01	MOD = 10	R/M或REG	W=0	W=1
000	[BX]+[SI]	[BX]+[SI]+D8	[BX]+[SI] +D16	000	AL	AX
001	[BX]+[DI]	[BX]+[DI] +D8	[BX]+[DI] +D16	001	CL	CX
010	[BP]+[SI]	[BP]+[SI] +D8	[BP]+[SI] +D16	010	DL	DX
011	[BP]+[DI]	[BP]+[DI] +D8	[BP]+[DI] +D16	011	BL	BX
100	[SI]	[SI] +D8	[SI] +D16	100	AH	SP
101	[DI]	[DI] +D8	[DI] +D16	101	CH	BP
110	直接地址	[BP] +D8	[BP] +D16	110	DH	SI
111	[BX]	[BX] +D8	[BX] +D16	111	BH	DI

1) 8086寻址方式



□ 为什么要设置多种寻址方式?

- 缩短指令长度，扩大寻址空间
- 对数据访问的灵活性和有效性提供支持

□ 两个重要概念

■ 形式地址(Formal Address)

- ◆ (机器)指令中给出的操作数地址

■ 有效地址(Effective Address)

- ◆ 形式地址按一定的规则变换后，得到能直接访问操作数的地址

□ 寻址方式：从形式地址产生有效地址的方法

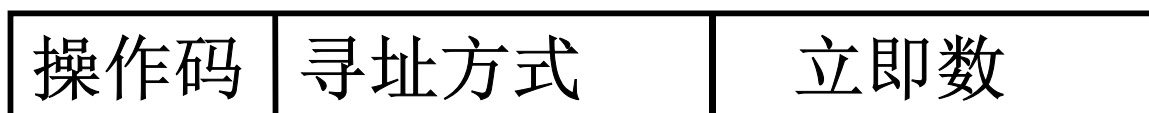
- 对于具有多个操作数的指令，**每个操作数都可以有各自的寻址方式**

2) 立即 (数) 寻址



□ 实例：MOV AL, 5

指令



目的操作数
寻址方式

取指令时此操作数
即被一并取出

□ 特点

- 操作数直接出现在机器指令中，作为机器指令的一部分，即形式地址不是操作数地址，而是操作数本身
- 取指令时操作数被一起取出，寻址速度快
- 只能用于表示源操作数

3) 直接寻址



□ 实例

MOV AL, DS:2000H

MOV BX, DS:[ADDR]

MOV VAR, AX

MOV BX, VAR+4 或者 MOV BX, VAR[4]

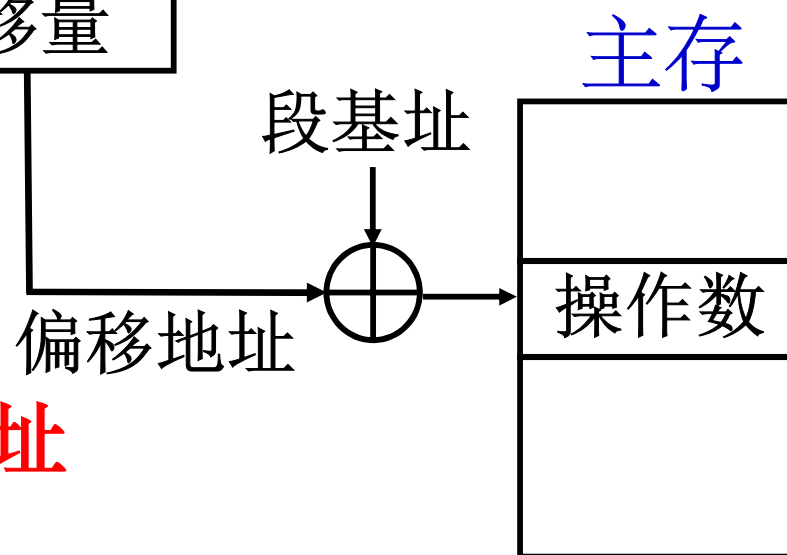
注意：必须使用段超越前缀，目的是为了与立即数寻址相区别

指令

操作码	寻址方式	位移量
-----	------	-----

□ 特点

■ 形式地址本身就是有效地址



4) 寄存器寻址



□ 实例

■ **MOV AX, 364AH**

■ **MOV CL, AL**

□ 特点

■ 类似于直接寻址，**机器指令中给出的操作数地址是寄存器编号**

■ **指令短，执行速度快**

5) 寄存器间接寻址 (1/2)



□ 实例

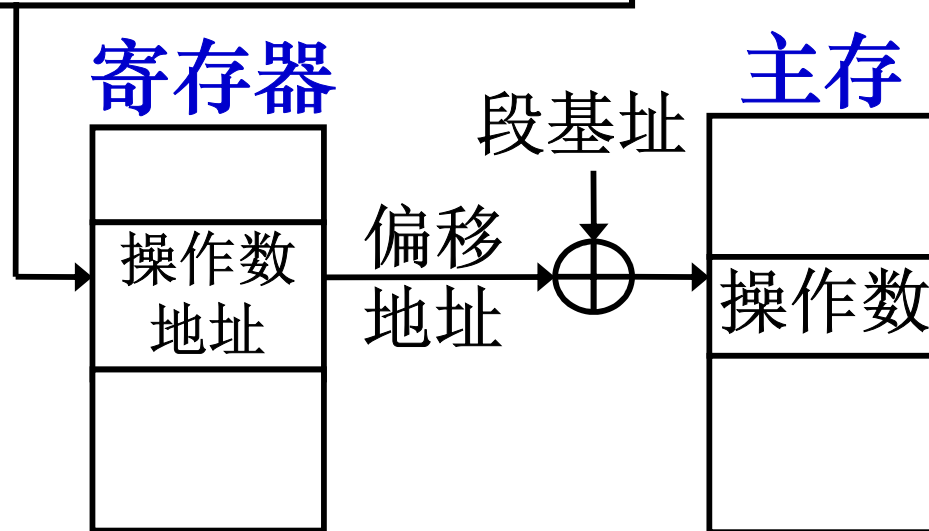
■ `MOV AX, [DI]` ;访问DS段、从[DI]单元取一个字→AX

□ 可用于寄存器间接寻址的寄存器有：

■ 基址寄存器 **BX**

■ 变址寄存器 **SI**、**DI**

操作码 寻址方式：基址/变址寄存器编号





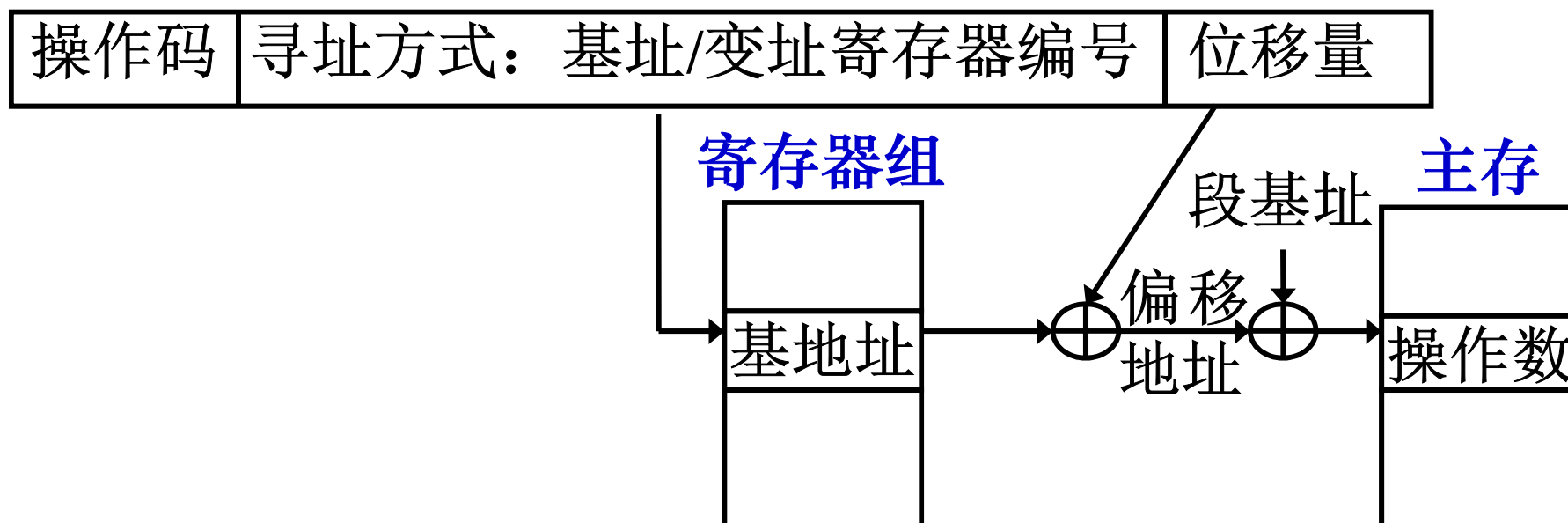
□特点

- 形式地址不直接给出操作数地址，而是给出**存放操作数地址的寄存器**
- 间接寻址可以是一次间接，也可以是多次间接
- 优点：指令短但寻址空间大，编写数组访问等程序灵活方便
- 缺点：需两次以上的访问才能得到操作数，执行速度慢

6) 基址寻址和变址寻址



□ 8086使用基址寄存器**BX**、**BP**，变址寄存器**DI**、**SI**；形式地址由位移量字段给出



7) 基址变址寻址



□实例:

- **DALTA[BP][DI][-100H]、DALTA[-100H][BP][DI]、
DALTA[DI][BP-100H]**
- **DELTA[BX][SI][5]、DELTA[5][BX][SI]、
DELTA[5][SI][BX]、DELTA[BX+SI][5]、
DELTA[BX+SI+5]**
- **MOV AL, [BX*2+10H][SI]**

□特点： 基址寻址 + 变址寻址

8) 相对寻址 (1/3)



