#### 王爽汇编第七章,更灵活的定位内存地址的方法

- 1. and 和 or指令
- 2. 关于ASCII码
- 3. 显示hello,world!
- 4. 大小写转换
- 5. [bx+idata]寻址
- 6. si和di
- 7. 不同的寻址方式的灵活应用

### 1. and 和 or指令

首先,介绍两条指令 and 和 or,因为我们下面的例程中要用到它们。

(1) and 指令:逻辑与指令,按位进行与运算。

例如指令:

 $01100011 \\ 00111011$ 

mov al,01100011B and al,00111011B

00100011

执行后: al=00100011B

and: 全是1才为1

通过该指令可将操作对象的相应位设为0,其他位不变。

例如:

将 al 的第 6 位设为 0 的指令是: and al,10111111B 将 al 的第 7 位设为 0 的指令是: and al,01111111B 将 al 的第 0 位设为 0 的指令是: and al,11111110B

(2) or 指令:逻辑或指令,按位进行或运算。

例如指令:

 $01100011 \\ 00111011$ 

mov al,01100011B or al,00111011B

<sup>执行后: al=01111011B</sub> **0111111**</sup>

or: 有1说1

通过该指令可将操作对象的相应位设为1,其他位不变。

例如:

将 al 的第 6 位设为 1 的指令是: or al,01000000B 将 al 的第 7 位设为 1 的指令是: or al,10000000B 将 al 的第 0 位设为 1 的指令是: or al,00000001B

#### 2. 关于ASCII码

ASCII(发音: /ˈæski/ ASS-kee [1], American Standard Code for Information Interchange, 美国信息交换标准代码)是基于 拉丁字母 的一套 电脑 编码 系统。它主要用于显示 现代英语,而其扩展版本 延伸美国标准信息交换码 则可16以部分支持其他 西欧 语言,并等同于国际标准 ISO/IEC 646。

ASCII 由电报码发展而来。第一版标准发布于1963年[3][4],1967年经历了一次主要修订[5][6],最后一次更新则是在1986年,至今为止共定义了128个字符;其中33个字符无法显示(一些终端提供了扩展,使得这些字符可显示为诸如笑脸、扑克牌花式等8-bit符号),且这33个字符多数都已是陈废的控制字符。控制字符的用途主要是用来操控已经处理过的文字。在33个字符之外的是95个可显示的字符。用键盘敲下空白键所产生的空白字符也算1个可显示字符(显示为空白)。

高四位		ASCII非打印控制字符									ASCII 打印字符													
		0000				0001					0010 0011		0100		0101		0110		0111					
低四位	7		字符	O 符 ctrl 代		码 字符解释	十进制	字符	1	1 代码	3 字符解释	2	2	+進制 字符		4 + 讲制 字符		+ 進制 字符		+進制字符		+進制字符		
0000	- \ 0	十進制	F1T BLANK	^@	NUL.	子付册样	16	±10	ctrl ^P	25451	子付併存 数据链路转意	32	子19	48	7-10	+亚和 64	11117	+理刷	P	+理制 96	<del>1</del> 10	+理制	2201	ctri
0000	1		MATT	^ A	SOH	工 头标开始	17		^0	DC1	设备控制 1	33		49	1	65	@ A	81	Q	97		113	p	
		1	44.254	^ B		正文开始			^R		设备控制 2				Cities C	1077114	В			122310	a		q	
0010	2	2	•	^C	STX	正文开始	18	11	^ S	DC2	设备控制 3	34	#	50	2	66	С	82	R	98	b	114		
0011	3	3	٧		ETX		19	••	in the	DC3		35	Biblio	51	Hilliam	67	350	83	Hilliam	99	C	115	S	
0100	4	4	•	^D	EOT	传输结束	20	1	^ T	DC4	设备控制 4	36	\$	52	4	68	D	84	I	100	d	116	I	
0101	5	5	*	^ E	ENQ	查询	21	∮	^ U	NAK	反确认	37	%	53	5	69	E	85	U	101	е	117	u	
0110	6	6	A	^F	ACK	确认	22		^ V	SYN	同步空闲	38	&	54	6	70	F	86	V	102	f	118	٧	
0111	7	7	•	^G	BEL	震铃	23	I	^ W	ETB	传输块结束	39	•	55	7	71	G	87	W	103	g	119	W	
1000	8	8		^н	BS	退格	24	1	^ X	CAN	取消	40	(	56	8	72	Н	88	Х	104	h	120	Х	
1001	9	9	0	^I	TAB	水平制表符	25	Į.	^ Y	EM	媒体结束	41	)	57	9	73		89	Υ	105	Í	121	У	
1010	A	10	0	^J	LF	换行/新行	26	$\rightarrow$	^ Z	SUB	替换	42	*	58		74	J	90	Z	106	j	122	z	
1011	В	11	Ō	^ K	VT	竖直制表符	27	<b>←</b>	^ [	ESC	转意	43	+	59		75	Κ	91	ſ	107	k	123	{	
1100	С	12	Q	^L	FF	换页/新页	28		^\	FS	文件分隔符	44	,	60	<	76	L	92		108	1	124		
1101	D	13	J	^ M	CR	回车	29	<b>(-)</b>	^]	GS	组分隔符	45		61		77	М	93		109	m	125	}	
1110	Е	14	.1	^ N	SO	移出	30	A	^6	RS	记录分隔符	46		62	>	78	Ν	94	^	110	n	126	~	
1111		15	n	^0	SI	移入	31	v	^_	US	单元分隔符	47	1	63	2	79	0	95		111	0	127	Δ	^Back

