

2020/12/25_16位汇编_第2课_8086相关寄存器

笔记本： 16位汇编

创建时间： 2020/12/25 星期五 10:03

作者： ileemi

- [指令系统](#)
- [寄存器](#)
- [8086 寄存器](#)
 - [通用寄存器](#)
 - [指令指针寄存器](#)
 - [Flag \(标志寄存器\)](#)
 - [debug 命令](#)
 - [将书写的汇编指令以二进制方式进行保存](#)
 - [溢出和进位](#)
- [编写HelloWorld](#)
 - [汇编编写 HelloWorld](#)
 - [二进制编写 HelloWorld](#)
- [使用Debug测试所有标志寄存器的作用](#)
 - [CF 进位标志](#)
 - [ZF 零标志](#)
 - [SF 符号标志](#)
 - [OF 溢出标志](#)
 - [PF 奇偶标志位](#)
 - [AF 辅助进位标志位](#)
 - [DF 方向标志位](#)
 - [IF 中断允许标志位](#)
 - [TF 陷间标志](#)
- [各标志数值对应的标志](#)

指令系统

ALU：算术逻辑单元（主要负责 算术运算（加法）、逻辑运算、移位运算）。

补码（数值取反加1）：将负数运算转换成加数运算。

3 - 1 ==》 3 + 255 = 2 (3 - 1 可直接写成：3 + 255 可提供运算速度)

- 计算机的指令系统就是指该计算机能够执行的全部指令的集合
- 每种计算机都有它支持的指令集合
- 16位8086指令系统是 Intel80x86系列微处理器指令系统的基础

8086：16位CPU

8080x86：32位汇编

寄存器

不同 CPU 表达的方式不一样。了解寄存器需要查看 CPU 对应的手册。

每条汇编就是一条二进制指令

硬件不具备判断能力

8086 寄存器

内部结构图：

11: 25

对程序员来说，8086内部结构的最重要的是其寄存器组：

- 8个通用寄存器
- 1个指令指针寄存器
- 1个标志寄存器
- 4个段寄存器

通用寄存器

16位通用寄存器：AX、BX、CX、DX、SP、BP、SI、DI

其中前4个数据寄存器都还可以分成高8位和低8位两个独立的寄存器（**Hight**、**Lower**）。8086的8位通用寄存器是：AH、BH、CH、DH、AL、BL、CL、DL。对其中某8位的操作，并不影响另外对应8位寄存器中的数据。

内部暂存器：CS、DS、SS（存放栈的段地址）、ES、IP

- AX（累加器）： $AX = AX + BX$ 运算效率比 $BX = BX + CX$ 要快。使用频率最高，用于算术、逻辑运算以及与外设传送信息等。
- BX（基址寄存器）：基址 + 偏移，寻址操作，基址需要放置到 "BX" 寄存器中（不是必须，取数据的时候就必须）。常用做存放存储器地址。
- CX（计数器）：循环操作时，指令放置到（不是必须）"CX" 寄存器中。作为循环和串操作等指令中的隐含计数器。
- DX（数据寄存器）：常用来存放双字长数据的高16位，或存放外设端口地址。
- SP（栈顶）：存储栈顶的偏移地址（必选），可以基址 + 偏移（区分CPU）
- BP（栈底）：存储栈底的偏移地址（可选），可以基址 + 偏移
- SI（源寄存器）：源变址寄存器，变址寄存器常用于存储器寻址时提供地址，可用来存放相对于**DS段（附加段）**之源变址指针。
- DI（目标寄存器）：目的变址寄存器，可用来存放相对于**ES段**之目的寄存器。

指令指针寄存器

- IP（指令指针寄存器）：告诉CPU执行哪个内存地址上的代码

Flag（标志寄存器）

标志（Flag）用于反映指令执行结果或控制指令执行形式，8086 处理器的各种标志形成了一个16位的标志寄存器 FLAGS（程序状态字PSW寄存器）。

程序设计需要利用标志的状态。

15	12	11	10	9	8	7	6	5	4	3	2	1	0
		OF	DF	IF	TF	SF	ZF		AF		PF		CF

CF、PF、AF、ZF、SF、TF、IF、DF、OF

状态标志：用来记录程序运行结果的状态信息，许多指令的执行都将相应地设置它

CF、PF、AF、ZF、SF、OF

- **CF（Carry Flag）：进位标志。**当运算结果的最高有效位有进位（加法）或借位（减法）时，进位标志置 "1"，即 $CF = 1$ ，反之 $CF = 0$ ；
- **ZF（Zero Flag）：零标志位。**若运算结果位0，则 $ZF = 1$ ，反之 $ZF = 0$ ；注意：ZF 为 1 表示的结果为 0；
- **SF（Sign Flag）：符号标志。**运算结果最高位为 "1"，则 $SF = 1$ ，反之 $SF = 0$ ；有符号数据用最高有效位表示数据的符号所以，最高有效位就是符号标志的状态。
- **OF（Overflow Flag）：溢出标志。**若算数运算的结果有溢出，则 $OF = 1$ ，反之 $OF = 0$ ；
- **PF（Parity Flag）：奇偶标志位。**记录相关指令执行后，其结果的最低位中（低八位）1 的个数是否为偶数（计算1的数量）。如果 1 的个数是偶数，则 $PF = 1$ ，如果为奇数，则 $PF = 0$ ；（现在很少用，硬件已经会进行自动判断，保留下来的标志。多用于奇偶校验）
- **AF（Auxiliary Flag）：辅助进位标志位。**运算时低半字节有进位或借位时， $AF = 1$ ；否则 $AF = 0$ ；（不常用，保留下来的标志）

