阿里云实时计算训练营重磅升级 结营抢天猫精灵 & 独家实物好礼 点我免费报名

云水

MyGitee - https://gitee.com/lsgx/

MyGithub - https://github.com/lsgxeva/

汇编指令速查

https://www.cnblogs.com/findumars/p/3498714.html

GAS中每个操作都是有一个字符的后缀，表明操作数的大小。

C声明	GAS后缀	大小(字节)
char	b	1
short	w	2
(unsigned) int / long / char*	l	4
float	s	4
double	l	8
long double	t	10/12

注意：GAL使用后缀“l”同时表示4字节整数和8字节双精度浮点数，这不会产生歧义因为浮点数使用的是完全不同的指令和寄存器。

操作数格式：

格式	操作数值	名称	样例（GAS = C语言）
$Imm	Imm	立即数寻址	$1 = 1
Ea	R[Ea]	寄存器寻址	%eax = eax
Imm	M[Imm]	绝对寻址	0x104 = *0x104
（Ea）	M[R[Ea]]	间接寻址	（%eax）= *eax

公告

昵称： lsgxeva
园龄： 5年4个月
粉丝： 183
关注： 1
+加关注

< 2021年3月 >

日	一	二	三	四	五	六
28	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	1	2	3
4	5	6	7	8	9	10

搜索

常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签

随笔分类

a10(4)
AI(1)
algorithm(1)
Android(39)
ans(55)
bitcoin(4)
c++(104)
c++11(38)
ccna(35)
cmake(5)
cnns(18)
cocos(1)
CSharp(29)
DataStructure(6)
DesignPattern(39)
更多

随笔档案

Imm(Ea)	M[Imm+R[Ea]]	(基址+偏移量)寻址	4(%eax) = *(4+eax)
（Ea,Eb）	M[R[Ea]+R[Eb]]	变址	(%eax,%ebx) = *(eax+ebx)
Imm （Ea,Eb）	M[Imm+R[Ea]+R[Eb]]	寻址	9(%eax,%ebx)= *(9+eax+ebx)
(,Ea,s)	M[R[Ea]*s]	伸缩化变址寻址	(,%eax,4)= *(eax*4)
Imm(,Ea,s)	M[Imm+R[Ea]*s]	伸缩化变址寻址	0xfc(,%eax,4)= *(0xfc+eax*4)
(Ea,Eb,s)	M(R[Ea]+R[Eb]*s)	伸缩化变址寻址	(%eax,%ebx,4) = *(eax+ebx*4)
Imm(Ea,Eb,s)	M(Imm+R[Ea]+R[Eb]*s)	伸缩化变址寻址	8(%eax,%ebx,4) = *(8+eax+ebx*4)

注：M[xx]表示在存储器中xx地址的值，R[xx]表示寄存器xx的值，这种表示方法将寄存器、内存都看出一个大数组的形式。

数据传送指令：

指令	效果	描述
movl S,D	D <-- S	传双字
movw S,D	D <-- S	传字
movb S,D	D <-- S	传字节
movsbl S,D	D <-- 符号扩展S	符号位填充(字节->双字)
movzbl S,D	D <-- 零扩展S	零填充(字节->双字)
pushl S	R[%esp] <-- R[%esp] – 4; M[R[%esp]] <-- S	压栈
popl D	D <-- M[R[%esp]]; R[%esp] <-- R[%esp] + 4;	出栈

注：均假设栈往低地址扩展。

算数和逻辑操作地址：

指令	效果	描述
leal S,D	D = &S	movl地版，S地址入D，D仅能是寄存器

最新评论

阅读排行榜

评论排行榜

推荐排行榜

incl D	D++	加1
decl D	D--	减1
negl D	D = -D	取负
notl D	D = ~D	取反
addl S,D	D = D + S	加
subl S,D	D = D – S	减
imull S,D	D = D*S	乘
xorl S,D	D = D ^ S	异或
orl S,D	D = D | S	或
andl S,D	D = D & S	与
sall k,D	D = D << k	左移
shll k,D	D = D << k	左移(同sall)
sarl k,D	D = D >> k	算数右移
shrl k,D	D = D >> k	逻辑右移

特殊算术操作：

指令	效果	描述
imull S	R[%edx]:R[%eax] = S * R[%eax]	无符号64位乘
mull S	R[%edx]:R[%eax] = S * R[%eax]	有符号64位乘
cltd S	R[%edx]:R[%eax] = 符号位扩展R[%eax]	转换为4字节
idivl S	R[%edx] = R[%edx]:R[%eax] % S; R[%eax] = R[%edx]:R[%eax] / S;	有符号除法，保存余数和商
divl S	R[%edx] = R[%edx]:R[%eax] % S; R[%eax] = R[%edx]:R[%eax] / S;	无符号除法，保存余数和商

注：64位数通常存储为，高32位放在edx，低32位放在eax。

条件码：

条件码寄存器描述了最近的算数或逻辑操作的属性。

CF：进位标志，最高位产生了进位，可用于检查无符号数溢出。

OF：溢出标志，二进制补码溢出——正溢出或负溢出。

ZF：零标志，结果为0。

SF：符号标志，操作结果为负。

比较指令：

指令	基于	描述
cmpb S2,S1	S1 – S2	比较字节，差关系
testb S2,S1	S1 & S2	测试字节，与关系
cmpw S2,S1	S1 – S2	比较字，差关系
testw S2,S1	S1 & S2	测试字，与关系
cmpl S2,S1	S1 – S2	比较双字，差关系
testl S2,S1	S1 & S2	测试双字，与关系

访问条件码指令：

指令	同义名	效果	设置条件
sete D	setz	D = ZF	相等/零
setne D	setnz	D = ~ZF	不等/非零
sets D		D = SF	负数
setns D		D = ~SF	非负数
setg D	setnle	D = ~(SF ^OF) & ZF	大于（有符号>）
setge D	setnl	D = ~(SF ^OF)	小于等于(有符号>=)
setl D	setnge	D = SF ^ OF	小于(有符号<)
setle D	setng	D = (SF ^ OF) | ZF	小于等于(有符号<=)
seta D	setnbe	D = ~CF & ~ZF	超过(无符号>)
setae D	setnb	D = ~CF	超过或等于(无符号>=)

setb D	setnae	D = CF	低于(无符号<)
setbe D	setna	D = CF | ZF	低于或等于(无符号<=)

跳转指令：

指令	同义名	跳转条件	描述
jmp Label		1	直接跳转
jmp *Operand		1	间接跳转
je Label	jz	ZF	等于/零
jne Label	jnz	~ZF	不等/非零
js Label		SF	负数
jnz Label		~SF	非负数
jg Label	jnle	~(SF^OF) & ~ZF	大于(有符号>)
jge Label	jnl	~(SF ^ OF)	大于等于(有符号>=)
jl Label	jnge	SF ^ OF	小于（有符号<）
jle Label	jng	(SF ^ OF) | ZF	小于等于(有符号<=)
ja Label	jnbe	~CF & ~ZF	超过(无符号>)
jae Label	jnb	~CF	超过或等于(无符号>=)
jb Label	jnae	CF	低于(无符号<)
jbe Label	jna	CF | ZF	低于或等于(无符号<=)

转移控制指令：（函数调用）：

指令	描述
call Label	过程调用，返回地址入栈，跳转到调用过程起始处，返回地址是call后面那条指令的地址
call *Operand	

leave	为返回准备好栈，为ret准备好栈，主要是弹出函数内的栈使用及%ebp

用GCC在C中潜入汇编代码：

asm(code-string [:output-list [: input-list [:overwrite-list]]]);

注意，后面的参数（如overwrite-list）如果为空则不要相应的“：”，而如果前面参数(如output-list）为空则需要用“：”占位。

如：

```
asm ("..."
    :                   //output需要占位
    : "r" (src)         //后面的Overwrites不能写，我测试的结果是写了编译不过
};
```

如：

```
Int ok_umul(unsigned x,unsigned y,unsigned *dest)
{
  int result;
asm(“movl %2 , %%eax; mull %3; movl %%eax,%0;\
        setae %dl; movzbl %%dl,%1”
        : “=r” (*dest) ， “=r” (result)         //output
        : “r” (x) ， “r” (y)                    //inputs
        : “%ebx” , “%edx”                       //Overwrites
);

return result;
}
```

我们用%0--%n表示输入的参数，”r”表示整数寄存器，”=”表示对其进行了赋值。%eax要写成%%eax，这是c语言字符串的规则，别忘了code-string就是一个c语言的字符串。

汇编指令速查

指令	功能
AAA	调整加
AAD	调整除
AAM	调整乘
AAS	调整减
ADC	进位加
ADD	加
AND	与
ARPL	调整优先级

BOUND	检查数组
BSF	位右扫描
BSR	位左扫描
BSWAP	交换字节
BT	位测试
BTC	位测试求反
BTR	位测试清零
BTS	位测试置一
CALL	过程调用
CBW	转换字节
CDQ	转换双字
CLC	进位清零
CLD	方向清零
CLI	中断清零
CLTS	任务清除
CMC	进位求反
CMOVA	高于传送
CMOVB	低于传送
CMOVE	相等传送
CMOVG	大于传送
CMOVL	小于传送
CMOVNA	不高于传送
CMOVNB	不低于传送
CMOVNE	不等传送
CMOVNG	不大于传送
CMOVNL	不小于传送
CMOVNO	不溢出传送
CMOVNP	非奇偶传送
CMOVNS	非负传送
CMOVO	溢出传送
CMOVP	奇偶传送
CMOVS	负号传送
CMP	比较
CMPSB	比较字节串
CMPSD	比较双字串
CMPSW	比较字串
CMPXCHG	比较交换

CMPXCHG486	比较交换486
CMPXCHG8B	比较交换8字节
CPUID	CPU标识
CWD	转换字
CWDE	扩展字
DAA	调整加十
DAS	调整减十
DEC	减一
DIV	除
ENTER	建立堆栈帧
HLT	停
IDIV	符号整除
IMUL	符号乘法
IN	端口输入
INC	加一
INSB	端口输入字节串
INSD	端口输入双字串
INSW	端口输入字串
JA	**高于跳转**
JB	**低于跳转**
JBE	**不高于跳转**
JCXZ	**计数一六零跳转**
JE	**相等跳转**
JECXZ	**计数三二零跳转**
JG	**大于跳转**
JL	**小于跳转**
JMP	**跳转**
JMPE	**跳转扩展**
JNB	**不低于跳转**
JNE	**不等跳转**
JNG	**不大于跳转**
JNL	**不小于跳转**
JNO	**不溢出跳转**
JNP	**非奇偶跳转**
JNS	**非负跳转**
JO	**溢出跳转**
JP	**奇偶跳转**

JS	负号跳转
LAHF	加载标志低八
LAR	加载访问权限
LDS	加载数据段
LEA	加载有效地址
LEAVE	清除过程堆栈
LES	加载附加段
LFS	加载标志段
LGDT	加载全局描述符
LGS	加载全局段
LIDT	加载中断描述符
LMSW	加载状态字
LOADALL	加载所有
LOADALL286	加载所有286
LOCK	锁
LODSB	加载源变址字节串
LODSD	加载源变址双字串
LODSW	加载源变址字串
LOOP	计数循环
LOOPE	相等循环
LOOPNE	不等循环
LOOPNZ	非零循环
LOOPZ	为零循环
LSL	加载段界限
LSS	加载堆栈段
LTR	加载任务
MONITOR	监视
MOV	传送
MOVSB	传送字节串
MOVSD	传送双字串
MOVSW	传送字串
MOVSX	符号传送
MOVZX	零传送
MUL	乘
MWAIT	
NEG	求补

NOP	空
NOT	非
OR	或
OUT	端口输出
OUTSB	端口输出字节串
OUTSD	端口输出双字串
OUTSW	端口输出字串
POP	出栈
POPA	全部出栈
POPF	标志出栈
PUSH	压栈
PUSHA	全部压栈
PUSHF	标志压栈
RCL	进位循环左移
RCR	进位循环右移
RDMSR	读专用模式
RDPMC	读执行监视计数
RDSHR	
RDTSC	读时间戳计数
REP	重复
REPE	相等重复
REPNE	不等重复
RET	过程返回
RETF	远过程返回
RETN	近过程返回
ROL	循环左移
ROR	循环右移
RSM	恢复系统管理
SAHF	恢复标志低八
SAL	算术左移
SALC	
SAR	算术右移
SBB	借位减
SCASB	扫描字节串
SCASD	扫描双字串
SCASW	扫描字串
SETA	高于置位

SETB	低于置位
SETE	相等置位
SETG	大于置位
SETL	小于置位
SETNA	不高于置位
SETNB	不低于置位
SETNE	不等置位
SETNG	不大于置位
SETNL	不小于置位
SETNO	不溢出置位
SETNP	非奇偶置位
SETNS	非负置位
SETO	溢出置位
SETP	奇偶置位
SETS	负号置位
SGDT	保存全局描述符
SHL	逻辑左移
SHLD	双精度左移
SHR	逻辑右移
SHRD	双精度右移
SIDT	保存中断描述符
SLDT	保存局部描述符
SMI	
SMINT	
SMINTOLD	
SMSW	保存状态字
STC	进位设置
STD	方向设置
STI	中断设置
STOSB	保存字节串
STOSD	保存双字串
STOSW	保存字串
STR	保存任务
SUB	减
SYSCALL	系统调用
SYSENTER	系统进入
SYSEXIT	系统退出

SYSRET	系统返回
TEST	数测试
UD0	未定义指令0
UD1	未定义指令1
UD2	未定义指令2
UMOV	
VERW	校验写
WAIT	等
WBINVD	回写无效高速缓存
WRMSR	写专用模式
WRSHR	
XADD	交换加
XBTS	
XCHG	交换
XLAT	换码
XOR	异或
XSTORE	

http://files.cnblogs.com/findumars/ASM_Detail.pdf

指令	**功能**
EMMS	媒体空MMX状态
F2XM1	浮点栈顶绝对值
FADD	浮点加
FADDP	浮点加出栈
FBLD	浮点加载十数
FBSTP	浮点保存十数出栈
FCHS	浮点正负求反
FCLEX	浮点检查错误清除
FCMOVB	浮点低于传送
FCMOVBE	浮点不高于传送
FCMOVE	浮点相等传送
FCMOVNB	浮点不低于传送
FCMOVNBE	浮点高于传送
FCMOVNE	浮点不等传送
FCMOVNU	浮点有序传送
FCMOVU	浮点无序传送

FCOM	浮点比较
FCOMI	浮点比较加载标志
FCOMIP	浮点比较加载标志出栈
FCOMP	浮点比较出栈
FCOMPP	浮点比较出栈二
FCOS	浮点余弦
FDECSTP	浮点栈针减一
FDISI	浮点检查禁止中断
FDIV	浮点除
FDIVP	浮点除出栈
FDIVR	浮点反除
FDIVRP	浮点反除出栈
FENI	浮点检查禁止中断二
FFREE	浮点释放
FFREEP	浮点释放出栈
FIADD	浮点加整数
FICOM	浮点比较整数
FICOMP	浮点比较整数出栈
FIDIV	浮点除整数
FIDIVR	浮点反除
FILD	浮点加载整数
FIMUL	浮点乘整数
FINCSTP	浮点栈针加一
FINIT	浮点检查初始化
FIST	浮点保存整数
FISTP	浮点保存整数出栈
FISTTP	
FISUB	浮点减整数
FISUBR	浮点反减整数
FLD	浮点加载数
FLD1	浮点加载一
FLDCW	浮点加载控制器
FLDENV	浮点加载环境
FLDL2E	浮点加载L2E
FLDL2T	浮点加载L2T
FLDLG2	浮点加载LG2
FLDLN2	浮点加载LN2

