

汇编部分

1 个字(word)是 2 个字节(byte), 1 个字节是 8 位(bit)

8086 的常用寄存器

AH/AL	AX	数据寄存器	通用寄存器
BH/BL	BX		
CH/CL	CX		
DH/DL	DX		
SP	栈顶指针	指针寄存器	
BP	基址指针		
SI	源地址	变址寄存器	
DI	目的地址		
IP	指令指针	控制寄存器	
FLAGS	标志寄存器		
CS	代码段	段寄存器	
DS	数据段		
SS	堆栈段		
ES	附加段		

8086 的标志寄存器 FLAGS

最低字节就是最低的 byte, 只看最后 8 位;

一共 8 位的结果看全部 8 位, 一共 16 位的结果看后 8 位;

状态标志	含义
进位标志 CF	当 <b>无符号整数</b> 加减运算的最高有效位进位(加法)或借位(减法)时 CF = 1, 否则 CF = 0;
奇偶标志 PF	当运算结果的 <b>最低字节</b> 中, 1 的个数为零或偶数时 PF = 1; 否则 PF = 0;
辅助进位标志 AF	当运算结果的 <b>最低字节</b> (同 PF) 中低位的一半向高位(看最后 8 位的低 4 位向高 4 位)有进位或借位时, AF=1; 该标志与操作数长度无关, 只看最低字节中间那两位就行;
零标志 ZF	若运算结果为 0 则 ZF = 1, 否则 ZF = 0;
符号标志 SF	若运算结果最高位为 1 则 SF = 1, 否则 SF = 0;
溢出标志 OF	当 <b>有符号整数</b> 加减结果有溢出则 OF = 1, 否则 OF = 0; 程序员: 无符号数关心进位, 有符号数关心溢出, 相同符号数相加后符号相反(如正+正=负), 则有溢出; 处理器硬件判断: 最高位和次高位同时进位或不进位, 则无溢出; 反之则有溢出;
方向标志	仅用于串操作指令, 控制地址的变化方向:

状态标志	含义
DF	1. 设置 DF = 0, 每次串操作后的存储器地址就自动增加, 即从低向高地址处理数据串; 2. 设置 DF = 1, 每次串操作后的存储器地址就自动减少, 即从高向低地址处理数据串; 3. 可以执行 CLD 指令设置 DF = 0; 执行 STD 指令设置 DF = 1;
中断允许标志 IF	主要针对外中断中可屏蔽中断的开放或禁止: 1. 当 IF=1 时, CPU 允许响应可屏蔽中断, 中断当前程序, 转去执行中断处理程序; 2. 当 IF = 0 时, 则不允许响应可屏蔽中断; 3. 可以执行 STI 指令设置 IF=1; 执行 CLI 指令设置 IF=0;
追踪标志 TF	用于单步调试程序: 1. 当 TF=1 时在执行完一条指令后, 产生单步中断; 2. 追踪标志 TF=1 时为调试程序时所用, 当程序调试成功后让 TF=0

8086 存储器的组织

分段方式, 20 位的物理地址由 16 位的段地址和 16 位的偏移地址形成, 每个段的**最大寻址空间为 64KB**;

例: 存储单元的地址

已知:

(0004H) = 5678H

字地址      字内容

(0004H) = 78H

字节地址      字节内容

问:

((0004H)) = ?

地址的地址      字或字节内容

78H	0004H
56H	0005H
34H	0006H
12H	0007H
1EH	5678H
2FH	5679H

注意一下按字读取的方式是【高字节+低字节】拼接出结果

DATA\_WORD DW 100,100H,-5,0 结果如下:

071A1: 64 100

071A2: 00 000

071A3: 00 000

071A4: 01 001

071A5: FB 251

071A6: FF 255

071A7: 00 000

071A8: 00 000

8086 缺省段 + 偏移寻址组合

记偏移寄存器→段寄存器方向;

段寄存器	偏移寄存器	用途
CS 代码段	IP	指令寻址
DS 数据段	BX, DI, SI 或 16 位数	数据寻址
SS 堆栈段	SP 或 BP	堆栈寻址
ES 附加段	DI	目标串寻址

ASCII 码的关键分界线

48=30H='0', 65=41H='A', 97=61H='a'

