哈爾濱工業大學(深圳)
HARBIN INSTITUTE OF TECHNOLOGY, SHENZHEN

汇编语言程序设计

第4&5讲：8086/8088的指令系统

裴文杰

第4&5讲：8086/8088的指令系统

8086/8088指令概述

数据传送指令

算术指令

逻辑指令

串处理指令

处理机控制与杂项操作指令

wenjiecoder@outlook.com

重点关注：

- 指令的汇编格式及特点
- 指令的基本功能
- 指令的执行对标志位的影响
- 指令的特殊要求

wenjiecoder@outlook.com

8086/8088指令概述

数据传送指令

算术指令

逻辑指令

串处理指令

处理机控制与杂项操作指令

wenjiecoder@outlook.com

8086/8088指令概述

计算机是通过执行指令序列来解决问题的，每条指令代表一种基本功能，这些基本功能是在程序运行期间由计算机硬件来实现的。

每种计算机都有对应于自己硬件的一组指令集，供用户使用，这组指令集就是这种计算机的指令系统。

本课程学习基于80x86 CPU的指令系统。

wenjiecoder@outlook.com

指令格式　　操作码　　操作数

例　　ADD　　AL，10H

(1) 操作码

指明CPU要执行什么样的操作。

是一条指令必不可少的部分，用助记符表示。

按功能指令分六类：

- 数据传送
- 算术运算
- 逻辑运算
- 串操作
- 控制转移
- 处理机控制

(2) 操作数

指明参与操作的数据或数据所在的地方。

了解操作数的来源、个数、类型、执行速度

例：result=a+b

```
data segment
  a       db  1
  b       db  2
  result  db  ?
  string  db  'result=$'
data ends
code segment
  assume cs:code, ds:data
start:  mov   ax,data
        mov   ds,ax
              mov   al,a
        add   al,b
        mov   result,al
        lea   dx,string
        mov   ah,09
        int   21h
        add   result,30h
        mov   dl,result
        mov   ah,2
        int   21h
              mov   ah,4ch
              int   21h
code    ends
        end   start
```

8086/8088指令概述

操作数三种来源：

① 操作数在指令中，称立即数操作数

如　MOV AL, 9

② 操作数在寄存器中，称寄存器操作数

指令中给出用符号表示的寄存器名。

如　MOV AL, 9; MOV AL, BL

③ 操作数在内存单元中，称存储器操作数或内存操作数

指令中给出该内存单元的地址。用[ ]表示存储器操作数

如　MOV AL , [2000H]

wenjiecoder@outlook.com

❑ 操作数个数

按指令格式中，操作数个数的多少分为四类：

- **无操作数：** 指令只有一个操作码，没有操作数
- **单操作数：** 指令中给出一个操作数
- **双操作数：** 指令中给出两个操作数
- **三操作数：** 指令中给出三个操作数

wenjiecoder@outlook.com

① 无操作数：指令只有一个操作码，没有操作数。

有两种可能：

▲ 有些操作不需要操作数。

如 HLT（暂停），NOP（无操作）等处理机控制指令。

▲ 操作数隐含在指令中。

如 STC（CF=1），CLC（CF=0）等处理机控制命令。

wenjiecoder@outlook.com

② 单操作数：指令中给出一个操作数。

有两种可能：

▲有些操作只需要一个操作数

如　INC　AL　　；（AL）←（AL）+ 1

▲有些操作将另一个操作数隐含在指令中

如　MUL　BL　　；（AX）←（AL）×（BL）

wenjiecoder@outlook.com

③ 双操作数： 指令中给出两个操作数。

如 ADD AL，BL ；(AL)←(AL)+(BL)

目的操作数 源操作数

操作后的结果通常存放在目的操作数中。

wenjiecoder@outlook.com

③ 三操作数： 指令中给出三个操作数。

如 IMUL BX, Array[100], 7 ;7*(Array[100])->BX

SHLD EBX, ECX, 16 ;双精度左移

目的操作数：1，源操作数：2

操作后的结果通常存放在目的操作数中。

wenjiecoder@outlook.com

❑ 操作数类型

8086/8088:

有的操作既可对字节操作，又可对字操作

有的操作只允许对字操作

- 指令应指明参与操作的数是字节还是字，即操作数的类型。
- 通常操作数的类型可由操作数本身隐含给出。
- 在特殊情况下需要指明。

wenjiecoder@outlook.com

8086/8088指令概述

① 指令中有寄存器操作数，由寄存器操作数决定类型。

例：　MOV　[BX], AL　　;字节操作, [BX] ← AL

　　MOV　[BX] , AX　　;字操作, [BX] ←AL, [BX+1] ←AH

　　MOV　BX, AL　　;???

wenjiecoder@outlook.com

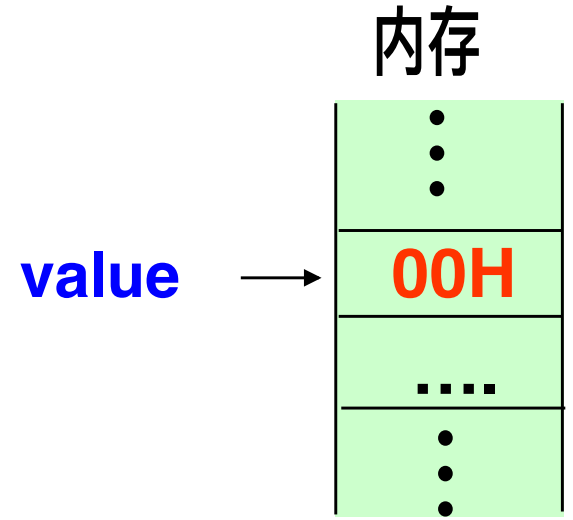
8086/8088指令概述

② 指令操作数中无寄存器，则由内存操作数的类型决定。

例　value 是一个变量 (即内存操作数)；

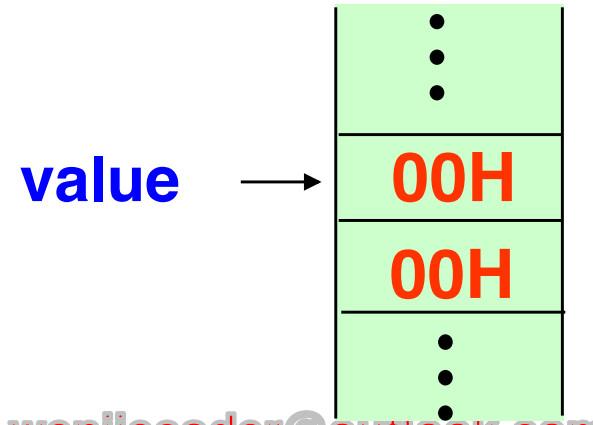
若定义value 为字节类型：

则 MOV value , 0　是字节操作。



若定义value 为字类型：

则 MOV value , 0　是一个字操作。



③ 指令中无类型的依据，需对存储器操作数加类型说明。

例　MOV [BX], 0　　✗

用 PTR 属性伪操作说明类型。

MOV　byte PTR　[BX] , 0

字节操作，[BX] ← 0

MOV　word PTR　[BX] , 0

字操作，[BX] ← 0, [BX+1] ← 0

wenjiecoder@outlook.com

❑ 执行速度

寄存器操作数　　立即数操作数　　存储器操作数

例

```
mov   AL , BL
mov   AL , 0
mov   AL , [ BX ]
```

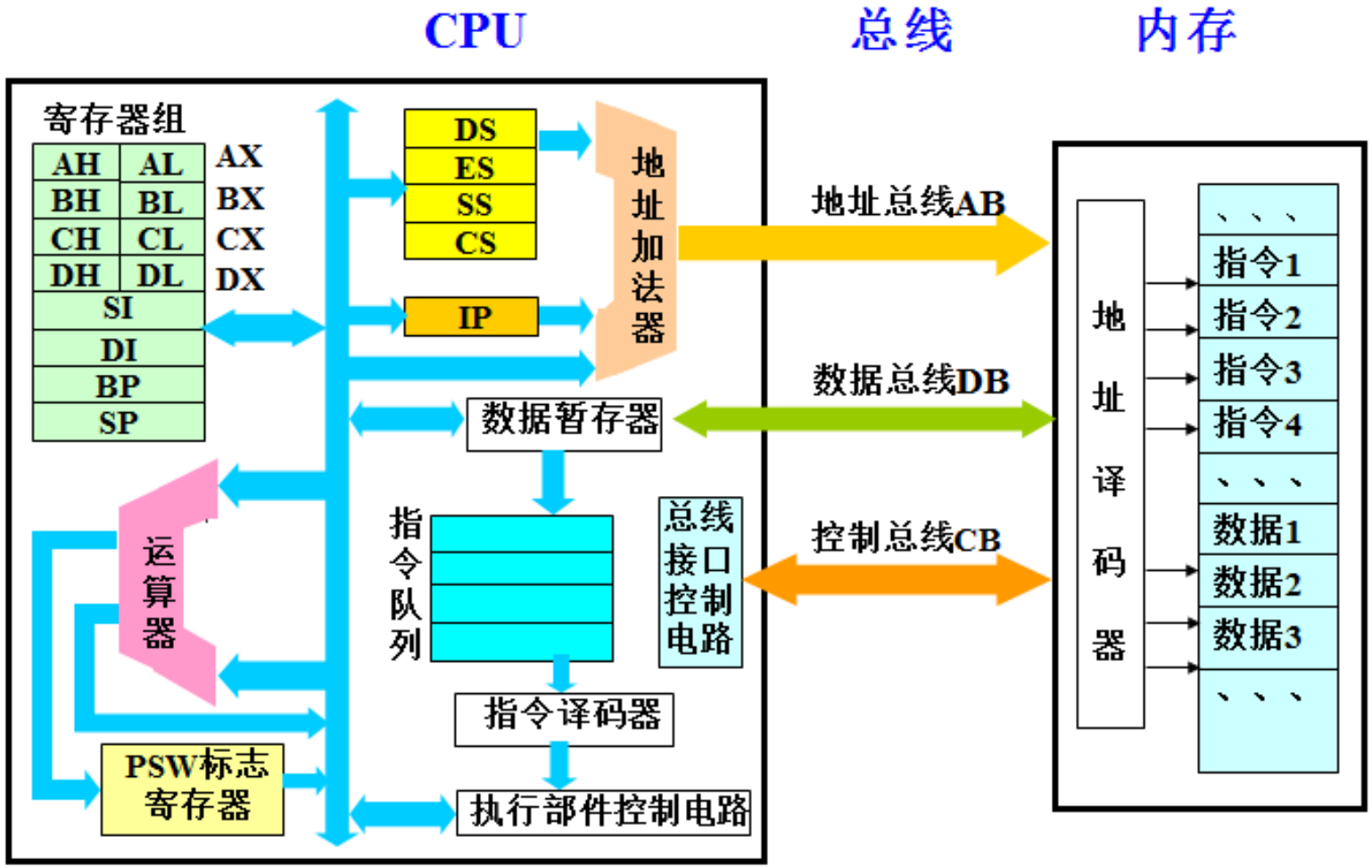
} 哪条指令执行速度快?