- 相对寻址与基址寻址和变址寻址相似，都属于偏移寻址的范畴
- 相对寻址采用**指令指示器(程序计数器)**参与寻址，寻找下一条要执行的指令地址
- 在8086符号指令中，相对寻址操作数用目标指令的标号给出，**短程(SHORT)**标号对应字节位移量，**近程(NEAR)**标号对应字位移量



1. 段内直接寻址

- 转移指令的有效地址是当前IP/EIP寄存器的内容和指令中指定的位移量之和
- 用于条件转移指令、无条件转移指令、循环指令和CALL
- 转移标号类型默认为NEAR

例1: **JMP SHORT X**

2. 段内间接寻址

- 转移指令的有效地址是一个寄存器或是一个存储单元的内容，所得到的转移地址用来取代IP/EIP寄存器的内容

例2： 设(DS)=2000H, (BX)=1000H, 变量TABLE的有效地址为1000H, (21000H)=0040H, (22000H)=5678H, 则：

JMP TABLE[BX]

3. 段间直接寻址

- 完成从一个段到另一个段的转移操作
- 转移指令中直接提供转移的段地址和偏移地址，用指令中提供的偏移地址取代IP/EIP寄存器，用指令中提供的段地址取代CS寄存器的内容
- 转移标号类型默认为FAR PTR

例3: **JMP FAR PTR X**



```
1  assume cs:codesg
2  codesg segment
3  start:mov ax,0
4         mov bx,0
5         jmp far ptr s
6         db 256 dup(0)
7         s:add ax,1
8         inc ax
9  codesg ends
10 end start
```

DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DEBUG

```
AX=0000 BX=0000 CX=010F DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=075A ES=075A SS=0769 CS=076A IP=0003  NU UP EI PL NZ NA PO NC
076A:0003 BB0000      MOV     BX,0000
-t

AX=0000 BX=0000 CX=010F DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=075A ES=075A SS=0769 CS=076A IP=0006  NU UP EI PL NZ NA PO NC
076A:0006 EA0B016A07  JMP     076A:010B
-t

AX=0000 BX=0000 CX=010F DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=075A ES=075A SS=0769 CS=076A IP=010B  NU UP EI PL NZ NA PO NC
076A:010B 050100      ADD     AX,0001
-t

AX=0001 BX=0000 CX=010F DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=075A ES=075A SS=0769 CS=076A IP=010E  NU UP EI PL NZ NA PO NC
076A:010E 40          INC     AX
-t
```



4. 段间间接寻址

■用存储器中的字的来取代IP/EIP和CS寄存器中的原始内容，以达到段间转移的目的

■转移标号类型默认为FAR PTR

例4：设(DS)=2500H,(SI)=1300H,

(26426H)=4500H, (26428H)=32F0H,

则：JMP DWORD PTR[SI+0126H]; 执行后

CS= , IP=

□ I/O设备(接口)中的寄存器，称之为**I/O端口**

□ X86使用专门的I/O指令进行I/O端口的寻址

■ **直接端口寻址**：在指令中直接给出要访问的端口地址，使用1个字节立即数寻址，最多允许寻址256个端口

例1：IN AL, 20H ;从20H端口读入1个字节给寄存器AL

■ **间接端口寻址**：由DX寄存器间接地给出I/O端口地址，为两个字节，最多可寻址 $2^{16}=64\text{K}$ 个端口地址

例2：MOV DX, 3FCH

OUT DX, AL ;将寄存器AL的内容写入3FCH端口

OUT AL, DX ;将3FCH端口内容写入寄存器AL



- 1 8086微机系统结构简介
- 2 8086指令系统简介
- 3 8086汇编语言程序设计



3.1 8086指令类型

3.2 汇编语言源程序的结构

3.3 伪指令

3.4 子程序设计

3.5 系统功能调用

□ 一个完备的计算机指令系统至少包含以下四大类指令：

- 数据传送

- 算术/逻辑运算

- 程序控制

- 输入输出

□ 以8086指令系统为蓝本，分类介绍一些常用指令



1. 传送类指令

- 数据传送类指令
- 地址传送类指令
- 输入输出类指令

1) 数据传送类指令之MOV (1/2)



□ 正确的MOV指令举例

MOV AH, BL	;AH←BL
MOV BL, 254	;BL←254
MOV [DI], ES	;DS:[DI+1] ←ES的高8位、 ;DS:[DI]←ES的低8位
MOV DS, AX	;DS←AX
MOV SS, [BX]	;SS←DS: [BX+1、 BX]
MOV VARB[SI], 1	;VARB [SI]←1
MOV VARW[DI], AX	;VARW[DI+1]←AH、 ;VARW[DI]←AL或写为 ;VARW[DI+1、 DI]←AX

1) 数据传送类指令之MOV (2/2)



□ 传送指令使用**注意事项**:

- 两操作数不得同时为存储器操作数: **MOV [SI], [BX] (X)**
- 操作数的数据类型要一致: **MOV AX, BL (X)**
- 两操作数的数据类型不能都是不确定的, 即操作数的数据类型不得出现二义性: **MOV [BX], 1 (X)**
- 两操作数不得同时为段寄存器操作数: **MOV DS, ES (X)**
- 段寄存器作目的的操作数时, CS不能作目的的操作数, 源操作数不得为立即数: **MOV CS, AX; (X)**
MOV DS, 3542H (X)

1) 数据传送类指令之PUSH与POP



□ 进栈指令

■ MOV AX, 5000H

■ MOV SS, AX

■ MOV SP, 1000H

■ MOV BX, 3000H

■ MOV DS, BX

■ PUSHF

■ **PUSH AX**

■ PUSH DS

■ POP DS

■ **POP AX**

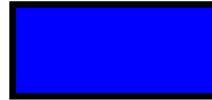
■ 操作数必须为16位!

■ POP指令与PUSH类似

AX:



BX:



SP:



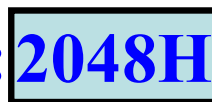
DS:



SS:



FLAGS:



50FF9H

50FFAH

50FFBH

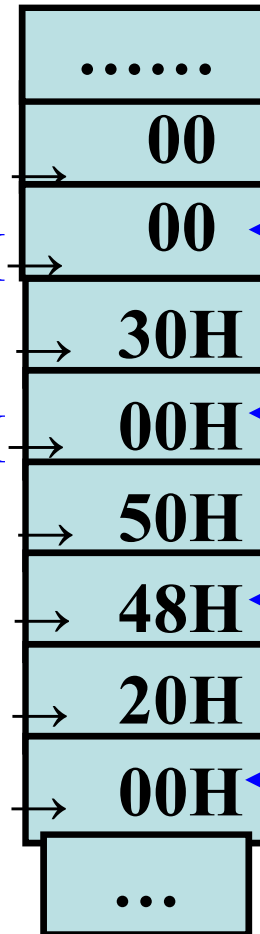
50FFCH

50FFDH

50FFEH

50FFFH

51000H



SS:SP

SS:SP

SS:SP

SS:SP

2) 地址传送类指令之LEA与LDS



□ 传送偏移地址到寄存器指令

■ **LEA DI, VARW ;DI←VARW的16位偏移地址**

■ **LEA DX, [BP][SI] ;DX←BP+SI的和**

□ 传送数据段地址指针指令

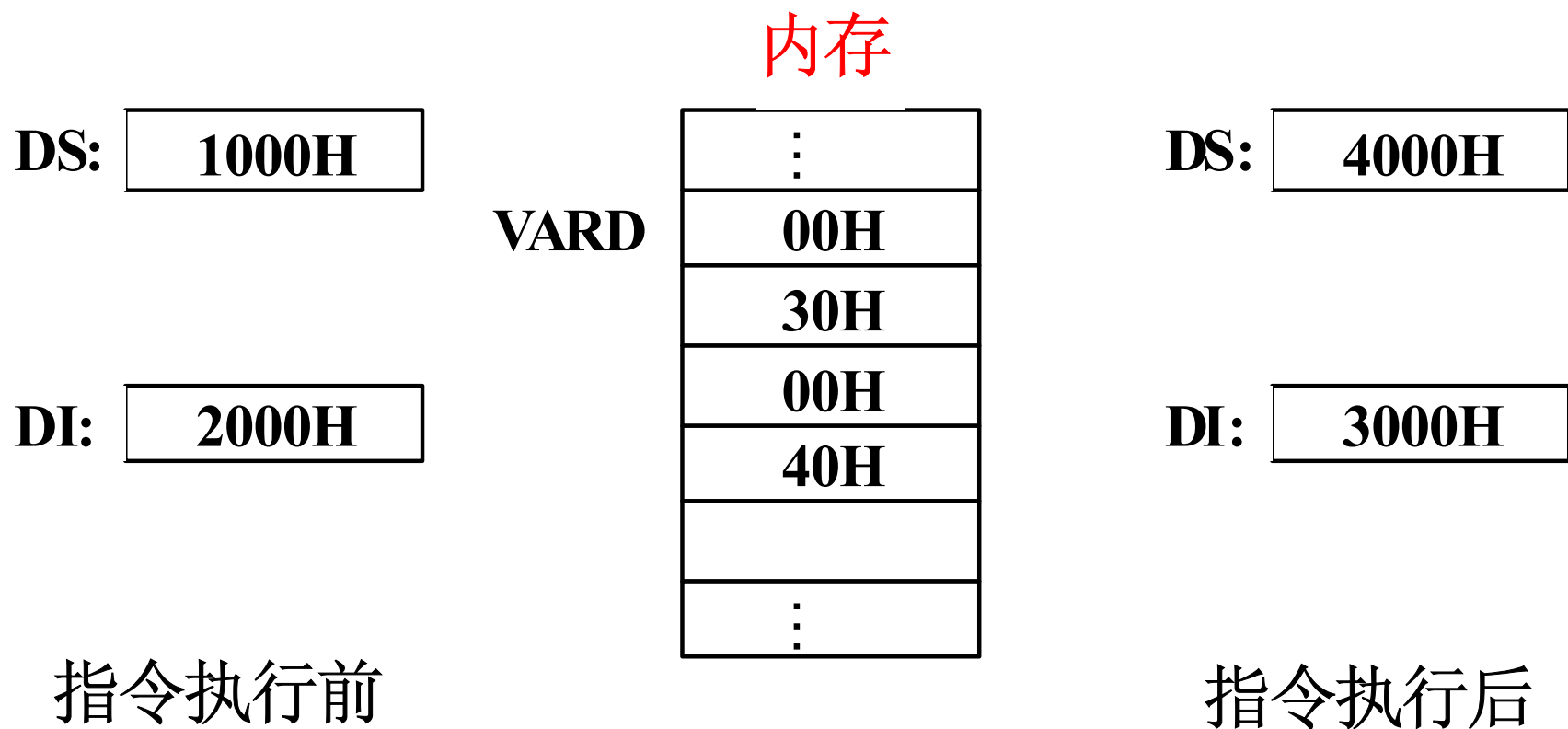
■ **LDS DI, VARD ;DI←VARD内容的低位字,**

DS←VARD内容的高位字

2) 地址传送类指令之LDS



□实例LDS DI, VARD 图示



3) 输入输出类指令 (1/2)