FLDPI	浮点加载PI
FLDZ	浮点加载零
FMUL	浮点乘
FMULP	浮点乘出栈
FNCLEX	浮点不检查错误清除
FNDISI	浮点不检查禁止中断
FNENI	浮点不检查禁止中断二
FNINIT	浮点不检查初始化
FNOP	浮点空
FNSAVE	浮点不检查保存状态
FNSTCW	浮点不检查保存控制器
FNSTENV	浮点不检查保存环境
FNSTSW	浮点不检查保存状态器
FPATAN	浮点部分反正切
FPREM	浮点部分余数
FPREM1	浮点部分余数二
FPTAN	浮点部分正切
FRNDINT	浮点舍入求整
FRSTOR	浮点恢复状态
FSAVE	浮点检查保存状态
FSCALE	浮点比例运算
FSETPM	浮点设置保护
FSIN	浮点正弦
FSINCOS	浮点正余弦
FSQRT	浮点平方根
FST	浮点保存
FSTCW	浮点检查保存控制器
FSTENV	浮点检查保存环境
FSTP	浮点保存出栈
FSTSW	浮点检查保存状态器
FSUB	浮点减
FSUBP	浮点减出栈
FSUBR	浮点反减
FSUBRP	浮点反减出栈
FTST	浮点比零
FUCOM	浮点无序比较
FUCOMI	浮点反比加载标志

FUCOMIP	浮点反比加载标志出栈
FUCOMP	浮点无序比较出栈
FUCOMPP	浮点无序比较出栈二
FWAIT	浮点等
FXAM	浮点检查
FXCH	浮点交换
FXTRACT	浮点分解
FYL2X	浮点求L2X
FYL2XP1	浮点求L2XP1
MOVED	媒体双字传送
MOVEQ	媒体四字传送
PACKSSDW	媒体符号双字压缩
PACKSSWB	媒体符号字压缩
PACKUSWB	媒体无符号字压缩
PADDB	媒体截断字节加
PADDD	媒体截断双字加
PADDSB	媒体符号饱和字节加
PADDSIW	
PADDSW	媒体符号饱和字加
PADDUSB	媒体无符号饱和字节加
PADDUSW	媒体无符号饱和字加
PADDW	媒体截断字加
PAND	媒体与
PANDN	媒体与非
PAVEB	
PCMPEQB	媒体字节比等
PCMPEQD	媒体双字比等
PCMPEQW	媒体字比等
PCMPGTB	媒体字节比大
PCMPGTD	媒体双字比大
PCMPGTW	媒体字比大
PDISTIB	
PMACHRIW	
PMADDWD	
PMAGW	
PMULHRIW	

PMULHRWC	
PMULHW	
PMVGEZB	
PMVLZB	
PMVNZB	
PMVZB	
POR	媒体或
PSLLD	媒体双字左移
PSLLQ	媒体四字左移
PSLLW	媒体字左移
PSRAD	媒体双字算术右移
PSRAW	媒体字算术右移
PSRLD	媒体双字右移
PSRLQ	媒体四字右移
PSRLW	媒体字右移
PSUBB	媒体截断字节减
PSUBSB	媒体符号饱和字节减
PSUBSIW	
PSUBSW	媒体符号饱和字减
PSUBUSB	媒体无符号饱和字节减
PSUBUSW	媒体无符号饱和字减
PSUBW	媒体截断字减
PUNPCKHBW	媒体字节高位解压
PUNPCKHDQ	媒体双字高位解压
PUNPCKHWD	媒体字高位解压
PUNPCKLBW	媒体字节低位解压
PUNPCKLDQ	媒体双字低位解压
PUNPCKLWD	媒体字低位解压

Delphi 2010 VCL、JCL 源码中用到的汇编指令(只是粗略统计):

按名称排序	**使用次数**	**按使用频率排序**	**使用次数**
ADC	15	MOV	4053
ADD	659	PUSH	1505
AND	162	CMP	1372

BSF	8	POP	1187
BSR	7	JE	952
BSWAP	12	CALL	847
BT	13	JMP	771
BTC	9	ADD	659
BTR	10	JNE	503
BTS	10	TEST	452
CALL	847	SUB	400
CDQ	6	DEC	332
CLD	10	LEA	288
CMP	1372	RET	280
CPUID	3	INC	261
CWD	1	JZ	252
DB	241	OR	248
DD	189	DB	241
DEC	332	DD	189
DIV	40	JNZ	167
DW	63	MOVZX	166
ELSE	2	AND	162
END	2	FLD	154
F2XM1	6	SHR	131
FABS	7	JB	101
FADD	9	JG	92
FADDP	15	JA	86
FBSTP	3	REP	83
FCHS	5	JBE	81
FCLEX	5	XCHG	79
FCOM	7	JLE	79
FCOMP	7	FSTP	76
FCOMPP	3	LODSB	74
FCOS	4	JL	72
FDIV	11	FWAIT	72
FDIVP	5	NEG	70
FDIVRP	11	DW	63
FFREE	13	LOCK	61
FIADD	6	STOSB	58
FIDIV	2	STOSW	54

FILD	32	MOVSX	53
FIMUL	4	FLDCW	52
FINCSTP	1	FLD1	52
FISTP	30	SHL	48
FLD	154	JAE	48
FLD1	52	DIV	40
FLDCW	52	JGE	35
FLDL2E	6	REPNE	33
FLDLG2	2	LODSW	33
FLDLN2	9	IMUL	32
FLDZ	8	FMUL	32
FMUL	32	FILD	32
FMULP	26	JNS	31
FNCLEX	11	FISTP	30
FNINIT	2	FXCH	28
FNSTCW	20	FMULP	26
FNSTSW	6	JS	24
FPATAN	15	SBB	22
FPREM	3	FSTSW	22
FPTAN	4	LOOP	20
FRNDINT	14	FNSTCW	20
FSCALE	8	FSTCW	18
FSIN	3	NOT	17
FSINCOS	7	JECXZ	17
FSQRT	15	FYL2X	17
FST	5	MUL	16
FSTCW	18	JNC	16
FSTP	76	SAHF	15
FSTSW	22	ROR	15
FSUB	11	FSQRT	15
FSUBP	5	FPATAN	15
FSUBR	2	FADDP	15
FSUBRP	4	ADC	15
FTST	4	FRNDINT	14
FWAIT	72	FFREE	13
FXAM	1	BT	13
FXCH	28	SAR	12

FXTRACT	1	ROL	12
FYL2X	17	RCL	12
FYL2XP1	1	JO	12
HLT	1	BSWAP	12
IMUL	32	REPE	11
INC	261	FSUB	11
INT	8	FNCLEX	11
JA	86	FDIVRP	11
JAE	48	FDIV	11
JB	101	WAIT	10
JBE	81	CLD	10
JC	6	BTS	10
JE	952	BTR	10
JECXZ	17	SETC	9
JG	92	FLDLN2	9
JGE	35	FADD	9
JL	72	BTC	9
JLE	79	INT	8
JMP	771	FSCALE	8
JNA	1	FLDZ	8
JNC	16	BSF	8
JNE	503	PUSHFD	7
JNG	2	FSINCOS	7
JNL	1	FCOMP	7
JNS	31	FCOM	7
JNZ	167	FABS	7
JO	12	BSR	7
JRCXZ	2	NOP	6
JS	24	JC	6
JZ	252	FNSTSW	6
LEA	288	FLDL2E	6
LEAVE	1	FIADD	6
LOCK	61	F2XM1	6
LODSB	74	CDQ	6
LODSW	33	STOSD	5
LOOP	20	POPFD	5

MOV	4053	FSUBP	5
MOVSB	1	FST	5
MOVSX	53	FDIVP	5
MOVZX	166	FCLEX	5
MUL	16	FCHS	5
NEG	70	SHRD	4
NOP	6	PUSHF	4
NOT	17	POPF	4
OR	248	FTST	4
PAUSE	3	FSUBRP	4
POP	1187	FPTAN	4
POPF	4	FIMUL	4
POPFD	5	FCOS	4
PUSH	1505	RCR	3
PUSHF	4	PAUSE	3
PUSHFD	7	FSIN	3
RCL	12	FPREM	3
RCR	3	FCOMPP	3
REP	83	FBSTP	3
REPE	11	CPUID	3
REPNE	33	STD	2
RET	280	SETNZ	2
ROL	12	SETE	2
ROR	15	JRCXZ	2
SAHF	15	JNG	2
SAL	1	FSUBR	2
SAR	12	FNINIT	2
SBB	22	FLDLG2	2
SETC	9	FIDIV	2
SETE	2	END	2
SETNC	1	ELSE	2
SETNZ	2	STC	1
SHL	48	SHLD	1
SHLD	1	SETNC	1
SHR	131	SAL	1
SHRD	4	MOVSB	1
STC	1	LEAVE	1

STD	2	JNL	1
STOSB	58	JNA	1
STOSD	5	HLT	1
STOSW	54	FYL2XP1	1
SUB	400	FXTRACT	1
TEST	452	FXAM	1
WAIT	10	FINCSTP	1
XCHG	79	CWD	1

常见汇编命令英文缩写

寄存器类（register）：
    通用寄存器:　　EAX、EBX、ECX、EDX：是ax,bx,cx,dx的延伸，各为32位
        AH＆AL=AX(accumulator)　　:累加寄存器
        BH＆BL=BX(base)　　:基址寄存器
        CH＆CL=CX(count)　　:计数寄存器
        DH＆DL=DX(data)　　:数据寄存器
    特殊功能寄存器:　　ESP、EBP、ESI、EDI、EIP：是sp,bp,si,di,ip的延伸，32位
        SP(Stack Pointer)　　:堆栈指针寄存器
        BP(Base Pointer)　　:基址指针寄存器
        SI(Source Index)　　:源变址寄存器
        DI(Destination Index)　　:目的变址寄存器
        IP(Instruction Pointer)　　:指令指针寄存器
    段寄存器:
        CS(Code Segment)　　:代码段寄存器
        DS(Data Segment)　　:数据段寄存器
        SS(Stack Segment)　　:堆栈段寄存器
        ES(Extra Segment)　　:附加段寄存器
    标志寄存器
        FR--flag register（程序状态字PSW--program status word）,PSW常用的标志有:
        标志　　值为1时的标记　　值为0时的标记　　说明
        OF(overflow flag)　　OV(overflow)　　NV(not overflow)　　溢出标志　操作数超出机器能表示的范围表
        ZF(zero flag)　　ZR(zero)　　NZ(not zero)　　零标志　运算结果等于0时为1.否则为0.
        PF(parity flag)　　PE(parity even)　　PO(parity odd)　　奇偶标志　运算结果操作数位为1的个数为偶
        CF(carry flag)　　CY(carried)　　NC(not carried)　　进位标志　最高有效位产生进位时为1.否则
        DF(direction flag)　　DN(down)　　UP(up)　　方向标志　用于串处理.DF=1时.每次操作后
        SF(sign flag)　　NG(negtive)　　PL(plus)　　符号标志　记录运算结果的符号.结果负时为
        TF(trap flag)　　陷阱标志　用于调试单步操作.
        IF(interrupt flag)　　中断标志　IF=1时.允许CPU响应可屏蔽中
        AF(auxiliary flag)　　辅助进位标志　运算时.第3位向第4位产生进
一、命令类
    1.通用数据传送指令.
        MOV----> move　　传送字或字节
        MOVSX---->extended move with sign data　　先符号扩展,再传送
        MOVZX---->extended move with zero data　　先零扩展,再传送
        PUSH---->push　　把字压入堆栈
        POP---->pop　　把字弹出堆栈
        PUSHA---->push all　　把AX,CX,DX,BX,SP,BP,SI,DI依次压入堆栈
        POPA---->pop all　　把DI,SI,BP,SP,BX,DX,CX,AX依次弹出堆栈
        PUSHAD---->push all data　　把EAX,ECX,EDX,EBX,ESP,EBP,ESI,EDI依次压入堆栈
        POPAD---->pop all data　　把EDI,ESI,EBP,ESP,EBX,EDX,ECX,EAX依次弹出堆栈
        BSWAP---->byte swap　　交换32位寄存器里字节的顺序
        XCHG---->exchange　交换字或字节.(至少有一个操作数为寄存器,段寄存器不可作为操作数)
        CMPXCHG---->compare and change　比较并交换操作数.第二个操作数为累加器AL/AX/EAX
        XADD---->exchange and add　　先交换再累加.(结果在第一个操作数里)
        XLAT---->translate　　字节查表转换
    2.输入输出端口传送指令.
        IN---->input　　I/O端口输入.(语法: IN 累加器,{ 端口号|DX })
        OUT---->output　　I/O端口输出.(语法: OUT { 端口号|DX },累加器)
    3.目的地址传送指令.
        LEA---->load effective address　　装入有效地址

LDS---->load DS 传送目标指针,把指针内容装入DS
LES---->load ES 传送目标指针,把指针内容装入ES
LFS---->load FS 传送目标指针,把指针内容装入FS
LGS---->load GS 传送目标指针,把指针内容装入GS
LSS---->load SS 传送目标指针,把指针内容装入SS

4.标志传送指令.

LAHF---->load AH from flag 标志寄存器传送,把标志装入AH.
SAHF---->save AH to flag 标志寄存器传送,把AH内容装入标志寄存器
PUSHF---->push flag 标志入栈
POPF---->pop flag 标志出栈
PUSHD---->push dflag 32位标志入栈
POPD---->pop dflag 32位标志出栈

二、算术运算指令

ADD---->add 加法
ADC---->add with carry 带进位加法
INC---->increase 1 加1
AAA---->ascii add with adjust 加法的ASCII码调整
DAA---->decimal add with adjust 加法的十进制调整
SUB---->substract 减法
SBB---->substract with borrow 带借位减法
DEC---->decrease 1 减1
NEC---->negative 求反(以 0 减之)
CMP---->compare 比较.两操作数作减法,仅修改标志位,不回送结果
AAS---->ascii adjust on substract 减法的ASCII码调整.
DAS---->decimal adjust on substract 减法的十进制调整
MUL---->multiplication 无符号乘法，结果回送AH和AL(字节运算),或DX和AX(字运算)
IMUL---->integer multiplication 整数乘法，结果回送AH和AL(字节运算),或DX和AX(字运算)
AAM---->ascii adjust on multiplication 乘法的ASCII码调整
DIV---->divide 无符号除法
IDIV---->integer divide 整数除法，商回送AL余数回送AH，字节运算，商回送AX余数回送DX，字运算
AAD---->ascii adjust on divide 除法的ASCII码调整
CBW---->change byte to word 字节转换为字.(把AL中字节的符号扩展到AH中去)
CWD---->change word to double word 字转换为双字.(把AX中的字的符号扩展到DX中去)
CWDE---->change word to double word with sign to EAX字转换为双字.(把AX中的字符号扩展到EAX中去)
CDQ---->change double word to quadrate word 双字扩展.把EAX中的字的符号扩展到EDX

三、逻辑运算指令

AND---->and 与运算
OR---->or 或运算
XOR---->xor 异或运算
NOT---->not 取反
TEST---->test 测试.(两操作数作与运算,仅修改标志位,不回送结果)
SHL---->shift left 逻辑左移
SAL---->arithmatic shift left 算术左移.(=SHL)
SHR---->shift right 逻辑右移
SAR---->arithmatic shift right 算术右移.(=SHR)
ROL---->rotate left 循环左移
ROR---->rotate right 循环右移
RCL---->rotate left with carry 通过进位的循环左移
RCR---->rotate right with carry 通过进位的循环右移

四、串指令

MOVS---->move string 串传送，MOVSB传送字符、MOVSW传送字、MOVSD传送双字
CMPS---->compare string 串比较，CMPSB比较字符、CMPSW比较字
SCAS---->scan string 串扫描，把AL或AX的内容与目标串作比较,比较结果反映在标志位
LODS---->load string 装入串，把源串中的元素(字或字节)逐一装入AL或AX中，LODSB传送字符、LODSW传
STOS---->store string 保存串，是LODS的逆过程
REP---->repeat 当CX/ECX<>0时重复
REPE---->repeat when equal 当比较结果相等,且CX/ECX<>0时重复
REPZ---->repeat when zero flag 当ZF=1,且CX/ECX<>0时重复
REPNE---->repeat when not equal 当比较结果不相等,且CX/ECX<>0时重复
REPNZ---->repeat when zero flag 当ZF=0,且CX/ECX<>0时重复
REPC---->repeat when carry flag 当CF=1且CX/ECX<>0时重复
REPNC---->repeat when not carry flag 当CF=0且CX/ECX<>0时重复