三条指令:

- ♦ 操作类型相同，都是传送指令
- ♦ 且目的操作数相同
- ♦ 仅源操作数不同

wenjiecoder@outlook.com

8086/8088指令概述



定长指令码格式：立即寻址最快，因为指令地址码即为操作数。
变长指令码格式：因为立即寻址操作数可能很长，取指令时可能需要两次访存。而寄存器寻址取指令只需一次访存，所以寄存器寻址最快。

wenjiecoder@outlook.com

对同一类型指令，执行速度：

寄存器操作数　　立即数操作数　　存储器操作数

快　　慢

例　mov　AL，BL

mov　AL，0

mov　AL，[BX]

快

慢

第4&5讲：8086/8088的指令系统

8086/8088指令概述

数据传送指令

算术指令

逻辑指令

串处理指令

处理机控制与杂项操作指令

重点关注：

- 指令的汇编格式及特点
- 指令的基本功能
- 指令的执行对标志位的影响
- 指令的特殊要求

wenjiecoder@outlook.com

数据传送指令

- 通用数据传送指令

  MOV、PUSH、POP、XCHG、XLAT

- 地址传送指令

  LEA、LDS、LES

- 标志寄存器传送指令

  LAHF、SAHF、PUSHF、POPF

- 类型转换指令

  CBW、CWD

wenjiecoder@outlook.com

数据传送指令

通用数据传送指令

传送指令：　MOV　DST，SRC

执行操作：　(DST) ← (SRC)

为什么设计指令时要有这么多限制：

1）段寄存器的内容，尤其是CS，不应该被随意改变。

2）考虑指令长度，解码难度和硬件实现复杂性等因素，比如两个存储器之间传送，那么两个存储器都需要寻址，指令过长，解码过于复杂等。

注意：

* 两个操作数的数据类型要相同：MOV　BL，AX
* DST、SRC 不能同时为段寄存器：MOV　DS
* 立即数不能直接送段寄存器：MOV　DS, 200
* DST 不能是立即数和CS：MOV　CS, AX；MOV　100H，AX（　）
* DST、SRC 不能同时为存储器寻址：MOV　VARA，VARB（　）
* 指令指针IP，不能作为MOV指令的操作数
* 不影响标志位

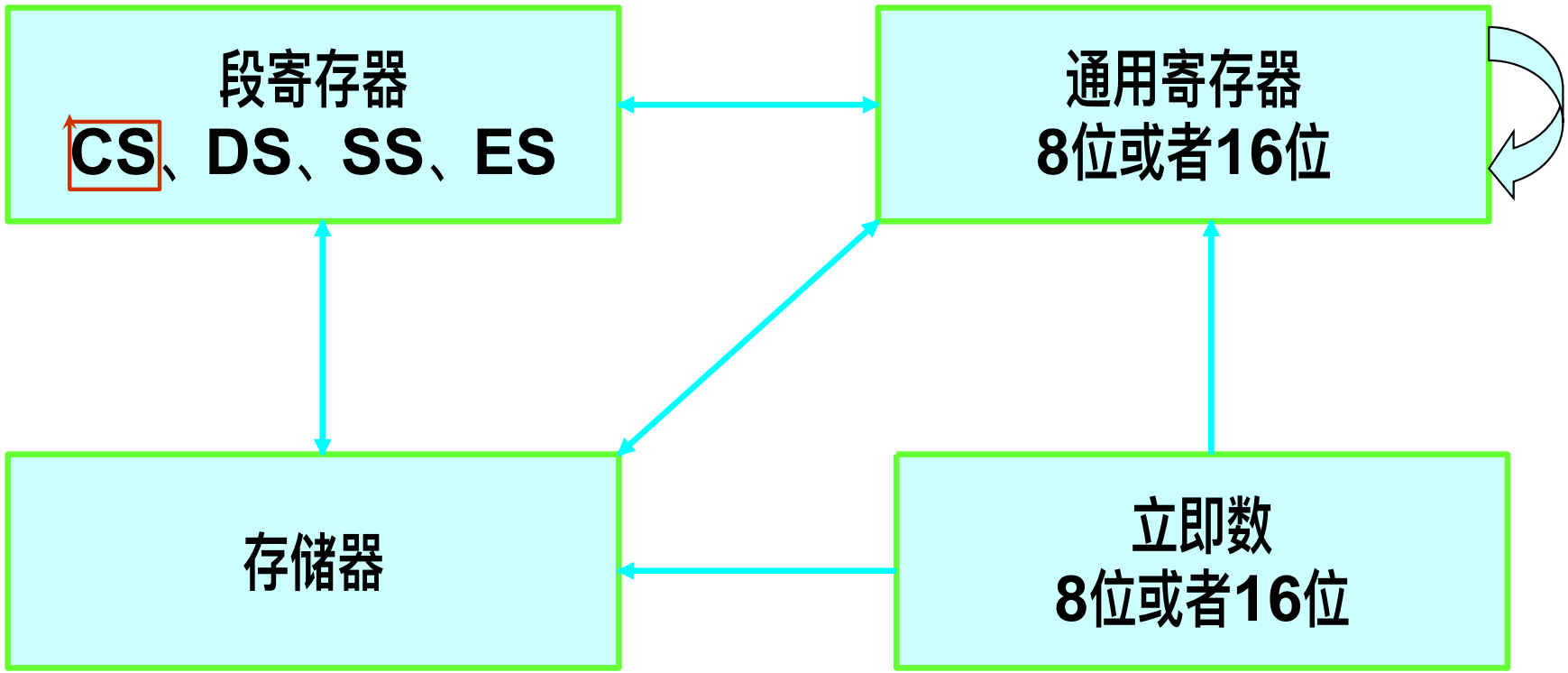
为什么IP不能作为MOV指令的操作数：

在目标程序运行时，IP的内容由微处理器硬件自动设置，程序不能直接访问IP，但一些指令却可改变IP的值，如转移指令、子程序调用指令等。

wenjiecoder@outlook.com

数据传送指令

在汇编语言中，主要的数据传送方式如下图所示。虽然一条MOV指令能实现其中大多数的数据传送方式，但也存在MOV指令不能实现的传送方式。

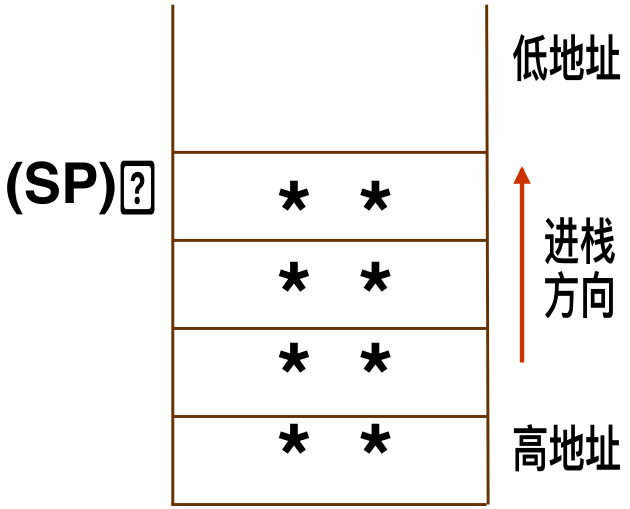


数据传送指令

进栈指令：PUSH SRC

执行操作：(SP) ← (SP) – 2

((SP)+1，(SP)) ← (SRC)



出栈指令：POP DST

执行操作：(DST) ← ((SP)+1，(SP))

(SP) ← (SP) + 2

堆栈：‘先进后出’的存储区，段地址存放在SS中，

SP在任何时候都指向栈顶，进出栈后自动修改SP。

注意：

* 堆栈操作必须以字为单位。
* 不影响标志位
* POP不能用立即寻址方式　　POP 1234H （×）
* POP指令DST不能是CS　　POP CS （×）