- 实现I/O设备(或I/O接口)与CPU之间的信息传送
- 一个I/O设备(或I/O接口)不仅配置数据寄存器，还会配置状态和控制命令寄存器，有的甚至有多多个数据和控制寄存器
- 通过端口地址区分I/O设备(或I/O接口)及其不同的寄存器
- 8086的端口地址16位，可寻址范围为0~65535
 - 端口地址用8位二进制表示，如8259A中断控制器的控制命令寄存器，端口地址为20H
 - 端口地址用16位二进制表示，如打印机专用适配器的数据寄存器的端口地址为378H



□ I/O指令有两种格式

■ 长格式指令

◆ **IN AL, Ioport; IN AX, Ioport**

◆ **Ioport**直接给出8位端口地址，称为**I/O操作的直接寻址方式**

◆ 输入**Ioport** 中的字节(字)信息，指令代码长度2个字节，端口地址**Ioport**占1个字节，寻址范围：0 ~ 255

3) 输入输出类指令 (2/2)



□ I/O指令有两种格式

■ 长格式指令

■ 短格式指令

◆ IN AL, DX; IN AX, DX

◆ DX寄存器给出端口地址，称为**I/O操作的间接寻址方式**。指令代码长度仅1个字节，寻址范围：0 ~ 65535

■ IO输出指令类似：OUT Ioport, AL; OUT DX, AX



2. 算术运算类指令

- 加法类指令

- 减法类指令

- 乘法类指令

- 除法类指令

1) 加法类指令之ADD



□ 加法指令格式: **ADD DOPD, SOPD**

■ 执行操作: **$DOPD \leftarrow DOPD + SOPD$**

■ 执行后, 影响**CF、ZF、OF、SF、PF、AF**标志位

□ ADD指令举例

■ **ADD AX, BX** ; **$AX \leftarrow AX + BX$**

■ **ADD AL, VARB[SI]** ; **$AL \leftarrow AL + VARB[SI]$**

■ **ADD BX, 1000** ; **$BX \leftarrow BX + 1000$**

■ **ADD WORD PTR [DI], 4567H**
; **$[DI+1, DI] \leftarrow [DI+1, DI] + 4567H$**

1) 加法类指令之ADC (1/2)



□ 带进位加法指令格式: **ADC DOPD, SOPD**

■ 执行操作: $DOPD \leftarrow DOPD + SOPD + CF$

■ 注意: CF是本指令执行前CF的值

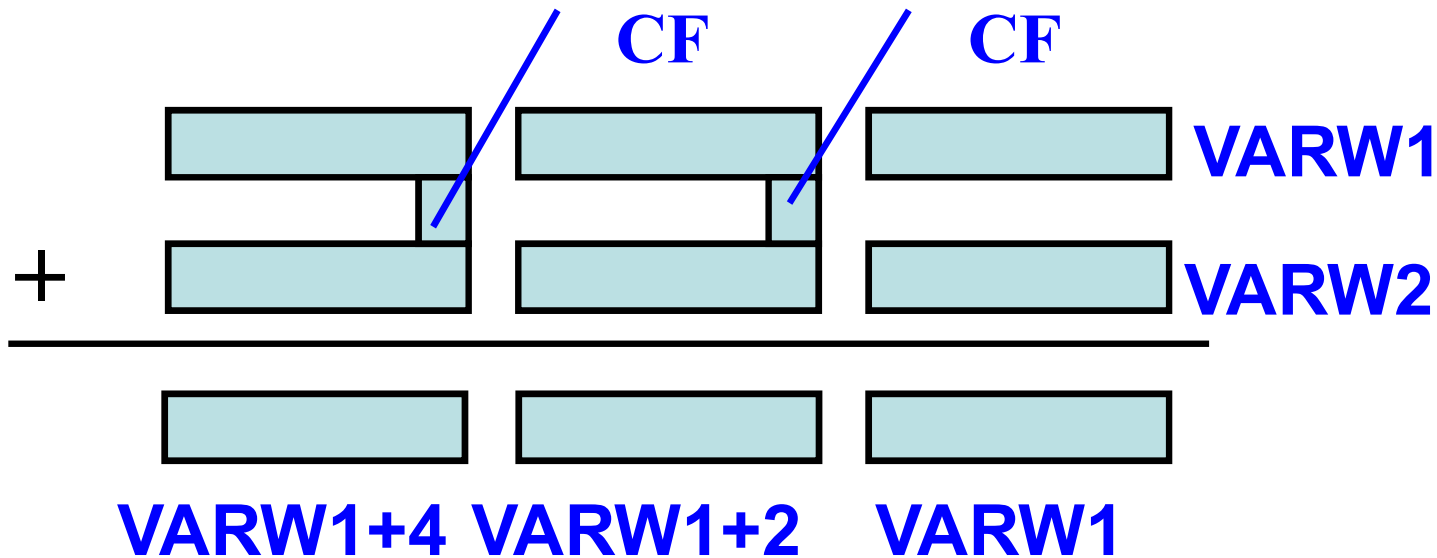
■ 用于多字节运算

■ 执行后, 影响**CF、ZF、OF、SF、PF、AF**标志位

□ ADC指令举例

■ 有两个3字长度的数据, 已分别存放在字变量名为VARW1和VARW2的存储区域中。存放次序是低位字在较小地址中存放, 假定两者之和不超过3字长度, 且存放在VARW1的存储区域中

1) 加法类指令之ADC (2/2)



```
MOV AX, VARW1
ADD AX, VARW2
MOV VARW1, AX
MOV AX, VARW1+2
ADC AX, VARW2+2
MOV VARW1+2, AX
```

```
MOV AX, VARW1+4
ADC AX, VARW2+4
MOV VARW1+4, AX
```

1) 加法类指令之INC



□ 增量指令格式: **INC OPR**

- 执行操作: $\text{OPR} \leftarrow \text{OPR} + 1$
- 用于循环程序中修改指针和循环次数
- 影响标志位ZF、SF、OF、PF和AF，但对进位标志CF没有影响

□ 增量指令举例

- **INC BX** **;BX ← BX + 1**
- **INC VARW[SI]**

2) 减法类指令之SUB、SBB、CMP、DEC



□ 减法指令格式: **SUB** DOPD, SOPD

■ SUB AX, BX ; $AX \leftarrow AX - BX$

■ SUB DX, 5000H ; $DX \leftarrow DX - 5000H$

■ SUB [BX][SI], AX ; $[BX+SI+1, BX+SI] \leftarrow [BX+SI+1, BX+SI] - AX$

□ 带借位减法指令格式: **SBB** DOPD, SOPD

■ SBB AX, CX ; $AX \leftarrow AX - CX - CF$

□ 比较指令格式: **CMP** OPR1, OPR2

■ CMP AX, BX ; $AX - BX$, 只影响标志位, 并不改变AX

□ 减量指令格式: **DEC** OPR

■ DEC AX ; $AX \leftarrow AX - 1$

□ 取负指令格式: **NEG** OPR

■ NEG AX ; $AX \leftarrow \overline{AX} + 1$; 0 - AX

3) 乘法类指令之MUL (1/2)



□ 无符号数乘法指令格式: **MUL** SOPD

■ 执行操作: **字节**操作数: $AX \leftarrow AL * SOPD$

字操作数: $DX, AX \leftarrow AX * SOPD$

■ 执行后, 影响**CF**、**OF**标志位

□ 无符号数乘法举例:

■ **MUL BL** ; $AX \leftarrow AL * BL$, 乘法一个操作数送AL中

■ **MUL BX** ; $DX:AX \leftarrow AX * BX$, 乘法一个操作数送AX中

□ 有符号数乘法指令格式: **IMUL** SOPD

■ 该指令除操作数是有符号数外, 其它同MUL指令

3) 乘法类指令之**MUL** (2/2)



例1：已知BH和CX中存放着两个无符号数，编制求其乘积的程序段。

■ **MOV BL, BH**

■ **MOV BH, 0** ;BX←BH中的8位无符号数变换为16位无符号数

■ **MOV AX, CX** ;AX←CX中的16位无符号数

■ **MUL BX** ;DX:AX←乘积

例2：设AL=0FFH，BL=1，分别执行下列指令，会得到不同结果。

MUL BL ;AX= ?

IMUL BL ;AX= ?