操作数寻址总结

做题注意一下哪边是高地址，哪边是低地址

- 立即数寻址 MOV AX , 3069H
- 寄存器寻址 MOV AL , BH
- 直接寻址 MOV AX , [2000H]
- 寄存器间接寻址 MOV AX , [BX]
- 寄存器相对寻址 MOV AX , NUM [SI]
- 基址变址寻址 MOV AX , [BP][DI]
- 有效地址 = 
$$\begin{matrix} (BX) & + & (SI) & \text{段地址在数据段DS} \\ (BP) & & (DI) & \text{段地址在堆栈段SS} \end{matrix}$$
- 相对基址变址寻址 MOV AX , MASK [BX][SI]

8086 的伪指令

段定义伪操作 SEGMENT, ENDS

程序由 4 个逻辑段组成：数据段 DATA、堆栈段 STACK、附加段 EXTRA 和代码段 CODE；

指定段地址伪指令 ASSUME（给程序员看的）

伪指令的格式：ASSUME <段寄存器名>:<段名>

程序结束伪操作 END

```
CODE SEGMENT           ; 定义代码段
    ASSUME CS:CODE, DS:DATA, ES:EXTRA
START:
    MOV  AX, DATA
    MOV  DS, AX          ; 段地址 -> 段寄存器
CODE ENDS
    END  START
```

数据定义及存储器分配伪操作 DB、DW、DD、DQ、DT

分别定义字节、字、双字、8 字节、10 字节

复制操作符 DUP，表示操作数重复若干次

例：DB 2 DUP (0,2 DUP (1,2),3)  
= 2 DUP (0, 1, 2, 1, 2, 3)  
= (0, 1, 2, 1, 2, 3, 0, 1, 2, 1, 2, 3)

表达式赋值伪操作 EQU

=可以对一个符号重复定义，EQU 不能对同一个符号重复定义

地址计数器与对准伪操作 \$

在数据段等于地址计数器的当前值，在代码段等于当前正在汇编的指令的地址

```
BUF2 DW 1,2,3,4,5
CNT2 EQU ($-BUF2)/2 ; (/2 后等于数组元素个数)
ORG $+8             ; 跳过 8 个字节的存储区
JNE $+6             ; 转向地址是 JNE 的地址 +6
JMP $+2             ; 转向下一条指令
```

表达式操作符

主要是对常量进行运算

运算符类型	运算符及说明
算术运算符	+, -, *, /、MOD（取余数）
逻辑运算符	AND（逻辑与）、OR（逻辑或）、XOR（位异或）、NOT（逻辑非）
位移运算符	SHL（逻辑左移）、SHR（逻辑右移）

运算符类型	运算符及说明
关系运算符	EQ (=)、NE (!=)、GT (>)、LT (<)、GE (>=)、LE (<=)

数值回送操作符 OFFSET、SEG、LENGTH、SIZE

OFFSET 取变量的偏地址，SEG 取变量段地址

```
MOV DX, SEG FUNC
```

LENGTH 取由 DUP 定义的变量的单元数（其他情况等于 1）

SIZE 取由 DUP 定义的变量的字节数（其他情况等于 1）

属性修改伪指令 PTR

新属性 PTR（旧属性的）表达式

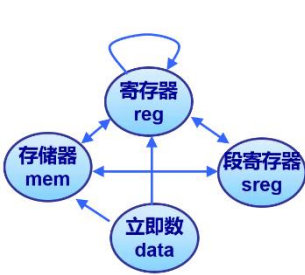
用于暂时改变内存变量或标号的原有属性，BYTE PTR 表示字节，WORD PTR 表示字，以此类推

程序段前缀结构 FAR

把整个程序定义成一个 FAR 型过程（或子程序），该过程能够让其它汇编文件中的段来调用（near 不能）

```
NAME PROC [TYPE]
; ...
NAME ENDP
```

8086 的指令系统



传送指令 MOV

其中，寄存器 REG: AX, AH, AL, DI, SI, BP, SP 等  
段寄存器 SREG: DS, ES, SS  
存储器 memory: [BX], [BX+SI+7]，变量等；  
立即数 immediate: 5, -24, 3Fh, 10001101b 等

注意：CS 只能作为操作源，MOV CS,AX 会报错 MOV AX,CS 可以

数据交换指令 XCHG

XCHG OPR1, OPR2 操作是交换 OPR1 和 OPR2 的值  
规定两个操作数必须有一个在寄存器中

换码/查表指令 XLAT 或 XLAT OPR

这两个指令默认了操作数，执行操作：(AL) ← ( BX + AL )

```
MOV BX, OFFSET TABLE ; 令 BX=0040H
MOV AL, 3
XLAT TABLE ; 指令执行后：(AL)=43H
```