wenjiecoder@outlook.com

数据传送指令

PUSH指令的三种格式：

PUSH　reg（寄存器）

PUSH　mem（存储器）

PUSH　segreg（段寄存器）

POP指令的三种格式：

POP　reg

POP　mem

POP　segreg（no CS）

8086不允许push指令使用立即数寻址

例：

```
PUSH   temp
PUSH   AX
...
POP    AX
POP    BX
```

错误：

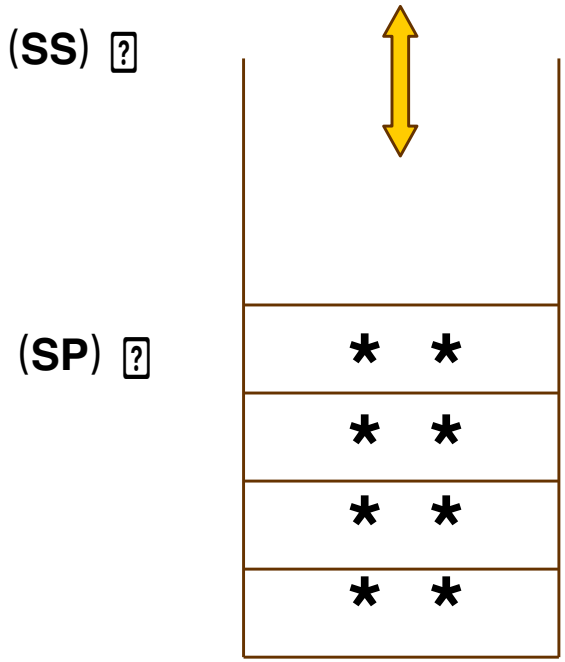
PUSH data（立即数）

POP　data（立即数）

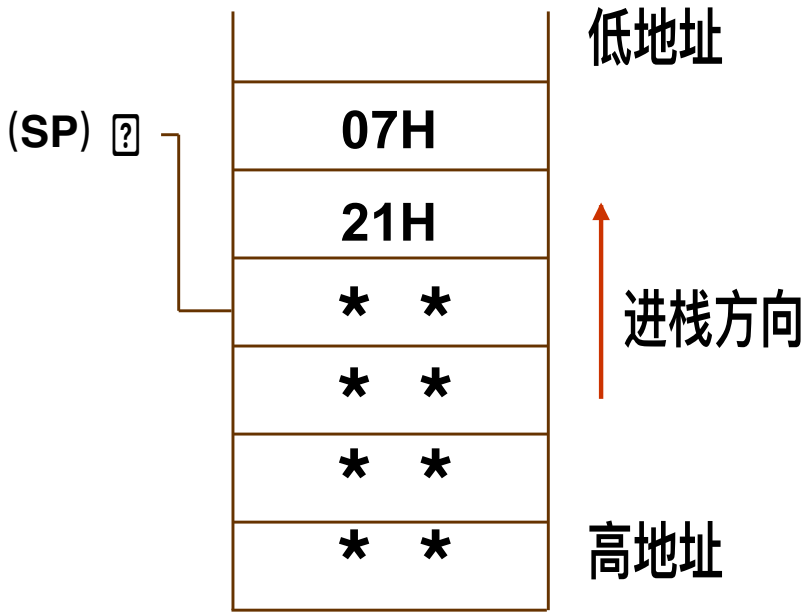
wenjiecoder@outlook.com

数据传送指令

例：假设 (AX) = 2107 H，执行 PUSH AX



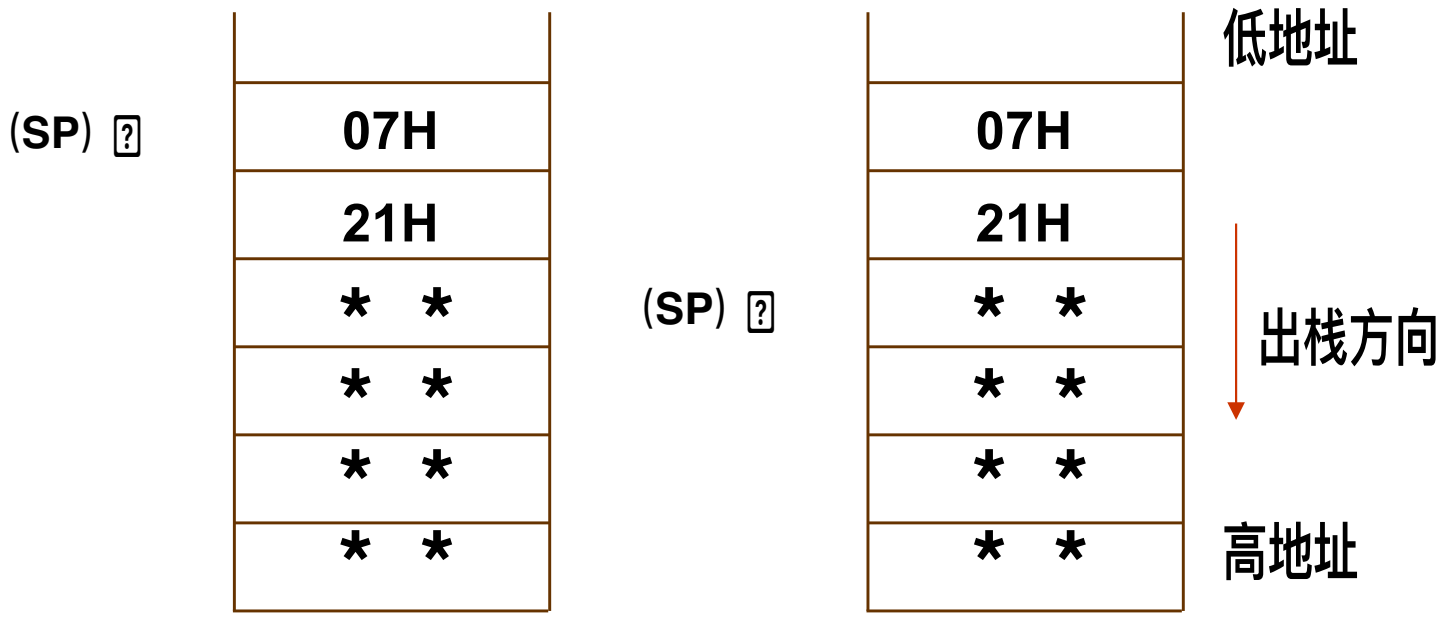
PUSH AX 执行前



PUSH AX 执行后

数据传送指令

例： POP BX



POP BX 执行前

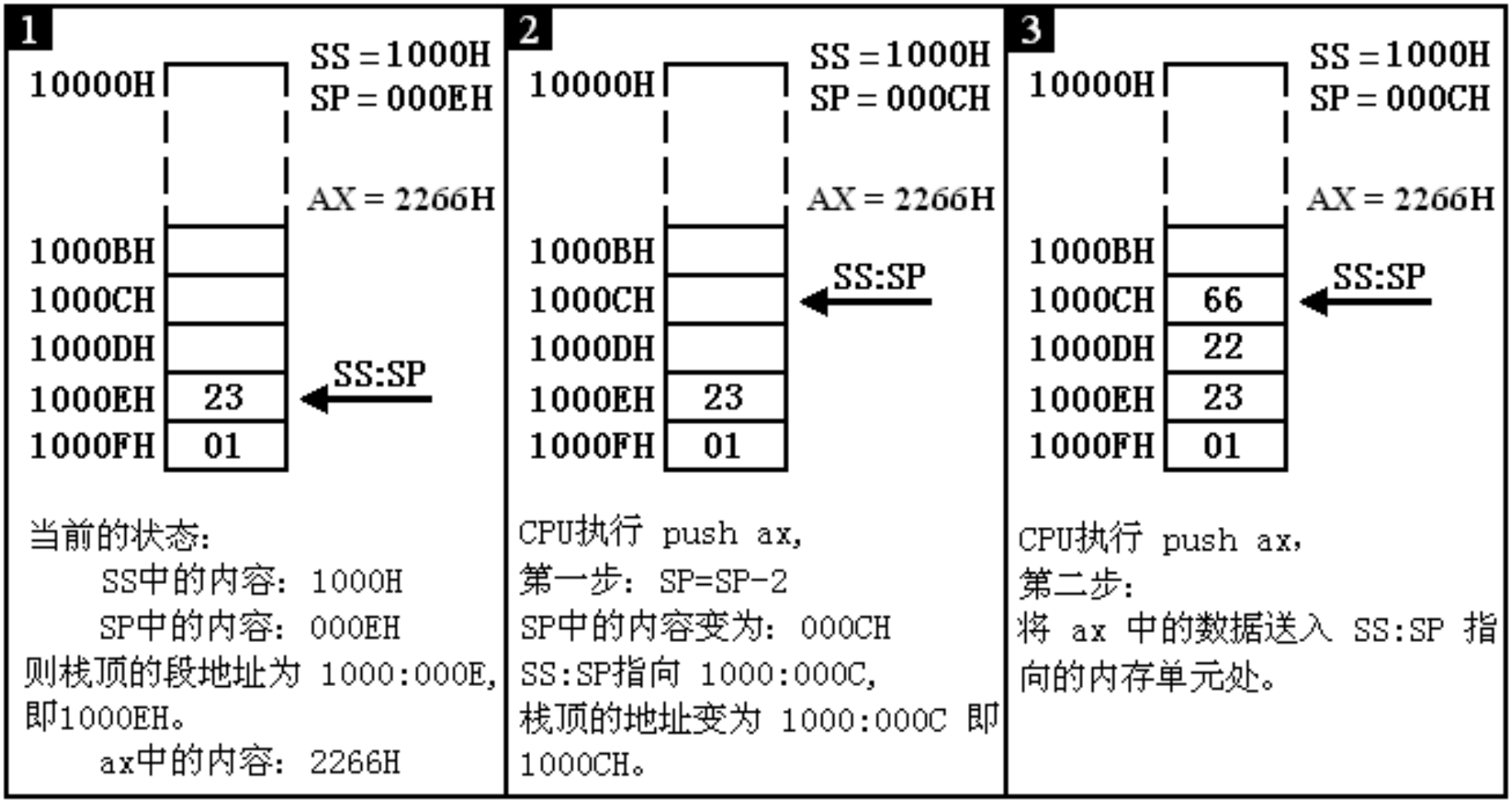
POP BX 执行后

(BX) = 2107H

数据传送指令

例：开辟一个16字节的堆栈，其SS=1000H，SP指向栈顶。即堆栈的内存地址范围为：10000H-1000FH。

PUSH AX

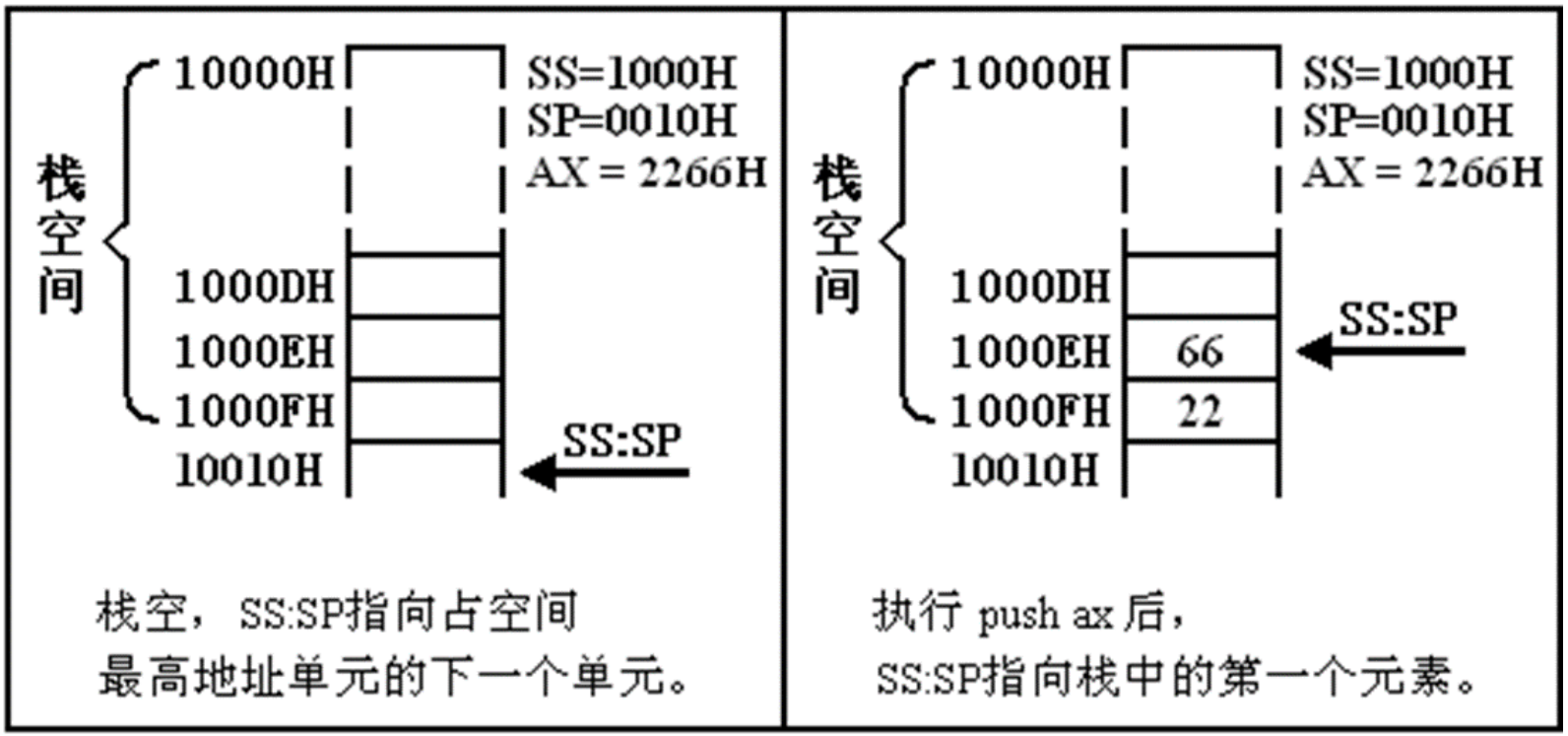


wenjiecoder@outlook.com

数据传送指令

思考：如果我们将10000H~1000FH
这段空间当作栈，初始状态栈是空的，此时，SS=1000H，SP=?

- **任意时刻，SS:SP指向栈顶，当栈中只有一个元素的时候，SS = 1000H，SP=000EH。**
- **栈为空，就相当于栈中唯一的元素出栈，出栈后，SP=SP+2，所以当栈为空的时候，SS=1000H，SP=0010H。**



栈满：SP? **SP=0000H**

wenjiecoder@outlook.com

数据传送指令

所有寄存器进栈指令：PUSHA

指令的格式为：PUSHA

功能：16位通用寄存器依次进栈，次序为AX、CX、DX、BX，指令执行前的SP、BP、SI、DI。指令执行后(SP)-16→(SP)仍指向栈顶。

所有寄存器出栈指令：POPA

指令的格式为：POPA

功能：16位通用寄存器依次出栈，次序为DI、SI、BP、SP，指令执行前的BX、DX、CX、AX。指令执行后(SP)+16→(SP) 仍指向栈顶。

需要说明的是：

SP出栈只是修改了指针使其后的BX能够出栈，而堆栈中原先由PUSHA指令存入的SP的原始内容被丢弃，并未真正送到SP寄存器中。

上述两条堆栈指令均不影响标志位。

wenjiecoder@outlook.com

数据传送指令

例：在有子程序或中断调用的程序中，若有些寄存器的内容在子程序或中断调用后还要用到，则可以用堆栈来保存。

```
PUSH  AX
PUSH  BX
PUSH  CX
PUSH  DX
……          ;其间用到AX、BX、CX、DX
POP   DX    ; 后进先出
POP   CX
POP   BX
POP   AX
```

```
PUSHA
…
POPA