4) 除法类指令之DIV



□ 无符号数除法指令格式: **DIV SOPD**

■ 执行操作: **字节**操作数: $AL \leftarrow AX \div \text{SOPD的商}$

$AH \leftarrow AX \div \text{SOPD的余数}$

字操作数: $AX \leftarrow (DX, AX) \div \text{SOPD的商}$

$DX \leftarrow (DX, AX) \div \text{SOPD的余数}$

■ 执行后, 不影响**任何标志位**

□ 无符号数除法举例:

■ **DIV BL** ; $AL \leftarrow (AX \div BL)$ 的商, $AH \leftarrow (AX \div BL)$ 的余数

■ **DIV BX** ; $AX \leftarrow (DX:AX \div BX)$ 的商, $DX \leftarrow (DX:AX \div BX)$ 的余数

■ **被除数**送AX或DX:AX中

□ 有符号整数除法指令格式: **IDIV SOPD**

□ 被除数的位数应是除数位数的2倍长度

■ 有符号数的扩展

- ◆ 使用数据类型转换指令CBW或CWD

■ 无符号数的扩展

- ◆ 扩展的高位置零即可

□ 需注意商溢出和“0”作除数的问题

- 当除数为字节/字时，若被除数的高8位/高16位的绝对值大于或等于除数的绝对值，商就会溢出，即商会超出相应寄存器的表示范围，产生溢出中断。这时，可通过使用数据类型转换指令，避免溢出错误

例：假定寄存器AX、BL分别存放着无符号数65534和4，试编制求两数商和余数的程序段，要求将它们分别存储到字变量X和Y中。

```
MOV DX, 0      ;扩展被除数为双字
MOV BH, 0      ;扩展除数为字
DIV BX         ;作除法：DX:AX÷BX
MOV X, AX      ;存商
MOV Y, DX      ;存余数
```

5) 符号扩展指令



□ 符号扩展指令，隐含对AL或者AX进行符号扩展，不影响标志位

□ CBW指令

■ 执行操作：AX ← AL

◆ 若(AL)的最高位有效位为0，则(AH)=00H;

◆ 若(AL)的最高位有效位为1，则(AH)=0FFH;

□ CWD指令

■ 执行操作：(DX,AX) ← AX

◆ 若(AX)的最高位有效位为0，则(DX)=0000H;

◆ 若(AX)的最高位有效位为1，则(DX)=0FFFFH;

例：设(AX)=0BA45H，则：

① CBW ;(AX)= ?

② CWD ;(DX)= ?



3. 逻辑类指令

- 逻辑运算类指令

- 移位类指令

- 串操作类指令

1) 逻辑运算类指令之NOT



□ 逻辑非指令

■ 指令格式及功能

◆ NOT DSOPD ;DSOPD $\leftarrow \overline{DSOPD}$

◆ 该指令对标志位无影响。

■ 实例

◆ NOT AX ;AX $\leftarrow \overline{AX}$

◆ NOT VARW[BX]

;VARW[BX+1、BX] $\leftarrow \overline{VARW[BX+1、BX]}$

1) 逻辑运算类指令之AND等



□ 逻辑与、逻辑或、逻辑异或、测试指令

■ 指令格式及功能

◆ **AND** DOPD, SOPD ;DOPD←DOPD \wedge SOPD

◆ **OR** DOPD, SOPD ;DOPD←DQPD \vee SOPD

◆ **XOR** DOPD, SOPD ;DOPD←DOPD \oplus SOPD

◆ **TEST** DOPD, SOPD ;DOPD \wedge SOPD

■ 标志位CF和OF被清零，标志位 PF、ZF和SF受影响正常变化，AF随机

4) 串操作类指令



- 串操作指令的操作对象可以是字符序列，也可以是任意的字节或字序列
- 串操作类指令共同的特点是：
 - **源串**的偏移首地址存放在变址寄存器**SI**中，在无段超越前缀的情况下，段基址取自**DS**段寄存器
 - **目的串**的偏移首地址存放在变址寄存器**DI**中，因它**不允许**使用段超越前缀，所以段基址总是取自**ES**段寄存器

- 用在串操作指令之前，例：REP MOVSB
- 使紧接其后的串操作指令重复执行，重复次数由CX内容决定
 - 重复之前，先判CX：若CX=0，则串操作指令一次也不被执行；否则重复，每重复一次，CX减1，直到CX=0止
- 重复执行期间，每次重复操作完成，可以被中断，且中断返回后，串操作指令继续重复执行剩余的次数



4. 程序控制类指令

- 转移类指令

- 循环控制类指令

- 子程序类指令

□特点

- 改变程序顺序执行的规律，跳转到程序员安排的目的指令开始执行

□实现方式

■远程(FAR)转移

- ◆同时改变段基址CS和段内偏移地址IP

■近程(NEAR)转移

- ◆仅改变段内偏移地址IP
- ◆相对寻址的近程转移中，把转移范围为 $-128 \sim +127$ 的转移指令独立出来，称为短程 (SHORT) 转移

1) 转移类指令之JMP



□ 无条件转移指令JMP

■ **JMP L1** ;假定L1为标号

■ **JMP CX**

■ **JMP WORD PTR [BX]**

■ **JMP VARD** ;假定VARD为双字变量

■ 操作数为标号时，直接跳转到标号处；为寄存器或内存时，跳转到寄存器或内存的**内容**为目的地址处

1) 转移类指令之JE等



□ 条件转移指令

- 条件转移指令根据**标志寄存器的状态**决定是否发生转移。若满足条件，转向**短标号**所指指令，否则顺序执行
- 条件转移指令对标志位无影响
- 条件转移指令都是**段内相对寻址短程转移指令**

条件转移指令列表 (1/2)



指令格式			功能描述
用于 单一 标志 位的 判断	JE/JZ	短标号	ZF 标志位为 1 ，则转移至短标号处
	JNE/JNZ	短标号	ZF 标志位为 0 ，则转移至短标号处
	JC	短标号	CF 标志位为 1 ，则转移至短标号处
	JNC	短标号	CF 标志位为 0 ，则转移至短标号处
	JO	短标号	OF 标志位为 1 ，则转移至短标号处
	JNO	短标号	OF 标志位为 0 ，则转移至短标号处
	JS	短标号	SF 标志位为 1 ，则转移至短标号处
	JNS	短标号	SF 标志位为 0 ，则转移至短标号处
	JP/JPE	短标号	PF 标志位为 1 ，则转移至短标号处
	JNP/JPO	短标号	PF 标志位为 0 ，则转移至短标号处

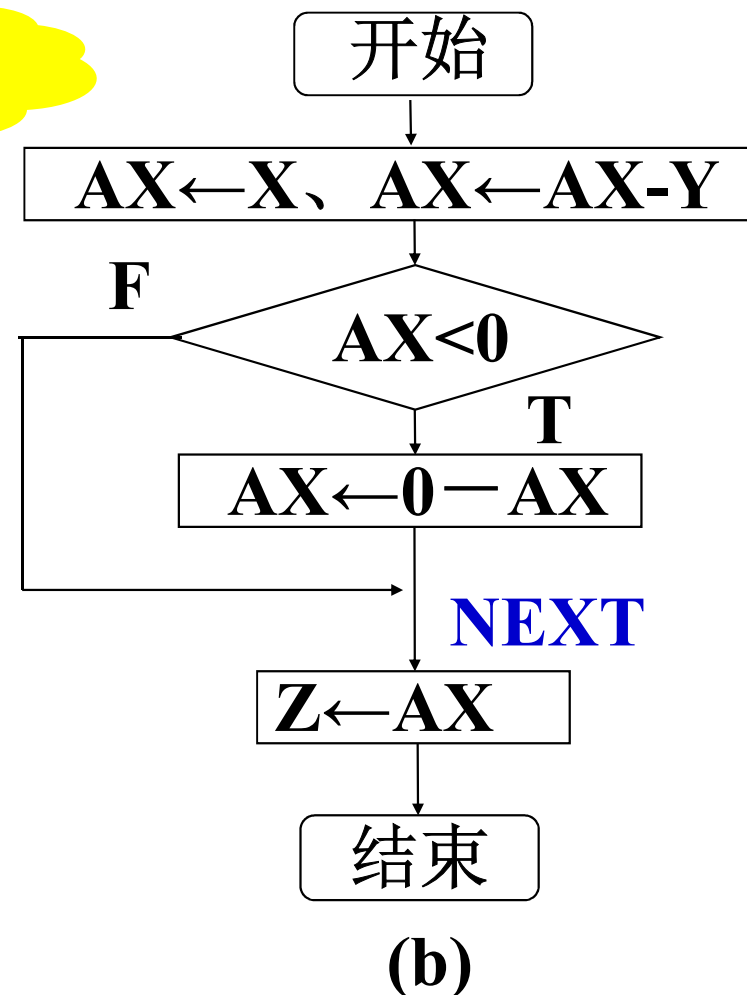
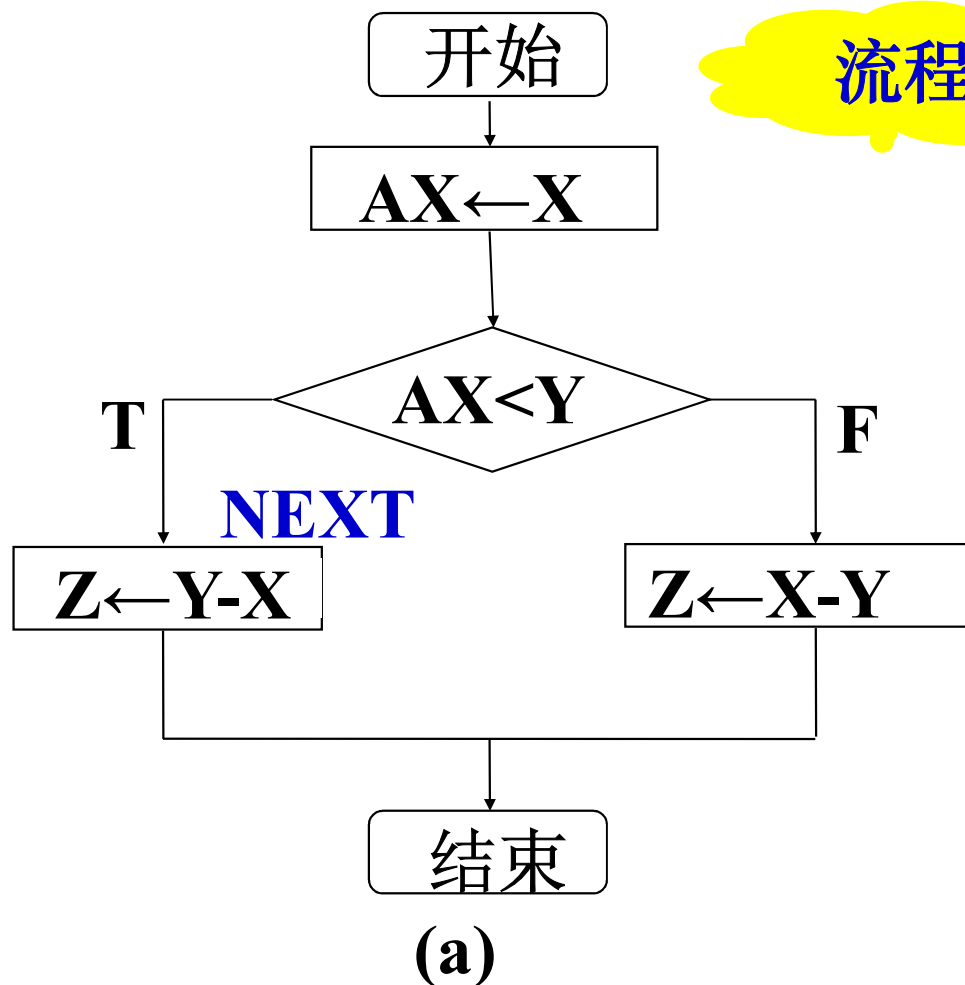
条件转移指令列表 (2/2)



