

2020/12/30_16位汇编_第5课_存储器寻址方式、asm文件的编写

笔记本： 16位汇编

创建时间： 2020/12/30 星期三 10:06

作者： ileemi

- [存储器寻址方式](#)
 - [直接寻址方式](#)
 - [寄存器间接寻址方式](#)
 - [寄存器相对寻址方式](#)
 - [基址变址寻址方式（基址+变址）](#)
 - [相对基址变址寻址方式](#)
- [对所有寻址方式的性能做一个排序](#)
- [测试寄存器间接寻址方式使用的默认段](#)
- [跨段拷贝数据](#)
- [.com 文件的缺点](#)
- [文件格式](#)
- [数据段的定义](#)
- [栈段的定义](#)

存储器寻址方式

8086设计了五种存储器寻址方式：

- 直接寻址方式
- 寄存器间接寻址方式
- 寄存器相对寻址方式
- 基址变址寻址方式
- 相对基址变址寻址方式

多次通过内存进行数据访问会导致得出最终结果的时间效率变慢。解决效率慢的方法可以通过对代码进行优化，减少内存访问次数。

直接寻址方式

有效地址在指令中直接给出，汇编指令示例：

```
mov ax,[2000]
```

```
mov [2000],ax
```

```
mov [2000],[3000] --> 错误写法
```

只取一个字节：mov al,[2000] 等价于 mov al,byte ptr [2000] (byte ptr：一个字节的指针)

只取一个字节：mov ax,[2000] 等价于 mov ax,word ptr [2000] (word ptr: 两个字节的指针)

将代码段偏移100位置上的数据送到寄存器ah中：

汇编代码：

mov ah, byte ptr cs:[200] --> 在xp中语法不通过，有些编译器允许这样写

需要使用下面的汇编语法：

-a

0B24:0100 cs:

0B24:0101 mov ah,byte ptr [200]

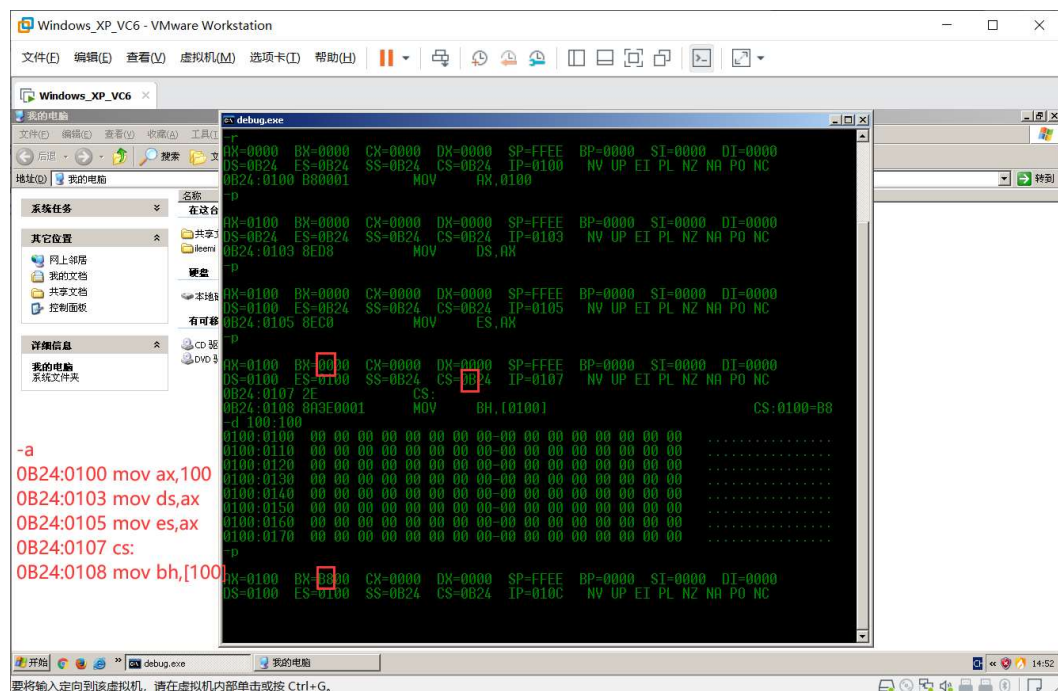
0B24:0105

-u

0B24:0100 2E CS:

