

2020/12/28_16位汇编_第3课_控制寄存器、段寄存器

笔记本： 16位汇编

创建时间： 2020/12/28 星期一 9:54

作者： ileemi

- [Flag 标记（控制）寄存器](#)
 - [DF 方向标志位](#)
 - [IF 屏蔽中断允许标志位](#)
 - [TF 陷间标志](#)
- [数据信息的表达单元](#)
- [存储单元及其存储内容](#)
- [数据的地址对齐](#)
- [存储器的分段管理](#)
 - [物理地址和逻辑地址](#)
 - [逻辑地址](#)
 - [物理地址和逻辑地址的转换](#)
- [段寄存器](#)
 - [如何分配各个逻辑段](#)
 - [测试汇编指令](#)

Flag 标记（控制）寄存器

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。**

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：

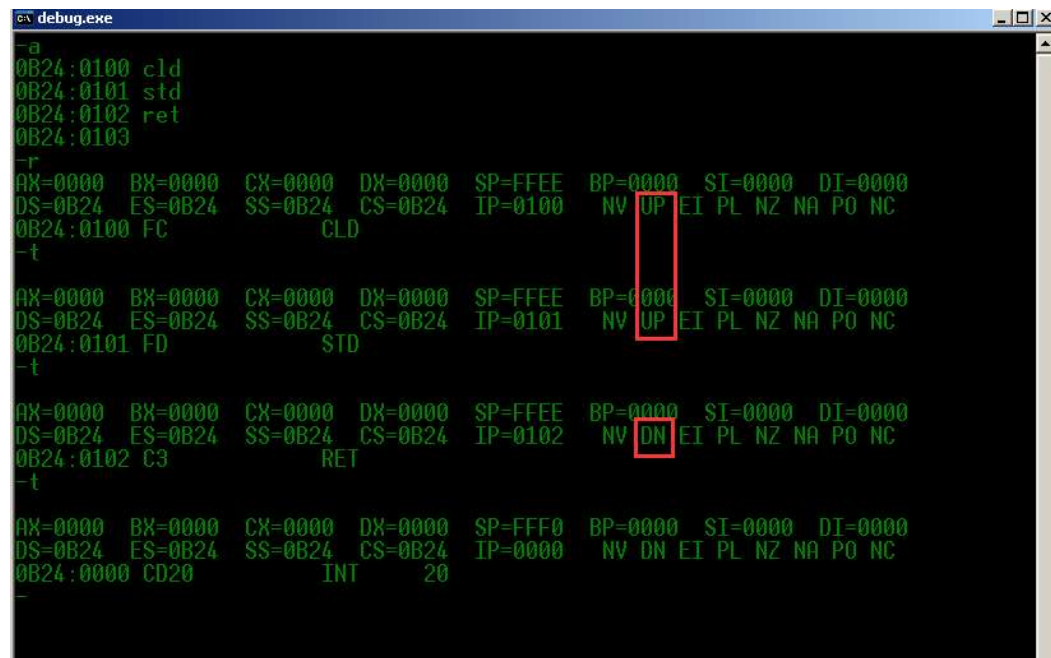
-a

0B24:0100 cld

0B24:0101 std

0B24:0102 ret

0B24:0103



```
debug.exe
-a
0B24:0100 cld
0B24:0101 std
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 FC          CLD
-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=0101 NV UP EI PL NZ NA PO NC
0B24:0101 FD          STD
-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 DN EI PL NZ NA PO NC
0B24:0102 C3          RET
-t
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 NV DN EI PL NZ NA PO NC
0B24:0000 CD20        INT     20
-
```

IF 屏蔽中断允许标志位

IF (Interrupt Flag)：屏蔽中断允许标志位

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：**IF**

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。

数据信息的表达单元

1. 计算机中信息的单位：

- 二进制：存储一位二进制数（0或1）
- 字节：8个二进制位（D7~D0）
- 字：16位，两个字节（D15~D0）
- 双字节：32位，4个字节（D31~D0）