地址传送指令 LEA、LDS、LES

LEA REG, SRC 是把 SRC 的有效地址 EA 送到 REG

LDS REG, SRC 是把源操作数存放的地址指针段基址:偏移地址的低 16 位送入目标寄存器，高 16 位送入 DS 段寄存器

LES REG, SRC 是把源操作数存放的地址指针段基址:偏移地址的低 16 位送入目标寄存器，高 16 位送入 ES 段寄存器

标志寄存器传送指令 LAHF、SAHF、PUSHF、POPF

分别是标志位低字节送 AH 指令，AH 送标志位低字节，标志位 2 字节进栈，栈顶 2 字节送标志位

加法指令 ADD、ADC、INC

分别是加法，带进位加法，+=1

减法指令 SUB、SBB、DEC、NEG

分别是减法，带借位减法，-=1，求补码

## 比较指令 CMP

CMP OPR1, OPR2 执行的是  $(OPR1) - (OPR2)$ ，但 CMP 不保存结果只影响标志位，配合条件跳转指令使用

## 乘法指令 MUL、IMUL SRC / 除法指令 DIV、IDIV

MUL/IMUL 是无符号和有符号乘法，执行操作：

(有符号乘法：乘积低半部分的最高位符号扩展到高半部分)

字节操作数  $(AX) \leftarrow (AL) * (SRC)$

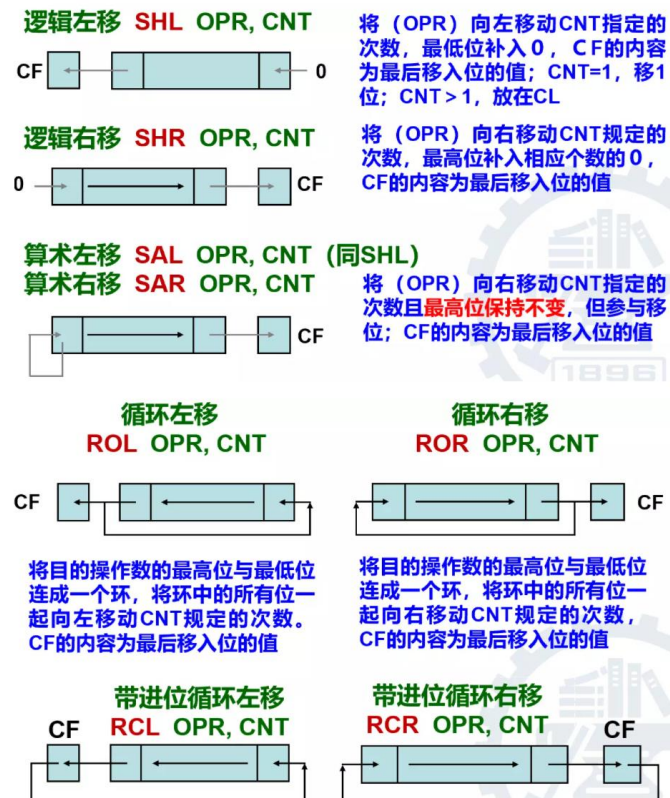
字操作数  $(DX, AX) \leftarrow (AX) * (SRC)$

## 逻辑运算指令 AND、OR、NOT、XOR、TEST

TEST 是取两个操作符按位与，配合条件跳转指令

## 移位指令 SHL、SHR、SAL、SAR、ROL、ROR、RCL、RCR

不能用立即数寻址，注意区分十进制和 16 进制



无条件转移指令 JMP 和条件转移指令 JZ/JNZ、JE/JNE、JS/JNS、JO/JNO、JP/JNP、JB/JNB、JL/JNL、JBE/JNBE、JLE/JNLE、JCXZ

指令名	转移条件	说明
JZ/JE	ZF=1	相等/为零转移(=)
JNZ/JNE	ZF=0	不相等/不为零转移(≠)
JS	SF=1	为负转移
JNS	SF=0	为正转移
JO	OF=1	溢出转移
JNO	OF=0	不溢出转移
JP/JPE	PF=1	偶数个 1 转移
JNP/JPO	PF=0	奇数个 1 转移
JB/JBAE/JC	CF=1	低于转移(<)
JNB/JAE/JNC	CF=0	不低于转移(≥)
JBE/JNA	CF=1 或 ZF=1	不高于转移(≤)
JNBE/JA	CF=0 且 ZF=0	高于转移(>)
JL/JNGE	SF≠OF	小于转移(<)
JNL/JGE	SF=OF	不小于转移(≥)
JLE/JNG	(SF≠OF) 且 ZF=1	不大于转移(≤)
JNLE/JG	(SF=OF) 且 ZF=0	大于转移(>)