五、程序转移指令

1>无条件转移指令(长转移)

JMP---->jump 无条件转移指令
CALL---->call 过程调用
RET---->return 过程返回
RETF---->return far 过程返回

2>条件转移指令(短转移,-128到+127的距离内) : 当且仅当(SF XOR OF)=1时,OP1<OP2

JAE---->jump when above or equal 不小于时转移
JNB---->jump when not below 不小于时转移
JB---->jump when below 小于时转移
JNAE---->jump when not above or equal 小于时转移

```
        JBE---->jump when below or equal                小于等于时转移
        JNA---->jump when not above                     小于等于时转移
          以上条目,测试无符号整数运算的结果(标志C和Z)
        JG---->jump when greater                        大于转移
        JNLE---->jump when not less or equal            大于转移
        JGE---->jump when greater or equal              大于等于转移
        JNL---->jump when not less                      大于等于转移
        JL---->jump when less                           小于转移
        JNGE---->jump when not greater or equal         小于转移
        JLE---->jump when less or equal                 小于等于转移
        JNG---->jump when not greater                   小于等于转移
          以上条目,测试带符号整数运算的结果(标志S,O和Z).
        JE---->jump when equal                          等于转移
        JZ---->jump when has zero flag                  结果为0转移
        JNE---->jump when not equal                     不等于转移
        JNZ---->jump when not has zero flag             结果不为0转移
        JC---->jump when has carry flag                 有进位转移
        JNC---->jump when not has carry flag            无进位转移
        JNO---->jump when not has overflow flag         不溢出时转移
        JNP---->jump when not has parity flag           奇偶性为奇数时转移
        JPO---->jump when parity flag is odd            奇偶性为奇数时转移
        JNS---->jump when not has sign flag             符号位为0时转移
        JO---->jump when has overflow flag              溢出时转移
        JP---->jump when has parity flag                奇偶性为偶数时转移
        JPE---->jump when parity flag is even           奇偶性为偶数时转移
        JS---->jump when has sign flag                  符号位为0时转移
    3>循环控制指令(短转移)
        LOOP---->loop                                   CX不为零时循环
        LOOPE---->loop equal                CX不为零且结果相等时循环（相等时Z=1）
        LOOPZ---->loop zero                             CX不为零且标志Z=1时循环
        LOOPNE---->loop not equal           CX不为零且结果不相等时循环（相等时Z=0）
        LOOPNZ---->loop not zero                        CX不为零且标志Z=0时循环
        JCXZ---->jump when CX is zero                   CX为零时转移
        JECXZ---->jump when ECX is zero                 ECX为零时转移
    4>中断指令
        INT---->interrupt                               ECX为零时转移
        INTO---->overflow interrupt                     溢出中断
        IRET---->interrupt return                       中断返回
    5>处理器控制指令
        HLT---->halt                    处理器暂停,直到出现中断或复位信号才继续
        WAIT---->wait                   当芯片引线TEST为高电平时使CPU进入等待状态
        ESC---->escape                                  转换到外处理器
        LOCK---->lock                                   封锁总线
        NOP---->no operation                            空操作
        STC---->set carry                               置进位标识位
        CLC---->clear carry                             清进位标识位
        CMC---->carry make change                       进位标识取反
        STD---->set direction                           置方向标识位
        CLD---->clear direction                         清方向标识位
        STI---->set interrupt                           置中断允许位
        CLI---->clear interrupt                         清中断允许位
六、伪指令
    DW---->definw word                                  定义字(2字节)
    PROC---->procedure                                  定义过程
    ENDP---->end of procedure                           过程结束
    SEGMENT---->segment                                 定义段
    ASSUME---->assume                                   建立段寄存器寻址
    ENDS---->end segment                                段结束
    END---->end                                         程序结束
```

AAA-添加后进行ASCII调整
AAD-ASCII分割前调整AX
AAM-ASCII调整AX后乘以
AAS-ASCII减法后调整AL
ADC-带进位加法
ADCX - 带进位标志的两个操作数的无符号整数相加
ADD-加

ADDPD-Add打包的双精度浮点值
ADDPS-Add打包的单精度浮点值
ADDSD-Add标量双精度浮点值
ADDSS - add标量单精度浮点值
ADDSUBPD-压缩双FP加/减
ADDSUBPS-压缩单FP加/减
ADOX - 带有溢出标志的两个操作数的无符号整数
AESDEC - 执行一轮AES解密流程
AESDECLAST - 执行AES解密流的最后一轮
AESENC - 执行一轮AES加密流程
AESENCLAST - 执行AES加密流的最后一轮
AESIMC - 执行AES InvMixColumn转换
AESKEYGENASSIST-AES轮回密钥生成辅助
AND-逻辑与
ANDN - 逻辑AND NOT
ANDPD-压缩双精度浮点值的按位逻辑与
ANDPS—压缩单精度浮点值的按位逻辑与
ANDNPD—压缩双精度浮点值的按位逻辑AND NOT
ANDNPS—压缩单精度浮点值的按位逻辑AND NOT
ARPL—调整段选择器的RPL字段
BLENDPD — 混合封装双精度浮点值
BEXTR — 位字段提取
BLENDPS — 混合封装的单精度浮点值
BLENDVPD — 可变混合封装双精度浮点值
BLENDVPS — 可变混合封装单精度浮点值
BLSI — 提取最低设置隔离位
BLSMSK — 获取掩码到最低设置位
BLSR — 复位最低设置位
BNDCL—检查下限
BNDCU/BNDCN—检查上边界
BNDLDX—使用地址转换加载扩展边界
BNDMK—制作界限
BNDMOV—移动边界
BNDSTX—使用地址转换存储扩展边界
BOUND—检查阵列索引对边界
BSF—位扫描转发
BSR—位扫描反转
BSWAP—字节交换
BT—位测试
BTC—位测试和补码
BTR—位测试和复位
BTS—位测试和设置
BZHI — 从指定位位置开始的零高位
CALL—调用过程
CBW/CWDE/CDQE—将字节转换为字/将字转换为双字/将双字转换为四字
CLAC—清除EFLAGS寄存器中的AC标志
CLC—清除进位标志
CLD—清除方向标志
CLFLUSH—刷新缓存行
CLFLUSHOPT—刷新缓存行已优化
CLI — 清除中断标志
CLTS—清除CR0中的任务切换标志
CLWB—高速缓存行回写
CMC—补充进位标志
CMOVcc—条件移动
CMP—比较两个操作数
CMPPD—比较打包的双精度浮点值
CMPPS—比较打包的单精度浮点值
CMPS/CMPSB/CMPSW/CMPSD/CMPSQ—比较字符串操作数
CMPSD—比较标量双精度浮点值
CMPSS—比较标量单精度浮点值
CMPXCHG—比较和交流
CMPXCHG8B/CMPXCHG16B—比较和交换字节
COMISD—比较标量有序双精度浮点值和设置EFLAGS
COMISS—比较标量有序单精度浮点值和设置EFLAGS
CPUID—CPU识别
CRC32 — 累加CRC32值
CVTDQ2PD—将打包的双字整数转换为打包的双精度浮点值
CVTDQ2PS—将打包的双字整数转换为打包的单精度浮点值
CVTPD2DQ—将打包的双精度浮点值转换为打包的双字整数
CVTPD2PI—将打包的双精度FP值转换为打包的双字整数
CVTPD2PS—将打包的双精度浮点值转换为打包的单精度浮点值
CVTPI2PD—将打包的双字整数转换为打包的双精度FP值

CVTPI2PS—将打包的双字整数转换为打包的单精度FP值
CVTPS2DQ—将打包的单精度浮点值转换为打包签名的双字整数值
CVTPS2PD—将打包的单精度浮点值转换为打包的双精度浮点值
CVTPS2PI—将打包的单精度FP值转换为打包的双字整数
CVTSD2SI—将标量双精度浮点值转换为双字整数
CVTSD2SS—将标量双精度浮点值转换为标量单精度浮点值
CVTSI2SD—将双字整数转换为标量双精度浮点值
CVTSI2SS—将双字整数转换为标量单精度浮点值
CVTSS2SD—将标量单精度浮点值转换为标量双精度浮点值
CVTSS2SI—将标量单精度浮点值转换为双字整数
CVTTPD2DQ—转换为截断打包的双精度浮点值到打包的双字整数
CVTTPD2PI—转换为截断打包的双精度FP值到打包的双字整数
CVTTPS2DQ—将截断转换为打包的单精度浮点值到打包签名的双字整数值
CVTTPS2PI—转换为截断打包的单精度FP值到打包的双字整数
CVTTSD2SI—将截断的标量双精度浮点值转换为有符号整数
CVTTSS2SI—将截断标量单精度浮点值转换为整数
CWD/CDQ/CQO—将字转换为双字/将双字转换为四字
DAA—十进制加法后调整AL
DAS—十进制减法后调整AL
DEC—减少1
DIV—无符号除法
DIVPD—除法包装的双精度浮点值
DIVPS—分割打包的单精度浮点值
DIVSD—除法标量双精度浮点值
DIVSS—除法标量单精度浮点值
DPPD — 封装双精度浮点值的点积
DPPS — 封装单精度浮点值的点积
EMMS—空MMX技术状态
ENTER—为过程参数创建堆栈框架
EXTRACTPS—提取打包的浮点值
F2XM1—计算2x-1
FABS—绝对值
FADD/FADDP/FIADD—Add
FBLD—加载二进制编码十进制
FBSTP—存储BCD整数和弹出
FCHS—更改标志
FCLEX/FNCLEX—清除例外
FCMOVcc—浮点条件移动
FCOM/FCOMP/FCOMPP—比较浮点值
FCOMI/FCOMIP/ FUCOMI/FUCOMIP—比较浮点值和设置EFLAGS
FCOS— 余弦
FDECSTP—减少堆栈顶指针
FDIV/FDIVP/FIDIV—划分
FDIVR/FDIVRP/FIDIVR—反向分割
FFREE—自由浮点寄存器
FICOM/FICOMP—比较整数
FILD—加载整数
FINCSTP—增加堆栈顶指针
FINIT/FNINIT—初始化浮点单元
FIST/FISTP—存储整数
FISTTP—存储整数与截断
FLD—加载浮点值
FLD1/FLDL2T/FLDL2E/FLDPI/FLDLG2/FLDLN2/FLDZ—负载常数
FLDCW—加载x87 FPU控制字
FLDENV—加载x87 FPU环境
FMUL/FMULP/FIMUL—乘
FNOP—无操作
FPATAN—部分反正切
FPREM—部分剩余
FPREM1—部分剩余
FPTAN—部分切线
FRNDINT—舍入为整数
FRSTOR—恢复x87 FPU状态
FSAVE/FNSAVE—存储x87 FPU状态
FSCALE—规模
FSIN—正弦
FSINCOS—正弦和余弦
FSQRT—平方根
FST/FSTP—存储浮点值
FSTCW/FNSTCW—存储x87 FPU控制字
FSTENV/FNSTENV—存储x87 FPU环境
FSTSW/FNSTSW—存储x87 FPU状态字
FSUB/FSUBP/FISUB—减去

FSUBR/FSUBRP/FISUBR—反向减
FTST—TEST
FUCOM/FUCOMP/FUCOMPP—无序比较浮点值
FXAM—检查浮点
FXCH—交换寄存器内容
FXRSTOR—恢复x87 FPU，MMX，XMM和MXCSR状态
FXSAVE—保存x87 FPU，MMX技术和SSE状态
FXTRACT—提取指数和指标
FYL2X—计算y * log2x
FYL2XP1—计算y * log2（x + 1）
HADDPD—包装双FP水平添加
HADDPS—包装单FP水平添加
HLT—停
HSUBPD—压缩双FP水平减法
HSUBPS—打包单FP水平减法
IDIV—签名除法
IMUL—签名乘法
IN—从端口输入
INC—递增1
INS/INSB/INSW/INSD—从端口到字符串的输入
INSERTPS—插入标量单精度浮点值
INT n/INTO/INT 3—调用中断过程
INVD—无效内部缓存
INVLPG—使TLB条目无效
INVPCID—使过程上下文标识符无效
IRET/IRETD—中断返回
Jcc—如果条件满足则跳转
JMP—跳
KADDW/KADDB/KADDQ/KADDD—ADD两个面具
KANDW/KANDB/KANDQ/KANDD—按位逻辑和掩码
KANDNW/KANDNB/KANDNQ/KANDND—按位逻辑AND NOT掩码
KMOVW/KMOVB/KMOVQ/KMOVD—从和到掩码寄存器
KNOTW/KNOTB/KNOTQ/KNOTD—NOT屏蔽寄存器
KORW/KORB/KORQ/KORD—按位逻辑或掩码
KORTESTW/KORTESTB/KORTESTQ/KORTESTD—OR Masks And Set Flags
KSHIFTLW/KSHIFTLB/KSHIFTLQ/KSHIFTLD—移位左掩码寄存器
KSHIFTRW/KSHIFTRB/KSHIFTRQ/KSHIFTRD—Shift右掩码寄存器
KTESTW/KTESTB/KTESTQ/KTESTD—打包位测试掩码和设置标志
KUNPCKBW/KUNPCKWD/KUNPCKDQ—解包掩码寄存器
KXNORW/KXNORB/KXNORQ/KXNORD—按位逻辑XNOR掩码
KXORW/KXORB/KXORQ/KXORD—按位逻辑异或掩码
LAHF—将状态标志加载到AH寄存器中
LAR—加载访问权限字节
LDDQU—加载未对齐的整数128位
LDMXCSR—加载MXCSR寄存器
LDS/LES/LFS/LGS/LSS—加载远程指针
LEA—加载有效地址
LEAVE—高级过程退出
LFENCE—负载栅栏
LGDT/LIDT—加载全局/中断描述符表寄存器
LLDT—加载本地jubu描述符表寄存器
LMSW—加载机器状态字
LOCK—置位LOCK＃信号前缀
LODS/LODSB/LODSW/LODSD/LODSQ—加载字符串
LOOP/LOOPcc—根据ECX计数器循环
LSL—负载段限制
LTR—加载任务寄存器
LZCNT— 计数前导零位的数量

MASKMOVDQU—存储双字双字的所选字节
MASKMOVQ—存储选定的四字节字节
MAXPD—最大打包双精度浮点值
MAXPS—最大打包单精度浮点值
MAXSD—返回最大标量双精度浮点值
MAXSS—返回最大标量单精度浮点值
MFENCE—内存围栏
MINPD—最小包装双精度浮点值
MINPS—最小打包单精度浮点值
MINSD—返回最小标量双精度浮点值
MINSS—返回最小标量单精度浮点值
MONITOR—设置监视器地址
MOV—移动
MOVAPD—移动对齐打包的双精度浮点值