0B24:0101 8A260002 MOV AH,[0200]

"CS:": 这里的 "cs:" 只针对下一条汇编语法有效，这样的写法叫 "段超越前缀指令"。段超越前缀指令默认为 ds 或者 es (由寄存器决定)。



段超越前缀指令

汇编指令：

cs::

mov ah, byte ptr [2000]

立即数寻址、直接寻址 默认使用的都是 "ds"。

通过 debug 单步走的时候，可以使用 "-r ip" 命令重新走 ip 指定地址上的代码。使用 "-r f" 命令可以修改指定标志寄存器中的值。汇编代码示例：

-r f

NV UP EI PL NZ NA PO NC -ZR DN CY

-r f

NV DN EI PL ZR NA PO CY -

寄存器间接寻址方式

有效地址**只能**存放在基址寄存器 bx、bp 或 变址寄存器 si、di 中，默认的段地址在 DS 段寄存器。

汇编指令：mov ax,[bx] ==> **mov ax,word ptr [bx]**

对应的反汇编：8B07 MOV AX,[BX]

8B: mov

07: ax bx

07 (00 000 111) : 前两位决定是 "寄存器间接寻址" 还是 "寄存器相对寻址方式" (00 -- 寄存器间接寻址, 01/11 -- 寄存器相对寻址方式)

间接寻址时，源寄存器只能是 基址寄存器 (bx、bp) 或者变址寄存器 (si、di) 。
16位汇编的7种寻址方式决定了16位汇编的语法。

寄存器相对寻址方式

相对寻址方式，源寄存器只能是基址寄存器或者是变址寄存器，偏移由编码决定的。间接寻址时，源寄存器只能是 基址寄存器 (bx、bp) 或者变址寄存器 (si、di) 。

有以下四种书写方式：

MOV AX,[bx+06H]

MOV AX,[bp+06H] --> AX←SS:[bp+06H]

MOV AX,[si+06H] --> AX←DS:[si+06H]

MOV AX,[di+06H] --> AX←SS:[di+06H]

汇编指令示例：mov ax,1000[si] == mov ax,[si+1000]

对应的反汇编为：8B840010 MOV AX,[SI+1000]

8B: mov

84 (10 000 100) : ax si

0010: 1000

84 (10 000 100) : 前两位决定是 "寄存器间接寻址" 还是 "寄存器相对寻址方式" (00 -- 寄存器间接寻址, 01/11 -- 寄存器相对寻址方式)