```


wenjiecoder@outlook.com

数据传送指令

思考：

SS和SP只记录了栈顶的地址，依靠SS和SP可以保证在入栈和出栈时找到栈顶。可是，如何能够保证在入栈、出栈时，栈顶不会超出栈空间?

- 当栈满的时候再使用push指令入栈，栈空的时候再使用pop指令出栈，都将发生栈顶超界问题。
- 因为栈空间之外的空间里很可能存放了具有其他用途的数据、代码等，这些数据、代码可能是我们自己的程序中的，也可能是别的程
- 但是由于我们在入栈出栈时的不小心，而将这些数据
一连串的错误。
- **栈顶超界是危险的。8086CPU不保证对栈的操作**
- 这就是说， 8086CPU
只知道栈顶在何处（由SS:SP指示），而不知道安排
，CPU
只知道当前要执行的指令在何处（由CS:SP指示）而不知道读者要执行的指令有多少。

对于16位8086系列CPU，
因为无法自动检测栈顶超界现象，所以编程要人为控制。而对于当代32或者64位CPU，会自动检测并对栈顶超界抛出异常。

要根据可能用到的最大栈空间，来安排栈的大小，防止入栈的数据太多而导致的超界；防止出栈时栈空了仍然继续出栈而导致的超界。

数据传送指令

交换指令：XCHG OPR1, OPR2

执行操作：(OPR1) ? (OPR2)

用于寄存器之间或者寄存器与存储器之间交换信息。

注意：

* 不影响标志位
* 两个操作数中必须有一个是寄存器
* 不允许使用段寄存器

例：XCHG BX, [BP+SI]

XCHG AL, BH

例 假设(AL)=2AH，(204DH)=5BH，则：
指令XCHG AL，[204DH]
执行后：(AL)=5BH，(204DH)=2AH

数据传送指令

换码指令：XLAT 或 XLAT OPR （OPR只是为了增加可读性，首地址需要预先送到BX）

执行操作：(AL) ? ((BX) + (AL))

用于代码转换，例如：把字符扫描码转换成ASCII码，或者把数字0-9转换成7段数码管所需要的相应代码等。

在使用这条指令之前，应先建立一个字节表格：

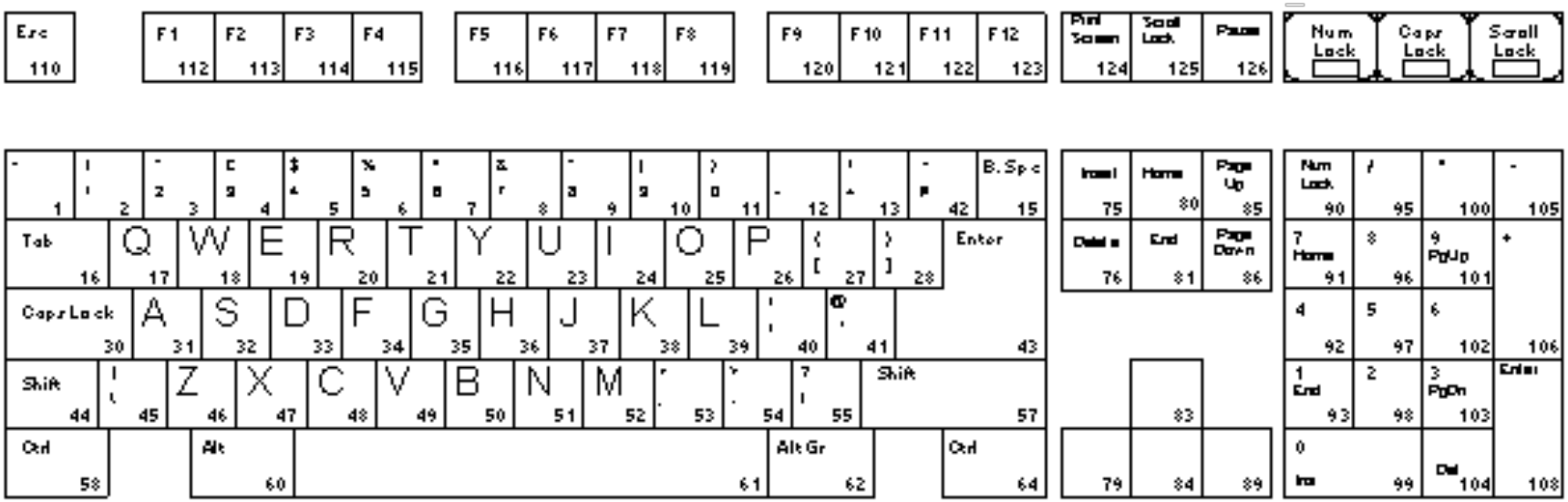
表格的首地址? (BX)

需转换的代码相对于首地址的位移量 ? (AL)

指令执行完后，AL中即为转换后的代码。

也叫
表格查询转译指令，用于两种不同编码之间的映射转换。比如可用于数据加密。

通过转译索引表格来实现。

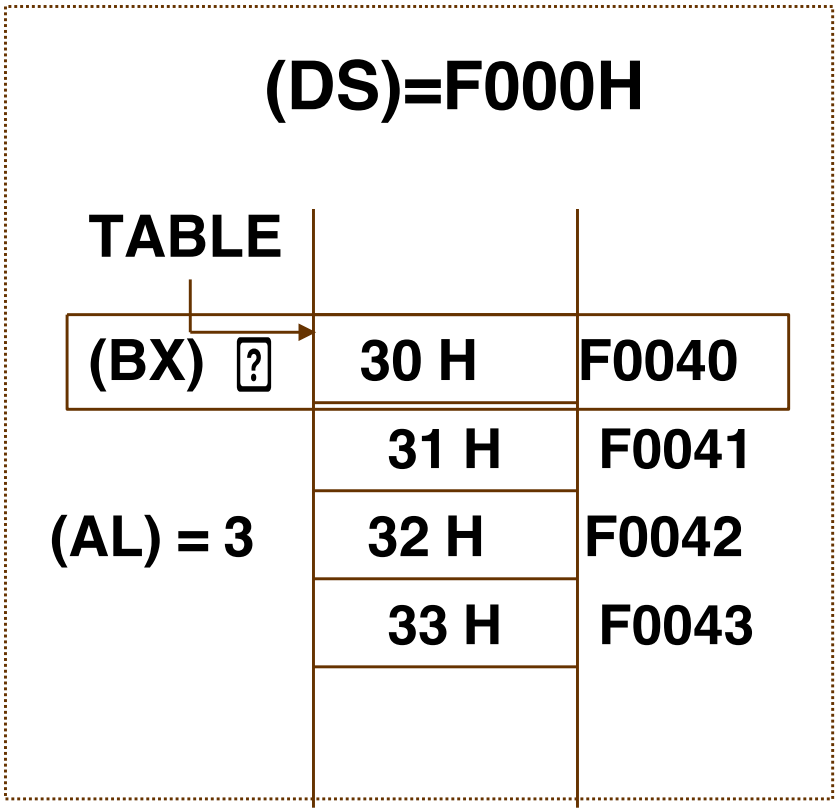


wenjiecoder@outlook.com

数据传送指令

换码指令：XLAT 或 XLAT OPR
执行操作：(AL) ← ((BX) + (AL))

例：MOV BX, OFFSET TABLE; (BX)=0040H
MOV AL, 3
XLAT TABLE
指令执行后 (AL)=33H



注意:
*该指令不影响标志位
*字节表格长度不能超过256个字节

AL 只有8位

地址传送指令

有效地址送寄存器指令：　LEA　REG, SRC

执行操作：　(REG) ← SRC的有效地址

例: LEA　BX, [BX+SI+0F62H]

若指令执行前（BX）=0400H, （SI）=003CH

则指令执行后（BX）=0400+003C+0F62=139EH

思考:

(1) MOV　BX, [BX+SI+0F62H] ；该指令与上述指令有何区别?