指令格式		功能描述	备注
用于有符号数	JL/JNGE 短标号 JLE/JNG 短标号 JNL/JGE 短标号 JNLE/JG 短标号	小于 /不大于且不等于转移 小于或等于/不大于则转移 不小于/大于或等于转移 不小于且不等于/大于转移 至短标号处	OF≠SF OF≠SF或ZF=1 OF=SF OF=SF且ZF=0
用于无符号数	JB/JNAE 短标号 JBE/JNA 短标号 JNB/JAE 短标号 JNBE/JA 短标号	低于 /不高于且不等于转移 低于或等于/不高于转移 不低于/高于或等于转移 不低于且不等于/高于转移 至短标号处	CF=1 CF=1或ZF=1 CF=0 CF=ZF=0

例1：已知字的无符号数 x 、 y ，试编制求： $z = |x - y|$ 的程序段。假设 X 、 Y 、 Z 为数据的变量名称。

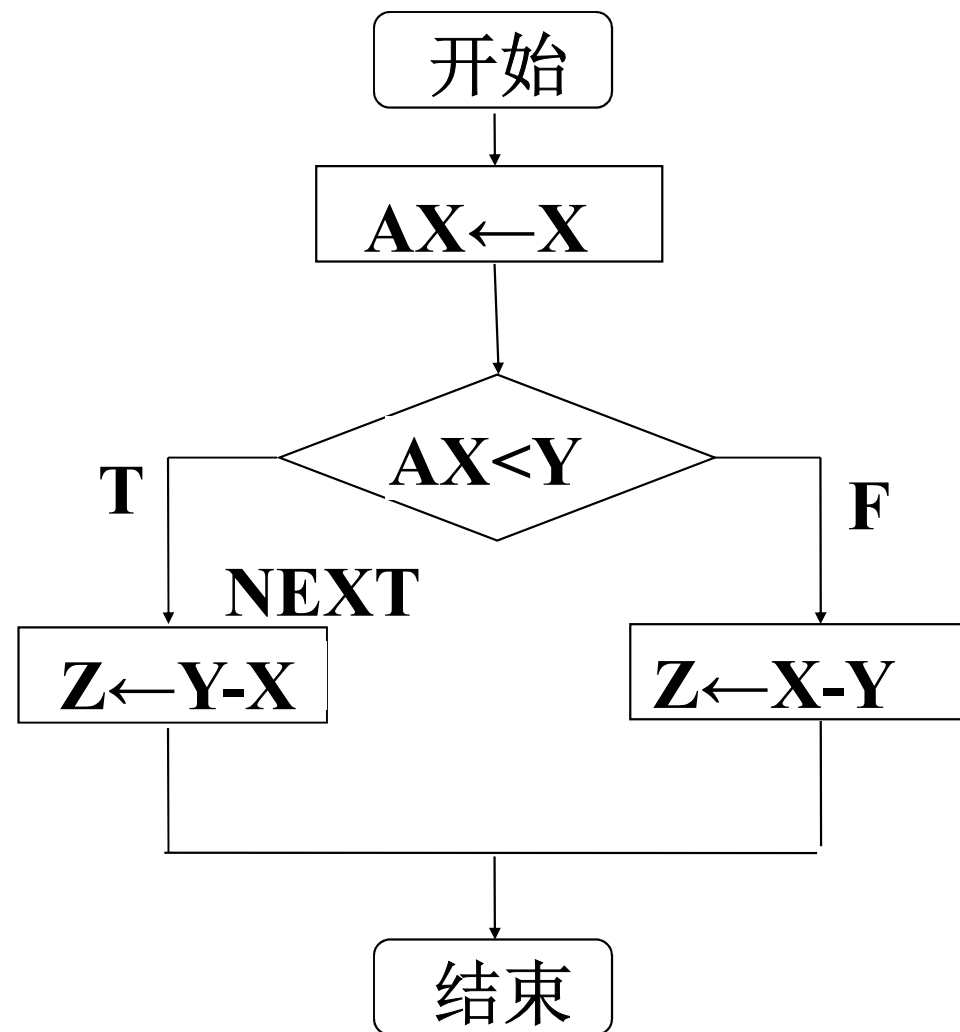
流程图



例1的方法一程序



```
MOV AX, X
CMP AX, y
JC NEXT
SUB AX, y
JMP DONE
NEXT: MOV AX, y
      SUB AX, X
DONE: MOV Z, AX
```

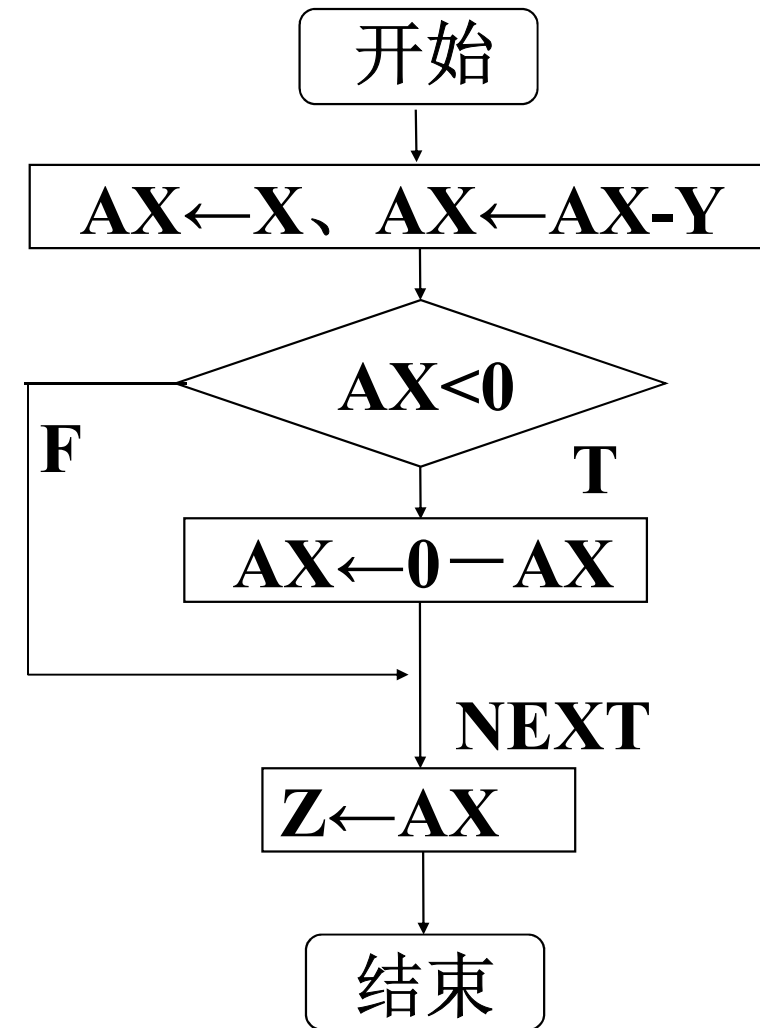


(a)

例1的方法二程序



```
MOV AX, X
SUB AX, y
JNC NEXT
NEG AX
NEXT: MOV Z, AX
```



(b)