控制标志：可由程序根据需要用指令进行设置，用于控制处理器执行指令的方式（不用在运算上，让CPU进行干活的标志）

DF、IF、TF

- **DF（Direction Flag）：方向标志位。**拷贝数据的时候用到。 $DF = 1$ ，拷贝数据地址递减， $DF = 0$ ，拷贝数据地址递增（cld 等价 $DF = 0$ （递增），std 等价 $DF = 1$ （递减））
- **IF（Interrupt Flag）：屏蔽中断允许标志位。**用于控制外部可屏蔽中断是否可以被处理器响应， $IF = 1$ ，则允许中断（类似上课时，手机有电话过来，会接电话，打断正在做的事情）； $IF = 0$ ，则禁止中断（类似于上课时，将手机设置为"飞行模式"）；硬件厂家或者 CPU 作者会用到。**特殊用途，不常用。**
- **TF（Trap Flag）：陷阱标志。** $TF = 1$ ，代码执行一行代码后会停止，不执行下一行代码（类似于 debug 时的 F11）。**特殊用途，不常用。**

Win7 以上的系统不支持 16 位汇编程序。Win XP 上自带了微软提供的汇编翻译器："debug"（既可以当翻译器也可以当作调试器）。

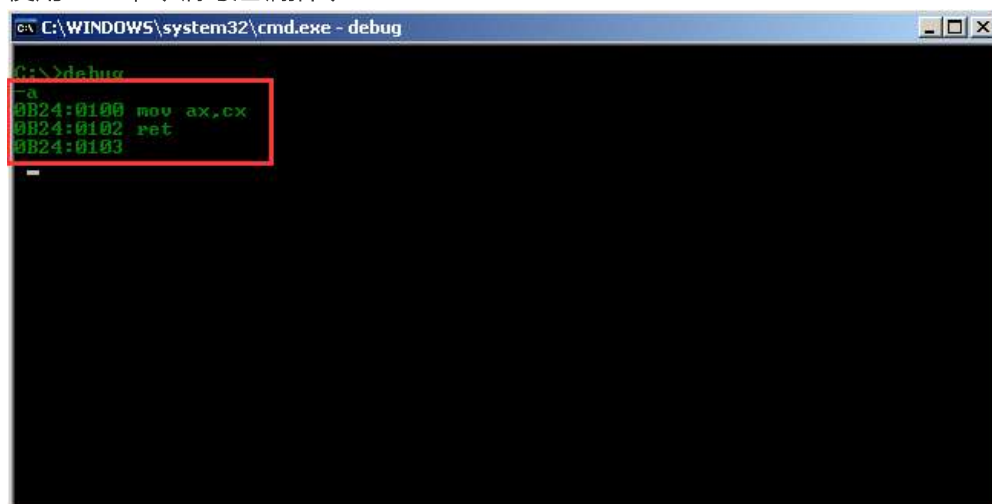
debug 命令

- p: 相当于 VS 调试时的 F10, 会跳过函数
- t: 相当于 VS 调试时的 F11, 会单步进入
- r: 查看当前CPU的各个寄存器数值信息
- e: 从指定地址开始, 修改内存数据
- d: 从指定地址开始查看其内存中的数据
- u: 查看反汇编
- g: debug Hello.com -g 直接运行程序

将书写的汇编指令以二进制方式进行保存

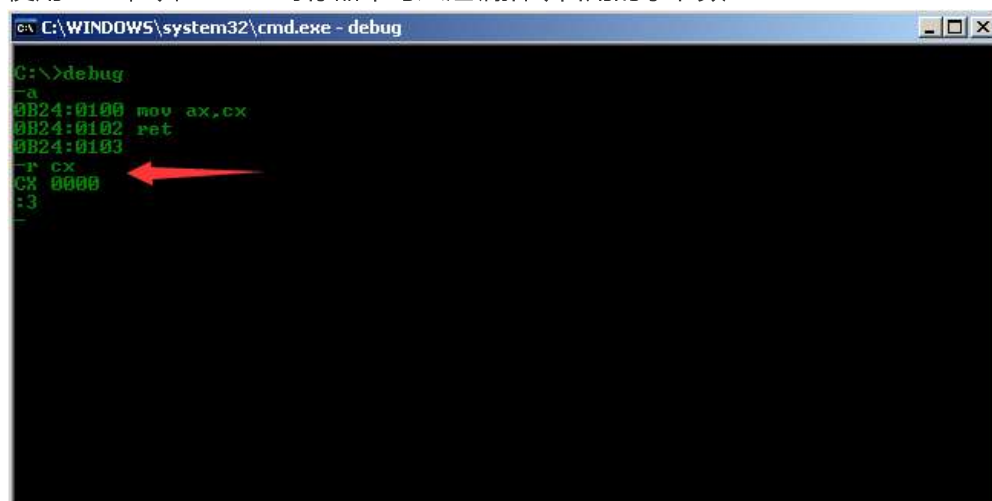
操作步骤:

- 使用 "a" 命令编写汇编指令:



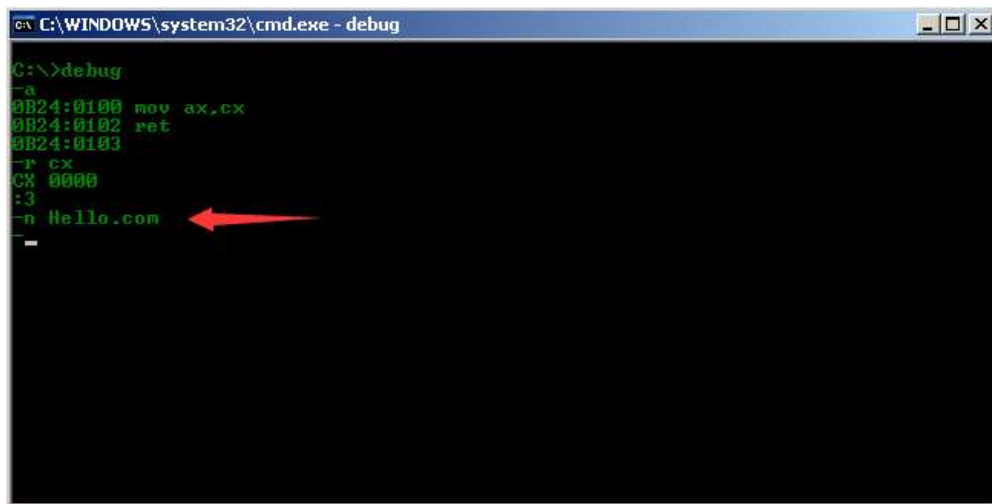
```
C:\WINDOWS\system32\cmd.exe - debug
C:\>debug
-a
0B24:0100 mov ax, cx
0B24:0102 ret
0B24:0103
-
```

- 使用 "r" 命令在 "CX" 寄存器中写入汇编指令占用的字节数:

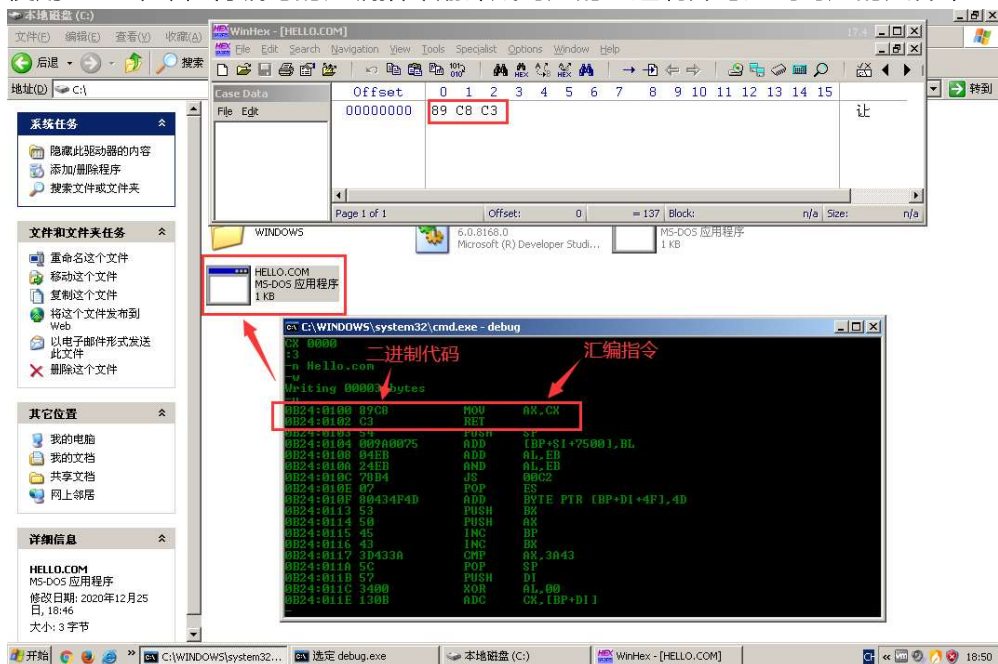


```
C:\WINDOWS\system32\cmd.exe - debug
C:\>debug
-a
0B24:0100 mov ax, cx
0B24:0102 ret
0B24:0103
-r cx
CX 0000
:3
-
```

- 使用 "n" 命令, 命名需要保存的二进制文件名 (.com后缀文件内容数据为纯二进制):



- 使用 "w" 命令，将编写的汇编指令翻译成对应的二进制并写入到对应的文件中：



溢出和进位

- 溢出标志OF和进位标志CF是两个意义不同的标志
- 进位标志表示无符号数运算结果是否超出范围，运算结果仍然正确
- 溢出标志表示有符号数运算结果是否超出范围，运算结果已经不正确。