MOVAPS—移动对齐打包的单精度浮点值
MOVBE—在交换字节后移动数据
MOVD/MOVQ—移动双字/移动四字
MOVDDUP—复制双精度浮点值
MOVDQA,VMOVDQA32/64—移动对齐的打包整数值
MOVDQU,VMOVDQU8/16/32/64—移动未对齐的打包整数值
MOVDQ2Q—将四字从XMM移动到MMX技术寄存器
MOVHLPS—将打包的单精度浮点值从高到低移动
MOVHPD—移动高压缩双精度浮点值
MOVHPS—移动高度封装的单精度浮点值
MOVLHPS—将打包的单精度浮点值从低到高移动
MOVLPD—移动低压缩双精度浮点值
MOVLPS—移动低压缩单精度浮点值
MOVMSKPD—提取封装的双精度浮点符号掩码
MOVMSKPS—提取打包的单精度浮点符号掩码
MOVNTDQA—加载双字体非时间对齐提示
MOVNTDQ—使用非时间提示存储打包的整数
MOVNTI—使用非时间提示存储双字
MOVNTPD—使用非时间提示存储打包的双精度浮点值
MOVNTPS—使用非时间提示存储打包的单精度浮点值
MOVNTQ—使用非时间提示的四字存储
MOVQ—移动四字
MOVQ2DQ—将四字从MMX技术移动到XMM寄存器
MOVS/MOVSB/MOVSW/MOVSD/MOVSQ—将数据从字符串移动到字符串
MOVSD—移动或合并标量双精度浮点值
MOVSHDUP—复制单个FP值
MOVSLDUP—复制单个FP值
MOVSS—移动或合并标量单精度浮点值
MOVSX/MOVSXD—使用符号扩展移动
MOVUPD—移动非对齐打包的双精度浮点值
MOVUPS—移动不对齐打包的单精度浮点值
MOVZX—用零扩展移动
MPSADBW — 计算多个封装的绝对差的和
MUL—无符号乘法
MULPD—乘法封装的双精度浮点值
MULPS—乘法封装的单精度浮点值
MULSD—乘以标量双精度浮点值
MULSS—乘以标量单精度浮点值
MULX — 无符号乘法不影响标志
MWAIT—监视等待
NEG—二的互补阴性
NOP—无操作
NOT—一个补码阴性
OR—逻辑包含OR
ORPD—压缩双精度浮点值的按位逻辑或
ORPS—压缩单精度浮点值的按位逻辑或
OUT—输出到端口
OUTS/OUTSB/OUTSW/OUTSD—输出字符串到端口
PABSB/PABSW/PABSD/PABSQ — 压缩绝对值
PACKSSWB/PACKSSDW—包含有符号饱和度
PACKUSDW—具有无符号饱和度的包
PACKUSWB—具有无符号饱和度的包
PADDB/PADDW/PADDD/PADDQ—添加打包的整数
PADDSB/PADDSW—添加带签名饱和度的打包签名整数
PADDUSB/PADDUSW—添加带有无符号饱和的打包的无符号整数
PALIGNR — 包装对齐
PAND—逻辑AND
PANDN—逻辑AND NOT
PAUSE—旋转环提示
PAVGB/PAVGW—平均打包整数
PBLENDVB — 可变混合打包字节
PBLENDW — 混合包装的词
PCLMULQDQ - 无载乘法四字
PCMPEQB/PCMPEQW/PCMPEQD— 比较打包数据以等于
PCMPEQQ — 比较打包的Qword数据
PCMPESTRI — 压缩比较显式长度字符串，返回索引
PCMPESTRM — 压缩比较显式长度字符串，返回掩码
PCMPGTB/PCMPGTW/PCMPGTD—比较打包签名的整数大于
PCMPGTQ — 比较打包数据大于
PCMPISTRI — 压缩比较隐式长度字符串，返回索引
PCMPISTRM — 压缩比较隐式长度字符串，返回掩码
PDEP — 平行位存款
PEXT — 平行位提取

PEXTRB/PEXTRD/PEXTRQ — 提取字节/双字/ Qword
PEXTRW—提取 Word
PHADDW/PHADDD — 包装水平添加
PHADDSW — 包装水平添加和饱和
PHMINPOSUW — 包装水平词最小
PHSUBW/PHSUBD — 打包水平减法
PHSUBSW — 包装水平扣除和饱和
PINSRB/PINSRD/PINSRQ — 插入字节/双字/ Qword
PINSRW—插入字
PMADDUBSW — 乘法和添加打包签名和无符号字节
PMADDWD—乘法和添加打包整数
PMAXSB/PMAXSW/PMAXSD/PMAXSQ—最大打包签名整数
PMAXUB/PMAXUW—最大打包的无符号整数
PMAXUD/PMAXUQ—最大打包的无符号整数
PMINSB/PMINSW—最小打包签名整数
PMINSD/PMINSQ—最小打包签名整数
PMINUB/PMINUW—最小打包的无符号整数
PMINUD/PMINUQ—最小打包的无符号整数
PMOVMSKB—移动字节掩码
PMOVSX—包装移动与符号扩展
PMOVZX—带零扩展的打包移动
PMULDQ—乘以压缩双字整数
PMULHRSW — 包装高与圆和规模
PMULHUW—乘法打包的无符号整数和存储高结果
PMULHW—乘法打包签名整数和存储高结果
PMULLD/PMULLQ—乘以压缩整数和存储低结果
PMULLW—乘以打包的有符号整数和存储低结果
PMULUDQ—乘法打包的无符号双字整数
POP—从堆栈中弹出一个值
POPA/POPAD—弹出所有通用寄存器
POPCNT — 返回设置为1的位数计数
POPF/POPFD/POPFQ—弹出堆栈到 EFLAGS 寄存器
POR—按位逻辑或
PREFETCHh—预取数据到缓存
PREFETCHW—在写入预期中将数据预取到缓存中
PREFETCHWT1—将向量数据预取到具有意图写入和T1提示的高速缓存中
PSADBW—计算绝对差的和
PSHUFB — 打包的随机字节
PSHUFD—随机打包双字
PSHUFHW—随机包装高字
PSHUFLW—随机包装低字
PSHUFW—随机包装的词
PSIGNB/PSIGNW/PSIGND — 打包标志
PSLLDQ—移位双四字左逻辑
PSLLW/PSLLD/PSLLQ—移位数据打包左逻辑
PSRAW/PSRAD/PSRAQ—移位打包数据右算术
PSRLDQ—移位双四字右逻辑
PSRLW/PSRLD/PSRLQ—移位打包数据右逻辑
PSUBB/PSUBW/PSUBD—减去打包的整数
PSUBQ—减去打包的四字整数
PSUBSB/PSUBSW—减去带有符号饱和的打包有符号整数
PSUBUSB/PSUBUSW—使用无符号饱和度减去打包的无符号整数
PTEST- 逻辑比较
PTWRITE - 将数据写入处理器跟踪数据包
PUNPCKHBW/PUNPCKHWD/PUNPCKHDQ/PUNPCKHQDQ— 解压高数据
PUNPCKLBW/PUNPCKLWD/PUNPCKLDQ/PUNPCKLQDQ—解压低数据
PUSH—将字，双字或四字推到堆栈上
PUSHA/PUSHAD—推送所有通用寄存器
PUSHF/PUSHFD—将EFLAGS寄存器推送到堆栈
PXOR—逻辑异或
RCL/RCR/ROL/ROR—旋转
RCPPS—计算包装的单精度浮点值的倒数
RCPSS—计算标量单精度浮点值的倒数
RDFSBASE/RDGSBASE—读取FS / GS段基址
RDMSR—从模型专用寄存器读取
RDPID—读取处理器ID
RDPKRU—读取用户页面的保护关键权限
RDPMC—读取性能监视计数器
RDRAND—读随机数
RDSEED—阅读随机SEED
RDTSC—读取时间戳计数器
RDTSCP—读取时间戳计数器和处理器ID
REP/REPE/REPZ/REPNE/REPNZ—重复字符串操作前缀

RET—从程序返回
RORX — 向右旋转逻辑而不影响标志
ROUNDPD — 圆形双精度浮点值
ROUNDPS — 圆形封装单精度浮点值
ROUNDSD — 圆形标量双精度浮点值
ROUNDSS — 圆形标量单精度浮点值
RSM—从系统管理模式恢复
RSQRTPS—计算压缩单精度浮点值的平方根的倒数
RSQRTSS—计算标量单精度浮点值的平方根的倒数
SAHF—将AH存储到标志
SAL/SAR/SHL/SHR—转移
SARX/SHLX/SHRX — 转移不影响标志
SBB—借用整数减法
SCAS/SCASB/SCASW/SCASD—扫描字符串
SETcc—在字段上设置字节
SFENCE—商店栅栏
SGDT—存储全局描述符表寄存器
SHA1RNDS4—执行四轮SHA1操作
SHA1NEXTE—计算四轮后的SHA1状态变量E.
SHA1MSG1—对下四个SHA1消息双字执行中间计算
SHA1MSG2—对下四个SHA1消息双字执行最终计算
SHA256RNDS2—执行两轮SHA256操作
SHA256MSG1—对下四个SHA256消息双字执行中间计算
SHA256MSG2—对下四个SHA256消息双字执行最终计算
SHLD—双精度位移左
SHRD—双精度平移
SHUFPD—包装交错双精度浮点值对的随机
SHUFPS—Packed Interleave单精度浮点值四重串交替
SIDT—存储中断描述符表寄存器
SLDT—存储本地描述符表寄存器
SMSW—存储机器状态字
SQRTPD—双精度浮点值的平方根
SQRTPS—单精度浮点值的平方根
SQRTSD—计算平方根的标量双精度浮点值
SQRTSS—计算标量单精度值的平方根
STAC—在EFLAGS寄存器中设置AC标志
STC—设置进位标志
STD—设置方向标志
STI—设置中断标志
STMXCSR—存储MXCSR寄存器状态
STOS/STOSB/STOSW/STOSD/STOSQ—存储字符串
STR—存储任务寄存器
SUB—减去
SUBPD—减去打包的双精度浮点值
SUBPS—减去打包的单精度浮点值
SUBSD—减去标量双精度浮点值
SUBSS—减去标量单精度浮点值
SWAPGS—交换GS基址寄存器
SYSCALL—快速系统调用
SYSENTER—快速系统调用
SYSEXIT—从快速系统调用快速返回
SYSRET—从快速系统调用返回
TEST—逻辑比较
TZCNT — 计算零位的位数
UCOMISD—无序比较标量双精度浮点值并设置EFLAGS
UCOMISS—无序比较标量单精度浮点值并设置EFLAGS
UD2—未定义指令
UNPCKHPD—解包和交织高压双精度浮点值
UNPCKHPS—解包和交织高度封装的单精度浮点值
UNPCKLPD—解包和交织低压双精度浮点值
UNPCKLPS—解包和交织低压缩单精度浮点值

VALIGND/VALIGNQ—对齐双字/四字向量
VBLENDMPD/VBLENDMPS—使用OpMask控件的Blend Float64 / Float32向量
VBROADCAST—加载广播浮点数据
VPBROADCASTM—广播掩码到向量寄存器
VCOMPRESSPD—将稀疏压缩双精度浮点值存储到密集存储器中
VCOMPRESSPS—将稀疏打包的单精度浮点值存储到密集存储器中
VCVTPD2QQ—将打包的双精度浮点值转换为打包的四字整数
VCVTPD2UDQ—将打包的双精度浮点值转换为打包的无符号双字整数
VCVTPD2UQQ—将打包的双精度浮点值转换为打包的无符号四字整数
VCVTPH2PS—将16位FP值转换为单精度FP值
VCVTPS2PH—将Single-Precision FP值转换为16位FP值

VCVTPS2UDQ—将打包的单精度浮点值转换为打包的无符号双字整数值
VCVTPS2QQ—将打包的单精度浮点值转换为打包的有符号四字整数值
VCVTPS2UQQ—将打包的单精度浮点值转换为打包的无符号四字整数值
VCVTQQ2PD—将打包的四字整数转换为打包的双精度浮点值
VCVTQQ2PS—将打包的四字整数转换为打包的单精度浮点值
VCVTSD2USI—将标量双精度浮点值转换为无符号双字整数
VCVTSS2USI—将标量单精度浮点值转换为无符号双字整数
VCVTTPD2QQ—将截断打包的双精度浮点值转换为打包的四字整数
VCVTTPD2UDQ—转换为截断打包的双精度浮点值到打包的无符号双字整数
VCVTTPD2UQQ—将截断的双精度浮点值转换为打包的无符号四字整数
VCVTTPS2UDQ—将截断转换为打包的单精度浮点值到打包的无符号双字整数值
VCVTTPS2QQ—将截断转换为打包的单精度浮点值到打包的有符号四字整数值
VCVTTPS2UQQ—将截断转换为打包的单精度浮点值到打包的无符号四字整数值
VCVTTSD2USI—将截断的标量双精度浮点值转换为无符号整数
VCVTTSS2USI—将截断标量单精度浮点值转换为无符号整数
VCVTUDQ2PD—将打包的无符号双字整数转换为打包的双精度浮点值
VCVTUDQ2PS—将打包的无符号双字整数转换为打包的单精度浮点值
VCVTUQQ2PD—将打包的无符号四字整数转换为打包的双精度浮点值
VCVTUQQ2PS—将打包的无符号四字整数转换为打包的单精度浮点值
VCVTUSI2SD—将无符号整数转换为标量双精度浮点值
VCVTUSI2SS—将无符号整数转换为标量单精度浮点值
VDBPSADBW—对无符号字节的双块打包和绝对差（SAD）
VEXPANDPD—从密集存储器加载稀疏压缩双精度浮点值
VEXPANDPS—从密集存储器加载稀疏打包的单精度浮点值
VERR/VERW—验证读取或写入的段
VEXP2PD—近似于指数2 ^ x的打包双精度浮点值，小于2 ^ -23相对误差
VEXP2PS—近似于包装的单精度浮点值的指数2 ^ x小于2 ^ -23相对误差
VEXTRACTF128/VEXTRACTF32x4/VEXTRACTF64x2/VEXTRACTF32x8/VEXTRACTF64x4—Extr act打包浮点值
VEXTRACTI128/VEXTRACTI32x4/VEXTRACTI64x2/VEXTRACTI32x8/VEXTRACTI64x4—提取打包的整数值
VFIXUPIMMPD—修复特殊打包的Float64值
VFIXUPIMMPS—修复特殊打包的Float32值
VFIXUPIMMSD—修复特殊标量Float64价值
VFIXUPIMMSS—修复特殊标量Float32价值
VFMADD132PD/VFMADD213PD/VFMADD231PD—熔丝乘法 - 加上双精度浮点值
VFMADD132PS/VFMADD213PS/VFMADD231PS—融合乘法 - 加上单精度浮点值
VFMADD132SD/VFMADD213SD/VFMADD231SD—熔丝乘加 - 标量双精度浮点值
VFMADD132SS/VFMADD213SS/VFMADD231SS—融合乘法 - 加上标量单精度浮点值
VFMADDSUB132PD/VFMADDSUB213PD/VFMADDSUB231PD—熔断乘法 - 交替加/减包装的双精度浮点值
VFMADDSUB132PS/VFMADDSUB213PS/VFMADDSUB231PS—封装单精度浮点值的融合乘法交替加法/减法
VFMSUBADD132PD/VFMSUBADD213PD/VFMSUBADD231PD—融合乘法交替减法/增加的双精度浮点值
VFMSUBADD132PS/VFMSUBADD213PS/VFMSUBADD231PS—融合乘法交替减法/添加的单精度浮点值
VFMSUB132PD/VFMSUB213PD/VFMSUB231PD—压缩双精度浮点值的乘法减法
VFMSUB132PS/VFMSUB213PS/VFMSUB231PS—压缩单精度浮点值的乘法减法
VFMSUB132SD/VFMSUB213SD/VFMSUB231SD—标量双精度浮点值的融合乘法 - 减法
VFMSUB132SS/VFMSUB213SS/VFMSUB231SS—标量单精度浮点值的融合乘减
VFNMADD132PD/VFNMADD213PD/VFNMADD231PD—熔丝负精度浮点值的乘积 - 加法
VFNMADD132PS/VFNMADD213PS/VFNMADD231PS—熔丝负精度浮点值的乘积 - 加法
VFNMADD132SD/VFNMADD213SD/VFNMADD231SD—熔丝负精度浮点值的乘积 - 加法
VFNMADD132SS/VFNMADD213SS/VFNMADD231SS—熔丝负精度浮点值的乘积 - 加法
VFNMSUB132PD/VFNMSUB213PD/VFNMSUB231PD—压缩双精度浮点值的负值乘法减法
VFNMSUB132PS/VFNMSUB213PS/VFNMSUB231PS—封装单精度浮点值的熔丝负负乘法 - 减法
VFNMSUB132SD/VFNMSUB213SD/VFNMSUB231SD—标量双精度浮点值的熔融负乘法 - 减法
VFNMSUB132SS/VFNMSUB213SS/VFNMSUB231SS—标量单精度浮点值的熔点负乘减
VFPCLASSPD—测试打包的Float64值的类型
VFPCLASSPS—测试打包的Float32值的类型
VFPCLASSSD—测试标量Float64值的类型
VFPCLASSSS—测试Scalar Float32值的类型
VGATHERDPD/VGATHERQPD — 使用有符号的双字/ Qword指数收集打包的DP FP值
VGATHERDPS/VGATHERQPS — 使用签名的双字/ Qword索引收集打包的SP FP值
VGATHERDPS/VGATHERDPD—收集包装单，包装双与签署双剑
VGATHERPF0DPS/VGATHERPF0QPS/VGATHERPF0DPD/VGATHERPF0QPD—稀疏预取打包SP / DP数据值与签名双字，签名的Q
VGATHERPF1DPS/VGATHERPF1QPS/VGATHERPF1DPD/VGATHERPF1QPD—稀疏预取打包SP / DP数据值与签名双字，签名的Q
VGATHERQPS/VGATHERQPD—收集包装单，包装双与签名的Qword指数
VPGATHERDD/VPGATHERQD — 使用签名的双字/ Qword指数收集打包的双字值
VPGATHERDD/VPGATHERDQ—收集包装双剑，包装的Qword和双重指数
VPGATHERDQ/VPGATHERQQ — 使用签名的双字/ Qword指数收集打包的Qword值
VPGATHERQD/VPGATHERQQ—收集包装双剑，包装的Qword与已签名的Qword指数
VGETEXPPD—将打包的DP FP值的指数转换为DP FP值
VGETEXPPS—将打包的SP FP值的指数转换为SP FP值
VGETEXPSD—将标量DP FP值的指数转换为DP FP值
VGETEXPSS—将标量SP FP值的指数转换为SP FP值
VGETMANTPD—从Float64 Vector中提取Float64 Vector of Normalized Mantissas
VGETMANTPS—从Float32 Vector中提取Float32向量的Normalized Mantissas