2. 最低有效位（LSB）：数据的最低位（D0位）。

3. 最高有效位（MSB）：数据的最高位，对应字节、字、双字分别值D7、D15、D31位。

Inter CPU 数据在内存中是以小尾方式进行存储的（十六制数据：0x1234，在内存中以 "1234" 方式存放为 "大尾方式"，在内存中以 "4321" 方式存放为 "小尾方式"）。

存储单元及其存储内容

1. 每个存储单元都有一个编号（地址）；被称为存储器地址。

2. 每个存储器单元存放一个字节的內容（内存中存放数据的最低单位为：字节，不能少于8个字节），示例：

在内存单元内 0002H 的位置存放一个数据 34H 可以表达为[0002H] = 34H；

在内存单元内 0002H 的位置存放一个数据 03H 可以表达为[0002H] = 03H（不能写成3H）；

3. 多字节数据在内存中存放数据的方式：

- 存放时，低字节存入低地址，高字节存入高地址；
- 表达时，用它的低地址表示多字节数据占据的地址空间。

"字" 单元的存储可表示为：[0002H] = 1234H；

"双字" 单元的存储可表示为：[0002H] = 87651234H；

80x86处理器采用 "低对低、高对高" 的存储形式，被称为 "小端（小尾）方式Little Endian"。相对应还有 "大端（大尾）方式Big Endian"。

数据的地址对齐

1. 同一个存储器地址可以是字节单元地址、字单元地址、双字单元地址等等。

2. 字单元安排在偶地址（xxx0B）、双字单元安排在模4地址（xx00B）等，被称为 "地址对齐"。

3. 对于不对齐地址的数据，处理器访问时，需要额外的访问存储器时间。

4. 应该将数据的地址对齐，以取得较高的存取速度。

16位 CPU 从内存中获取一个字节两个字的时间效率是一样的。在设计的时候，16位 CPU 规定每次获取数据的时候就传输 2 个字节的数据（需要一个字节数据给你一个字节数据）。导致 CPU 只能访问地址为 2 的幂的内存地址。

存放数据时，要注意数据的存储效率（16位CPU：数据尽量存放到地址为2的倍数上，32位CPU：数据尽量存放到地址为4的倍数上）。这就叫做内存对齐，内存对齐

可以使数据的访问效率加快。

16位处理器能处理的最大数据范围是 $2^{16} = 0 \sim 0\text{xFFFF}$;

16位处理器支持的最大内存地址为: $65536 / 1024 = 64\text{KB}$;

20位处理器能处理的最大数据范围是:

$2^{20} = 0\text{x}000000 \sim 0\text{x}\text{FFFFFF}$;

16位处理器支持的最大内存地址为:

$1048576 / 1024 = 1024\text{KB}$;

32位处理器支持的最大内存地址为: $2^{32} = 4\text{GB}$;

内存存储的发展要比CPU的发展要快。

16位 CPU 支持 1 MB的内存, 可行办法就是, 继续使用 16 位CPU, 用两个寄存器去表达一个地址 (**内存地址升级为 20 位, CPU 总线也从 16 根升级到 20 根, 寄存器保留 16 位**)。

存储器的分段管理

代码和数据分开, 方便软件的编写, 防止代码写错, 错误代码后面的数据需要挪动。主要解决数据的寻址问题。

汇编编程的第一步就是先进行内存分段。

1. 8086CPU有20条地址线:

- 最大可寻址控件为: $2^{20} = 1\text{MB}$;
- 物理地址范围从 $0\text{x}000000\text{H} \sim 0\text{x}\text{FFFFFFH}$;

2. 8086PU将1MB空间分成许多逻辑段 (Segment) :