(2) LEA　BX, LIST

MOV　BX, OFFSET　LIST

相同功能。MOV指令比LEA更快，但是OFFSET只能与简单的符号地址相连，而LEA适用范围更广，可以与更复杂的寻址方式相连。

wenjiecoder@outlook.com

数据传送指令

地址传送指令

指针送寄存器和DS指令： LDS REG, SRC

执行操作：

(REG) ← (SRC)

(DS) ← (SRC+2)

相继二字 → 寄存器、DS

可用于同时传送段地址和有效地址，高地址数据装入段寄存器，低地址数据装入相应的存储有效地址的寄存器。

指针送寄存器和ES指令： LES REG, SRC

执行操作：

(REG) ← (SRC)

(ES) ← (SRC+2)

相继二字 → 寄存器、ES

注意：

* 不影响标志位
* REG 不能是段寄存器
* SRC 必须为存储器寻址方式

wenjiecoder@outlook.com

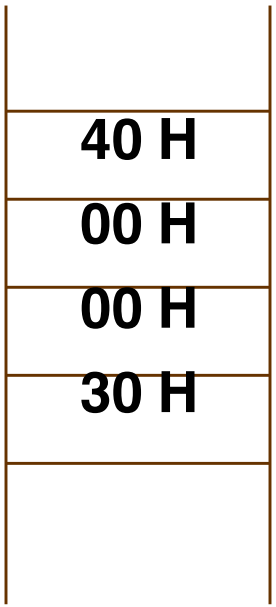
数据传送指令

例: LDS DI, [BX]

如指令执行前（DS）=B000H, （BX）=080AH,
（B080AH）=05AEH, （B080CH）=4000H

则指令执行后（DI）= 05AEH , （DS）=4000H

DS: TABLE
(0000H):1000H




```
MOV  BX, TABLE              ; (BX)=0040H
MOV  BX, OFFSET TABLE       ; (BX)=1000H
LEA  BX, TABLE              ; (BX)=1000H
LDS  BX, TABLE              ; (BX)=0040H
                            ; (DS)=3000H
LES  BX, TABLE              ; (BX)=0040H
                            ; (ES)=3000H
```

数据传送指令

标志寄存器传送指令

标志送AH指令：　LAHF（Load AH with Flags）
执行操作：　(AH) ← (FLAGS的低字节)

AH送标志寄存器指令：　SAHF（Save AH into Flags）
执行操作：　(FLAGS的低字节) ← (AH)

标志进栈指令：　PUSHF
执行操作：　(SP) ← (SP) - 2
((SP)+1, (SP)) ← (FLAGS)

标志出栈指令：　POPF
执行操作：　(FLAGS) ← ((SP)+1, (SP))
(SP) ← (SP) + 2

* SAHF，POPF影响标志位

数据传送指令

类型转换指令

保持有符号数的值不变

CBW　　AL → AX　（AL中的内容符号扩展到AX）

执行操作：若(AL)的最高有效位为0，则(AH)= 00H

若(AL)的最高有效位为1，则(AH)= FFH

CWD　　AX → (DX, AX)　（AX中的内容符号扩展到DX：AX）

执行操作：若(AX)的最高有效位为0，则(DX)= 0000H

若(AX)的最高有效位为1，则(DX)= FFFFH

例：(AX) = 0BA45H

1011 1010 0100 0101

```
CBW          ; (AX)=0045H
CWD          ; (DX)=0FFFFH  (AX)=0BA45H
```

注意：

* 无操作数指令
* 隐含对AL 或AX 进行符号扩展
* 不影响条件标志位

8086/8088指令概述

数据传送指令

算术指令

逻辑指令

串处理指令

处理机控制与杂项操作指令

wenjiecoder@outlook.com

算术指令

- 加法指令

  ADD、ADC、INC

- 减法指令

  SUB、SBB、DEC、NEG、CMP

- 乘法指令

  MUL、IMUL

- 除法指令

  DIV、IDIV

wenjiecoder@outlook.com

算术指令

- 加法指令

加法指令: ADD DST, SRC
执行操作: (DST) ← (SRC) + (DST)

带进位加法指令: ADC DST, SRC
执行操作: (DST) ← (SRC) + (DST) + CF

加1指令: INC OPR
执行操作: (OPR) ← (OPR) + 1

注意:

* 除INC指令不影响CF标志外，均对条件标志位有影响。

算术指令

举例： n=8 bit 带符号数(-128~127) ，无符号数(0~255)

```
  0000 0100
+ 0000 1011
-----------
  0000 1111
```

带：(+4)+(+11)=+15 OF=0

无：4+11=15 CF=0

带符号数和无符号数都不溢出

```
  1000 0111
+ 1111 0101
-----------
1 0111 1100
```

带：(-121)+(-11)=-132 超过范围

现为：124 OF=1

无：135+245=380 超过范围

现为：124 CF=1

带符号数和无符号数都溢出

```
  0000 0111
+ 1111 1011
-----------
1 0000 0010
```

带：(+7)+(-5)=+2 OF=0

无：7+251=258 超过范围

现为：2 CF=1

无符号数溢出

```
  0000 1001
+ 0111 1100
-----------
  1000 0101
```

带：(+9)+(+124)=133 超过范围

现为：-123 OF=1

无：9+124=133 CF=0

带符号数溢出

算术指令

对无符号数，考虑进位标志CF，则结果正确。

```
  0000 0100
+ 0000 1011
-----------
  0000 1111
```

带：(+4)+(+11)=+15　OF=0

无：4+11=15　　CF=0

带符号数和无符号数都不溢出

```
  1000 0111
+ 1111 0101
-----------
1 0111 1100
```

带：(-121)+(-11)=-132

现为：124　　OF=1 ？

无：135+245=380

现为：124　　CF=1 √

带符号数和无符号数都溢出

```
  0000 0111
+ 1111 1011
-----------
1 0000 0010
```

带：(+7)+(-5)=+2　OF=0

无：7+251=258

现为：2　　CF=1 √

无符号数溢出

```
  0000 1001
+ 0111 1100
-----------
  1000 0101
```

带：(+9)+(+124)=133

现为：-123　　OF=1 ？

无：9+124=133　　CF=0

带符号数溢出

算术指令

```
    1000 0111
  + 1111 0101
  ------------
  1 0111 1100
```

带：(-121)+(-11)=-132

现为：124　　OF=1 ？

无：135+245=380

现为：124　　CF=1 √

带符号数和无符号数都溢出

```
    0000 1001
  + 0111 1100
  ------------
    1000 0101
```

带：(+9)+(+124)=133

现为：-123　　OF=1 ？

无：9+124=133　　CF=0

带符号数溢出

对带符号数，若溢出标志OF为1，则计算结果错误。

溢出产生的原因是所表示的数超过的所给位数的最大能力，8位带符号数的表示范围在-128~+127。

处理方法：增加位数。左侧例题只要把位数扩展到16位就不会溢出了

为什么加法指令没有必要区分操作数是有符号数还是无符号数：

无符号数的加减法与有符号数的补码的加减法是恰巧是一致的。因此有符号数与无符号数可以使用同一指令，得到同一结果。而这一结果是作为有符号数还是无符号数，是由程序员来定义的：如果程序员认为是有符号数的加减法，则他将关心运算结果的溢出标志，以判别结果是否可用；如果程序员认为是无符号数的加减法，则他将关心运算结果的进位标志，看运算结果是否有进(借)位。

算术指令

加法指令对条件标志位的影响

SF= { 1 结果为负；0 否则

ZF= { 1 结果为0；0 否则

CF= { 1 和的最高有效位 有 向高位的进位；0 否则

OF= { 1 两个操作数符号相同，而结果符号与之相反；0 否则

两个操作数如果符号相反呢?

CF 位表示 无符号数 相加的溢出。

OF 位表示 带符号数 相加的溢出。

算术指令

例：双精度数的加法DX、AX与BX、CX，两个双字长的数相加。DX、BX分别存放高位字。

(DX) = 0002H　(AX) = 0F365H
(BX) = 0005H　(CX) = 0E024H

指令序列　ADD　AX, CX　; (1)
　　ADC　DX, BX　; (2)

带符号的双精度数的溢出，应该根据ADC指令的OF来判断，低位加法的ADD指令的溢出没有意义。

(1) 执行后，(AX) = 0D389H
　　CF=1　OF=0　SF=1　ZF=0

(2) 执行后，(DX) = 0008H
　　CF=0　OF=0　SF=0　ZF=0

算术指令

减法指令

减法指令: SUB DST, SRC
执行操作: (DST) ← (DST) - (SRC)

带借位减法指令: SBB DST, SRC
执行操作: (DST) ← (DST) - (SRC) - CF

减1指令: DEC OPR
执行操作: (OPR) ← (OPR) - 1

求补指令: NEG OPR
执行操作: (OPR) ← 0FFFFH - (OPR)+1

比较指令: CMP OPR1, OPR2
执行操作: (OPR1) - (OPR2)，不保存结果，只是根据结果设置标志位。

注意:
* 除DEC指令不影响CF标志外，均对条件标志位有影响。

wenjiecoder@outlook.com

算术指令

减法指令对条件标志位（CF/OF/ZF/SF）的影响：

CF= { 1 被减数的最高有效位有向高位的借位
      0 否则

或转换成加法：

CF= { 1 减法转换为加法运算时无进位
      0 否则

OF= { 1 两个操作数符号相反，而结果的符号与减数相同
      0 否则

或转换成加法：

OF= { 1 两个操作数符号相同，而结果符号与之相反
      0 否则

CF 位表示 无符号数 减法的溢出。

OF 位表示 带符号数 减法的溢出。

NEG 指令对CF/OF的影响

$$CF = \begin{cases} 0 & \text{操作数为0} \\ 1 & \text{否则} \end{cases}$$

NEG:
0FFFFH-操作数+1 等效于 0-操作数，即用0减去操作数，只有操作数为0时，不需要借位。

$$OF = \begin{cases} 1 & \text{操作数为 -128（字节运算）或} \\ & \text{操作数为 -32768（字运算）} \\ 0 & \text{否则} \end{cases}$$