基址变址寻址方式（基址+变址）

有效地址 由 基址寄存器 (BX或BP) 的内容加上 变址寄存器 (SI或DI) 的内容构成（**没有相对就没有偏移**）：

有效地址 = BX/BP + SI/DI

段地址对应BX基址寄存器默认是DS，对应BP基址寄存器默认是SS，可用段超越前缀改变。

mov ax,[bx+si] ds段

mov ax,[bp+si] ss段

```
mov ax,[bp+di] ss段
mov ax,[bx-1] --> -1 == +FF
```

汇编指令示例: `mov ax,[bx][si]`
对应的反汇编为: `8B00 MOV AX,[BX+SI]`

相对基址变址寻址方式

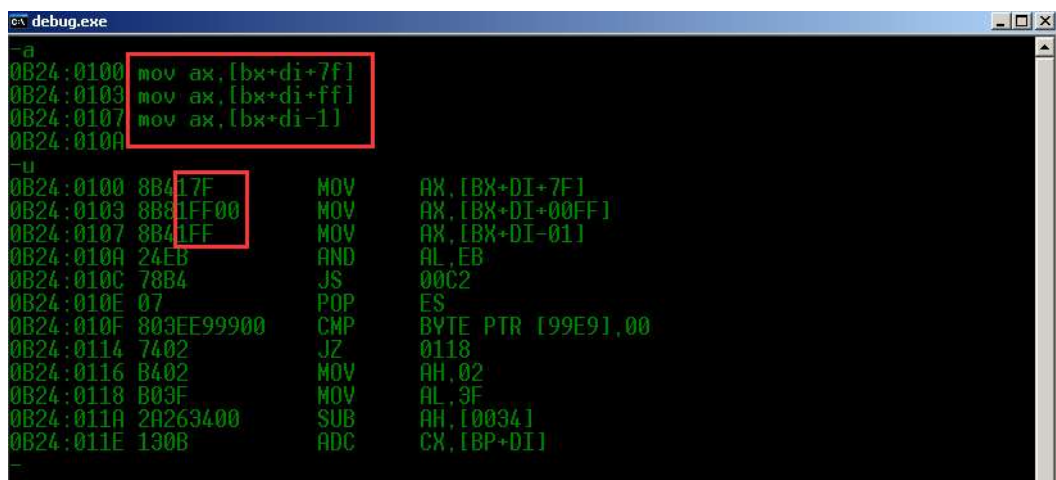
有效地址是基址寄存器 (BX/BP)、变址寄存器 (SI/DI) 与一个 8 位 或 16 位位移量之和:

有效地址 = BX/BP + SI/DI + 8/16 位位移量 (位移量可用符号表示)

段地址对应 BX 基址寄存器默认是 DS, 对应 BP 基址寄存器默认是 SS。

汇编指令示例: `mov ax,1000[bx][si]` 等价 **`mov ax,[bx+si+1000]`**

对应的反汇编为: `8B800010 MOV AX,[BX+SI+1000]`



```
debug.exe
-a
0B24:0100 mov ax,[bx+di+7f]
0B24:0103 mov ax,[bx+di+ff]
0B24:0107 mov ax,[bx+di-1]
0B24:010A
-u
0B24:0100 8B417F MOV AX,[BX+DI+7F]
0B24:0103 8B81FF00 MOV AX,[BX+DI+00FF]
0B24:0107 8B41FF MOV AX,[BX+DI-01]
0B24:010A 24EB AND AL,EB
0B24:010C 78B4 JS 00C2
0B24:010E 07 POP ES
0B24:010F 809EE99900 CMP BYTE PTR [99E9],00
0B24:0114 7402 JZ 0118
0B24:0116 B402 MOV AH,02
0B24:0118 B09F MOV AL,9F
0B24:011A 2A263400 SUB AH,[0034]
0B24:011E 130B ADC CX,[BP+DI]
```

对所有寻址方式的性能做一个排序

由快到慢:

1. 寄存器寻址方式
2. 立即数寻址方式
3. 寄存器间接寻址方式
4. 直接寻址方式 (理论上比基址变址寻址方式快一点, 有缓存的情况下, `mov` 指令执行完后, 偏移值就已经到达缓存中了)
5. 基址变址寻址方式
6. 寄存器相对寻址方式
7. 相对基址变址寻址方式

测试寄存器间接寻址方式使用的默认段

汇编代码：

```
-a
0B24:0100 mov ax,100
0B24:0103 mov ds,ax
0B24:0105 mov ax,200
0B24:0108 mov es,ax
0B24:010A mov ax,[si]
0B24:010C mov ax,[di]
0B24:010E
```

通过测试 寄存器间接寻址方式 使用的默认段为 ds。

跨段拷贝数据

ds、es 同时使用的场景有拷贝（模式使用es）两个段中的数据时，可以将ds、es分别指向两个段首地址（不跨段，两个寄存器的数值设置成一样即可）。

di 可以指向 ds，si 可以指向 es。

.com 文件的缺点

".com" 文件不分段的情况下，文件大小不能超过64kb。程序运行的时候，操作系统会将文件中的代码加载到内存中，超过 64kb 的 ".com" 文件代码不会加载到内存。操作系统遇到一个 ".com" 程序也不会进行内存分段。

文件格式

通过文件格式可以方便的进行内存分段，在文件中记录代码段、数据段的大小，并告诉操作系统将代码、数据加载到哪些位置上。操作系统负责去申请对应的内存空间。寄存器cs 的值供操作系统自动进行修改。

这个文件格式就叫做 ".exe" 文件格式。".exe" 文件中除了编写的代码外还内置了分段的信息。".com" 文件中没有描述分段信息，所以其不能进行分段（操作系统也就不知道代码、数据在何处）。