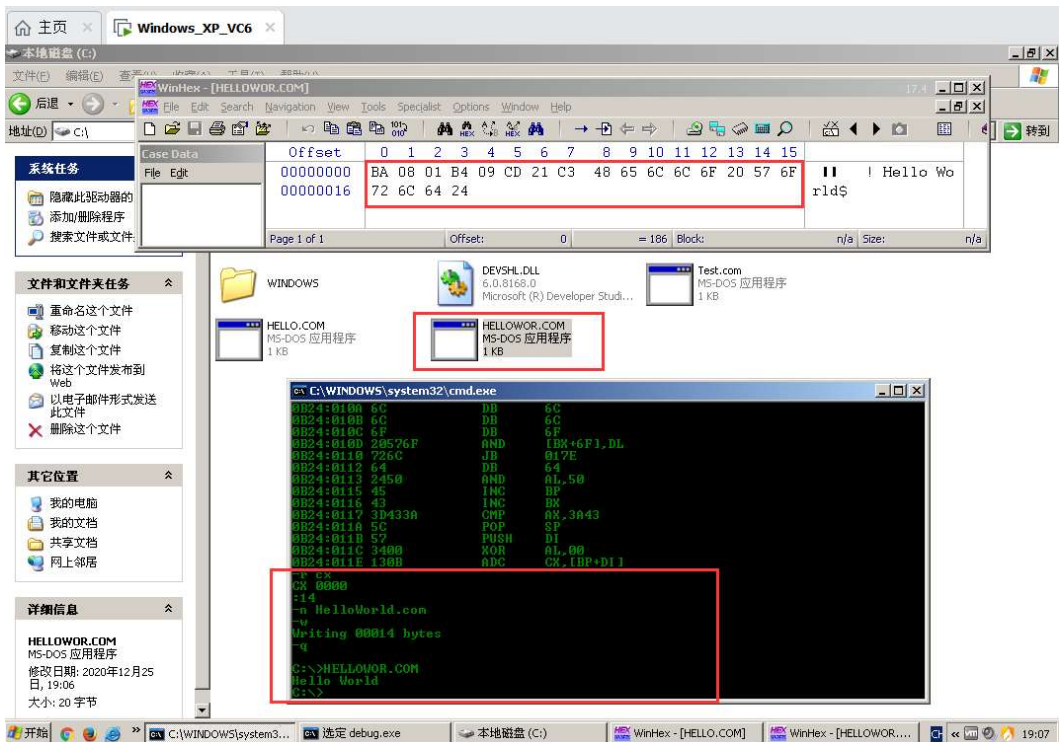
编写HelloWorld

汇编编写 HelloWorld

在指定内存进行写入：

```
C:\WINDOWS\system32\cmd.exe - debug
C:\>debug
-a
0B24:0100 mov dx,100
0B24:0103 mov ah,09
0B24:0105 int 21
0B24:0107 ret
0B24:0108
-e 100 "Hello World$"
-d 100
0B24:0100 48 65 6C 6C 6F 20 57 6F Hello Wo
0B24:0110 72 6C 64 24 50 45 43 3D-43 3A 5C 57 34 00 13 0B rld$PEC=C:\W4...
0B24:0120 57 53 5C 53 59 53 54 45-4D 33 32 5C 43 4F 4D 4D WS\SYSTEM32\COMM
0B24:0130 41 4E 44 2E 43 4F 4D 00-41 4C 4C 55 53 45 52 53 AND.COM.ALUSERS
0B24:0140 50 52 4F 46 49 4C 45 3D-43 3A 5C 44 4F 43 55 4D PROFILE=C:\DOCU
0B24:0150 45 7E 31 5C 41 4C 4C 55-53 45 7E 31 00 41 50 50 E~1\ALLUSE~1.APP
0B24:0160 44 41 54 41 3D 43 3A 5C-44 4F 43 55 4D 45 7E 31 DATA=C:\DOCU~1
0B24:0170 5C 35 31 61 73 6D 5C 41-50 50 4C 49 43 7E 31 00 \51asm\APPLIC~1.
0B24:0180 43 4C 49 45 4E 54 4E 41 CLIENTNA
```