8位二进制有符号数表示范围：-128~127

$$\begin{array}{r} 10000000 \\ 01111111 \\ +\ 00000001 \\ \hline 10000000 \end{array}$$

算术指令

例：SUB [SI+14H], 0136H

若（DS）=3000H，（SI）=0040H，（30054H）=4336H

则，指令执行后：（30054H）=4200H

SF=0，CF=0，OF=0，ZF=0

SUB DH, [BP+4]

若（SS）=0000H，（BP）=00E4H，

（DH）=41H，（000E8）=5AH

则，指令执行后：（DH）=0E7H

SF=1，CF=1，OF=0，ZF=0

wenjiecoder@outlook.com

算术指令

例：x、y、z 均为双精度数，分别存放在地址为X, X+2；　　Y, Y+2；Z, Z+2的存储单元中，存放时，高字节存放在高地址中，低字节存放在低地址中，用指令序列实现：

W ? x + y + 24 - z，并用W, W+2单元存放W

```
MOV  AX,  X
MOV  DX,  X+2
ADD  AX,  Y
ADC  DX,  Y+2      ;  x+y
ADD  AX,  24
ADC  DX,  0        ;  x+y+24
SUB  AX,  Z
SBB  DX,  Z+2      ;  x+y+24-z
MOV  W,   AX
MOV  W+2, DX       ;  结果存入W, W+2单元
```

算术指令

乘法指令

无符号数乘法指令：　　MUL　SRC

带符号数乘法指令：　　IMUL　SRC

执行操作：

字节操作数　(AX) ← (AL) * (SRC)

字操作数　　(DX, AX) ← (AX) * (SRC)

注意：

* AL (AX) 为隐含的乘数寄存器。
* AX (DX,AX) 为隐含的乘积寄存器。
* SRC不能为立即数。
* 除CF和OF外，对条件标志位无定义。

无定义不是没影响，无定义是指指令执行之后的标志位状态不定。

wenjiecoder@outlook.com

算术指令

注意：MUL和IMUL指令的使用条件是由数的格式决定的。如：

(11111111b) * (11111111b)

若是无符号数：255d*255d=65025d

若是带符号数： (-1) * (-1) =1

因此，必须根据所要相乘的数的格式来决定选用哪一种命令。

wenjiecoder@outlook.com

算术指令

例：如（AL）=0B4H，（BL）=11H，求：

```
IMUL   BL
MUL    BL
```

分析：

（AL）=0B4H为无符号数的180，有符号数的-76；

（BL）=11H为无符号数的17，有符号数的17.

因此：

MUL BL的执行结果为：（AX）=0BF4H（3060）

IMUL BL的执行结果为：（AX）=0FAF4H（-1292）

wenjiecoder@outlook.com

算术指令

可以用来检验字节（字）相乘的结果是字节还是字（字还是双字）；

乘法指令对 CF/OF 的影响：

MUL指令：CF, OF =
- **00** 乘积的高一半为零
- **11** 否则

IMUL指令：CF, OF =
- **00** 乘积的高一半是低一半的符号扩展
- **11** 否则

A5：1010 0101

求补：0101 1011

例：(AX) = 16A5H，(BX) = 0611H

```
(1)   IMUL  BL      ; (AX) ? (AL) * (BL)
                    ; A5*11 ? 5B*11=060B ? F9F5
                    ; (AX) = 0F9F5H    CF=OF=1

(2)   MUL  BX       ; (DX, AX) ? (AX) * (BX)
                    ; 16A5*0611=0089 5EF5
                    ; (DX)=0089H  (AX)=5EF5H   CF=OF=1
```

算术指令

- 除法指令

无符号数除法指令：　DIV　SRC

带符号数除法指令：　IDIV　SRC　(余数的符号和被除数相同)

执行操作：

字节操作　(AL) ← (AX) / (SRC) 的商

(AH) ← (AX) / (SRC) 的余数

字操作　(AX) ← (DX, AX) / (SRC) 的商

(DX) ← (DX, AX) / (SRC) 的余数

注意：
* AX (DX,AX) 为隐含的被除数寄存器。
* AL (AX) 为隐含的商寄存器。
* AH (DX) 为隐含的余数寄存器。
* SRC不能为立即数。
* 对所有条件标志位均无定义。

算术指令

思考：

```
MOV  AX, 2EE0H    ;12000
MOV  BL, 8
DIV   BL
```

计算结果：

商：1500=>AL

余数：0=>AH

问题：

只要字节除法操作AH绝对值大于等于除数的绝对值、或者字除法操作DX的绝对值大于等于除数绝对值，都会发生除法溢出

!

如何解决?

操作数先扩展，再做除法。

```
MOV  AX, 2EE0H    ;12000
CWD               ???
MOV  BX, 8
DIV   BX
```

MOV DX, 0 更合理

计算结果：

商：1500=>AX

余数：0=>DX

wenjiecoder@outlook.com

算术指令

例：x , y , z , v 均为16位带符号数，计算(v-(x*y+z–540))/x

```
MOV   AX, X
IMUL  Y        ; x*y → (DX,AX)
MOV   CX, AX
MOV   BX, DX
MOV   AX, Z
CWD            ; Z → (DX, AX)
ADD   CX, AX
ADC   BX, DX    ; x*y+z → (BX, CX)
SUB   CX, 540
SBB   BX, 0     ; x*y+z-540
MOV   AX, V
CWD            ; V → (DX, AX)
SUB   AX, CX
SBB   DX, BX    ; v-(x*y+z-540)
IDIV  X        ; (v-(x*y+z-540))/x→ (AX) , 余数→ (DX)
```

8086/8088指令概述

数据传送指令

算术指令

逻辑指令

串处理指令

处理机控制与杂项操作指令

wenjiecoder@outlook.com

逻辑指令

逻辑指令：

- 逻辑运算指令

  AND、OR、NOT、XOR、TEST

- 移位指令和循环移位指令

  SHL、SHR 、 SAL 、SAR、

  ROL、ROR、RCL、RCR

wenjiecoder@outlook.com

逻辑指令

逻辑运算指令：

逻辑非指令：NOT OPR　　* OPR不能为立即数
执行操作：(OPR) → (OPR)取反　　* 不影响标志位

逻辑与指令：AND DST, SRC
执行操作：(DST) → (DST) ∧ (SRC)

逻辑或指令：OR DST, SRC
执行操作：(DST) → (DST) ∨ (SRC)

异或指令：XOR DST, SRC
执行操作：(DST) → (DST) ⊕ (SRC)

测试指令：TEST OPR1, OPR2
执行操作：(OPR1) ∧ (OPR2)

CF	OF	SF	ZF	PF	AF	
0	0	*	*	*		无定义

根据运算结果设置

wenjiecoder@outlook.com

逻辑指令

例：屏蔽AL的第0、1两位

```
AND  AL, 0FCH
```

```
  * * * *  * * * *
AND 1 1 1 1  1 1 0 0
  * * * *  * *  0 0
```

例：置AL的第5位为1

```
OR  AL, 20H
```

```
  * * * *  * * * *
OR  0 0 1 0  0 0 0 0
  * *  1 *  * * * *
```

例：使AL的第0、1位变反

```
XOR  AL, 3
```

```
  * * * *  * *  0 1
XOR 0 0 0 0  0 0 1 1
  * * * *  * *  1 0
```

例：测试某些位是0是1

```
TEST  AL, 1
JZ    EVEN
```

```
  * * * *  * * * *
AND 0 0 0 0  0 0 0 1
  0 0 0 0  0 0 0 *
```

移位指令和循环移位指令

逻辑左移 SHL OPR, CNT

逻辑右移 SHR OPR, CNT

算术左移 SAL OPR, CNT（同逻辑左移）

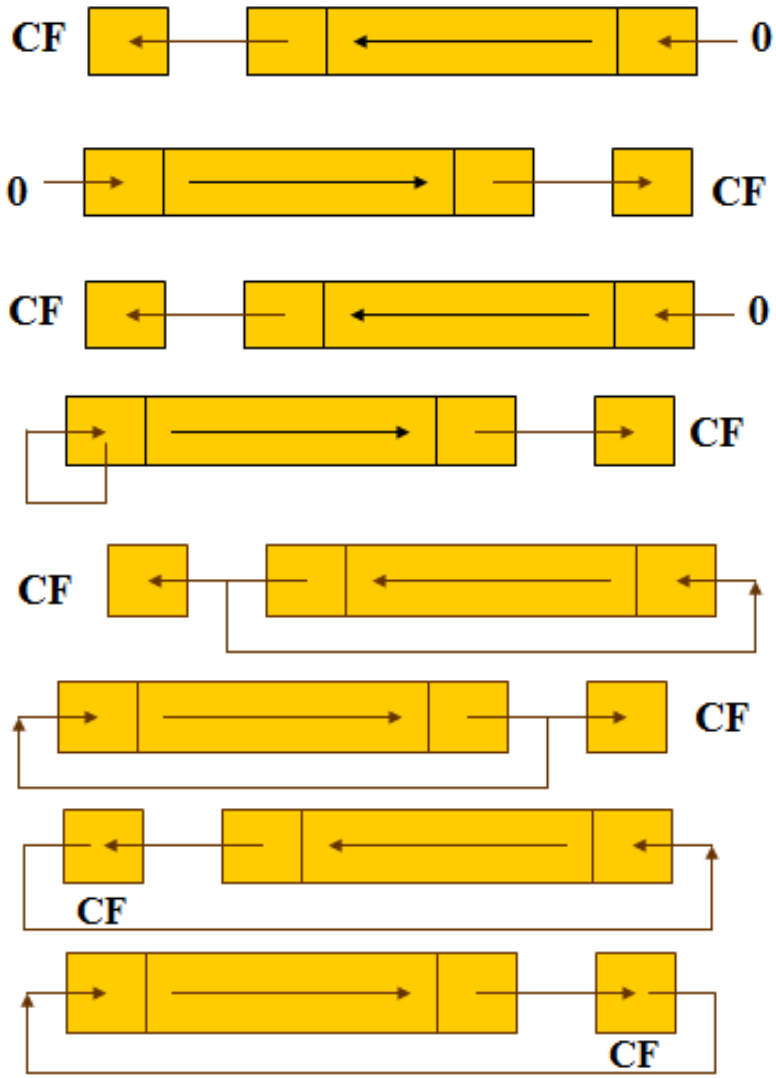
算术右移 SAR OPR, CNT

循环左移 ROL OPR, CNT

循环右移 ROR OPR, CNT

带进位循环左移 RCL OPR, CNT

带进位循环右移 RCR OPR, CNT



逻辑指令

注意:

* OPR可用除立即数以外的任何寻址方式
* CNT=1，SHL　OPR, 1

  CNT>1，MOV　CL,　CNT

  　　SHL　OPR, CL　　; 以SHL为例
* 条件标志位:

  CF = 移入的数值

  OF = $\begin{cases} 1 & \text{CNT=1时，最高有效位的值发生变化} \\ 0 & \text{CNT=1时，最高有效位的值不变} \end{cases}$

  无定义，　CNT>1时。

  移位指令：SF、ZF、PF 根据移位结果设置，AF无定义

  循环移位指令：不影响 SF、ZF、PF、AF
* 移位指令可以用来作乘2或除2的操作，其中算术移位指令用于带符号数运算，逻辑移位用于无符号数运算。

逻辑指令

例：(AX)= 0012H，(BX)= 0034H，把它们装配成(AX)= 1234H

```
MOV  CL, 8
ROL  AX, CL
ADD  AX, BX
```

例：(BX) = 84F0H

(1) (BX) 为带符号数，求 (BX) ×2

```
SAL  BX, 1      ; (BX) = 09E0H, OF=1
```

(2) (BX) 为无符号数，求 (BX) / 2

```
SHR  BX, 1      ; (BX) = 4278H
```

(3) (BX) 为带符号数，求 (BX) / 4

```
MOV  CL, 2
SAR  BX, CL     ; (BX) = 0E13CH
```


wenjiecoder@outlook.com

逻辑指令

循环移位和带进位循环移位的区别：

以ROL和RCL为例：
简单的说就是ROL只是二进制本身首尾相衔接的循环左移，每左移一位就将最高位送入CF（进位标志），但是CF不参与循环。
而RCL是把CF也参与进循环里面，以8位字节循环为例，相当于8位二进制循环变成了9位二进制循环。

比如，二进制数00110101，假设进位CF为1。
那么ROL 3: 表示00110101 ->01101010 ->11010100 ->10101001
RCL 3: 表示1+00110101 ->0+01101011（原借位的值1进入现在值的最低位，此时原最高位的0进入借位） ->0+11010110 -> 1+10101100,
相当于变成九位100110101循环，结果取最低的八位。

8086/8088指令概述

数据传送指令

算术指令

逻辑指令

串处理指令

处理机控制与杂项操作指令

wenjiecoder@outlook.com

串处理指令

串处理指令：

- 设置方向标志指令

  CLD、STD

- 串处理指令

  串传送：MOVS

  存入串：STOS

  取出串：LODS

  串比较：CMPS

  串扫描：SCAS

- 串重复前缀

  重复：REP

  相等/为零则重复：REPE/REPZ

  不相等/不为零则重复：REPNE/REPNZ

串处理指令

与 REP 配合工作的 MOVS / STOS

REP：重复操作直到计数器（CX）的值为0为止。

执行操作：

(1) 如 (CX)=0 则退出 REP，否则转(2)；

(2) (CX) ? (CX) -1；

(3) 执行 MOVS / STOS ；

(4) 重复 (1) ~ (3)。

wenjiecoder@outlook.com

串处理指令

MOVS 串传送指令：

MOVS　DST, SRC　（在操作数明确是字节、还是字传送）

MOVSB　　（字节）

MOVSW　　（字）

REP MOVS：将数据段中的整串数据传送到附加段中。

源串（默认数据段，可用段前缀修改）→ 目的串（必须附加段）

例：MOVS　ES: BYTE PTR [DI]，DS: [SI]

执行操作：

(1) ((DI)) ← ((SI))

(2) 字节操作：(SI)←(SI)±1，(DI)←(DI)±1

字操作：(SI)←(SI)±2，(DI)←(DI)±2

方向标志　DF=0 时用 +　（从前往后）

DF=1 时用 -　（从后向前）

wenjiecoder@outlook.com

串处理指令

执行 REP MOVS 之前，应先做好：

源串首地址（或末地址）→ SI；

目的串首地址（或末地址）→ DI；

串长度 → CX；

建立方向标志。

（ CLD 使 DF=0，STD 使 DF=1 ）

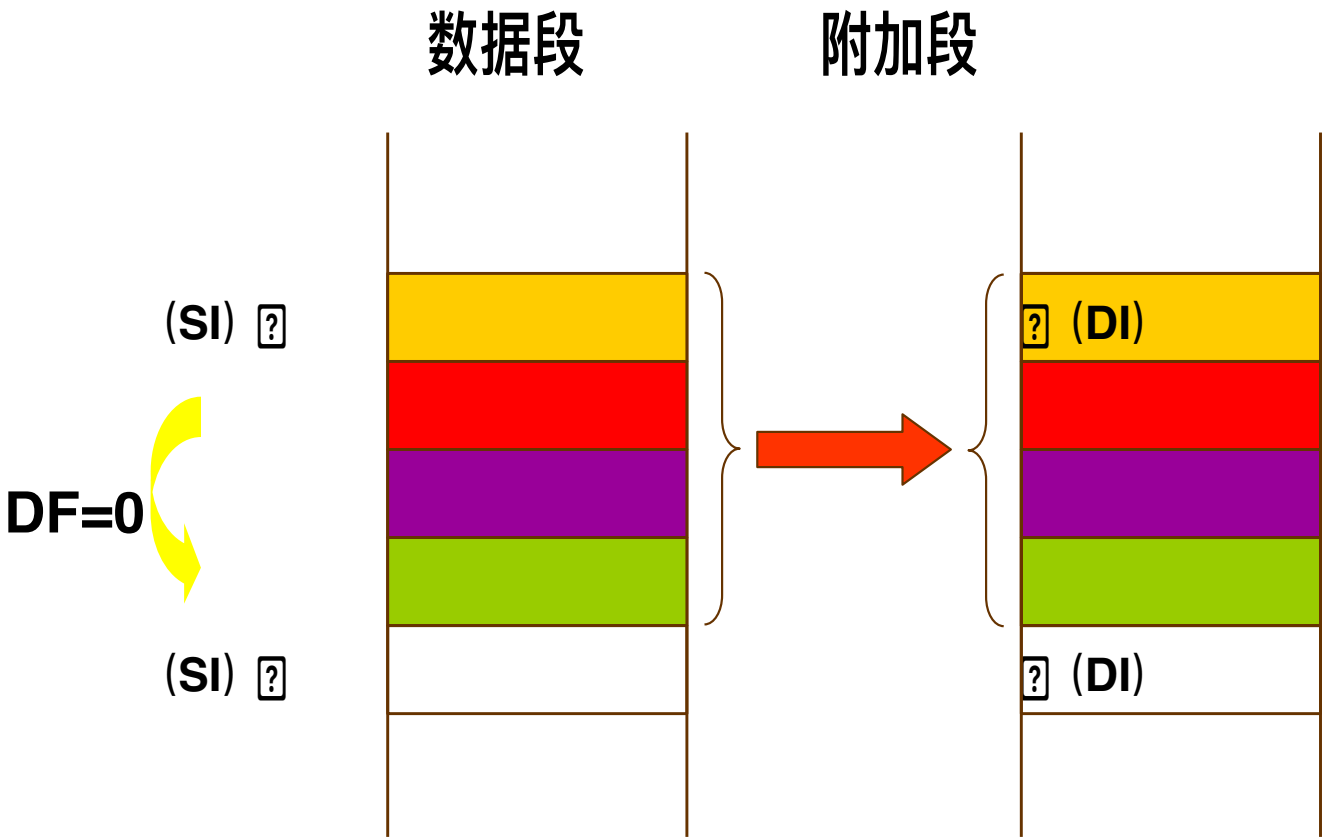
串处理指令

DF=0，执行操作：

(1) ((DI)) ← ((SI))

(2) 字节操作：(SI)←(SI)+1，(DI)←(DI)+1

字操作：(SI)←(SI)+2，(DI)←(DI)+2



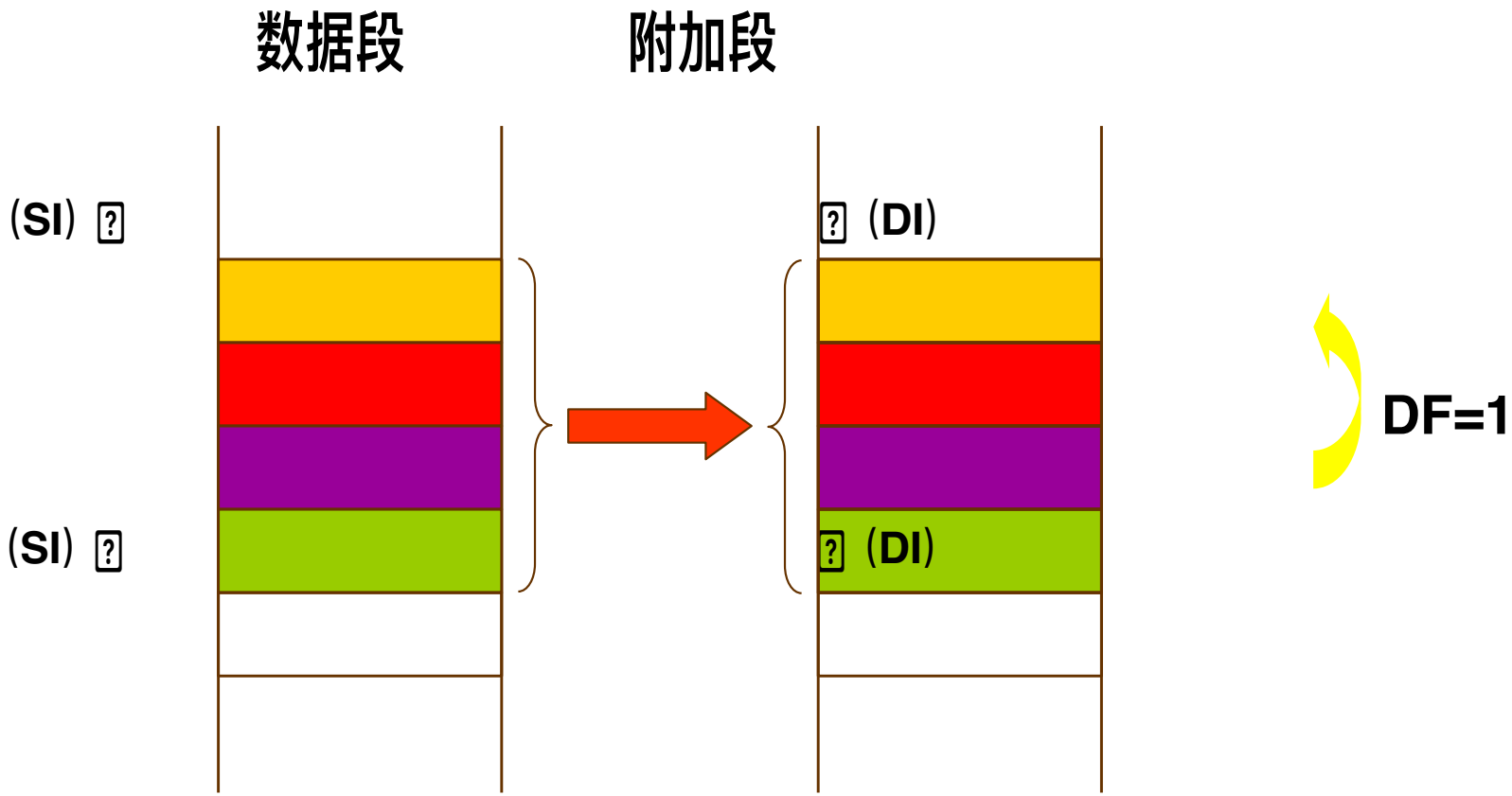
串处理指令

DF=1，执行操作：

(1) ((DI)) ← ((SI))