无条件转移 short 是 8 位位移量，near 是 16 位位移量；

条件转移只有 8 位位移量，范围 -128 ~ +127

## 循环指令 LOOP、LOOPZ/LOOPE、LOOPNZ/LOOPNE

共同前提都需要 CX≠0，分别是循环，为 0 或相等时循环，不为零或不相等时循环；

## 子程序调用和返回指令

### 段内直接调用 CALL DST，其中 DST 为子程序名

执行操作：

$(SP) \leftarrow (SP) - 2$ ；断点压入堆栈

$((SP)+1, (SP)) \leftarrow (IP)$ ；主程序地址压栈

$(IP) \leftarrow (IP) + 16$  位位移量

注意：IP 为 Call 指令的下一条指令的地址，其与 DST 子程序的地址间的相对地址或位移量是固定的

### 段内间接调用 CALL DST，其中 DST 为寄存器如 BX，或存储器地址 WORD PTR [BX]

执行操作：

$(SP) \leftarrow (SP) - 2$

$((SP)+1, (SP)) \leftarrow (IP)$

$(IP) \leftarrow (EA)$  (DST 为内存地址)

### 段内近返回 RET

功能：将堆栈中保存的 2 字节断点的偏移地址恢复到 IP

执行操作： $(IP) \leftarrow ((SP)+1, (SP))$   $(SP) \leftarrow (SP)+2$

### 段内带立即数近返回 RET EXP

功能：EXP 表示，弹出断点之后，使 SP 内容再回退 EXP 个字节单元，作用是使断点之后的 EXP 个字节单元分数数据失效

## 中断与中断返回指令 INT、INTO、IRET

分别是调用中断号，溢出中断，中断返回

### ※ 几个常用的 BIOS 和 DOS 中断

中断号	调用参数	功能
INT 16H	AH=00H	从键盘读一字符，AL=字符码，AH=扫描码（对应键盘的物理按键情况）
	AH=01H	读键盘缓冲区字符，若 ZF=0，AL=字符码，AH=扫描码；若 ZF=1，缓冲区空
	AH=02H	读键盘，AL=键盘状态字节
INT 21H	AH=01H	从键盘输入一个字符并回显在屏幕上，AL=字符
	AH=06H DL=0FFH	若有字符可取，AL=字符，ZF=0 若无字符可取，AL=0，ZF=1
	AH=07H	从键盘输入一个字符不回显，AL=字符
	AH=08H	从键盘输入一个字符不回显，AL=字符
	AH=0AH	输入字符串，DS:DX=缓冲区首址
	AH=0BH	读键盘状态，AL=0FFH 有键入，AL=00 无键入
	AH=4CH	返回操作系统
	AH=02H	显示字符，DL=待显示字符
	AH=06H	显示字符串，DL=待显示字符
	AH=09H	显示字符串，DS:DX=串地址，串必须以 \$ 结束
	AH=35H	读中断向量，AL=向量号，ES:BX=中断向量
	AH=25H	写入中断向量，AL=向量号，DS:DX=中

		断向量
--	--	-----



## 接口部分

### IN 指令

从 I/O 端口向 AL (8 位数据) 或 AX (16 位数据) 输入字节 (字)

### OUT 指令

从 AL (8 位数据) 或 AX (16 位数据) 向 I/O 端口输出字节 (字)

### I/O 指令使用格式

OUT8 MACRO PORT, VAL

MOV DX, PORT

MOV AL, VAL

OUT DX, AL

ENDM

IN8 MACRO VAL, PORT

MOV DX, PORT

IN AL, DX

MOV VAL, AL

ENDM

### I/O 端口地址固定式译码设计

例：使用 74LS138 (3-8 译码器) 设计一个系统板上的地址译码电路，每个接口芯片内部的端口数目为 32 个，每个芯片的基址为 00H、20H、40H、60H、80H、A0H、C0H、E0H。(只有 A9-A0 参与译码，其余地址线为 0)。