2) 循环控制类指令



□ 8086有四种循环控制指令

- LOOP指令

- LOOPZ/LOOPE指令

- LOOPNZ/LOOPNE指令

- JCXZ指令

□ 操作数和条件转移指令一样，必须是**短标号**

□ 使用循环控制指令前，必须先把循环次数送入**CX**

□ 循环控制类指令都不影响标志位

2) 循环控制类指令之LOOP



□ 无条件循环指令

■ 指令格式：LOOP 短标号

■ 功能

◆ 先将CX减1，然后判断CX。若CX≠0，转移到短标号所指的语句，否则顺序执行

◆ 一条LOOP指令，相当于以下两条指令的组合：

DEC CX

JNZ 短标号

■ 注意

◆ 如果在循环准备时将“1”送到CX中，将一次也不会执行循环体；若将“0”送CX，将会循环 2^{16} 次

3) 子程序类指令



□ 子程序的概念是程序设计最重要的创新之一

- 一些经常使用的、能独立完成某一特定功能的程序段，常常将其独立出来作为子程序，在需要时由主程序调用，而不必多次重复编写

- 子程序的概念也是模块化程序设计的基础

□ 除了用户自己编写的子程序外，现代计算机系统往往也提供大量的通用子程序

□ 为实现调用程序和子程序之间的转返，8086微机设置了调用和返回两个指令

3) 子程序类指令之CALL



□调用指令

■指令一般形式

◆ **CALL NEAR**过程名/ **FAR**过程名/ **REG16/MEM16/**
MEM32

■分为四种情况

- ◆ 段内直接寻址近程调用
- ◆ 段间直接寻址远程调用
- ◆ 段内间接寻址近程调用
- ◆ 段间间接寻址远程调用

3) 子程序类指令之RET



□ 返回指令

■ 指令格式: RET

- ◆ 段内(近程)或段间(远程)返回, 符号指令格式都为RET, 但对应机器指令不同, 代表的操作也有差别

■ 功能

- ◆ 控制子程序返回到主程序调用处继续执行
- ◆ **近程**返回指令将栈顶字内容出栈送**IP**
- ◆ **远程**返回指令将栈顶双字内容出栈送**CS和IP**

■ 执行操作

- ◆ 近程返回: $IP \leftarrow (SP+1)(SP)$ 、 $SP \leftarrow SP+2$
- ◆ 远程返回: $CS \leftarrow (SP+3)(SP+2)$ 、 $IP \leftarrow (SP+1)(SP)$ 、 $SP \leftarrow SP+4$



3.1 8086指令类型

3.2 汇编语言源程序的结构

3.3 伪指令

3.4 子程序设计

3.5 系统功能调用

3.2 汇编语言源程序的结构



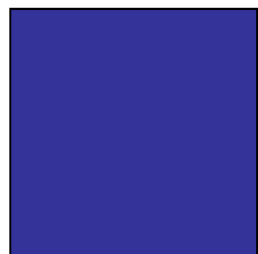
数据段



堆栈段



附加段



代码段

说明:

①各段顺序无关

②除代码段外，均可缺省

③可有若干个数据段，若干个代码段

END [标号]

数据段	{	DATA SEGMENT	
		BUF1 DB 34H	;变量定义
		BUF2 DB 2AH	
		SUM DB ?	
		DATA ENDS	
代码段	{	CODE SEGMENT	
		ASSUME CS:CODE, DS:DATA	
		START: MOV AX, DATA	
		MOV DS, AX	; 段寄存器赋值
		MOV AL, BUF1	
		ADD AL, BUF2	
		MOV SUM, AL	
		MOV AH, 4CH	
		INT 21H	; 程序退出
		CODE ENDS	
END START			

3.2 汇编语言源程序的结构



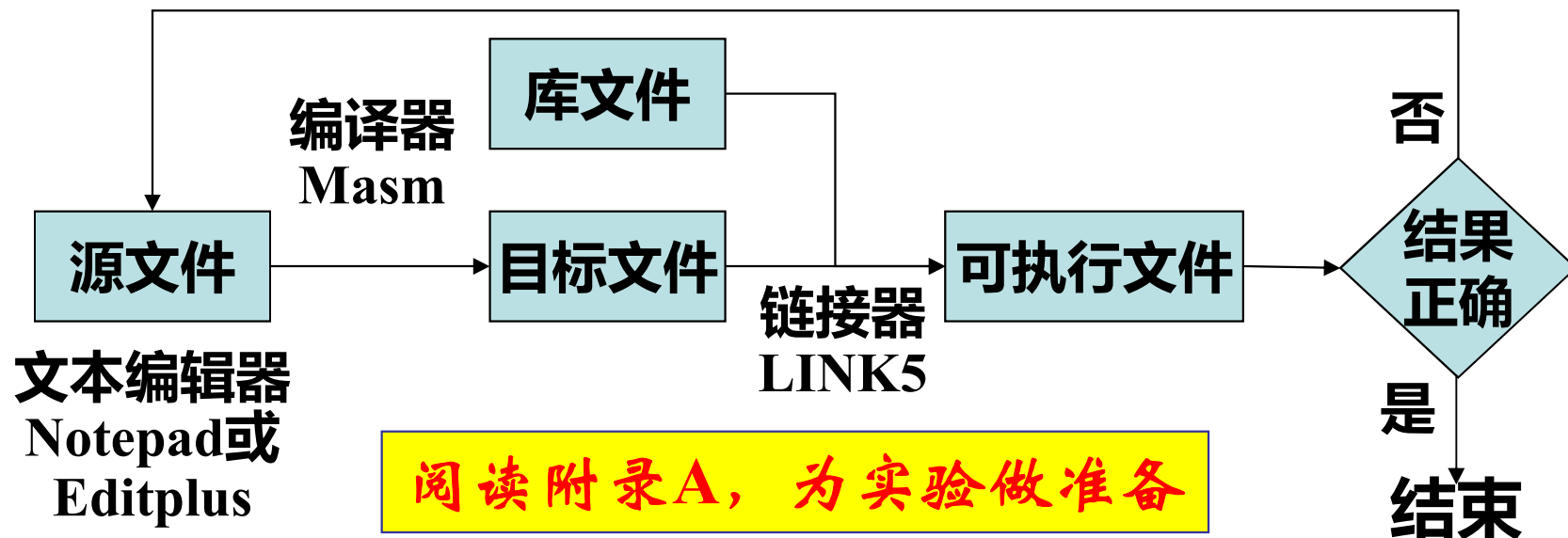
□ 编译、链接和运行程序

步骤1：用文本编辑器编辑汇编语言源程序(.asm)

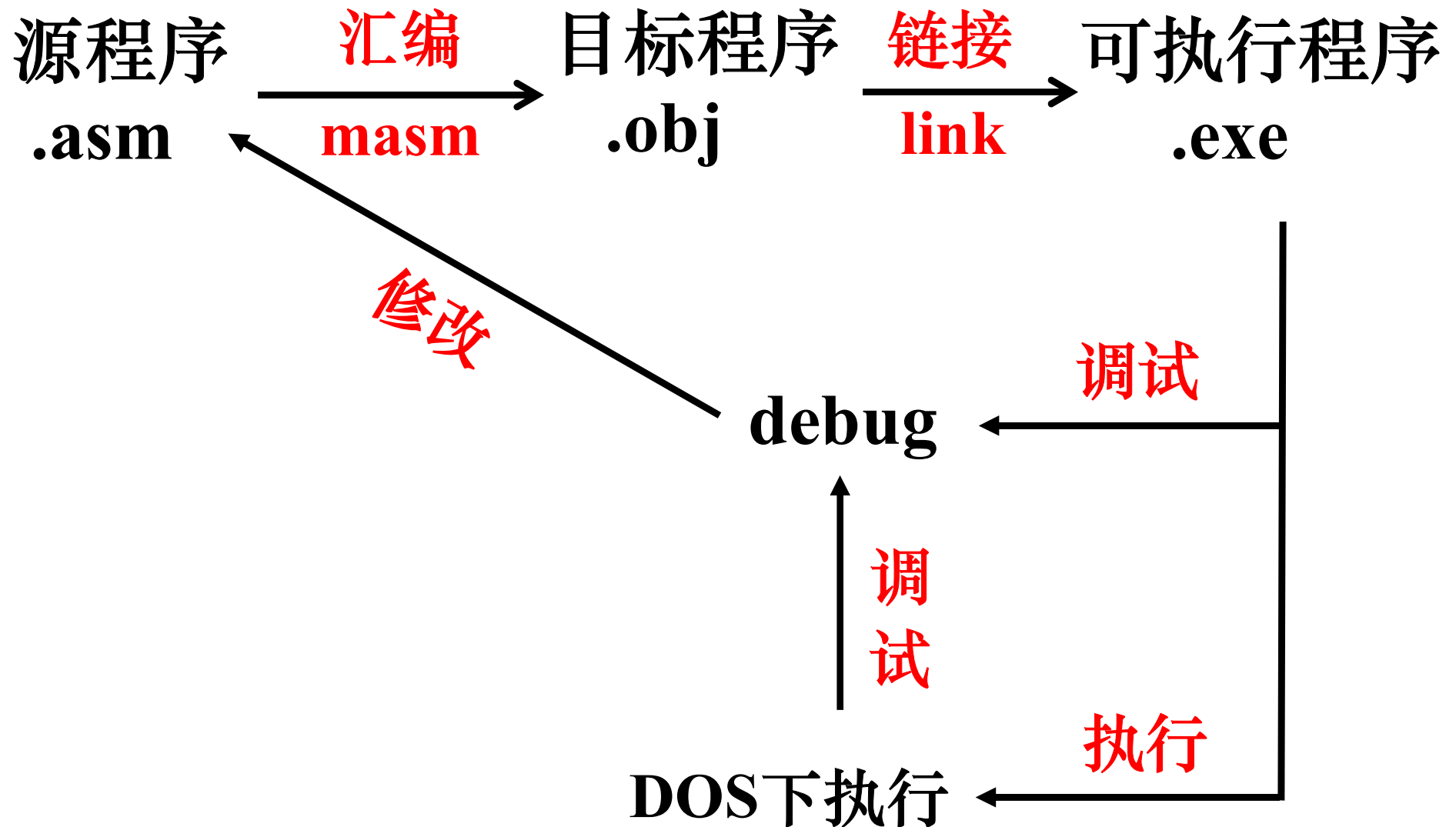
步骤2：编译源程序，生成目标文件(.obj)

步骤3：将目标文件和库文件(.inc)链接在一起，得到可执行文件(.exe)