运行效果：

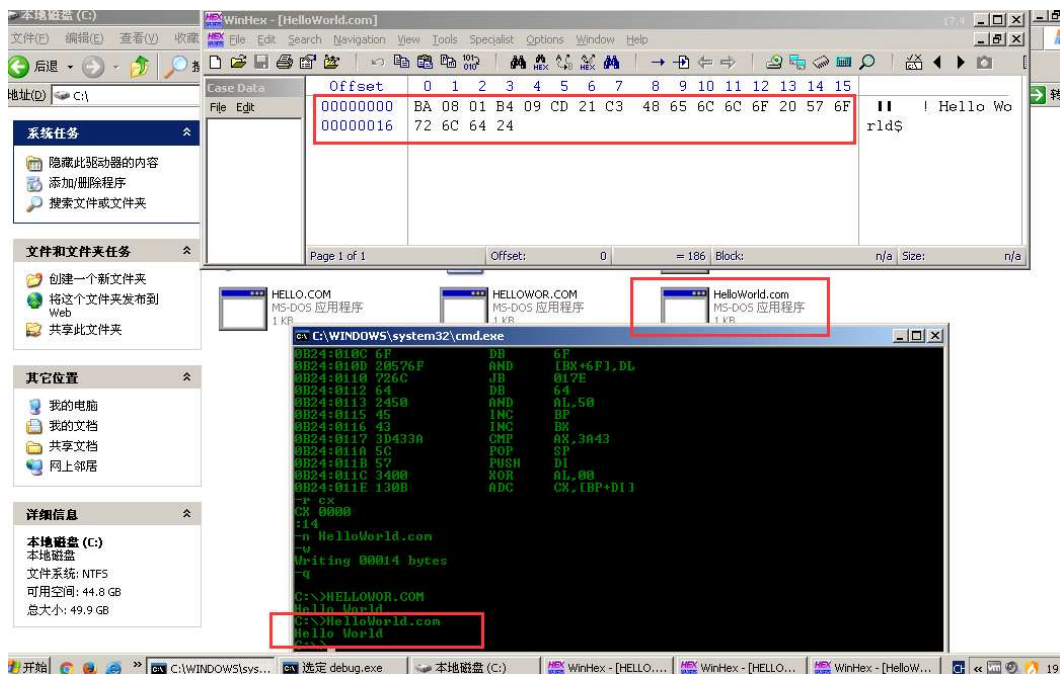


二进制编写 HelloWorld

打开 "WinHex" 进行编写：



运行效果：



"BA" --> 指令 "mov dx" 的二进制代码

"B4" --> 指令 "mov ah" 的二进制代码 (mov ah,09 简单认为就是调用操作系统的API)。

在 Win XP 系统中，只要运行一个 ".com" 程序就会将文件中的代码加载到内存 "0x100" 的位置 (类似于模块基址，其是固定的)。

使用Debug测试所有标志寄存器的作用

例如：ZF = 1 ==> ZR ZF = 0 ==> NZ

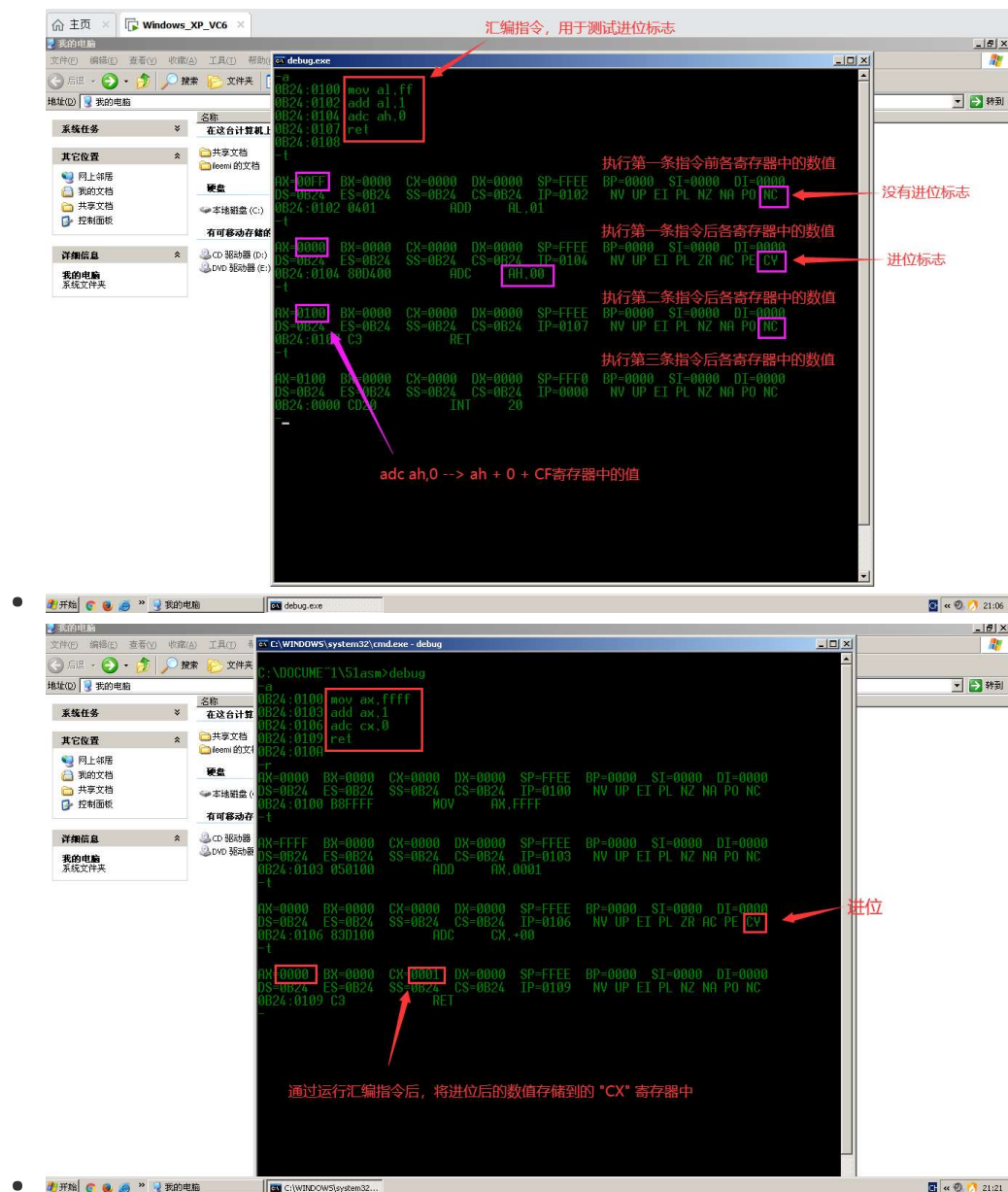
CF 进位标志

CF (Carry Flag 进位标志)：当运算结果的最高有效位有进位 (加法) 或借位 (减法) 时，进位标志置 "1"，即 CF = 1，反之 CF = 0；