分析：系统板上的 I/O 地址分配在 000~0FFH 范围内，故只使用低 8 位地址，则 A9 和 A8 应赋 0 值。为了实现每个接口芯片内部拥有 32 个端口，只要留出 5 根低位地址线不参加译码，即地址线低 5 位作为芯片内部端口译码 32=2<sup>5</sup>，其余的高位地址线作为 74LS138 输入线参加译码，或作为控制线控制译码是否有效即可。其控制/译码线与地址线的分配如下：

地址线	0 0 A <sub>9</sub> A <sub>8</sub>	A <sub>7</sub> A <sub>6</sub> A <sub>5</sub>	A <sub>4</sub> A <sub>3</sub> A <sub>2</sub> A <sub>1</sub> A <sub>0</sub>
用途	控制	片选	片内端口寻址
十六进制	0H	0~7H	0~1FH

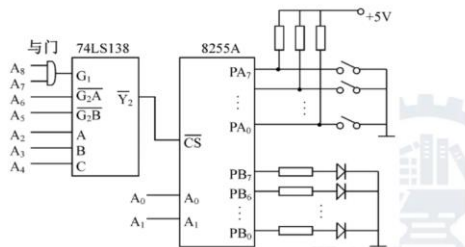
例：留有 2 个地址线，可以片内寻址 4 个

使能端确定 A8-A5=1100，ABC=010

范围 188H-18BH

A <sub>9</sub>	A <sub>8</sub>	A <sub>7</sub>	A <sub>6</sub>	A <sub>5</sub>	A <sub>4</sub>	A <sub>3</sub>	A <sub>2</sub>	A <sub>1</sub>	A <sub>0</sub>
0	1	1	0	0	0	1	0	x	x

2、一个系统的地址译码使用 74LS138 实现，除图中的地址线外，其余地址线为 0 电平，请问 8255 的端口地址是多少？



### ※ 可编程并行接口芯片 82C55A

外部特性：端口地址依次为：A 口，B 口，C 口，命令口

- D<sub>0</sub>~D<sub>7</sub>：双向数据线，用于 CPU 向 8255A 发送命令、数据和 8255A 向 CPU 回送状态、数据等。
- $\overline{CS}$ ：片选信号，低电平有效。
- A<sub>1</sub>，A<sub>0</sub>：片内端口地址信号可以形成 4 个端口地址。
- $\overline{RD}$ ：读信号，低电平有效。

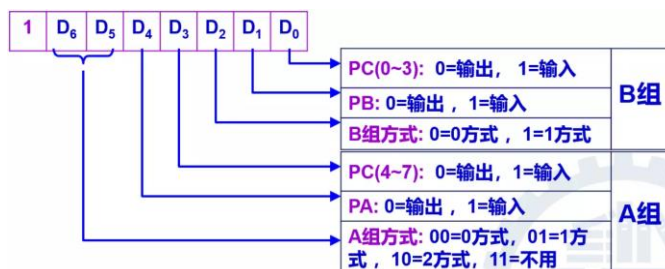
- $\overline{WR}$ ：写信号，低电平有效。
- RESET：复位信号，高电平有效。清除控制寄存器，并将 8255A 的 A、B、C 三个 8 位端口均置为 0 方式输入，直到在初始化程序段中用方式命令才能改变，使其进入用户所选的状态。

### 8255 的工作方式

(只讲过) 方式 0：基本输入/输出方式

- 单向 I/O，一次初始化只能指定端口 (PA、PB 和 PC) 作输入或输出，不能指定端口同时又输入又输出
- 适用于无条件或查询方式传送，不能用中断方式交换数据
- A 端口：数据端口，8 位并行
- B 端口：数据端口，8 位并行
- C 端口：数据端口，4 位并行 (分高 4 位和低 4 位)，或作位控，按位输出高/低电平

### 8255 工作方式命令

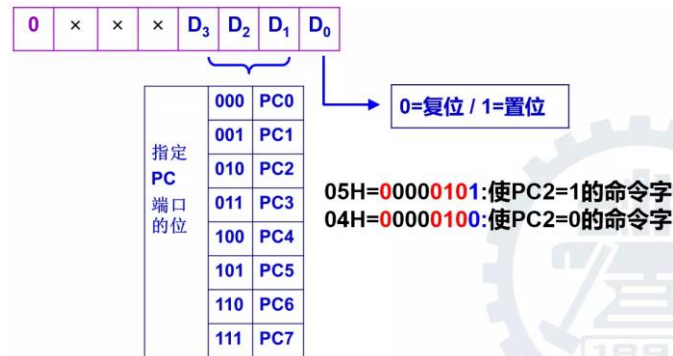


例：把 A 口指定为 1 方式，输入，C 口上半部为输出；B 口指定为 0 方式，输出，C 口下半部定为输入，则工作方式命令代码是：10110001B=B1H

```
MOV DX, 303H ; 8255A 命令口地址 (假设)
MOV AL, 0B1H ; 初始化命令
OUT DX, AL ; 送到命令口
```