### 3. 显示hello,world!

注意这里我调用了 int 21h , 这是一个DOS的中断, 其中ah=09是他的功能号用来打印字符串。

关于DOS中断的内容会在后续章节详细介绍。

## 4. 大小写转换

小写字母的ASCII码值比大写字母的ASCII码值大20H(32)

大写字母ASCII码的第5位为0,小写字母的第5位为1(其他一致)

大写	十六进制	二进制	小写	十六进制	二进制	
A	41	01000001	a	61	01100001	
В	42	01000010	b	62	01100010	
С .	43	01 <mark>0</mark> 00011	C	63	01100011	
D	44	01000100	d	64	01100100	
E	45	01000101	е	65	01100101	
F	46	01000110	f	66	01100110	

```
26 mov ds:[bx],al
27 inc bx
28 loop s
29
30 ;输出字符串
31 mov dx,0; ds:[0]
32 mov ah,09h
33 int 21h
34
35 ;retun 2 dos
36 mov ax,4c00h
37 int 21h
38 code ends
39 end main
```

## 5. [bx+idata]寻址

在前面,我们用[bx]的方式来指明一个内存单元,还可以用一种更为灵活的方式来指明内存单元: [bx+idata]表示一个内存单元,他的偏移地址为(bx)+idata [bx中的数值加上数字]

例如:

mov ax, [bx+200]

也可以写成如下的格式:(注意这里指的是masm所支持的伪指令,具体其他编译器看情况而定.)

```
1 mov ax,[bx+200]
2 mov ax,[200+bx]
3 mov ax,[bx].200
4 mov ax,200[bx]
```

[bx+idata]处理数组例子:

```
code segment
main:
     mov bx,0
    s:mov al,0[bx]
      mov 0[bx],al
     or al,00100000b;转小写
     mov 3[bx],al
      inc bx
     ;输出字符串
     mov dx,0
     mov ah,09h
     mov ax,4c00h
```

#### 翻译成C语言

```
#include <stdio.h>
int main()

{
    char a[]="doS";
    char b[]="ASM";

int i=0;

do
    {
        a[i] = a[i] & 0xDF;
        b[i] = b[i] | 0x20;
        i++;

}while(i<3);

printf("%s%s\n",a,b);
return 0;
}</pre>
```

## 6. si和di

这两个寄存器其实之前介绍寄存器篇章的时候,已经介绍过了,si和di都是偏移寄存器并且注意在x86架构下,他们是字符串相关的寄存器。

```
(1) mov bx,0
mov ax,[bx]

(2) mov si,0
mov ax,[si]

(3) mov di,0
mov ax,[di]

下面的 3 组指令也实现了相同的功能。

(1) mov bx,0
mov ax,[bx+123]

(2) mov si,0
mov ax,[si+123]

(3) mov di,0
mov ax,[di+123]
```

# 7. 不同的寻址方式的灵活应用

如果我们比较一下前面用到的集中定位内存地址的方法(可以称为寻址方式),就可以发现:

- [idata]用一个常量来表示地址,可用于直接定位一个内存单元;
- [bx]用一个变量来表示内地地址,可用于间接定位一个内存单元(比如循环);

- [bx+idata]用一个变量和常量表示地址,可在一个起始地址的基础上用变量间接定位一个内存单元(比如多个数组);
- [bx+si]用两个变量表示地址;
- [bx+si+idata]用连个变量和一个常量表示地址。

可以看到,从[idata]一直到[bx+si+idata],我们可以用更加灵活的方式来定位一个内存单元的地址。 <mark>这使得我们可以从更结构化的角度</mark>来看待所要处理的数据。

#### 参考文献:

https://blog.csdn.net/trochiluses/article/details/20008631 int 21—dos系统调用 表 https://blog.csdn.net/qq\_39654127/article/details/88698911 王爽汇编笔记

《王爽汇编第4版》 第7章