汇编指令：

- 测试1：
 - a
 - 0B24:0100 mov al,ff
 - 0B24:0102 add al,1
 - 0B24:0104 adc ah,0
 - 0B24:0107 ret
- 测试2：
 - a
 - 0B24:0100 mov ax,ffff
 - 0B24:0103 add ax,1
 - 0B24:0106 adc cx,0
 - 0B24:0109 ret
 - 0B24:010A

操作示例：



结论：有进位寄存器 "CF" 的标志为 "CY", 没有进位寄存器 "CF" 的标志为 "NC"。"CX" 寄存器在产生进位的时候才会有作用。

ZF 零标志

ZF (Zero Flag 零标志位)：若运算结果位0, 则 ZF = 1, 反之 ZF = 0; 注意：ZF 为 1 表示的结果为 0;

对 "AX 寄存器中的数据和 BX 寄存器中的数据进行相减" 进行测试。

汇编指令：

-r ax

AX 0000

:21

-r bx

BX 0000

:21

-a

0B24:0100 mov cx,ax

0B24:0102 sub cx,bx

0B24:0104 ret

0B24:0105

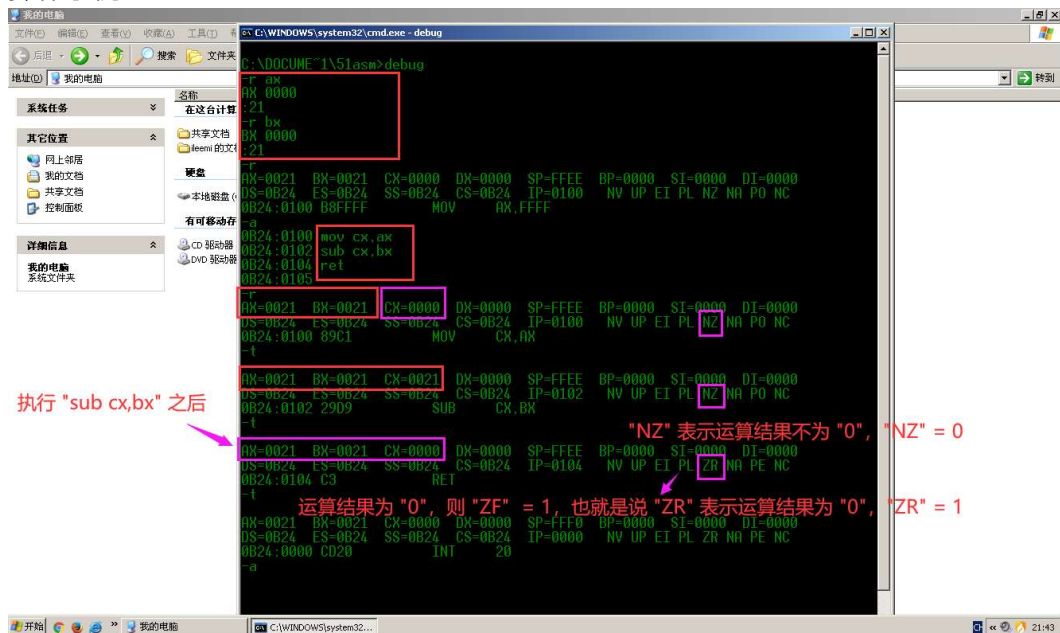
-r

AX=0021 BX=0021 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000

DS=0B24 ES=0B24 SS=0B24 CS=0B24 IP=0100 NV UP EI PL NZ NA PO NC

0B24:0100 89C1 MOV CX,AX

操作示例：



结论：运算结果为 "0"，零标志寄存器 "ZF" 的标志为 "ZR"，运算结果不为 "0"，零标志寄存器 "ZF" 的标志为 "NZ"。

影响标志寄存器的指令：add、sub、mul、div、

SF 符号标志

运算结果最高位为 "1"，则 "SF" = 1，反之 "SF" = 0；有符号数据用最高有效位表示数据的符号所以，最高有效位就是符号标志的状态。

示例 ("H" 表示十六进制数据)：

3AH + 7CH = B6H；对应的二进制为：1011 0110

B6H 对应的二进制数据的最高位为 "1"，所以 "SF" = 1

85H + 7CH = (1)00H；对应的二进制为：1（进位） 0000 0000

(1)00H 对应的二进制数据的最高位为 "0"，所以 "SF" = 0

汇编指令：

-a

0B24:0100 mov ax,0

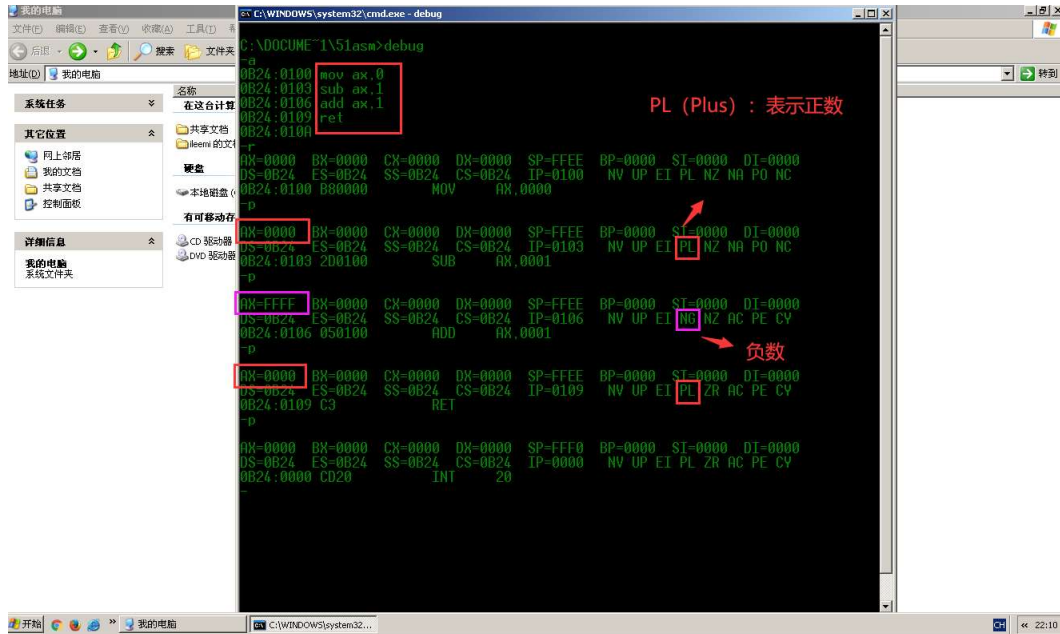
0B24:0103 sub ax,1

0B24:0106 add ax,1

0B24:0109 ret

0B24:010A

操作示例:



结论: "SF" = 0, 其表示正数, 汇编中对应的标志为 "PL", "SF" = 1, 其表示负数, 汇编中对应的标志为 "NG".

OF 溢出标志

"进位" 计算结果正确

"溢出" 计算结果错误

若算数运算的结果有溢出, 则 "OF" = 1, 反之 "OF" = 0;

示例 ("H" 表示十六进制数据):

3AH + 7CH = B6H; 对应的二进制为: 1011 0110, 产生溢出: OF = 1

- 无符号数运算: $58 + 124 = 182$ (范围内, 无进位)
- 有符号数运算: $58 + 124 = 182$ (范围外, 有溢出)

AAH + 7CH = (1)26H; 对应的二进制为: 1 (进位) 0010 0110

没有溢出: OF = 0

- 无符号数运算: $170 + 124 = 294$ (范围外, 有进位)
- 有符号数运算: $-86 + 124 = 38$ (范围内, 无溢出)

两个寄存器做运算:

- 无符号数据进行加法或者减法运算, 只需要查看 "CF" 标志为即可
- 有符号数据进行加法或者减法运算, 只需要查看 "OF" 标志为即可, 如果溢出, 其结果不能使用。

CPU 硬件判断溢出的依据 (正负数判断最高位是否为1):