- 每个段最大限制为: 64KB
- 段地址的低 4 位为: 0000B

3. 这样, 一个存储单元除具有一个**唯一的物理地址**外, 还具有**多个逻辑地址**。

分段操作由程序员自己去分配。**CPU访问内存地址必须通过分段 (段首地址 + 偏移) 访问**。编程中不能直接给物理地址, 寄存器放不下。

物理地址和逻辑地址

1. 对应每个物理存储单元都有一个唯一的20位编号, 就是物理地址, 从 $0\text{x}000000 \sim 0\text{x}\text{FFFFFF}$ 。

2. 分段后在用户编程时, 采用逻辑地址, 形式为:

- 段基地址 : 段内偏移地址 (":" 为分隔符)

一个寄存器表示偏移最多为16位 (64KB)。另外一个寄存器来表示段的首地址 (段的首地址位20位, 一个寄存器最多可以表示16位)。

16位的寄存器存储20位的地址，需要保证20位地址的低4位都为0，为零的地址部分可以不进行存储。只需将不是0的地址存储到16位的寄存器中即可（例如：FFFF0，只需将FFFF告诉CPU，CPU拿到FFFF后自动将低四位补0）。通俗的讲就是内存条给CPU16位的数据，CPU将对应的20位地址告诉内存条。

例如：访问0x12345内存地址

寄存器1：0x1234

寄存器2：0x5

寄存器1 和 寄存器2 分别将两个地址告诉 CPU，CPU 会将寄存器2 的地址左移 4 位在 和 寄存器1 给的地址相加，之后访问相加后的内存地址。

段的首地址必须是16的倍数（便于寄存器存储）。16 位 CPU，一个段最小位 16 字节。

逻辑地址

1. 段地址说明逻辑段在主存中的起始位置。
2. 8086规定段地址必须是 16 的倍数：0x1234。
3. 省略低 4 位0000B，段地址就可以用16位数据表示，就能用 16 位段寄存器表达段地址。
4. 偏移地址说明主存单元距离段起始位置的偏移量。
5. 每段不超过 64 KB,偏移地址也可用 16 位数据表示。

物理地址和逻辑地址的转换

1. 将逻辑地址中的段地址左移 4 位，加上偏移地址（16位）就得到 20 位的物理地址。
2. 一个物理地址可以有多个逻辑地址。

逻辑地址：0x1114:100，对应的物理地址为：11240H

逻辑地址：0x1034:F00，对应的物理地址为：11240H

段寄存器

8086有 4 个 16 位的段寄存器，分别是：**CS、DS、ES、SS**

- CS（代码段）：指明代码段的起始地址
- DS（数据段）：指明数据段的起始地址
- ES（附加段）：指明附加段的起始地址
- SS（堆栈段）：指明堆栈段的起始地址

每个段寄存器用来确定一个逻辑段的起始地址，每种逻辑段均有各自的用途

在程序中只能段加偏移去指明内存要访问的内存地址。

0AFB:0100 --> 对应的物理地址为：0AFB0+0100 == 00B0B

如何分配各个逻辑段

- 程序的指令序列必须安排在代码段。
- 程序使用的堆栈一定在堆栈段。
- 程序中的数据默认是安排在数据段，也经常安排在附加段，尤其是串操作的目的区必须是附加段。
- 数据的存放比较灵活，实际上可以存放在任何种逻辑段中。

测试汇编指令

-d 1234:0

-e 1234:0 a

-d 1234:0

-d 1230:40

-d 1224:100

-d

-d 100 默认段来自 DS 寄存器

mov ax, ex:[2000]

mov ax,100

mov dx,ax

mov ax,[2000]

告诉 CPU 将数据放到 0100 段开始地址处。

mov ax,[2000] 等价于 mov ax,0100:2000

最终访问的物理内存地址为：01000+2000 = 03000

CPU访问代码是根据 CS:IP 指向的内存地址进行访问的。

防止程序出现问题，段寄存器一般不要存放数据，存放数据一般使用通用寄存器。