VGETMANTSD—从Float64标量提取Float64的规范化尾数
VGETMANTSS—从Float32向量提取标准化尾数Float32矢量
VINSERTF128/VINSERTF32x4/VINSERTF64x2/VINSERTF32x8/VINSERTF64x4—插入打包的浮点值
VINSERTI128/VINSERTI32x4/VINSERTI64x2/VINSERTI32x8/VINSERTI64x4—插入打包的整数值
VMASKMOV—条件SIMD打包荷载和商店
VPBLENDD — 混合包装双字
VPBLENDMB/VPBLENDMW—使用opmask控制的混合字节/字向量
VPBLENDMD/VPBLENDMQ—使用OpMask控件混合Int32 / Int64向量
VPBROADCASTB/W/D/Q—从通用寄存器加载广播整数数据
VPBROADCAST—加载整数和广播
VPCMPB/VPCMPUB—将打包的字节值比较到掩码中
VPCMPD/VPCMPUD—将打包的整数值与掩码进行比较
VPCMPQ/VPCMPUQ—将打包的整数值与掩码进行比较
VPCMPW/VPCMPUW—比较打包的词值到掩码
VPCOMPRESSD—将稀疏压缩双字整数值存储到密集存储器/寄存器中
VPCOMPRESSQ—将稀疏压缩四字整数值存储到密集存储器/寄存器中
VPCONFLICTD/Q—检测在打包的双字/ Qword值的向量内的冲突到密集存储器/寄存器中
VPERM2F128 — 允许浮点值
VPERM2I128 — 允许整数值
VPERMD/VPERMW—Permute打包双字/字元素
VPERMI2W/D/Q/PS/PD—两个表的完全权限覆盖索引
VPERMILPD—允许双精度浮点值对的内部通道
VPERMILPS—允许四精度浮点值的内部通道
VPERMPD—允许双精度浮点元素
VPERMPS—允许单精度浮点元素
VPERMQ—四字节元素置换
VPEXPANDD—从密集存储器/寄存器加载稀疏压缩双字整数值
VPEXPANDQ—从密集存储器/寄存器加载稀疏压缩四字整数值
VPLZCNTD/Q—计算打包双字，打包的Qword值的前导零位数
VPMASKMOV — 条件SIMD整数打包荷载和商店
VPMOVM2B/VPMOVM2W/VPMOVM2D/VPMOVM2Q—将屏蔽寄存器转换为向量寄存器
VPMOVB2M/VPMOVW2M/VPMOVD2M/VPMOVQ2M—将矢量寄存器转换为掩码
VPMOVQB/VPMOVSQB/VPMOVUSQB—Down将QWord转换为字节
VPMOVQW/VPMOVSQW/VPMOVUSQW—Down将QWord转换为Word
VPMOVQD/VPMOVSQD/VPMOVUSQD—Down将QWord转换为DWord
VPMOVDB/VPMOVSDB/VPMOVUSDB—Down将DWord转换为字节
VPMOVDW/VPMOVSDW/VPMOVUSDW—Down将DWord转换为Word
VPMOVWB/VPMOVSWB/VPMOVUSWB—向下将字转换为字节
PROLD/PROLVD/PROLQ/PROLVQ—位向左旋转
PRORD/PRORVD/PRORQ/PRORVQ—位向右旋转
VPSCATTERDD/VPSCATTERDQ/VPSCATTERQD/VPSCATTERQQ—分散包装双字，带有符号双字的打包字，签名的Qword指数
VPSLLVW/VPSLLVD/VPSLLVQ—可变位移左逻辑
VPSRAVW/VPSRAVD/VPSRAVQ—可变位移右算术
VPSRLVW/VPSRLVD/VPSRLVQ—可变位移右逻辑
VPTERNLOGD/VPTERNLOGQ—按位三进制逻辑
VPTESTMB/VPTESTMW/VPTESTMD/VPTESTMQ—逻辑与和设置掩码
VPTESTNMB/W/D/Q—逻辑NAND和设置
VRANGEPD—对于Float64值的打包对的范围限制计算
VRANGEPS—对于Float32值的打包对的范围限制计算
VRANGESD—范围限制计算从一对Scalar Float64值
VRANGESS—从一对标量Float32值计算范围限制
VRCP14PD—计算压缩Float64值的近似倒数
VRCP14SD—计算标量Float64值的近似倒数
VRCP14PS—计算压缩Float32值的近似倒数
VRCP14SS—计算标量Float32值的近似倒数
VRCP28PD—近似于具有小于2 ^ -28相对误差的封装双精度浮点值的倒数
VRCP28SD—近似于标量双精度浮点值的倒数，小于2 ^ -28相对误差
VRCP28PS—近似于具有小于2 ^ -28相对误差的封装单精度浮点值的倒数
VRCP28SS—近似于标量单精度浮点值的倒数，小于2 ^ -28相对误差
VREDUCEPD—对压缩的Float64值执行压缩转换
VREDUCESD—对标量Float64值执行减少转换
VREDUCEPS—对已压缩的Float32值执行减少转换
VREDUCESS—对标量Float32值执行减少转换
VRNDSCALEPD—圆形压缩Float64值以包括给定数目的分数位
VRNDSCALESD—圆形标量浮点值包括给定数量的比特位
VRNDSCALEPS—圆形压缩Float32值以包括给定数目的分数位
VRNDSCALESS—圆形标量浮点值包括给定数量的比特位
VRSQRT14PD—计算压缩Float64值的平方根的近似倒数
VRSQRT14SD—计算标量Float64值的平方根的近似倒数
VRSQRT14PS—计算压缩Float32值的平方根的近似倒数
VRSQRT14SS—计算标量Float32值的平方根的近似倒数
VRSQRT28PD—近似于包装的双精度浮点值的倒数平方根，小于2 ^ -28相对误差
VRSQRT28SD—近似于标量双精度浮点值的倒数平方根，小于2 ^ -28相对误差

```
VRSQRT28PS—近似于包装的单精度浮点值的倒数平方根，小于2 ^ -28相对误差
VRSQRT28SS—近似于标量单精度浮点值的倒数平方根，小于2 ^ -28相对误差
VSCALEFPD—使用Float64值缩放打包的Float64值
VSCALEFSD—使用Float64值缩放Scalar Float64值
VSCALEFPS—使用Float32值缩放打包的Float32值
VSCALEFSS—使用Float32值缩放标量浮点32值
VSCATTERDPS/VSCATTERDPD/VSCATTERQPS/VSCATTERQPD—分散包装单，包装双与签署的双剑和指数
VSCATTERPF0DPS/VSCATTERPF0QPS/VSCATTERPF0DPD/VSCATTERPF0QPD—稀疏预取打包SP / DP数据值与签名双字，签
VSCATTERPF1DPS/VSCATTERPF1QPS/VSCATTERPF1DPD/VSCATTERPF1QPD—稀疏预取打包SP / DP数据值与签名双字，签
VSHUFF32x4/VSHUFF64x2/VSHUFI32x4/VSHUFI64x2—随机播放128位粒度的打包值
VTESTPD/VTESTPS—压缩位测试
VZEROALL—零所有YMM寄存器
VZEROUPPER—YMM寄存器的零上限位
WAIT/FWAIT—等待
WBINVD—回写并使缓存无效
WRFSBASE/WRGSBASE—写FS / GS段基
WRMSR—写入模型专用寄存器
WRPKRU—将数据写入用户页密钥寄存器
XACQUIRE/XRELEASE — 硬件锁Elision Prefix提示
XABORT — 事务中止
XADD—交换和添加
XBEGIN — 事务开始
XCHG—与寄存器交换寄存器/存储器
XEND — 事务结束
XGETBV—获取扩展控制寄存器的值
XLAT/XLATB—表查找翻译
XOR—逻辑异或
XORPD—压缩双精度浮点值的按位逻辑异或
XORPS—压缩单精度浮点值的按位逻辑异或
XRSTOR—恢复处理器扩展状态
XRSTORS—恢复处理器扩展状态主管
XSAVE—保存处理器扩展状态
XSAVEC—使用压缩保存处理器扩展状态
XSAVEOPT—保存处理器扩展状态优化
XSAVES—保存处理器扩展状态主管
XSETBV—设置扩展控制寄存器
XTEST — 测试如果在事务执行
```

AAA	未组合的十进制加法调整指令 AAA(ASCII Adgust for Addition) 格式: AAA 功能: 对两个组合的十进制数相加运算(存在AL中)的结果进行调整,产生一个未组合的十进制数放在AX中.	说明: 1. 组合的十进制数和未组合的十进制数:在计算中,十进制数可用四位二进制数编码,称为BCD码. 当一个节(8位)中存放一位BCD码,且放在字节的低4位, 高4位为时称为未组合的BCD码. 2. AAA的调整操作 若(AL) and 0FH>9 或 AF=1,则调整如下: (AL)<--(AL)+6,(AH)<--(AH)+1,AF=1,CF<--AF,(AL)<--(AL) and 0FH
AAD	未组合十进制数除法调整指令 AAD(ASCII Adjust for Division) 格式: AAD 功能: 在除法指令前对AX中的两个未组合十进制数进行调整,以便能用DIV指令实现两个未组合的十进制数的除法运算,其结果为未组合的十进制数,商(在AL中)和余数(在AH中).	说明: 1. AAD指令是在执行除法DIV之前使用的,以便得到二进制结果存于AL中,然后除以OPRD,得到的商在AL中,余数在AH中. 2. 示例: MOV BL,5 MOV AX,0308H AAD ;(AL)<--1EH+08H=26H,(AH)<--0 DIV BL ;商＝07H-->(AL),余数＝03H-->(AH).
AAM	未组合十进制数乘法调整指令 AAM(ASCII Adjust MULtiply) 格式: AAM 功能: 对两个未组合的十进制数相乘后存	说明: 1. 实际上是两个未组合的十进制数字节相乘,一个0～9的数与另一个0～9的数相乘其积最大为81.为了得到正确的结果,应进行如下调整:

	于AX中的结果进行调整,产生一个未组合的十进制数存在AL中.	乘积: (AH)<--(AL)/10 (AL)<--(AL)MOD10 2. 本指令应跟在MUL指令后使用,乘积的两位十进制结果,高位放在AH中,低位放在AL中.AH内容是MUL指令的结果被10除的商,即(AL)/10,而最后的AL内容是乘积被10整除的余数(即个位数).
AAS	未组合十进制减法调整指令 AAS(ASCII Adjust for Subtraction) 格式: AAS 功能: 对两个未组合十进制数相减后存于AL中的结果进行调整,调整后产生一个未组合的十进制数数且仍存于AL中.	说明: 1. 本指令影响标志位CF及AF. 2. 调整操作 若(AL) and 0FH > 9 或 AF=1 则(AL)<--(AL)-6,(AH)<--(AH)-1,CF<--AF,(AL)<--(AL) and 0FH, 否则(AL)<--(AL) and 0FH
ADC	带进位加法指令 ADC(Addition Carry) 格式: ADC OPRD1,OPRD2 功能: OPRD1<--OPRD1 + OPRD2 + CF	说明: 1. OPRD1为任一通用寄存器或存储器操作数,可以是任意一个通用寄存器,而且还可以是任意一个存储器操作数. OPRD2为立即数,也可以是任意一个通用寄存器操作数.立即数只能用于源操作数. 2. OPRD1和OPRD2均为寄存器是允许的,一个为寄存器而另一个为存储器也是允许的,但不允许两个都是存储器操作数. 3. 加法指令运算的结果对CF、SF、OF、PF、ZF、AF都会有影响.以上标志也称为结果标志. 4. 该指令对标志位的影响同ADD指令.
ADD	加法指令 ADD(Addition) 格式: ADD OPRD1,OPRD2 功能: 两数相加	说明: 1. OPRD1为任一通用寄存器或存储器操作数,可以是任意一个通用寄存器,而且还可以是任意一个存储器操作数. OPRD2为立即数,也可以是任意一个通用寄存器操作数.立即数只能用于源操作数. 2. OPRD1和OPRD2均为寄存器是允许的,一个为寄存器而另一个为存储器也是允许的, 但不允许两个都是存储器操作数. 3. 加法指令运算的结果对CF、SF、OF、PF、ZF、AF都会有影响.以上标志也称为结果标志.加法指令适用于无符号数或有符号数的加法运算.
AND	逻辑与运算指令 AND 格式: AND OPRD1,OPRD2 功能: 对两个操作数实现按位逻辑与运算,结果送至目的操作数.本指令可以进行字节或字的'与'运算, OPRD1<--OPRD1 and OPRD2.	说明: 1. 目的操作数OPRD1为任一通用寄存器或存储器操作数.源操作数OPRD2为立即数,任一通用寄存器或存储器操作数. 2. 示例: AND AL,0FH ;(AL)<--(AL) AND 0FH AND AX,BX ;(AX)<--(AX) AND (BX) AND DX,BUFFER[SI+BX] AND BETA[BX],00FFH 注意: 两数相与，有一个数假则值为假
CALL	过程调用指令 CALL 格式: CALL OPRD 功能: 过程调用指令	说明: 1. 其中OPRD为过程的目的地址. 2. 过程调用可分为段内调用和段间调用两种.寻址方式也可以分为直接寻址和间接寻址两种. 3. 本指令不影响标志位.