步骤4：执行.exe文件，检查结果是否正确。若不正确转1



汇编语言程序编译链接过程



3.2 汇编语言源程序的结构



例：三个整数相加，和的结果通过显示器输出。

①C语言源程序：

```
#include <stdio.h> //函数printf定义在头文件stdio.h中
int main()          // 声明函数main
{
    // 函数main开始
    int a, b=100, c=50, d=-79; // 定义要处理的数据
    a = b+c+d;                // 处理数据
    printf("a=%d\n", a);      // 调用函数printf
    return 0;                 // 返回
}                              // 函数main结束
```

3.2 汇编语言源程序的结构



②8086汇编语言源程序：

```
include    irvine.inc      ; DumpRegs定义在irvine.inc中
.data
    A      dw    100, 50, -75    ; 定义存储变量
.code      ; 定义代码段
main      PROC      ; 定义函数main
    mov     ax, A      ; AX = 100
    add     ax, A+2     ; AX = AX + 50
    sub     ax, A+4     ; AX = AX - 75
    call    DumpRegs   ; 调用子程序DumpRegs
                    ; 显示AX内的值
    .exit    ; 退出
main      ENDP      ; 函数main结束
END       main      ; 源程序结束，指明程序入口
```

- 这两个源程序的功能相同，在结构上也有很多相同的地方
 - 引用其它文件的语句(**#include**和**include**)
 - 函数定义语句(指明函数的名称，函数的开始和结束等)
 - 定义变量(**int**和**dw**)
 - 进行数据处理(完成加减运算)
 - 调用其它函数输出结果(**printf**和**call**)
 - 注释(**//**和**;**后的语句)
 - 函数返回语句(**return**和**exit**)

3.2 汇编语言源程序的结构



□但结构上也有一些不同的地方

■汇编源程序中，增加了.code (定义代码段) 和.data(定义数据段)的语句

■汇编源程序表示源程序结束的语句(end)

□程序设计 = 符合特定**模板** “**算法+数据结构**”

模板——程
序设计语言
对源程序结
构的规定

数据结构——
变量、常量、
符号...及其定
义

算法——处理数据的
运算(加减、输出),
如C语言的语句序列
或汇编的指令序列

□ 汇编语言语句格式

① **指令语句**：完成操作功能，能翻译成机器指令

指令语句格式：

[符号名:]指令助记符 [目的操作数[,源操作数]][;注释]

② **伪指令语句**：为汇编程序在翻译源程序时提供有关信息

伪指令语句格式：

[符号名] 伪操作 [操作数[,操作数,...]] [; 注释]

③ **宏指令语句**：由若干条指令语句组成的语句

3.2 汇编语言源程序的结构



□ 汇编语言源程序模板

.model small ; 定义存储模式

include *.inc ; 引用头文件

.data ; 定义数据段

; 在此插入变量、常量、符号...的定义

.stack ; 定义堆栈段

.code ; 定义代码段

.startup ; 定义程序入口

; 在此插入指令

.exit; 返回

; 在此插入其它数据段/代码段或其它子程序

End ; 源程序结束

伪指令



3.1 8086指令类型

3.2 汇编语言源程序的结构

3.3 伪指令

3.4 子程序设计

3.5 系统功能调用

□伪指令：帮助汇编程序正确翻译源程序的命令，本身不生成任何机器指令

- 处理器选择伪指令

- 数据定义和存储器分配伪指令

- 表达式赋值伪指令

- 地址计数器与对准伪指令

3.3 伪指令



□ 简化段定义伪指令

□ 符号定义伪指令

□ 数据定义伪指令

□ 过程定义伪指令

□ 模块定义伪指令



3.1 8086指令类型

3.2 汇编语言源程序的结构

3.3 伪指令

3.4 子程序设计

3.5 系统功能调用

过程调用实例



例: `int i;` ← `i` 是全局静态变量

`void set_array(int num)`

`{`
`int array[10];` ← `array` 数组是局部变量

`for (i = 0; i < 10; i++) {`

`array[i] = compare(num, i); }`

`set_array` 是调用过程
`compare` 是被调用过程

`}`
`int compare (int a, int b)`

`{`

`if (sub(a, b) >= 0)` ← `compare` 是调用过程
`return 1;`

`else`
`return 0;`

`compare` 是调用过程
`sub` 是被调用过程

`}`
`int sub (int a, int b)`

`{`

`return a-b;`

`}`

□ 过程控制是另一类无条件转移指令，与jmp指令相比

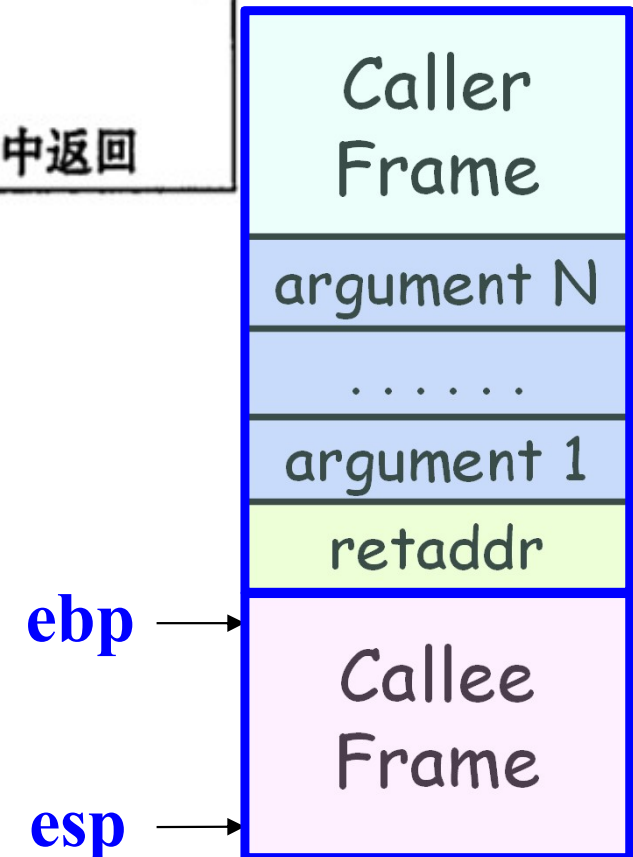
指令		描述
call	<i>Label</i>	过程调用
call	<i>*Operand</i>	过程调用
ret		从过程调用中返回

■ 相同点：

- ◆ 控制从程序的一部分跳转到另一部分

■ 不同点：

- ◆ 返回(Return)
- ◆ 传递数据 (arguments, return values)
- ◆ 局部变量(Local variable)
- ◆ 寄存器(Registers)





3.1 8086指令类型

3.2 汇编语言源程序的结构

3.3 伪指令

3.4 子程序设计

3.5 系统功能调用

3.5 系统功能调用



➤ 概念

◆DOS操作系统包含了很多涉及设备驱动和文件管理等方面的子程序——相对独立的程序模块，**配上不同的编号(功能类型码)**，程序员通过编号可方便的调用这些子程序

➤ 分类

◆存储管理

◆磁盘的读写与管理

◆基本输入输出(键盘、打印机、显示器等)

◆处理时间日期等

➤ 入、出口参数

◆大都采用约定寄存器法

➤关于DOS软中断

- ◆包含多个子功能的功能包，各子功能用**功能号区分**，用软中断指令调用，**中断类型码为21H**

➤DOS功能调用的基本步骤

- ◆将调用参数装入寄存器
- ◆将功能号（子程序号）装入AH寄存器
- ◆按中断类型号调用软中断指令（INT 21H）
系统功能调用总表

调用格式：

MOV AH, 功能号

<置相应参数>

INT 21H

➤ 单字符输入 (1号功能调用)

◆ 等待接收键盘输入一字符

- ◆ 若键入的字符不是控制字符，则将相应字符编码送AL且在显示器上显示出来。
- ◆ 若键入的字符是控制字符，则实现其控制功能。

◆ 注意

- ◆ 该功能一直等待程序员按键，若不按键程序永远等待

➤ 入口参数：无

➤ 出口参数：AL ← 键入字符的ASCII码

1) 带回显的键盘输入 (2/2)



□ 实例

■ 下列语句等待键盘输入
变量CHAR中:

MOV AH, 1 ;指出是

INT 21H ;进入系

MOV CHAR, AL ;

```
GET KEY:  MOV AH, 1
          INT 21H
          CMP AL, 'Y'
          JZ  YES
          CMP AL, 'N'
          JZ  NO
          JMP GET KEY
YES:      :
NO:       :
```

2) 输出字符到显示器



➤ 单字符输出 (2号功能调用)

◆ 在显示器上显示输出一个字符

➤ 入口参数: DL ← 要输出字符或它的ASCII码

➤ 出口参数: 无

➤ 实例

◆ 在显示器上将显示字符 'A'

例:

```
MOV AH, 2
```

```
MOV DL, 41H
```

```
INT 21H
```

MOV DL, 'A'	;送准备输出的字符, 即入口参数
MOV AH, 2	;指出是2号调用
INT 21H	;进入系统功能调用总入口
	;显示器输出一字符

2) 输出字符到显示器



```
.CODE
START:
    mov AH, 1
    INT 21H

    mov DL, 'A'
    mov AH, 2
    INT 21H
```

2) 输出字符到显示器



17:		↑	AX	=	0142
18:			BX	=	0000
19:	.CODE		CX	=	0000
20:	START:		DX	=	0041
21:	mov AH,1		SP	=	0200
22:	INT 21H		BP	=	0000
23:			SI	=	0000
24:	mov DL,'A'		DI	=	0000
25:	mov AH,2		DS	=	12CE
26:	INT 21H		ES	=	12CE
27:			SS	=	12E8
28:			CS	=	12DE

17:		↑	AX	=	0041
18:			BX	=	0000
19:	.CODE		CX	=	0000
20:	START:		DX	=	0041
21:	mov AH,1		SP	=	0200
22:	INT 21H		BP	=	0000
23:			SI	=	0000
24:	mov DL,'A'		DI	=	0000
25:	mov AH,2		DS	=	12CE
26:	INT 21H		ES	=	12CE
27:			SS	=	12E8
28:			CS	=	12DE
29:	MOV SI,4;		IP	=	000A