正数 + 正数 = 负数 (溢出)

负数 + 负数 = 正数 (溢出)

测试1:

- 汇编代码:

-a

0B24:0100 mov ax,7fff

0B24:0103 add ax,1

0B24:0106 ret

0B24:0107

操作示例:

```
C:\WINDOWS\system32\cmd.exe - debug
C:\DOCUMENT~1\51asm>debug
-a
0B24:0100 mov ax,7fff
0B24:0103 add ax,1
0B24:0106 ret
0B24:0107
-r
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0B24 ES=0B24 SS=0B24 CS=0B24 IP=0100 NV UP EI PL NZ NA PO NC
0B24:0100 B8FF7F MOV AX,7FFF
-p
AX=7FFF BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0B24 ES=0B24 SS=0B24 CS=0B24 IP=0103 NV UP EI PL NZ NA PO NC
0B24:0103 050100 ADD AX,0001
-p
AX=8000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0B24 ES=0B24 SS=0B24 CS=0B24 IP=0106 OV UP EI NG NZ AC PE NC
0B24:0106 C3 RET
-p
AX=8000 BX=0000 CX=0000 DX=0000 SP=FFF0 BP=0000 SI=0000 DI=0000
DS=0B24 ES=0B24 SS=0B24 CS=0B24 IP=0000 OV UP EI NG NZ AC PE NC
0B24:0000 CD20 INT 20
-
两个正数: 7fff, 1 进行 "add", 结果为负数: "8000",
通过溢出标志可知其结果产生溢出
```

测试2:

- 汇编代码:

-a

0B24:0100 mov ax,8000

0B24:0103 add ax,8000

0B24:0106 ret

0B24:0107

操作示例：

```
C:\WINDOWS\system32\cmd.exe - debug
C:\DOCUME~1\51asm>debug
~a
0B24:0100 mov ax,8000
0B24:0103 add ax,8000
0B24:0106 ret
0B24:0107
~r
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0B24 ES=0B24 SS=0B24 CS=0B24 IP=0100 NV UP EI PL NZ NA PO NC
0B24:0100 B80080 MOV AX,8000
~p
AX=8000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0B24 ES=0B24 SS=0B24 CS=0B24 IP=0103 NV UP EI PL NZ NA PO NC
0B24:0103 050080 ADD AX,8000
~p
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0B24 ES=0B24 SS=0B24 CS=0B24 IP=0106 OV UP EI PL ZR NA PE CY
0B24:0106 C3 RET
~p
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFF0 BP=0000 SI=0000 DI=0000
DS=0B24 ES=0B24 SS=0B24 CS=0B24 IP=0000 OV UP EI PL ZR NA PE CY
0B24:0000 CD20 INT 20
~
两个负数相加结果为正数, CPU 检测结果溢出了
8000 + 8000 = 0000
```