CBW	字节扩展指令 CBW(Convert Byte to Word) 格式: CBW 功能: 将字节扩展为字,即把AL寄存器的符号位扩展到AH中.	说明: 1. 两个字节相除时,先使用本指令形成一个双字节长的被除数. 2. 本指令不影响标志位. 3. 示例: MOV AL,25 CBW IDIV BYTE PTR DATA1
CLC	处理器控制指令－标志位操作指令 格式: CLC ;置CF=0 STC ;置CF=1 CMC ;置CF=(Not CF)进位标志求反 CLD ;置DF=0 STD ;置DF＝1 CLI ;置IF=0，CPU禁止响应外部中断 STI ;置IF=1，使CPU允许向应外部中断 功能: 完成对标志位的置位、复位等操作.	说明: 例如串操作中的程序,经常用CLD指令清方向标志使DF＝0,在串操作指令执行时,按增量的方式修改吕指针.
CLD	处理器控制指令－标志位操作指令 格式: CLC ;置CF=0 STC ;置CF=1 CMC ;置CF=(Not CF)进位标志求反 CLD ;置DF=0 STD ;置DF＝1 CLI ;置IF=0，CPU禁止响应外部中断 STI ;置IF=1，使CPU允许向应外部中断 功能: 完成对标志位的置位、复位等操作.	说明: 例如串操作中的程序,经常用CLD指令清方向标志使DF＝0,在串操作指令执行时,按增量的方式修改吕指针.
CLI	处理器控制指令－标志位操作指令 格式: CLC ;置CF=0 STC ;置CF=1 CMC ;置CF=(Not CF)进位标志求反 CLD ;置DF=0 STD ;置DF＝1 CLI ;置IF=0，CPU禁止响应外部中断 STI ;置IF=1，使CPU允许向应外部中断 功能: 完成对标志位的置位、复位等操作.	说明: 例如串操作中的程序,经常用CLD指令清方向标志使DF＝0,在串操作指令执行时,按增量的方式修改吕指针.
CMC	处理器控制指令－标志位操作指令 格式: CLC ;置CF=0 STC ;置CF=1 CMC ;置CF=(Not CF)进位标志求反 CLD ;置DF=0 STD ;置DF＝1 CLI ;置IF=0，CPU禁止响应外部中断 STI ;置IF=1，使CPU允许向应外部中断 功能: 完成对标志位的置位、复位等操作.	说明: 例如串操作中的程序,经常用CLD指令清方向标志使DF＝0,在串操作指令执行时,按增量的方式修改吕指针.
CMP	比效指令 CMP(CoMPare) 格式: CMP OPRD1,OPRD2 功能: 对两数进行相减,进行比较.	说明: 1. OPRD1为任意通用寄存器或存储器操作数. OPRD2为任意通用寄存器或存储器操作数,立即数也可用作源操作数OPRD2.

		2. 对标志位的影响同SUB指令,完成的操作与SUB指令类似,唯一的区别是不将OPRD1-OPRD2的结果送回OPRD1,而只是比较. 3. 在8088/8086指令系统中,专门提供了一组根据带符号数比较大小后,实现条件转移的指令.
CMPS	字符串比较指令 格式: CMPS OPRD1,OPRD2 CMPSB CMPSW 功能: 由SI寻址的源串中数据与由DI寻址的目的串中数据进行比较,比较结果送标志位,而不改变操作数本身. 同时SI,DI将自动调整.	说明: 1. 其中OPRD2为源串符号地址,OPRD1为目的串符号地址. 2. 本指令影响标志位AF、CF、OF、SF、PF、ZF.本指令可用来检查二个字符串是否相同,可以使用循环控制方法对整串进行比较. 3. 与MOVS相似,CMPS指令也可以不使用操作数,此时可用指令CMPSB或CMPSW分别表示字节串比较或字串比较.
CMPSB	字符串比较指令 格式: CMPS OPRD1,OPRD2 CMPSB CMPSW 功能: 由SI寻址的源串中数据与由DI寻址的目的串中数据进行比较,比较结果送标志位,而不改变操作数本身. 同时SI,DI将自动调整.	说明: 1. 其中OPRD2为源串符号地址,OPRD1为目的串符号地址. 2. 本指令影响标志位AF、CF、OF、SF、PF、ZF.本指令可用来检查二个字符串是否相同,可以使用循环控制方法对整串进行比较. 3. 与MOVS相似,CMPS指令也可以不使用操作数,此时可用指令CMPSB或CMPSW分别表示字节串比较或字串比较.
CMPSW	字符串比较指令 格式: CMPS OPRD1,OPRD2 CMPSB CMPSW 功能: 由SI寻址的源串中数据与由DI寻址的目的串中数据进行比较,比较结果送标志位,而不改变操作数本身. 同时SI,DI将自动调整.	说明: 1. 其中OPRD2为源串符号地址,OPRD1为目的串符号地址. 2. 本指令影响标志位AF、CF、OF、SF、PF、ZF.本指令可用来检查二个字符串是否相同,可以使用循环控制方法对整串进行比较. 3. 与MOVS相似,CMPS指令也可以不使用操作数,此时可用指令CMPSB或CMPSW分别表示字节串比较或字串比较.
CWD	字扩展指令 CWD(Convert Word to Double Word) 格式: CWD 功能: 将字扩展为双字长,即把AX寄存器的符号位扩展到DX中.	说明: 1. 两个字或字节相除时,先用本指令形成一个双字长的的被除数. 2. 本指令不影响标志位. 3. 示例: 在B1、B2、B3字节类型变量中,分别存有8们带符号数a、b、c,实现(a*b+c)/a运算。
DAA	组合的十进制加法调整指令 DAA(Decimal Adjust for Addition) 格式: DAA 功能: 对AL中的两个组合进制数相加的结果进行调整,调整结果仍放在AL中,进位标志放在CF中.	说明: 1. 调整操作如下 (1) 若(AL) and 0FH>9 或 AF=1,则(AL)<--(AL)+6,AF<--1,对低四位的调整. (2) 若(AL) and 0F0H>90H 或 CF=1,则(AL)<--(AL)+60H,CF<--1. 2. 示例: (AL)=18H,(BL)=06H ADD AL,BL ; (AL)<--(AL)+(BL) ; (AL)=1EH DAA ; (AL)
DAS	组合十进制减法调整指令 DAS(Decimal Adjust for Subtraction) 格式: DAS 功能: 对两个组合十进制数相减后存于	说明: 调整操作 若(AL) and 0FH > 9 或 AF=1,则(AL)<--(AL)-6,AF=1

	AL中的结果进行调整,调整后产生一个组合的十进制数且仍存于AL中.	若(AL) and 0F0H > 90H 或 CF=1,则(AL)<--(AL)-60,CF=1
DEC	减一指令 DEC(Decrement by 1) 格式: DEC OPRD 功能: OPRD<--OPRD-1	说明: 1. OPRD 为寄存器或存储器操作数. 2. 这条指令执行结果影响AF、OF、PF、SF、ZF标志位，但不影响CF标志位. 3. 示例 DEC AX DEC CL DEC WORD PTR[DI] DEC ALFA[DI+BX]
DIV	无符号数除法指令 DIV(DIVision) 格式: DIV OPRD 功能: 实现两个无符号二进制数除法运算.	说明: 1. 其中OPRD为任一个通用寄存器或存储器操作数. 2. 字节相除,被除数在AX中;字相除,被除数在DX,AX中,除数在OPRD中. 字节除法: (AL)<--(AX)/OPRD,(AH)<--(AX)MOD OPRD 字除法: (AX)<--(DX)(AX)/OPRD,(DX)<--(DX)(AX) MOD OPRD
ESC	处理器交权指令 ESC 格式: ESC EXTOPRD,OPRD 功能: 使用本指令可以实现协处理器出放在ESC指令代码中的6位常数,该常数指明协处理器要完成的功能. 当源操作数为存储器变量时,则取出该存储器操作数传送给协处理器.	说明: 1. 其中EXTOPRD为外部操作码,OPRD为源操作数. 2. 本指不影响标志位.
HLT	处理器暂停指令 HLT 格式: HLT 功能: 使处理器处于暂时停机状态.	说明: 1. 本指令不影响标志位. 2. 由执行HLT引起的暂停,只有RESET(复位)、NMI(非屏蔽中断请求)、INTR(可屏蔽的外部中断请求)信号可以使 其退出暂停状态.它可用于等待中断的到来或多机系统的同步操作.
IDIV	带符号数除法指定 IDIV(Interger DIVision) 格式: IDIV OPRD 功能: 这实现两个带符号数的二进制除法运算.	说明: 1. 其中OPRD为任一通用寄存器或存储器操作数. 2. 理由与IMUL相同,只有IDIV指令,才能得到符号数相除的正确结果. 3. 当被除数为8位,在进行字节除法前,应把AL的符号位扩充至AH中.在16位除法时,若被除数为16位,则应将AX中的符号位扩到DX中.
IMUL	带符号数乘法指令 IMUL(Integer MULtiply) 格式: IMUL OPRD 功能: 完成两个带符号数的相乘	说明: 1. 其中OPRD为任一通用寄存器或存储器操作数. 2. MUL指令对带符号相乘时,不能得到正确的结果. 例如: (AL)=255 (CL)=255 MUL CL (AX)=65025 注意: 这对无符号数讲,结果是正确的,但对带符号数讲,相当于(-1)*(-1)结果应为+1,而65025对应的带符号数为-511,显然是不正确的.

IN	输入指令 IN 格式: IN AL,n ;(AL)<--(n) IN AX,n ;(AX)<--(n+1),(n) IN AL,DX ;(AL)<--[(DX)] IN AX,DX ;(AX)<--[(DX)+1],[(DX)] 功能: 输入指令	说明: 1. 其中n为8位的端口地址,当字节输入时,将端口地址n+1的内容送至AH中,端口地址n的内容送AL中. 2. 端口地址也可以是16位的,但必须将16位的端口地址送入DX中.当字节寻址时,由DX内容作端口地址的内容送至AL中; 当输入数据字时,[(DX)+1]送AH,[(DX)]送AL中,用符号:(AX)<--[(DX)+1],[(DX)]表示.
INC	加1指令 INC(INCrement by 1) 格式: INC OPRD 功能: OPRD<--OPRD+1	说明: 1. OPRD 为寄存器或存储器操作数. 2. 这条指令执行结果影响AF、OF、PF、SF、ZF标志位,但不影响CF标志位. 3. 示例: INC SI;(SI)<--(SI)+1 INC WORD PTR[BX] INC BYTE PTR[BX+DI] INC CL;(CL)<--(CL)+1 注意: 上述第二,三两条指令,是对存储字及存储字节的内容加1以替代原来的内容.
INT	软中断指令 INT 格式: INT n 其中n为软中断的类型号. 功能: 本指令将产生一个软中断,把控制转向一个类型号为n的软中断,该中断处理程序入口地址在中断向量表的n*4地址处的二个存储器字(4个单元)中.	说明: 操作过程与INTO指令雷同,只需将10H改为n*4即可.所以,本指令也将影响标志位IF及TF.
INTO	溢出中断指令 INTO(INTerrupt if Overflow) 格式: INTO 功能: 本指令检测OF标志位,当OF=1时,说明已发生溢出,立即产生一个中断类型4的中断,当OF＝0时,本指令不起作用.	说明: 1. 本指令影响标志位IF及TF. 2. 本指令可用于溢出处理,当OF=1时,产生一个类型4的软中断.在中断处理程序中完成溢出的处理操作.
IRET	中断返回指令 IRET 格式: IRET 功能: 用于中断处理程序中,从中断程序的断点处返回,继续执行原程序.	说明: 1. 本指令将影响所有标志位. 2. 无论是软中断,还是硬中断,本指令均可使其返回到中断程序的断点处继续执行原程序.
JA	条件转移指令JA/JNBE 格式: JA/JNBE标号 功能: 为高于/不低于等于的转移指令	说明: 1. 例如两个符号数a,b比较时,a>b(即CF=0,ZF=0)时转移.因为单一标志位CF=0,只表示a>=b. 2. JA/JNBE是同一条指令的两种不同的助记符. 3. 该指令用于无符号数进行条件转移
JAE	条件转移指令JAE/JNB 格式: JAE/JNB 标号 功能: 为高于等于/不低于的转移指令	说明: 1. JAE/JNB是同一条指令的两种不同的助记符. 2. 该指令用于无符号数进行条件转移.
JB	条件转移指令JB/JNAE 格式: JB/JNAE 标号 功能: 低于/不高于等于时转移	说明: 该指令用于无符号数的条件转移

JBE	条件转移指令JBE/JNA 格式: JBE/JNA 标号 功能: 低于等于/不高于时转移	说明: 该指令用于无符号数的条件转移
JC	条件转移指令 JC 格式: JC 标号 功能: CF＝1,转至标号处执行	说明: JC为根据标志位CF进行转移的指令
JE	条件转移指令JE/JZ 格式: JE/JZ标号 功能: ZF＝1,转至标号处执	说明: 1. 指令JE与JZ等价,它们是根据标志位ZF进行转移的指令 2. JE,JZ均为一条指令的两种助记符表示方法
JG	条件转移指令JG/JNLE 格式: JG/JNLE 标号 功能: 大于/不小于等于时转移	说明: 用于带符号数的条件转移指令
JGE	条件转移指令JGE/JNL 格式: JGE/JNL标号 功能: 大于等于/不小于时转移	说明: 用于带符号数的条件转移指令
JL	条件转移指令JL/JNGE 格式: JL/JNGE标号 功能: 小于/不大于等于时转移	说明: 用于带符号数的条件转移指令
JLE	条件转移指令JLE/JNG 格式: JLE/JNG 标号 功能: 小于等于/不大于时转移	说明: 用于带符号数的条件转移指令
JMP	无条件转移指令JMP 格式: JMP OPRD 功能: JMP指令将无条件地控制程序转移到目的地址去执行.当目的地址仍在同一个代码段内,称为段内转移;当目标地址不在同一个代码段内,则称为段间转移.这两种情况都将产生不同的指令代码,以便能正确地生成目的地址,在段内转移时,指令只要能提供目的地址的段内偏移量即够了;而在段间转移时,指令应能提供目的地址的段地址及段内偏移地址值.	说明: 1. 其中OPRD为转移的目的地址.程序转移到目的地址所指向的指令继续往下执行. 2. 本组指令对标志位无影响. 3. <1> 段内直接转移指令: JMP NEAR 标号 <2> 段内间接转移指令: JMP OPRD <3> 段间直接转移指令: JMP FAR 标号 <4> 段间间接转移指令:JMP OPRD其中的OPRD为存储器双字操作数.段间间接转移只能通过存储器操作数来实现.
JNA	条件转移指令JBE/JNA 格式: JBE/JNA 标号 功能: 低于等于/不高于时转移	说明: 该指令用于无符号数的条件转移
JNAE	条件转移指令JB/JNAE 格式: JB/JNAE 标号 功能: 低于/不高于等于时转移	说明: 该指令用于无符号数的条件转移
JNB	条件转移指令JAE/JNB 格式: JAE/JNB 标号 功能: 为高于等于/不低于的转移指令	说明: 1. JAE/JNB是同一条指令的两种不同的助记符. 2. 该指令用于无符号数进行条件转移.
JNBE	条件转移指令JA/JNBE 格式: JA/JNBE标号 功能: 为高于/不低于等于的转移指令	说明: 1. 例如两个符号数a,b比较时,a>b(即CF=0,ZF=0)时转移.因为单一标志位CF=0,只

		表示a>=b. 2. JA/JNBE是同一条指令的两种不同的助记符. 3. 该指令用于无符号数进行条件转移
JNC	条件转移指令JNC 格式: JNC标号 功能: CF＝0,转至标号处执行	说明: JNC为根据标志位CF进行转移的指令
JNE	条件转移指令JNE/JNZ 格式: JNE/JNZ 标号 功能: ZF＝0,转至标号处执行	说明: 1. 指令JNE与JNZ等价,它们是根据标志位ZF进行转移的指令 2. JNE,JNZ均为一条指令的两种助记符表示方法
JNG	条件转移指令JLE/JNG 格式: JLE/JNG 标号 功能: 小于等于/不大于时转移	说明: 用于带符号数的条件转移指令
JNGE	条件转移指令JL/JNGE 格式: JL/JNGE标号 功能: 小于/不大于等于时转移	说明: 用于带符号数的条件转移指令
JNL	条件转移指令JGE/JNL 格式: JGE/JNL标号 功能: 大于等于/不小于时转移	说明: 用于带符号数的条件转移指令
JNLE	条件转移指令JG/JNLE 格式: JG/JNLE 标号 功能: 大于/不小于等于时转移	说明: 用于带符号数的条件转移指令
JNO	条件转移指令JNO 格式: JNO 标号 功能: OF＝0,转至标号处执行	说明: JNO是根椐溢出标志位OF进行转移的指令
JNP	条件转移指令JNP/JPO 格式: JNP/JPO 标号 功能: PF＝0,转至标号处执行	说明: 1. 指令JNP与JPO,它们是根据奇偶标志位PF进行转移的指令 2. JNP,JPO均为一条指令的两种助记符表示方法
JNS	条件转移指令JNS 格式: JNS 标号 功能: SF＝0,转至标号处执行	说明: JNS是根据符号标志位SF进行转移的指令
JNZ	条件转移指令JNE/JNZ 格式: JNE/JNZ 标号 功能: ZF＝0,转至标号处执行	说明: 1. 指令JNE与JNZ等价,它们是根据标志位ZF进行转移的指令 2. JNE,JNZ均为一条指令的两种助记符表示方法
JO	条件转移指令JO 格式: JO 标号 功能: OF＝1,转至标号处执行	说明: JO是根椐溢出标志位OF进行转移的指令
JP	条件转移指令JP/JPE	说明:

	格式: JP/JPE 标号 功能: PF＝1,转至标号处执行	1. 指令JP与JPE,它们是根据奇偶标志位PF进行转移的指令 2. JP,JPE均为一条指令的两种助记符表示方法
JPE	条件转移指令JP/JPE 格式: JP/JPE 标号 功能: PF＝1,转至标号处执行	说明: 1. 指令JP与JPE,它们是根据奇偶标志位PF进行转移的指令 2. JP,JPE均为一条指令的两种助记符表示方法
JPO	条件转移指令JNP/JPO 格式: JNP/JPO 标号 功能: PF＝0,转至标号处执行	说明: 1. 指令JNP与JPO,它们是根据奇偶标志位PF进行转移的指令 2. JNP,JPO均为一条指令的两种助记符表示方法
JS	条件转移指令JS 格式: JS 标号 功能: SF＝1,转至标号处执行	说明: JS是根据符号标志位SF进行转移的指令
JZ	条件转移指令JE/JZ 格式: JE/JZ标号 功能: ZF＝1,转至标号处执	说明: 1. 指令JE与JZ等价,它们是根据标志位ZF进行转移的指令 2. JE,JZ均为一条指令的两种助记符表示方法
LAHF	标志传送指令 LAHF 格式: LAHF 功能: 取FLAG标志寄存器低8位至AH寄存器.(AH)<--(FLAG)7~0	说明: 该指令不影响FLAG的原来内容,AH只是复制了原FLAG的低8位内容.