### 8255 的 C 端口按位置位/复位命令

这个命令是写入到命令口而不是 C 口；



### ※ 可编程定时/计数器 82C54A

外部特性：8254 是 8253 的升级版，端口地址依次为通道 0，通道 1，通道 2，命令寄存器

- 数据线：D<sub>0</sub>-D<sub>7</sub>
- 地址线：片选  $\overline{CS}$ ，片内地址 A<sub>0</sub>、A<sub>1</sub>
- 控制线： $\overline{WR}$ ， $\overline{RD}$
- 时钟信号：CLK<sub>0</sub>-CLK<sub>2</sub>
- 门控信号：GATE<sub>0</sub>-GATE<sub>2</sub>
- 输出信号：OUT<sub>0</sub>-OUT<sub>2</sub>

8254 的命令字：使用同一个端口，按方式命令在先，其它命令在后的顺序写入端口

## 8254 方式命令

初始化 82C54A, 包括选定计数通道、设定工作方式、确定字节读写顺序以及计数值码制

D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>
计数器选择		读写字节数		工作方式		码制	

位数	功能和取值
D7-D6	计数器选择 D7D6=00, 选择 0 号计数器; D7D6=01, 选择 1 号计数器; D7D6=10, 选择 2 号计数器; D7D6=11, 无效;
D5-D4	设置读写字节数 D5D4=00, 锁存命令 计数器值复制到锁存器, D3~0 无效; D5D4=01, 仅读/写一低字节, 高字节为 0 D5D4=10, 仅读/写一高字节, 低字节为 0 D5D4=11, 一次读/写 2 个字节, 先低字节, 后高字节
D3-D1	工作方式 D3D2D1=000, 选择 0 方式; D3D2D1=001, 选择 1 方式; D3D2D1=X10, 选择 2 方式; D3D2D1=X11, 选择 3 方式; D3D2D1=100, 选择 4 方式; D3D2D1=101, 选择 5 方式;
D0	码制 D0=0, 选择二进制计数; D0=1, 选择十进制 BCD 计数;

计数码制: 用二进制计数, 直接写十进制值不加 H

; 计数初值设为 10

out8 command\_addr, 00110100B

out16 cn0, 10

## 8254 锁存命令

D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>
SC <sub>1</sub>	SC <sub>0</sub>	0	0	X	X	X	X

00=计数器0  
01=计数器1  
10=计数器2

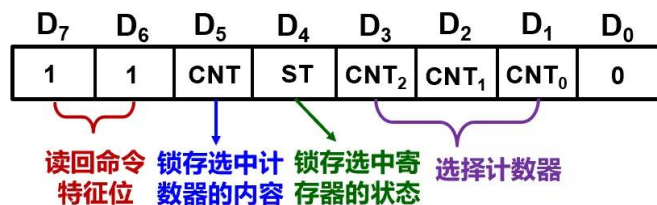
锁存命令  
特征位

未用

Q: 读当前计数值为什么要先锁存计数值?

A: 减 1 计数器是 16 位的, 而定时器的数据线是 8 位的, 必须读两次才能读出 16 位数据, 在两次读操作时可能使减 1 计数器的内容发生变化

## 8254 读回命令



注意: 读回命令只是一个锁存的功能, 读还要发一条读命令

D3-D1 用于选择 3 个计数器 1: 选中, 0: 未选中;