结论：

正数 + 正数 = 负数（溢出）

负数 + 负数 = 正数（溢出）

在8086中溢出标志 "OF" 用 "OV" 表示，没有溢出用 "NV" 表示。

PF 奇偶标志位

PF (Parity Flag)：奇偶标志位。记录相关指令执行后，其结果的所有 bit 位中 1 的个数是否为偶数。如果 1 的个数是偶数，则 PF = 1，如果为奇数，则 PF = 0；

汇编指令：

- 指令1：

~a

0B24:0100 add al,d

0B24:0102 ret

0B24:0103

执行后，结果为 0000 1101B，其中有3（奇数）个1，则 PF = 0；8086中对应的标志为：PO；

```
debug.exe
~a
0B24:0100 add al,d
0B24:0102 ret
0B24:0103
~r
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0B24 ES=0B24 SS=0B24 CS=0B24 IP=0100 NV UP EI PL NZ NA PO NC
0B24:0100 0400 ADD AL,0D
~t
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0B24 ES=0B24 SS=0B24 CS=0B24 IP=0102 NV UP EI PL NZ NA PO NC
0B24:0102 C3 RET
~
```

- 指令2:

-a

0B24:0100 add al,3

0B24:0102 ret

0B24:0103

执行后，结果为 0000 0011B，其中有2（奇数）个1，则 PF = 1；8086中对应的标志为：**PE**；

```

debug.exe
-a
0B24:0100 add al,3
0B24:0102 ret
0B24:0103
-r
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0B24 ES=0B24 SS=0B24 CS=0B24 IP=0100 NV UP EI PL NZ NA PO NC
0B24:0100 0403 ADD AL,03
-t
AX=0003 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0B24 ES=0B24 SS=0B24 CS=0B24 IP=0102 NV UP EI PL NZ NA PE NC
0B24:0102 C3 RET
-

```

AF 辅助进位标志位

AF (Auxiliary Flag) : 辅助进位标志位

这个位表示加减法做到一半时有没有形成进位或者借位，如果有则AF=1。

汇编指令：

-a

0B24:0100 mov al,e

0B24:0102 mov bl,8

0B24:0104 add al,bl

0B24:0106 ret

0B24:0107

最后结果为：al = 16H (0001 0110H)，这就是低4位向高4位进位。反之在减法中第3位不够减去向第4位借位（注意数位是从第0位开始数的）叫低4位向高4位借位。像前面的 AL中前四位为高四位,后四位为低四位。例如,当两个字节相加时，如果从低4位向高4位有进位时，则 AF = 1。8086中对应的标志为 **AC**。反之 AF = 0，对应的标志为 **NA**。

```
debug.exe
-a
0B24:0100 mov al,e
0B24:0102 mov bl,8
0B24:0104 add al,bl
0B24:0106 ret
0B24:0107
-r
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0B24 ES=0B24 SS=0B24 CS=0B24 IP=0100 NV UP EI PL NZ NA PO NC
0B24:0100 B00E MOV AL,0E
-t
AX=000E BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0B24 ES=0B24 SS=0B24 CS=0B24 IP=0102 NV UP EI PL NZ NA PO NC
0B24:0102 B308 MOV BL,08
-t
AX=000E BX=0008 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0B24 ES=0B24 SS=0B24 CS=0B24 IP=0104 NV UP EI PL NZ NA PO NC
0B24:0104 00D8 ADD AL,BL
-t
AX=0016 BX=0008 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0B24 ES=0B24 SS=0B24 CS=0B24 IP=0106 NV UP EI PL NZ AC PO NC
0B24:0106 C3 RET
-
```

0016H == 0001 0110H, 低四位向高四位进行了进位

DF 方向标志位

DF (Direction Flag) : 方向标志位

在串处理指令中, 控制每次操作后, si (指向原始偏移地址)、di (指向目标偏移地址) 的增减。

df=0时, 每次操作后, si、di递增; 8086中对应的标志为: **UP**

df=1时, 每次操作后, si、di递减。8086中对应的标志为: **DN**

汇编指令:

cld

std

IF 中断允许标志位

IF (Interrupt Flag) : 中断允许标志位

当 IF = 1时, CPU 在执行完当前指令后响应中断, 引发中断过程; 8086中对应的标志为: **IF**

当 IF = 0时, CPU 在执行完当前指令后不响应可屏蔽中断。8086中对应的标志为: **DI**

TF 陷间标志

TF (Trap Flag) : 定时器溢出标志。这个位主要是用来在debug 中进行 -t 指令时使用的。当 CPU 在执行完一条指令后, 如果检测到 TF 位的值为 1, 则产

生单步中断，引发中断过程。通过这个位，我们就可以在 debug 中对程序进行单步跟踪。

各标志数值对应的标志

标志	1	0
CF	CY	NC
PF	PE	PO
AF	AC	NA
ZF	ZR	NZ
SF	NG	PL
IF	EI	DI
DF	DN	UP
OF	OV	NV