LDS	从存储器取出32位地址的指令 LDS
	格式: LDS OPRD1,OPRD2
	功能: 从存储器取出32位地址的指令.
		说明:
		OPRD1 为任意一个16位的寄存器.
		OPRD2 为32位的存储器地址.
		示例: LDS SI,ABCD
		LDS BX,FAST[SI]
		LDS DI,[BX]
		注意: 上面LDS DI,[BX]指令的功能是把BX所指的32位地址指针的段地址送入DS,偏移地址送入DI.

LEA	有效地址传送指令 LEA 格式: LEA OPRD1,OPRD2 功能: 将源操作数给出的有效地址传送到指定的的寄存器中.	说明: 1. OPRD1 为目的操作数,可为任意一个16位的通用寄存器. OPRD2 为源操作数,可为变量名、标号或地址表达式. 示例: LEA BX,DATA1 LEA DX,BETA[BX+SI] LEA BX BX,[BP],[DI] 2. 本指令对标志位无影响。
LES	从存储器取出32位地址的指令 LES 格式: LES OPRD1,OPRD2 功能: 从存储器取出32位地址的指令.	说明: OPRD1 为任意一个16位的寄存器. OPRD2 为32位的存储器地址. 示例: LES SI,ABCD LES BX,FAST[SI] LES DI,[BX] 注意: 上面LES DI,[BX]指令的功能是把BX所

		指的32位地址指针的段地址送入ES,偏移地址送入DI.
LOCK	封锁总线指令 LOCK 格式: LOCK 功能: 指令是一个前缀,可放在指令的前面,告诉CPU在执行该指令时,不允许其它设备对总线进行访问.	无可用信息!用户可自行添加!
LODS	取字符串元素指令 LODS 格式: LODS OPRD 其中OPRD为源字符串符号地址. 功能: 把SI寻址的源串的数据字节送AL或数据字送AX中去, 并根据DF的值修改地址指针SI进行自动调整.	说明: 1. 本指令不影响标志位. 2. 当不使用操作数时,可用LODS(字节串)或LODSW(字串)指令.
LOOP	循环控制指令LOOP 格式: LOOP 标号 功能: (CX)<--(CX)-1,(CX)<>0,则转移至标号处循环执行, 直至(CX)=0,继续执行后继指令.	说明: 1. 本指令是用CX寄存器作为计数器,来控制程序的循环. 2. 它属于段内SHORT短类型转移,目的地址必须距本指令在-128到+127个字节的范围内.
LOOPE	循环控制指令LOOPZ/LOOPE 格式: LOOPZ/LOOPE 标号 功能: (CX)<--(CX)-1,(CX)<>0 且ZF＝1时,转至标号处循环	说明: 1. 本指令是用CX寄存器作为计数器,来控制程序的循环. 2. 它属于段内SHORT短类型转移,目的地址必须距本指令在-128到+127个字节的范围内. 3. 以上两种助记符等价.
LOOPNE	循环控制指令LOOPNZ/LOOPNE 格式: LOOPNZ/LOOPNE 标号 功能: (CX)<--(CX)-1,(CX)<>0 且ZF＝0时,转至标号处循环	说明: 1. 本指令是用CX寄存器作为计数器,来控制程序的循环. 2. 它属于段内SHORT短类型转移,目的地址必须距本指令在-128到+127个字节的范围内. 3. 以上两种助记符等价.
LOOPNZ	循环控制指令LOOPNZ/LOOPNE 格式: LOOPNZ/LOOPNE 标号 功能: (CX)<--(CX)-1,(CX)<>0 且ZF＝0时,转至标号处循环	说明: 1. 本指令是用CX寄存器作为计数器,来控制程序的循环. 2. 它属于段内SHORT短类型转移,目的地址必须距本指令在-128到+127个字节的范围内. 3. 以上两种助记符等价.
LOOPZ	循环控制指令LOOPZ/LOOPE 格式: LOOPZ/LOOPE 标号 功能: (CX)<--(CX)-1,(CX)<>0 且ZF＝1时,转至标号处循环	说明: 1. 本指令是用CX寄存器作为计数器,来控制程序的循环. 2. 它属于段内SHORT短类型转移,目的地址必须距本指令在-128到+127个字节的范围内. 3. 以上两种助记符等价.
MOVE	数据传送指令 MOV 格式: MOV OPRD1,OPRD2 功能: 本指令将一个源操作数送到目的操作数中,即OPRD1<--OPRD2.	说明: 1. OPRD1 为目的操作数,可以是寄存器、存储器、累加器. OPRD2 为源操作数,可以是寄存器、存储器、累加器和立即数. 2. MOV 指令以分为以下四种情况: <1> 寄存器与寄存器之间的数据传送指令 <2> 立即数到通用寄存器数据传送指令 <3> 寄存器与存储器之间的数据传送指令

		<4> 立即数到存储器的数据传送 3. 本指令不影响状态标志位
MOVS	字符串传送指令 MOVS 格式: MOVS OPRD1,OPRD2 MOVSB MOVSW 功能: OPRD1<--OPRD2.	说明: 1. 其中OPRD2为源串符号地址,OPRD1为目的串符号地址. 2. 字节串操作: 若DF=0,则作加, 若DF=1,则作减. 3. 对字串操作时: 若DF=0,则作加,若DF=1,则作减,. 4. 在指令中不出现操作数时,字节串传送格式为MOVSB、字串传送格式为MOVSW. 5. 本指令不影响标志位.
MOVSB	字符串传送指令 MOVS 格式: MOVS OPRD1,OPRD2 MOVSB MOVSW 功能: OPRD1<--OPRD2.	说明: 1. 其中OPRD2为源串符号地址,OPRD1为目的串符号地址. 2. 字节串操作: 若DF=0,则作加, 若DF=1,则作减. 3. 对字串操作时: 若DF=0,则作加,若DF=1,则作减,. 4. 在指令中不出现操作数时,字节串传送格式为MOVSB、字串传送格式为MOVSW. 5. 本指令不影响标志位.
MOVSW	字符串传送指令 MOVS 格式: MOVS OPRD1,OPRD2 MOVSB MOVSW 功能: OPRD1<--OPRD2.	说明: 1. 其中OPRD2为源串符号地址,OPRD1为目的串符号地址. 2. 字节串操作: 若DF=0,则作加, 若DF=1,则作减. 3. 对字串操作时: 若DF=0,则作加,若DF=1,则作减,. 4. 在指令中不出现操作数时,字节串传送格式为MOVSB、字串传送格式为MOVSW. 5. 本指令不影响标志位.
MUL	无符号数乘法指令 MUL(MULtiply) 格式: MUL OPRD 功能: 乘法操作.	说明: 1. OPRD为通用寄存器或存储器操作数. 2. OPRD为源操作数,即作乘数.目的操作数是隐含的,即被乘数总是指定为累加器AX或AL的内容. 3. 16位乘法时,AX中为被乘数.8位乘法时,AL为被乘数.当16位乘法时,32位的乘积存于DX及AX中;8位乘法的16位乘积存于AX中. 4. 操作过程: 字节相乘:(AX)<--(AL)*OPRD,当结果的高位字节(AH)不等于0时,则CF＝1、OF＝1.
NEG	取补指令 NEG(NEGate) 格式: NEG OPRD 功能: 对操作数OPRD进行取补操作,然后将结果送回OPRD.取补操作也叫作求补操作,就是求一个数的相反数的补码.	说明: 1. OPRD为任意通用寄存器或存储器操作数. 2. 示例: (AL)=44H,取补后,(AL)=0BCH(-44H). 3. 本指令影响标志位CF、OF、SF、PF、ZF及AF.
NOP	空操作指令 NOP 格式: NOP 功能: 本指令不产生任何结果,仅消耗几个时钟周期的时间,接着执行后续指令,常用于程序的延时等.	说明: 本指令不影响标志位.

NOT	逻辑非运算指令 NOT 格式: NOT OPRD 功能: 完成对操作数按位求反运算(即0变1,1变0),结果关回原操作数.	说明: 1. 其中OPRD可为任一通用寄存器或存储器操作数. 2. 本指梳令可以进行字或字节'非'运算. 3. 本指令不影响标志位.
OR	逻辑或指令 OR 格式: OR OPRD1,OPRD2 功能: OR指令完成对两个操作数按位的'或'运算,结果送至目的操作数中,本指令可以进行字节或字的'或'运算. OPRD1<--OPRD1 OR OPRD2.	说明: 1. 其中OPRD1,OPRD2含义与AND指令相同,对标志位的影响也与AND指令相同. 2. 两数相或,有一个数为真则值为真.
OUT	输出指令 OUT 格式: OUT n,AL ;(n)<--(AL) 功能: 输出指令	说明: 1. OUT n,AX ;(n+1),(n)<--(AX) OUT DX,AL ;[(DX)]<--(AL) OUT DX,AX ;[(DX)+1],[(DX)]<--(AX) 2. 输入指令及输出指令对标志位都不影响.
POP	堆栈操作指令 PUSH和POP 格式: PUSH OPRD POP OPRD 功能: 实现压入操作的指令是PUSH指令;实现弹出操作的指令是POP指令.	说明: 1. OPRD为16位(字)操作数,可以是寄存器或存储器操作数. 2. POP指令的操作过程是: POP OPRD:OPRD<--((SP)),(SP)<--(SP)+2 它与压入操作相反,是先弹出栈顶的数顶,然后再修改指针SP的内容. 3. 示例: POP AX POP DS POP DATA1 POP ALFA[BX][DI] 4. PUSH和POP指令对状态标志位没有影响.
POPF	标志传送指令 POPF 格式: POPF 功能: 本指令的功能与PUSHF相反,在子程序调用和中断服务程序中,往往用PUSHF指令保护FLAG的内容,用POPF指令将保护的FLAG内容恢复.	说明: 如果对堆栈中的原FLAG内容进行修改,如对TF等标志位进行修改,然后再弹回标志位寄存器FLAG.这是通过指令修改TF标志的唯一方法.
PUSH	堆栈操作指令 PUSH和POP 格式: PUSH OPRD POP OPRD 功能: 实现压入操作的指令是PUSH指令;实现弹出操作的指令是POP指令.	说明: 1. OPRD为16位(字)操作数,可以是寄存器或存储器操作数. 2. PUSH的操作过程是: (SP)<--(SP)-2,((sp))<--OPRD 即先修改堆栈指针SP(压入时为自动减2),然后,将指定的操作数送入新的栈顶位置. 此处的((SP))<--OPRD,也可以理解为:[(SS)*16+(SP)]<--OPRD 或 [SS:SP]<--OPRD
PUSHF	标志传送指令 PUSHF 格式: PUSHF 功能: 本指令可以把标志寄存器的内容保存到堆栈中去	
RCL	循环移位指令 格式: ROL OPRD1,COUNT ;不含进位标志位CF在循环中的左循环移位指令. ROR OPRD1,COUNT ;不含进位示志位CF在循环中的右循环移位指令.	说明: 1. 本指令组只影响标志CF、OF.OF由移入CF的内容决定,OF取决于移位一次后符号位是否改变,如改变,则OF=1. 2. 由于是循环移位,所以对字节移位8次; 对字

	RCL OPRD1,COUNT ;带进位的左循环移位指令. RCR OPRD1,COUNT ;带进位的右循环移位指令.	移位16次,就可恢复为原操作数.由于带CF的循环移位,可以将CF的内容移入, 所以可以利用它实现多字节的循环.
RCR	循环移位指令 格式: ROL OPRD1,COUNT ;不含进位标志位CF在循环中的左循环移位指令. ROR OPRD1,COUNT ;不含进位示志位CF在循环中的右循环移位指令. RCL OPRD1,COUNT ;带进位的左循环移位指令. RCR OPRD1,COUNT ;带进位的右循环移位指令.	说明: 1. 本指令组只影响标志CF、OF.OF由移入CF的内容决定,OF取决于移位一次后符号位是否改变,如改变,则OF=1. 2. 由于是循环移位,所以对字节移位8次; 对字移位16次,就可恢复为原操作数.由于带CF的循环移位,可以将CF的内容移入,所以可以利用它实现多字节的循环. 注意: 以上程序中的指令SHR AL,CL如改为SAR AL,CL,虽然最高4位可移入低4位,但最高位不为0,故应加入一条指令AND AL,0FH.否则,若最高位不为0时,将得到错误结果.
REP	重复前缀的说明 格式: REP ;CX<>0 重复执行字符串指令 REPZ/REPE ;CX<>0 且ZF＝1重复执行字符串指令 REPNZ/REPNE ;CX<>0 且ZF＝0重复执行字符串指令 功能: 在串操作指令前加上重复前缀,可以对字符串进重复处理.由于加上重复前缀后,对应的指令代码是不同的,所以指令的功能便具有重复处理的功能,重复的次数存放在CX寄存器中.	说明: 1. REP与MOVS或STOS串操作指令相结合使用,完成一组字符的传送或建立一组相同数据的字符串. 2. REPZ/REPE常用与CMPS串操作指令结合使用, 可以完成两组字符串的比较. 3. REPZ/REPE常与SCAS指令结合使用,可以完成在一个字符串中搜索一个关键字. 4. REPNZ/REPNE与CMPS指令结合使用,表示当串未结束(CX=1)且当对应串元素不相同(ZF=0)时,继续重复执行串比较指令.
REPE	重复前缀的说明 格式: REP ;CX<>0 重复执行字符串指令 REPZ/REPE ;CX<>0 且ZF＝1重复执行字符串指令 REPNZ/REPNE ;CX<>0 且ZF＝0重复执行字符串指令 功能: 在串操作指令前加上重复前缀,可以对字符串进重复处理.由于加上重复前缀后,对应的指令代码是不同的,所以指令的功能便具有重复处理的功能,重复的次数存放在CX寄存器中.	说明: 1. REPZ/REPE常用与CMPS串操作指令结合使用, 可以完成两组字符串的比较. 2. REPZ/REPE常与SCAS指令结合使用,可以完成在一个字符串中搜索一个关键字. 3. REPNZ/REPNE与CMPS指令结合使用,表示当串未结束(CX=1)且当对应串元素不相同(ZF=0)时,继续重复执行串比较指令. 4. REPNZ/REPNE与SCAS指令结合使用,表示串未结束(CX=1)且当关键字与串元素不相同(ZF=0)时,继续重复执行串搜索指令.
REPNE	重复前缀的说明 格式: REP ;CX<>0 重复执行字符串指令 REPZ/REPE ;CX<>0 且ZF＝1重复执行字符串指令 REPNZ/REPNE ;CX<>0 且ZF＝0重复执行字符串指令	说明: 1. REPZ/REPE常用与CMPS串操作指令结合使用, 可以完成两组字符串的比较. 2. REPZ/REPE常与SCAS指令结合使用,可以完成在一个字符串中搜索一个关键字. 3. REPNZ/REPNE与CMPS指令结合使用,表示当串未结束(CX=1)且当对应串元素不相同(ZF=0)时,继续重复执行串比较指令. 4. REPNZ/REPNE与SCAS指令结合使用,表示串未结束(CX=1)且当关键字与串元素不相同(ZF=0)时,继续重复执行串搜索指令.
REPNZ	重复前缀的说明 格式: REP ;CX<>0 重复执行字符串指令 REPZ/REPE ;CX<>0 且ZF＝1重复执行字符串指令 REPNZ/REPNE ;CX<>0 且ZF＝0重复执行字符串指令	说明: 1. REPZ/REPE常用与CMPS串操作指令结合使用, 可以完成两组字符串的比较. 2. REPZ/REPE常与SCAS指令结合使用,可以完成在一个字符串中搜索一个关键字. 3. REPNZ/REPNE与CMPS指令结合使用,表示当串未结束(CX=1)且当对应串元素不相同

