

第三章

2018302070001沈思源

3.13

3.13 若执行“MOV CX,0F081H”后,再执行“MOVZX EAX,CX”和“MOVSX EBX,CX”,问 EAX = ? EBX = ?

解

首先 MOV,MOVSX,MOVZX 都是数据传送类指令,格式均为:

OP OPD, OPS ;OPD表示目的操作数, OPS表示源操作数

执行 MOV CX 0F081H, 注意到后缀是H, 表示源操作数数字是16进制, 16进制的数如果开头是A-F, 那么开始数字前加上0, 所以16位通用寄存器CX将数据 F081H 送到CX寄存器地址中, (CX)=F081H.

接着执行

MOVZX EAX,CX MOVZX 指令表示的是将OPS内容-->OPD,对于OPD左边高位空缺的位采用**Zero填充**

MOVSX EBX,CX 指令与之类似, 但是对于OPD高位采用OPS的**符号填充**

因此, 32位寄存器EAX,EBX分别对应如下:

```
1 MOV CX 0F081H
2 -> (CX)= 1111 0000 1000 0001
3 MOVZX EAX CX ;Zero
4 -> (EAX)= 0000 0000 0000 0000 1111 0000 1000 0001 = 0000F081H
5 MOVSX EBX CX ;Symbol
6 -> (EBX)= 1111 1111 1111 1111 1111 0000 1000 0001 = FFFFFFF081H
```

3.14

3.14 设 SP/ESP = 180H, AX = 198H, 其余通用寄存器的值均为 0, 问执行语句序列

```
PUSHA
PUSHAD
:
POPA
```

后, SP/ESP = ? AX = ?

解

首先明确指令对 PUSHA/POPA 和 PUSHAD/POPAD

PUSHA 指令表示压入8个字类型通用寄存器, 依次顺序是:

AX,CX,DX,BX,TEMP(指令执行前的SP),BP,SI,DI

与之对应的 POPA 指令的出栈顺序就是:

DI,SI,BP,TEMP,BX,DX,CX,AX

注意, TEMP称作临时单元; PUSHAD指令执行时发生以下操作:

TEMP=SP/ESP, SP/ESP-16 -> SP/ESP

之所以栈指针寄存器-16是因为目前的计算机体系中栈向下增长(堆相反),而本处理机体系1字2字节,一共压入8字16字节,因此下移16字节地址

PUSHAD 指令表示的是压入8个双字通用寄存器(32bit),因此块的大小是 $2 \times 2 \times 8 = 32\text{bytes}$

TEMP=SP/ESP, SP/ESP-32->SP/ESP

依次将EAX,ECX,EDX,EBX,TEMP(指令执行前的ESP),EBP,ESI,EDI压入[SP]/[ESP]

因此,解答如下

```
1  PUSHA
2  -> SP/ESP = SP/ESP - 16  STACK:DI SI BP...AX
3  PUSHAD
4  -> SP/ESP = SP/ESP - 32  STACK:EDI ESI...EAX DI SI...AX
5  ...
6  POPA
7  -> SP/ESP = SP/ESP + 16    STACK:弹出16字节数据,即弹出EDI,ESI,EBP,TEMP给
   DI,SI,BP...AX
8
9  SP/ESP=180H-32=160H
10 而TEMP是指令执行前的ESP,TEMP的高16位给了CX,低16位(也就是SP)给了AX,指令执行前的
   SP/ESP有
11  SP/ESP=180H-16D=170H
12  = 0000 0000 0000 0000 0000 0001 0111 0000
13                                     -----
14                                     SP
15 因此AX=170H
16
```

3.15

3.15 若 $EDX = 1034H$, $ESI = 15H$, $BUF = 289H$,问执行指令

LEA EBX,BUF[EDX][4*ESI]

后,EBX=? ESI=?

解

先明确 **LEA** 指令: **LEA OPD,OPS**,指令将OPS的有效地址/偏移地址EA传送给OPD

首先看题干指令OPD部分, **BUF[EDX][4*ESI]**,这是**相对基址变址寻址**寻址类型,因此:

```
1  OPD : EA=BUF+EDX+4*ESI=289H+1034H+4*15H
2          =289H+1034H+60H
3          =131DH
```

3.16

3.16 若 $CX = 6$, $EDX = 0$, $EAX = 0F1501980H$, $CF = 1$,问执行指令

SHLD EDX,EAX,CL

后,EDX=? EAX=? CX=? CF=?