D5-D4 用于选择读当前状态还是当前计数值 0: 读取, 1: 不读取

## 8254 的 6 种工作方式

	方式0	方式1	方式2	方式3	方式4	方式5
OUT输出状态	写入控制字后变0, 计数结束变1, 并维持至重写控制字或计数初值	写入控制字后变1, GATE上升沿触发变0, 开始计数, 计数结束变1	写入控制字后变1, 计数到1变0, 维持一个CLK变1, 重装初值继续计数	写入控制字后变1, 装入初值且GATE=1则OUT变1, 计数到变0, 重装初值继续计数, 计数到则反向	写入控制字后变1, 计数结束变0, 维持一个CLK变1	写入控制字后变1, GATE上升沿触发开始计数, 计数结束输出一个CLK的负脉冲
初值自动重装	无	无	计数到0重装	根据初值奇偶分别重装	无	无
计数过程中改变初值	立即有效	GATE触发后有效	计数到1或GATE触发后有效	计数结束或GATE触发后有效	立即有效	GATE触发后有效
GATE	0	禁止计数	无影响	禁止计数	禁止计数	无影响
	下降沿	暂停计数	无影响	停止计数	停止计数	无影响
	上升沿	继续计数	从初值开始重新计数	从初值开始重新计数	从初值开始重新计数	从初值开始重新计数
	1	允许计数	无影响	允许计数	允许计数	无影响

计数初值换算方法如下:

产生时间间隔  $\tau$  的时间常数  $T_c$ :

$$T_c = \frac{\text{要求定时的时间}}{\text{时钟脉冲周期}} = \frac{\tau}{1/\text{CLK}} = \tau \times \text{CLK}$$

产生频率为  $f$  的信号波形的时间常数  $T_c$

$$T_c = \frac{\text{时钟脉冲的频率}}{\text{要求的波形频率}} = \frac{\text{CLK}}{f}$$

例: 8254 初始化, 要求选择 2 号计数器工作在 3 方式, 计数初值为 533H (2 个字节) 二进制计数

```
MOV DX, 307H ; 命令口
MOV AL, 10110110B ; 2号计数器的方式命令字
OUT DX, AL
MOV DX, 306H ; 2号计数器数据口
MOV AX, 533H ; 计数初值
OUT DX, AL ; 先送低字节到2号计数器
MOV AL, AH ; 取高字节送AL
OUT DX, AL ; 后送高字节到2号计数器
```

## 8259 中断实验例程

中断向量指针: 指出中断向量存放在中断向量表的地址

中断类型号×4 = 中断向量最低字节的指针

中断号×4 = 偏移地址 IP

中断号×4+2 = 段基址 CS

; 设置中断向量地址

```
setint macro intno, handler
    push ds
    mov ax, 0
    mov ds, ax ; 数据段定位到0
    mov di, intno * 4
    cli
    mov bx, offset handler
    mov [di], bx
    add di, 2
```

```

mov bx, seg handler
mov [di], bx
pop ds
endm

; 代码段开始, MIRQ3 的新中断程序
twinkle1 proc far
push ax
cli
xor light, 1
out8 PA, light
inc si
out8 M8259, 20h
sti
pop ax
iret
endp

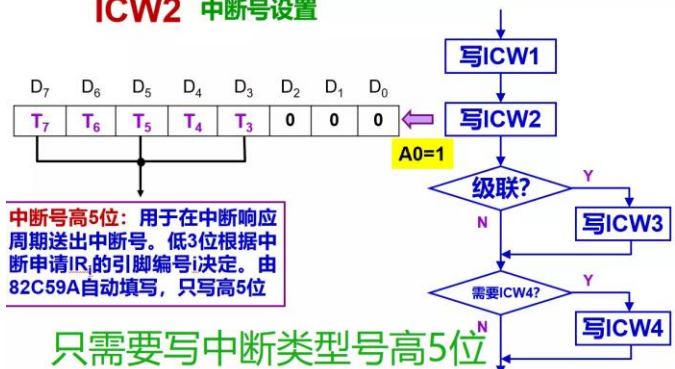
```

```

start:
; 定位 DS 到数据段
mov ax, data
mov ds, ax
; 设置 8255 工作命令
out8 cmd, 10000000b
; 设置中断向量指向新中断程序
setint MIRQ3, twinkle1
setint SIRQ10, twinkle2
; 分别设置主片和从片的中断屏蔽字
mask M8259+1, 11110011b
mask S8259+1, 11111011b
...

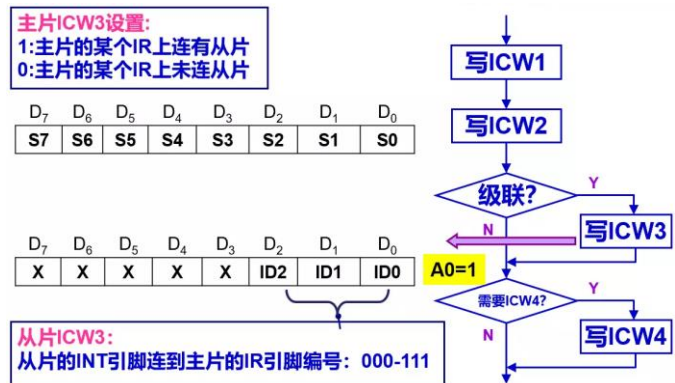
```