".exe" 程序内置了文件分段的信息。

微软官方提供了一个编译器："masm"，其汇编编译器为 "ml.exe"，链接器为："link.exe"。文件后缀名为：".asm"。

Masm615 --> 最后一个单独发布版本，高版本的系统需要下载 VS 后使用。在高版本系统中，编译C、C++程序，需要使用 "cl.exe" 将高级语言编译成汇编语言，接着使用 "ml.exe" 编译成 ".obj" 文件，最后使用 "link.exe" 将 ".obj" 文件链接成最终的可执行程序。

编写 ".asm" 文件时，需要提前进行分段，告诉操作系统如何分段。

db: 定义一个字节
dw: 定义两个字节
segment: 段
dup: 大小
end: 代码结束符
;; 注释
ret: 从当前栈地址中拿一个地址并返回

在 ".asm" 文件中, 段的先后顺序影响的是生成对应的二进制文件中段的存放位置 (先后)。推荐先定义数据段在定义代码段。

在代码段中给一个 "START:" 和 "end START" 可以告诉 CPU 哪里是代码段的开始和结束。

数据段的定义

一个数据段的容量最大为64KB

可以字符串的方式, 一个字节一个字节的方式, ASCII方式。

```
MyDatas segment
    db 256 dup(11h)
    ;db 03, 04, 05, 06
    MSG1 db "Hello World!$"
    org 512 ;在 Hello World! 后面申请512字节, 默认初始化为0
MyDatas ends
```

MSG1 --> 标号, 供代码段使用, 在使用的时候就不需要给偏移 (需要添加 **offset** 关键字, 使用时不添加关键字, 操作系统会认为 "取内容" (mov dx, word ptr [MSG1], mov dx, 0))。

栈段的定义

定义栈段时, 内部应该定义栈的大小, 栈中初始化存放数据没有意义 (注意: 栈的地址是倒的)。定义栈段时, 在segment关键字加 stack, 这样操作系统在申请栈空间的时候就会倒着申请 (不加stack: 100~200, 加stack: 200~100)。

栈段的定义:
db 256 dup(?) --> 栈中有256个字节, 不进行初始化
db 256 --> 栈中有1个字节

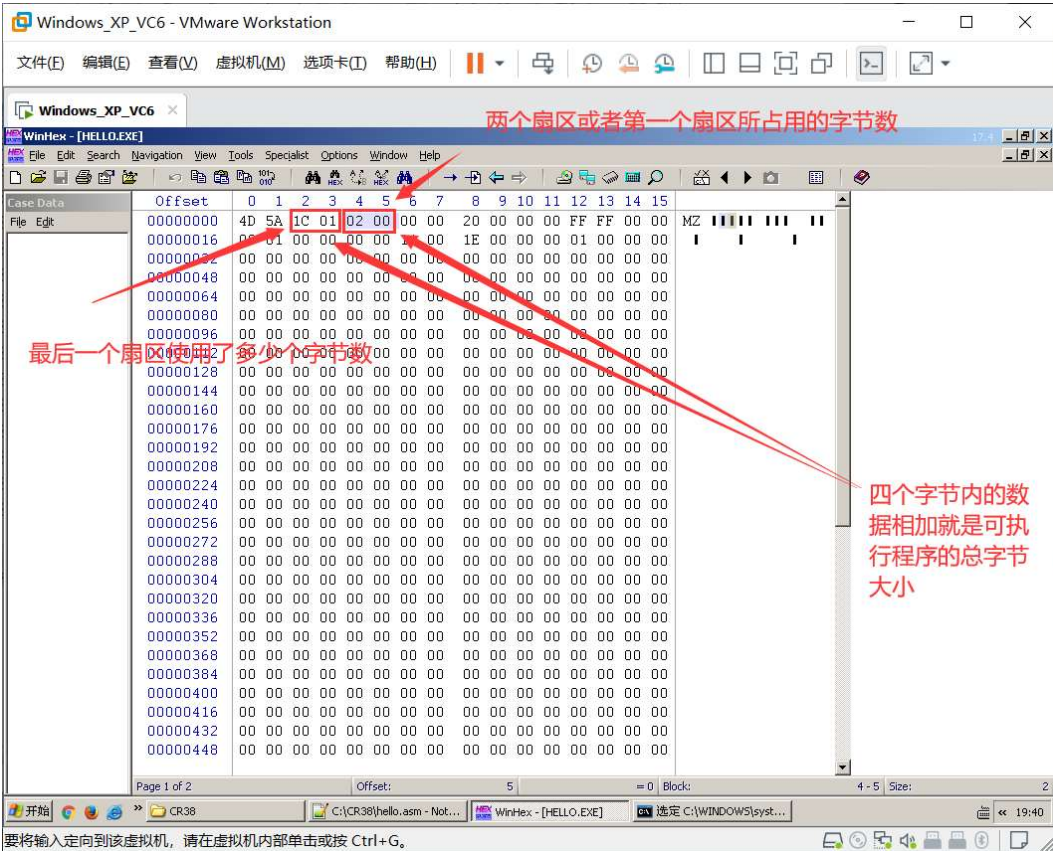
```
MyStack segment stack
    db 256 dup(?)
MyStack ends
```