		(ZF=0)时,继续重复执行串比较指令. 4. REPNZ/REPNE与SCAS指令结合使用,表示串未结束(CX=1)且当关键字与串元素不相同(ZF=0)时,继续重复执行串搜索指令.
REPZ	重复前缀的说明 格式: REP ;CX<>0 重复执行字符串指令 REPZ/REPE ;CX<>0 且ZF＝1重复执行字符串指令 REPNZ/REPNE ;CX<>0 且ZF＝0重复执行字符串指令 功能: 在串操作指令前加上重复前缀,可以对字符串进重复处理.由于加上重复前缀后,对应的指令代码是不同的,所以指令的功能便具有重复处理的功能,重复的次数存放在CX寄存器中.	说明: 1. REPZ/REPE常用与CMPS串操作指令结合使用, 可以完成两组字符串的比较. 2. REPZ/REPE常与SCAS指令结合使用,可以完成在一个字符串中搜索一个关键字. 3. REPNZ/REPNE与CMPS指令结合使用,表示当串未结束(CX=1)且当对应串元素不相同(ZF=0)时,继续重复执行串比较指令. 4. REPNZ/REPNE与SCAS指令结合使用,表示串未结束(CX=1)且当关键字与串元素不相同(ZF=0)时,继续重复执行串搜索指令.
RET	返回指令 RET 格式: RET 功能: 当调用的过程结束后实现从过程返回至原调用程序的下一条指令,本指令不影响标志位.	说明: 由于在过程定义时,已指明其近(NEAR)或远(FAR)的属性,所以RET指令根据段内调用与段间调用,执行不同的操作 对段内调用: 返回时,由堆栈弹出一个字的返回地址的段内偏移量至IP. 对段外调用: 返回时,由堆栈弹出的第一个字为返回地址的段内偏移量,将其送入IP中,由堆栈弹出第二个字为返回地址的段基址,将其送入CS中.
ROL	循环移位指令 格式: ROL OPRD1,COUNT ;不含进位标志位CF在循环中的左循环移位指令. ROR OPRD1,COUNT ;不含进位示志位CF在循环中的右循环移位指令. RCL OPRD1,COUNT ;带进位的左循环移位指令. RCR OPRD1,COUNT ;带进位的右循环移位指令.	说明: 1. 本指令组只影响标志CF、OF.OF由移入CF的内容决定,OF取决于移位一次后符号位是否改变,如改变,则OF=1. 2. 由于是循环移位,所以对字节移位8次; 对字移位16次,就可恢复为原操作数.由于带CF的循环移位,可以将CF的内容移入, 所以可以利用它实现多字节的循环.
ROR	循环移位指令 格式: ROL OPRD1,COUNT ;不含进位标志位CF在循环中的左循环移位指令. ROR OPRD1,COUNT ;不含进位示志位CF在循环中的右循环移位指令. RCL OPRD1,COUNT ;带进位的左循环移位指令. RCR OPRD1,COUNT ;带进位的右循环移位指令.	说明: 1. 本指令组只影响标志CF、OF.OF由移入CF的内容决定,OF取决于移位一次后符号位是否改变,如改变,则OF=1. 2. 由于循环移位,所以对字节移位8次; 对字移位16次,可恢复为原操作数.
SAHF	标志传送指令 SAHF 格式: SAHF 功能: 将AH存至FLAG低8位	说明: 本指令将用AH的内容改写FLAG标志寄存器中的SF、ZF、AF、PF、和CF标志,从而改变原来的标志位.
SAL	算术左移指令 SAL(Shift Arithmetic Left) 格式: SAL OPRD1,COUNT 功能: 其中OPRD1,COUNT与指令SHL相同.本指令与SHL的功能也完全相同,这是因为逻辑左移指令与算术左移指令所要完成的操作是一样的.	说明: 1. 其中OPRD1为目的操作数,可以是通用寄存器或存储器操作数. 2. COUNT代表移位的次数(或位数).移位一次,COUNT=1;移位多于1次时,COUNT=(CL),(CL)中为移位的次数.

SAR	算术右移指令 SAR 格式: SAR OPRD1,COUNT 功能: 本指令通常用于对带符号数减半的运算中,因而在每次右移时,保持最高位(符号位)不变,最低位右移至CF中.	说明: 1. 其中OPRD1为目的操作数,可以是通用寄存器或存储器操作数. 2. COUNT代表移位的次数(或位数).移位一次,COUNT=1;移位多于1次时,COUNT=(CL),(CL)中为移位的次数.
SBB	带借位减去指令 SBB(SuBtraction with Borrow) 格式: SBB OPRD1,OPRD2 功能: 是进行两个操作数的相减再减去CF进位标志位,即从OPRD1<--OPRD1-OPRD2-CF,其结果放在OPDR1中.	说明: 示例 SBB DX,CX SBB AX,DATA1 SBB BX,2000H SBB ALFA[BX+SI],SI SBB BETAP[DI,030AH
SCAS	字符串搜索指令 SCAS 格式: SCAS OPRD SCASB SCASW 功能: 把AL(字节串)或AX(字串)的内容与由DI寄存器寻址的目的串中的数据相减,结果置标志位,但不改变任一操作数本身.地址指针DI自动调整.	说明: 1. 其中OPRD为目的串符号地址. 2. 本指令影响标志AF、CF、OF、PF、SF、ZF.该指令可查找字符串中的一个关键字,只需在本指令执行前, 把关键字放在AL(字节)或AX(字串)中,用重复前缀可在整串中查找. 指令中不使用操作数时,可用指令格式SCASB,SCASW,分别表示字节串或字串搜索指令.
SCASB	字符串搜索指令 SCAS 格式: SCAS OPRD SCASB SCASW 功能: 把AL(字节串)或AX(字串)的内容与由DI寄存器寻址的目的串中的数据相减,结果置标志位,但不改变任一操作数本身.地址指针DI自动调整.	说明: 1. 其中OPRD为目的串符号地址. 2. 本指令影响标志AF、CF、OF、PF、SF、ZF.该指令可查找字符串中的一个关键字,只需在本指令执行前, 把关键字放在AL(字节)或AX(字串)中,用重复前缀可在整串中查找. 指令中不使用操作数时,可用指令格式SCASB,SCASW,分别表示字节串或字串搜索指令.
SCASW	字符串搜索指令 SCAS 格式: SCAS OPRD SCASB SCASW 功能: 把AL(字节串)或AX(字串)的内容与由DI寄存器寻址的目的串中的数据相减,结果置标志位,但不改变任一操作数本身.地址指针DI自动调整.	说明: 1. 其中OPRD为目的串符号地址. 2. 本指令影响标志AF、CF、OF、PF、SF、ZF.该指令可查找字符串中的一个关键字,只需在本指令执行前, 把关键字放在AL(字节)或AX(字串)中,用重复前缀可在整串中查找. 指令中不使用操作数时,可用指令格式SCASB,SCASW,分别表示字节串或字串搜索指令.
SHL	逻辑左移指令 SHL(Shift logical left) 格式: SHL OPRD1,COUNT 功能: 对给定的目的操作数左移COUNT次,每次移位时最高位移入标志位CF中,最低位补零.	说明: 1. 其中OPRD1为目的操作数,可以是通用寄存器或存储器操作数. 2. COUNT代表移位的次数(或位数).移位一次,COUNT=1;移位多于1次时,COUNT=(CL),(CL)中为移位的次数. 3. 例如: SHL AL,1 SHL CX,1 SHL ALFA[DI] 或者: MOV CL,3 SHL DX,CL SHL ALFA[DI],CL

SHR	逻辑右移指令 SHR 格式: SHR OPRD1,COUNT 功能: 本指令实现由COUNT决定次数的逻辑右移操作,每次移位时,最高位补0,最低位移至标志位CF中.	说明: 1. 其中OPRD1为目的操作数,可以是通用寄存器或存储器操作数. 2. COUNT代表移位的次数(或位数).移位一次,COUNT=1;移位多于1次时,COUNT=(CL),(CL)中为移位的次数. 3. 影响标志位OF,PF,SF,ZF,CF.
STC	处理器控制指令－标志位操作指令 格式: CLC ;置CF=0 STC ;置CF=1 CMC ;置CF=(Not CF)进位标志求反 CLD ;置DF=0 STD ;置DF＝1 CLI ;置IF=0，CPU禁止响应外部中断 STI ;置IF=1，使CPU允许向应外部中断 功能: 完成对标志位的置位、复位等操作.	说明: 例如串操作中的程序,经常用CLD指令清方向标志使DF＝0,在串操作指令执行时,按增量的方式修改吕指针.
STD	处理器控制指令－标志位操作指令 格式: CLC ;置CF=0 STC ;置CF=1 CMC ;置CF=(Not CF)进位标志求反 CLD ;置DF=0 STD ;置DF＝1 CLI ;置IF=0，CPU禁止响应外部中断 STI ;置IF=1，使CPU允许向应外部中断 功能: 完成对标志位的置位、复位等操作.	说明: 例如串操作中的程序,经常用CLD指令清方向标志使DF＝0,在串操作指令执行时,按增量的方式修改吕指针.
STI	处理器控制指令－标志位操作指令 格式: CLC ;置CF=0 STC ;置CF=1 CMC ;置CF=(Not CF)进位标志求反 CLD ;置DF=0 STD ;置DF＝1 CLI ;置IF=0，CPU禁止响应外部中断 STI ;置IF=1，使CPU允许向应外部中断 功能: 完成对标志位的置位、复位等操作.	说明: 例如串操作中的程序,经常用CLD指令清方向标志使DF＝0,在串操作指令执行时,按增量的方式修改吕指针.
STOS	字符串存储指令 STOS 格式: STOS OPRD 功能: 把AL(字节)或AX(字)中的数据存储到DI为目的串地址指针所寻址的存储器单元中去.指针DI将根据DF的值进行自动调整.	说明: 1. 其中OPRD为目的串符号地址. 2. 本指令不影响标志位.当不使用操作数时,可用STOSB或STOSW分别表示字节串或字串的操作.

SUB	减法指令SUB(SUBtract)
	格式: SUB OPRD1,OPRD2
	功能: 两个操作数的相减,即从OPRD1中减去OPRD2,其结果放在OPDR1中.
		说明:
		示例 SUB DX,CX
		SUB [BX+25],AX
		SUB DI,ALFA[SI]
		SUB CL,20
		SUB DATA1[DI][BX],20A5H

TEST	测试指令 TEST 格式: TEST OPRD1,OPRD2	说明: TEST与AND指令的关系,有点类似于CMP与SUB指令之间的关系.

	功能: 其中OPRD1、OPRD2的含义同AND指令一样,也是对两个操作数进行按位的'与'运算,唯一不同之处是不将'与'的结 果送目的操作数,即本指令对两个操作数的内容均不进行修改,仅是在逻辑与操作后,对标志位重新置位.	
WAIT	处理器等待指令 WAIT 格式: WAIT 功能: 本指令将使处理器检测TEST端脚,当TEST有效时,则退出等待状态执行下条指令,否则处理器处于等待状态,直到TEST有效.	说明: 本指令不影响标志位.
XCHG	数据交换指令 XCHG 格式: XCHG OPRD1,OPRD2 其中的OPRD1为目的操作数,OPRD2为源操作数 功能: 将两个操作数相互交换位置,该指令把源操作数OPRD2与目的操数OPRD1交换.	说明: 1. OPRD1及OPRD2可为通用寄存器或存储器,但是两个存储器之间是不能用XCHG指令实现的. 2. 段寄存器内容不能用XCHG指令来交换. 3. 若要实现两个存储器操作数DATA1及DATA2的交换,可用以下指令实现: 示例: PUSH DATA1 PUSH DATA2 POP DATA1 POP DATA2 4. 本指令不影响状态标志位.
XLAT	查表指令 XLAT 格式: XLAT TABLE其中TABLE为一待查表格的首地址. 功能: 把待查表格的一个字节内容送到AL累加器中.	说明: 1. 在执行该指令前,应将TABLE先送至BX寄存器中,然后将待查字节与在表格中距表首地址位移量送AL,即 (AL)<--((BX)+(AL)). 2. 本指令不影响状态标位,表格长度不超过256字节.
XOR	逻辑异或运算指令 XOR 格式: XOR OPRD1,OPRD2 功能: 实现两个操作数按位'异或'运算,结果送至目的操作数中. OPRD1<--OPRD1 XOR OPRD2	说明: 1. 其在OPRD1、OPRD2的含义与AND指令相同,对标志位的影响与与AND指令相同. 2. 相异为真,相同为假.

汇编程序从写出到执行的过程
编程（Edit）——1.asm——编译（编译）——1.obj—— 连接（link）——1.exe——加载（command）——内存中的程序——运行（CPU）

分类: Hacker

好文要顶 关注我 收藏该文

lsgxeva
关注 - 1
粉丝 - 183
+加关注

0 0

« 上一篇： 关于ida pro的插件keypatch
» 下一篇： 8086 CPU 寻址方式

posted @ 2018-04-25 23:29 lsgxeva 阅读(20950) 评论(1) 编辑 收藏

评论列表

#1楼 2019-09-25 19:56 **wydj**

感觉好些错误啊，比如mov D,S 是把S传给D；在比如cmp S1,S2，计算S1-S2的结果。

支持(0) 反对(0)

刷新评论 刷新页面 返回顶部

登录后才能发表评论，立即 登录 或 注册， 访问 网站首页

AWS免费产品:
· 如何在AWS上免费构建网站
· AWS免费云存储解决方案
· 在AWS上免费构建数据库
· AWS上的免费机器学习

最新新闻:
· 司机资格可以网上买？货拉拉合规整改屡屡“慢半拍”
· 科学家完成了迄今最小引力场的测量
· 为挖矿疯狂：有人定制高端水冷循环散热 有人用冷油降温...
· 新研究发现全球最古老陨石冲击坑只是正常地质作用结果
· Google祸不单行
» 更多新闻...

历史上的今天：
2018-04-25 ida动态调试笔记
2018-04-25 IDA 操作记录
2018-04-25 Win7 VS2015 NASM汇编语言环境配置
2018-04-25 逆向中静态分析工具——IDA初学者笔记之字符串分析
2018-04-25 静态分析：IDA逆向代码段说明 text、idata、rdata、data
2018-04-25 逆向中静态分析工具——IDA初学者笔记
2018-04-25 程序破解基本知识

Copyright © 2021 lsgxeva
Powered by .NET 5.0 on Kubernetes