解

先明确 **SHLD** 指令.

S表示这是逻辑和移位指令中的一般移位指令，**H**表示逻辑移位，**R**表示右移，逻辑左移和算术左移(SAL,SHL)在物理上一样，但是右移不同。

逻辑右移： 将各位向右移动，而当前的最低有效位移到CF，将0补到最高有效位MSB上

算术右移： 将各位向右移动，而当前的最低有效位移到CF，但是MSB位保持不变

SHLD 指令是**双精度移位指令**,L表示这是双精度左移，格式如下

SHLD OPD,OPS1,OPS2

OPS1(寄存器)存放待移入的OPD值，移位时，OPD根据OPS2的值进行移位，移出的位-->CF(16位标志寄存器EFLAGS的进位标志CF),空出位用OPS1的高位(SHLD向左移时)或者低位(SHRD右移时)来填补。

移位后，OPS1的值保持不变，CF等于最后一次移出的那个位值。

注意，移位数指的是二进制数的移位数，比如如果是4，那么对应的16进制数就是移位1

```
1  SHLD EDX,EAX,CL ;EDX=0,EAX=0F1501980H
2  ->注意到，CX=6=0000 0000 0000 0110
3
4          -----
4          CL
5
5  因此CL=6.
6  EAX由于16进制MSB位为字母表示，有前缀0，实际值为F150 1980
7  EAX = 1111 0001 0101 0000 0001 1001 1000 0000
8  EDX = 0
9  EDX左移6位，低位由EAX高位进行填补，因此移位后的EDX有
10 EDX= 0000 0000 0000 0000 0000 0000 0011 1100
11 CF则对应EDX最后一次移出的那个位值，显然是0
12 EAX作为OPS1，保持不变.
13 CX在ECX中，未受影响，保持不变
14
15 综上，移位后
16 EDX=3CH,EAX=0 F1501980H,CX=6,CF=0;
```

3.17

```
1  .586
2  STACKS  SEGMENT USE16
3          DB  256 DUP(0)
4  STACKS  ENDS
5
6  DATAS   SEGMENT USE16
7      N   EQU 26
8      BUF DB  N+1
9      COUNT DB  0
10     CHAR  DB  N+1 DUP(0)
11           DB  10
12           DB  13
13           DB  '$'
14     PROMPT DB  'PLEASE INPUT YOUR STRING:',13,10,'$'
15     ;NEXT  DB  13,10,'$'
16     RESTIP DB  'THE STRING YOU INPUT IS:',13,10,'$'
17  DATAS   ENDS
18
19  CODES   SEGMENT USE16
```

```
20      ASSUME  CS:CODES,DS:DATAS
21      START:
22      MOV  AX,DATAS ;段地址->AX
23      MOV  DS,AX    ;段地址->DS
24      LEA  DX,PROMPT ;取PROMPT的偏移地址->DX
25
26      MOV  AH,9      ;显示输入提示字符串
27      INT  21H
28
29      LEA  DX,BUF    ;缓冲区输入
30      MOV  AH,0AH
31      INT  21H
32
33      MOV  DL,10
34      MOV  AH,2
35      INT  21H
36
37      LEA  DX,restip
38      MOV  AH,9
39      INT  21H
40
41      LEA  DX,CHAR
42      MOV  SI,DX;SI存入输入字符串起始地址
43      MOV  BH,0;BX高8位清零
44      MOV  BL,[SI-1];低八位置为字符串长度数据存储的地址
45      MOV  BYTE PTR [SI+BX], '$' ;在字符串末尾处加上结束符号
46
47      LEA  DX,CHAR
48      MOV  AH,9
49      INT  21H
50
51
52      MOV  AH,4CH
53      INT  21H
54      CODES ENDS
55      END  START
```

BUF	N+1
COUNT	输入字符串的长度
CHAR	输入的字符串' *'
	' '
	' '
	' '
	' '
	' '