内存地址是动态申请的，一个程序的代码执行地址并不确定。

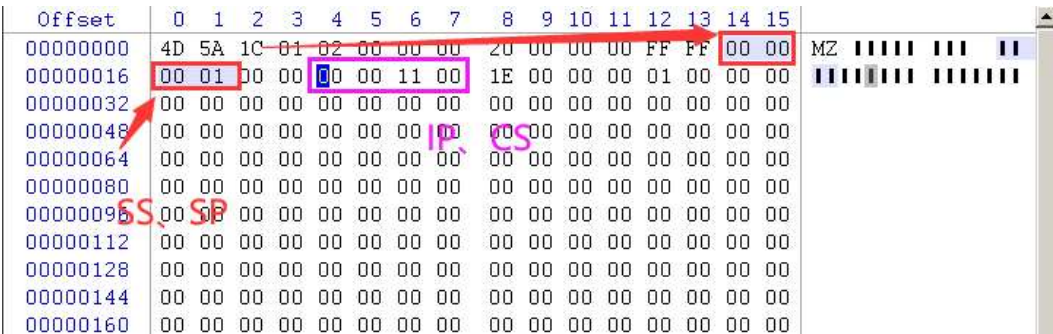
文件数据的对齐为：".exe" 文件的前 512 (200H) 个字节（没有数据的位置用 "0" 填充，对齐值和硬件相关），前两个字节为 "MZ"；操作系统读取数据的时候一个扇区一个扇区的进行读取（提供数据的读取效率）。

"dos" 文件格式也就是 ".exe" 文件格式同时统称为："MZ" 文件格式。

当可执行程序通过操作系统加载到内存的时候，操作系统需要为其申请内存空间，操作系统需要知道可执行程序的字节大小，为了方便，每个可执行程序中保存了 "自身" 的文件大小。



操作系统读取到可执行文件的字节大小后，将可执行文件中的数据按照 "一个扇区" 为单位将数据移动到内存中。



CS、SS通过操作系统将可执行文件代码移动到内存后自动进行获取。在文件中没有定义 DS、ES 数值的位置。在使用数据段的时候，需要自己进行设置（操作系统会自动计算代码段在内存中的偏移）。

汇编代码示例：


```

MyStack segment stack
    db 256 dup(?)
MyStack ends

MyDatas segment
    ;db 256 dup(11h)
    ;db 03,04,05,06
    MSG1 db "Hello World!$"
    ;org 512 ;在 Hello World! 后面申请512字节，默认初始化为0
MyDatas ends

MyCodes segment
MAIN:
    mov ax, MyDatas ;获取数据段应该获取地址，不应该获取偏移
    mov ds, ax
    mov es, ax
    mov dx, offset MSG1 ; ==> mov dx, word ptr [0] ==> mov d

    mov ah, 09h ;系统调用 类似调用printf
    int 21h

    mov ax, 4c00h ;强制退出进程，不需要在使用ret
    int 21h
MyCodes ends
end MAIN

```

The screenshot shows a Windows XP VM running a debugger (WinHex) on a Hello World program. The assembly source code is visible on the left, and the disassembled code is on the right. The program has terminated normally, displaying "Hello World!\$" in the output window.