(2) 字节操作：(SI)←(SI)-1，(DI)←(DI)-1

字操作：(SI)←(SI)-2，(DI)←(DI)-2



串处理指令

```
data   segment
            mess1 db  'personal_computer'
data   ends
extra   segment
            mess2 db  17 dup (?)
extra   ends
code   segment
            assume  cs:code, ds:data, es: extra
start:
            mov  ax, data
            mov  ds, ax
            mov ax, extra
            mov  es, ax
            lea  si, mess1
            lea  di, mess2
            mov  cx, 17
            cld
            rep  movsb
            mov   ah, 4ch
            int 21h
code   ends
            end start
```

例：在数据段中有一个字符串，其长度为17个字节，要求把他们转送到附加段中的一个缓冲区中。

```
lea         si, mess1+16
lea         di, mess2+16
mov         cx, 17
std
rep         movsb
```


wenjiecoder@outlook.com

串处理指令

```
data   segment
          mess1 db  'personal_computer'
          mess2 db  17 dup (?)
data   ends

code   segment
          assume  cs:code, ds:data, es:data
start:
          mov  ax, data
          mov  ds, ax
          mov  es, ax
          lea  si, mess1
          lea  di, mess2
          mov  cx, 17
          cld
          rep  movsb
          mov   ah, 4ch
          int 21h
code   ends
          end start
```

例：在数据段中有一个字符串，其长度为17个字节，要求把他们转送到附加段中的一个缓冲区中。

```
lea       si, mess1+16
lea       di, mess2+16
mov       cx, 17
std
rep       movsb
```

串处理指令

```
data   segment
          mess1 db  'personal_computer'
          mess2 db  17 dup (?)
data   ends
code   segment
          assume  cs:code, ds:data, es:data
start:
          mov  ax, data
          mov  ds, ax
          mov  es, ax
          lea  di, mess1
          lea  si, mess2
          mov  cx, 17
          cld
          rep  movsb
          mov   ah, 4ch
          int 21h
code   ends
end start
```

结果?

串处理指令

STOS 存入串指令：

STOS　DST

STOSB　　(字节)

STOSW　　(字)

可用来初始化某一缓冲区

执行操作：

字节操作：((DI))←(AL)，(DI)←(DI)±1

字操作：((DI))←(AX)，(DI)←(DI)±2

例：把附加段中首地址为mess2的10个字节缓冲区置为 20H。

```
lea  di, mess2
mov  al, 20H
mov  cx, 10
cld
rep  stosb
```

```
lea  di, mess2
mov  ax, 2020H
mov  cx, 5
cld
rep  stosw
```

串处理指令

LODS 从串取指令：

LODS SRC

LODSB (字节)

LODSW (字)

执行操作：

字节操作：(AL)←((SI))，(SI)←(SI)±1

字操作：(AX)←((SI))，(SI)←(SI)±2

注意：

* 源串一般在数据段中（允许使用段跨越前缀来修改）
* 不影响条件标志位
* LODS 指令一般不与 REP

联用，有时缓冲区中的一串字符需要依次取出来测试的时候，可以与后面即将要讲到循环指令连用来实现此功能。

wenjiecoder@outlook.com

串处理指令

与 REPE / REPZ（REPNE / REPNZ）配合工作的
CMPS 和 SCAS

REPE / REPZ（相等/为零则重复）

执行操作：

(1) 如 (CX)=0 或

ZF=0（即某次比较的结果两个操作数不相等）则退出串操作，否则转(2)

(2) (CX)←(CX) -1

(3) 执行 CMPS / SCAS

(4) 重复 (1) ~ (3)

实际上REPE和REPZ是完全相同的，只是表达的方式不同而已。与REP相比，除满足(CX)=0的条件可结束操作外，还增加了ZF=0的条件。也就是说，只要两数相等就可继续比较，如果遇到两数不相等时可提前结束操作。

串处理指令

与 REPE / REPZ（REPNE / REPNZ）配合工作的
CMPS 和 SCAS

REPNE / REPNZ（不相等/不为零则重复）

执行操作：

(1) 如 (CX)=0 或 ZF=1 则退出串操作，否则转(2)

(2) (CX)←(CX) -1

(3) 执行 CMPS / SCAS

(4) 重复 (1) ~ (3)

实际上REPNE和REPNZ是完全相同的，只是表达的方式不同而已。与REP相比，除满足(CX)=0的条件可结束操作外，还增加了ZF=1的条件。也就是说，只要两数不相等就可继续比较，如果遇到两数相等时可提前结束操作。

串处理指令

CMPS 串比较指令：

CMPS　SRC, DST

CMPSB　　(字节)

CMPSW　　(字)

用于比较两个字符串

执行操作：

(1) ((SI)) - ((DI))

根据比较结果置条件标志位：相等 ZF=1；不等 ZF=0

(2) 字节操作：(SI)←(SI)±1，(DI)←(DI)±1

字操作：(SI)←(SI)±2，(DI)←(DI)±2

例：比较两个8字节长度的字符串，找出它们不相匹配的位置。

```
lea   si, mess1
lea   di, mess2
mov   cx, 8
cld
repe  cmpsb
```

(di)：不匹配字符的下一个地址

(cx)：剩下还未比较的字符个数

串处理指令

例：比较两个字符串，找出它们不相匹配的位置

```
data segment
        string1 db 'Harbin Institute of Technology', '$'
        string2 db 'Harbin Institute of Technology', '$'
        mess1 db  'Match.',13,10,'$'
        mess2 db  'No Match!',13,10,'$'
data ends
code segment
        assume cs:code, ds:data, es:data
start:
        mov ax,data
        mov ds,ax
        mov es,ax
        lea si,string1
        lea di,string2
        cld
        mov cx, 30
        repz cmpsb
        jz match
        lea dx,mess2
        jmp near ptr disp
match:
        lea dx,mess1
disp:
        mov ah,09
        int 21h

        MOV AH,4CH
        INT 21H
code ends
        end start
```

显示代码
用来在显示器显示字符串的系统功能调用

wenjiecoder@outlook.com

串处理指令

SCAS

串扫描指令：把AL/AX的内容与由目的变址寄存器指向的在附加段中的一个字节或字进行比较。

SCAS DST

SCASB (字节)

SCASW (字)

可用于从一个字符串中查找一个指定的字符

执行操作：

字节操作：(AL) - ((DI)), (DI) ←(DI)±1

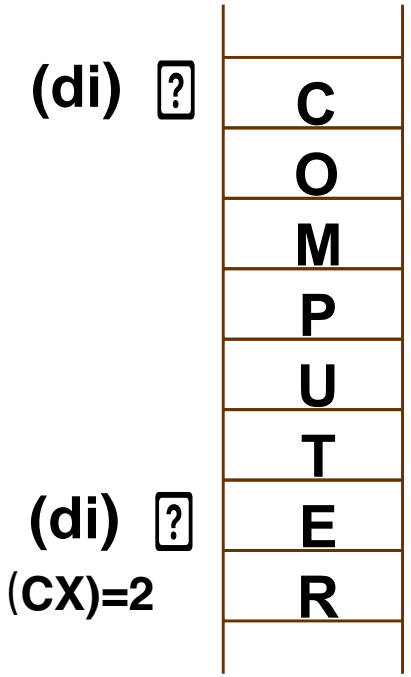
字操作： (AX) - ((DI)), (DI) ←(DI)±2

串处理指令

例：从一个字符串中查找一个指定的字符“T”

```
mess  db  ‘COMPUTER’

lea   di, mess
mov   al, ‘T’
mov   cx, 8
cld
repne scasb
```



(di)：相匹配字符的下一个地址

(cx)：剩下还未比较的字符个数

8086/8088指令概述

数据传送指令

算术指令

逻辑指令

串处理指令

处理机控制与杂项操作指令

wenjiecoder@outlook.com

处理机控制与杂项操作指令

- 标志处理指令

  CLC、STC、CMC、

  CLD、STD、

  CLI、STI

- 其他处理机控制与杂项操作指令

  NOP、HLT

wenjiecoder@outlook.com

处理机控制与杂项操作指令

1. 标志处理指令

CLC	进位位置0指令	; CF ← 0
STC	进位位置1指令	; CF ← 1
CMC	进位位求反指令	; CF ← Not(CF)
CLD	方向标志位置0指令	; DF ← 0
STD	方向标志位置1指令	; DF ← 1
CLI	中断标志位置0指令	; IF ← 0
STI	中断标志位置1指令	; IF ← 1

注意: * 以上指令只影响本指令指定的标志

2. CPU控制类指令

(1) 空操作指令　NOP（No Operation Instruction）

该指令不执行任何操作，但占用一个字节存储单元，空耗一个指令执行周期，常用于程序调试。

例如：在需要预留指令空间时用NOP填充，代码空间多余时也可以用NOP填充，还可以用NOP实现软件延时。

(2) 暂停指令　HLT（Enter Halt State Instruction）

在等待中断信号时，该指令使CPU处于暂停工作状态，CS:IP指向下一条待执行的指令。当产生了中断信号，CPU把CS和IP压栈，并转入中断处理程序。在中断处理程序执行完后，中断返回指令IRET弹出IP和CS，并唤醒CPU执行下条指令。

wenjiecoder@outlook.com

第4讲作业：

Page 108 -116：3.10、3.42

第5讲作业：

Page 111：3.23、3.27

wenjiecoder@outlook.com