3) 输出字符到打印机



➤ 5号功能调用

◆ 命令打印机输出一个字符

➤ 入口参数: **DL** ← 要输出字符或它的ASCII码

➤ 出口参数: 无

➤ 实例

◆ 在打印机上输出字符 ‘A’

MOV DL, 'A' ;送准备输出的字符, 即入口参数

MOV AH, 5 ;指出是5号调用

INT 21H ;进入系统功能调用总入口, 打印机输出一字符

4) 输出以 “\$”结尾的字符串



➤ 9号功能调用

◆ 将以 “\$”结尾的字符串输出到显示器， “\$”不输出

➤ 入口参数

◆ DS:DX ← 要输出字符串始地址指针

➤ 出口参数：无

AH ← 功能号**09H**

DS: DX ← 待输出字符串的偏移地址

INT 21H

5) 输入字符串 (1/2)



➤ 输入字符串 (10号功能调用)

◆ 等待接收键盘输入字符串并将它们送程序员指定的缓冲区

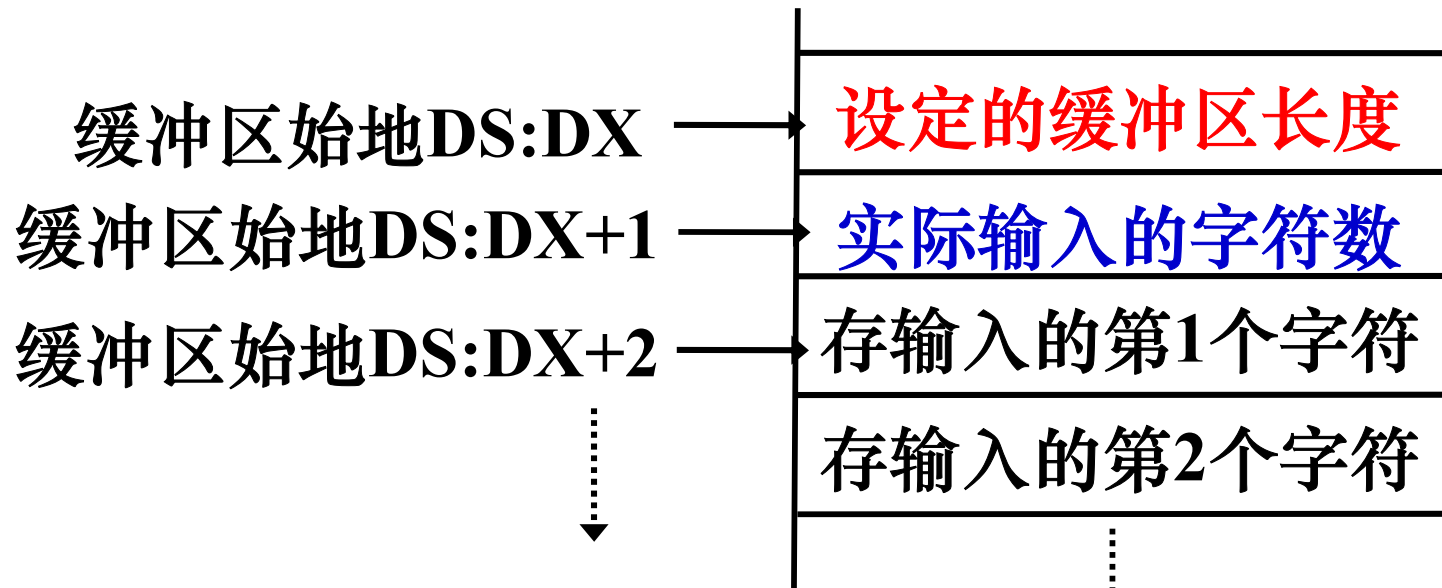
➤ 入口参数:

◆ $DS:DX \leftarrow$ 缓冲区始地址

➤ 出口参数:

◆ $DS:DX+2 \leftarrow$ 存放键入字符串的第一个字符

5) 输入字符串 (2/2)



- 缓冲区第1字节为缓冲区的最大长度，由程序员指定
- 缓冲区第2字节为实际输入字符数，不含回车键，由系统填入
- 缓冲区第3字节开始存输入的字符串
- 字符串以回车键结束，回车键是接收字符串的最后1个字符
- 若输入的字符超过缓冲区最大长度时，则随后输入的字符被丢掉并且响铃，直至键入回车键止

输入字符串程序段

```
DAT1 DB 20, ?, 20 DUP ( ? )
```

在数据段
中定义

缓冲区长度
的字符数
第1个字符

```
LEA
```

```
MOV
```

```
INT
```

定义后的输入缓冲
区初始状态:

- 缓冲区第
- 缓冲区第
- 字符串以
- 若输入的
- 丢掉并上

存储输入
的字符



20个字节

6) 返回操作系统 (DOS) 功能



➤ 功能号

◆ 4CH

➤ 调用格式:

MOV AH, 4CH

INT 21H

➤ 功能:

◆ 程序执行完该2条语句后能正常返回OS

◆ 常位于子程序结尾处

例1：编写由键盘提示输入初始数据，并检查输入数据正确与否？若正确，输入的数据送打印机输出；若不正确，重新输入直至正确并送打印机输出。

.MODEL SMALL

.STACK

.DATA

CR EQU 0DH ;回车字符(回车键)

LF EQU 0AH ;换行字符(换行键)

;定义字符串以便提示用

MESG1 DB 'PLEASE INPUT DATA',CR,LF, '\$'

MESG2 DB 'INPUT CORRECT? Y/N',CR,LF, '\$'

MESG3 DB 'INPUT ERROR AGAIN!',CR,LF, '\$'

BUFFER DB 200 ;规定缓冲区长度

DB ? ;留空由系统填入实际输入的字符数

DB 200 DUP ('\$')



```
.CODE
.STARTUP
AGAIN: LEA DX, MSG1
      MOV AH, 9
      INT 21H
      LEA DX, BUFFER
      MOV AH, 10
      INT 21H
      LEA DX,BUFFER+2
      MOV AH, 9
      INT 21H
```

```
      LEA DX,MSG2
      MOV AH,9
      INT 21H
      MOV AH,1
      INT 21H
      CMP AL,'Y'
      JZ RIGHT
      LEA DX,MSG3
      MOV AH,9
      INT 21H
      JMP SHORT AGAIN
RIGHT: LEA DI,BUFFER+2
```

MOV CX, WORD PTR BUFFER+1

;CX←要打印信息的个数

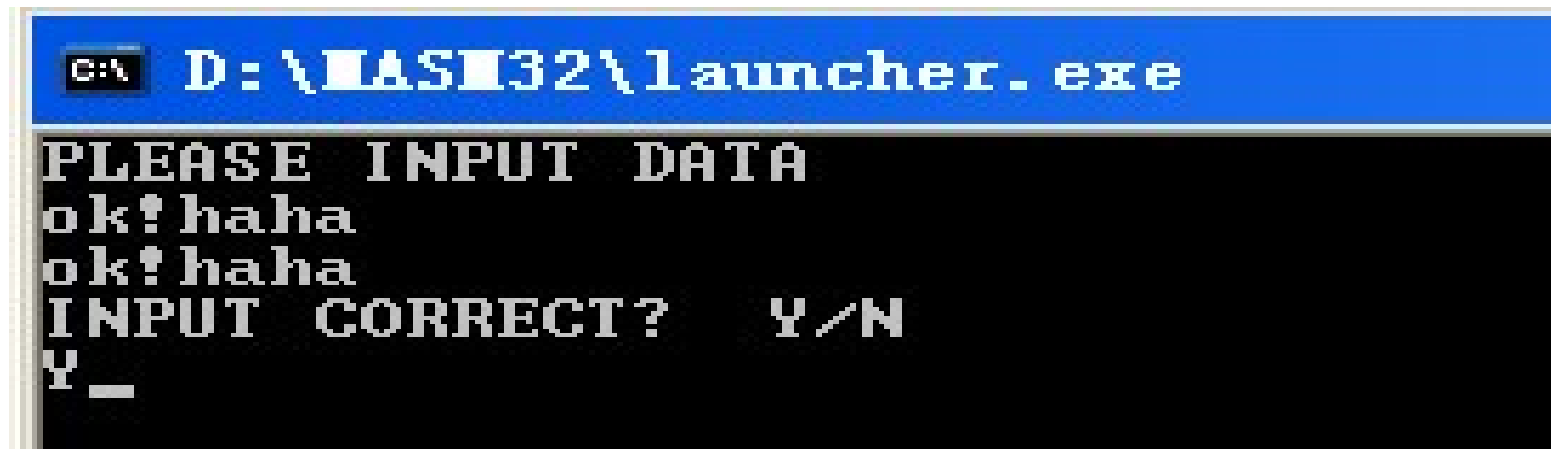
PRINT: MOV DL, [DI]

MOV AH, 5

INT 21H **;打印机输出**

INC DI

LOOP PRINT



```
C:\ D:\MASM32\launcher.exe
PLEASE INPUT DATA
ok!haha
ok!haha
INPUT CORRECT?  Y/N
Y_
```

以下数据区在内存中是如何存储的？

DATA SEGMENT

NAMES DB 'TOM..' ,20

DB 'CATE' ,25

DATA ENDS

每个字符在内存中以
ASCII码表示，占用1B

示例2

变量在内存中的存储



以下数据区在内存中是如何存储的?

DATA SEGMENT

NAMES DB 'TOM..' ,20

DB 'CATE' ,25

DATA ENDS

NAME

54

'T'

4F

'O'

4D

'M'

2E

.

2E

.

14

43

'C'

41

'A'

54

'T'

45

'E'

19

谢 谢!