Assembly language source file:

```

1
2
3 MySta
4 db
5 MySta
6
7
8 MyDat.
9 ;db
10 ;db
11 MSG
12 ;org
13 MyDat.
14
15 MyCod
16 MAIN:
17 m
18 m
19
20
21 m
22 m
23 i
24

```

Disassembled code:

```

00AC:0000 B8AC0B MOV AX,0BAC
00AC:0003 8ED8 MOV DS,AX
00AC:0005 8EC0 MOV ES,AX
00AC:0007 B80000 MOV DX,0000
00AC:000A B409 MOV AH,09
00AC:000C CD21 INT 21
00AC:000E B8004C MOV AX,4C00
00AC:0011 CD21 INT 21
00AC:0013 CC INT 3
00AC:0014 3464 XOR AL,64
00AC:0016 58 POP AX
00AC:0017 7504 JNZ 001D
00AC:0019 8A04 MOV AL,[SI]
00AC:001B 86C4 XCHG AL,AH
00AC:001D 80E438 AND AH,38
00AC:0000 48 65 6C 6C 6F 20 57 6F 72 6C 64 21 24 00 00 00 Hello World!$...
00AC:0010 58 AC 0B 8E 08 8E C0 BA 00 00 B4 09 CD 21 B8 00 .....!
00AC:0020 4C CD 21 CC 34 64 58 75 04 8A 04 86 C4 80 E4 38 L!.4dXu.....8
00AC:0030 3D FF 18 74 03 3D FF 28 1F 75 25 8B 76 1C 0F 00 =.t.=.(.u%.v...
00AC:0040 E6 75 1D 50 53 0F 02 C6 80 E4 9A 80 FC 9A 74 0E ..u.PS.....t.
00AC:0050 87 9A 8B C6 E8 F4 05 5B 58 07 1F 61 66 CB 5B 58 .....tX..af.tX
00AC:0060 07 1F 61 66 FF 2E 98 25 00 00 00 00 00 00 00 ..af...%.
00AC:0070 0E 1D 25 27 29 1F 97 1E 23 1D 36 1D A4 1D 30 1E ..%')...#.6...0.

```

Output window:

```

Program terminated normally
-q
C:\CR38>HELLO.EXE
Hello World!
C:\CR38>debug HELLO.EXE
-u
00AC:0000 B8AC0B MOV AX,0BAC
00AC:0003 8ED8 MOV DS,AX
00AC:0005 8EC0 MOV ES,AX
00AC:0007 B80000 MOV DX,0000
00AC:000A B409 MOV AH,09
00AC:000C CD21 INT 21
00AC:000E B8004C MOV AX,4C00
00AC:0011 CD21 INT 21
00AC:0013 CC INT 3
00AC:0014 3464 XOR AL,64
00AC:0016 58 POP AX
00AC:0017 7504 JNZ 001D
00AC:0019 8A04 MOV AL,[SI]
00AC:001B 86C4 XCHG AL,AH
00AC:001D 80E438 AND AH,38
-d bac:0
00AC:0000 48 65 6C 6C 6F 20 57 6F 72 6C 64 21 24 00 00 00 Hello World!$...
00AC:0010 58 AC 0B 8E 08 8E C0 BA 00 00 B4 09 CD 21 B8 00 .....!
00AC:0020 4C CD 21 CC 34 64 58 75 04 8A 04 86 C4 80 E4 38 L!.4dXu.....8
00AC:0030 3D FF 18 74 03 3D FF 28 1F 75 25 8B 76 1C 0F 00 =.t.=.(.u%.v...
00AC:0040 E6 75 1D 50 53 0F 02 C6 80 E4 9A 80 FC 9A 74 0E ..u.PS.....t.
00AC:0050 87 9A 8B C6 E8 F4 05 5B 58 07 1F 61 66 CB 5B 58 .....tX..af.tX
00AC:0060 07 1F 61 66 FF 2E 98 25 00 00 00 00 00 00 00 ..af...%.
00AC:0070 0E 1D 25 27 29 1F 97 1E 23 1D 36 1D A4 1D 30 1E ..%')...#.6...0.

```