## ICW2 中断号设置



例: 中断号 0DH(00001101B), 只需要写 00001000B

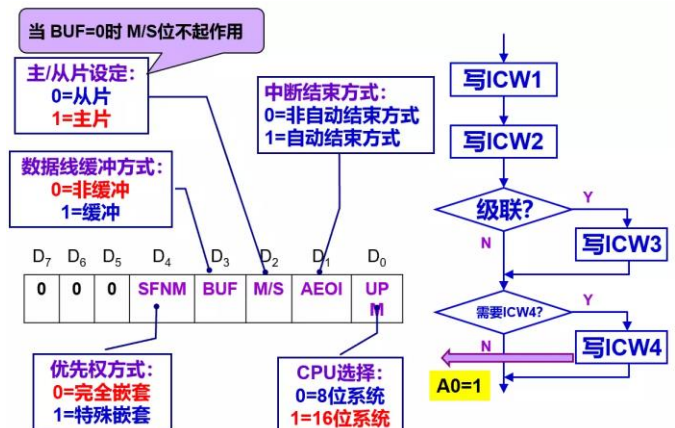
## ICW3: 进行级联设置



例: 从片 A 和从片 B 的请求线 INT 连到主片的 IR3 和 IR6

; 主片的 ICW3=01001000B=48H  
MOV AL, 48H ; 主片的 ICW3  
OUT 21H, AL  
; 从片 A 的 ICW3=00000011B=03H  
MOV AL, 03H ; 从片 A 的 ICW3, 写入 3  
OUT 0A1H, AL  
; 从片 B 的 ICW3=00000110B=06H  
MOV AL, 06H ; 从片 B 的 ICW3, 写入 6  
OUT 0A1H, AL

## ICW4: 进行优先级和结束方式设置, 一般选红色的



## OCW1: 进行常规的屏蔽开放操作

用来写入屏蔽字

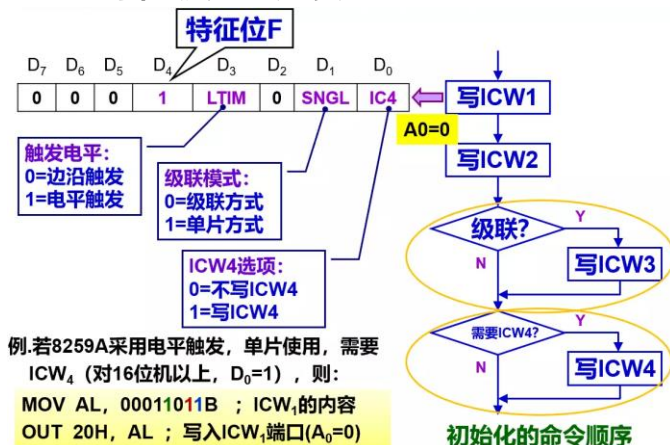
## ※ 中断控制器 82C59A

实验箱主片端口地址: 20H 和 21H

从片端口地址: 0A0H 和 0A1H

8259A 有两类编程命令, 初始化命令字 ICW (不允许用户设置) 和操作命令字 OCW

### ICW1: 进行中断触发和单片/多片设置

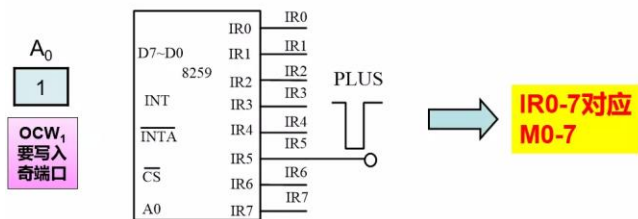


### ICW2: 进行中断号设置



0=开放, 1=屏蔽

D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>
M <sub>7</sub>	M <sub>6</sub>	M <sub>5</sub>	M <sub>4</sub>	M <sub>3</sub>	M <sub>2</sub>	M <sub>1</sub>	M <sub>0</sub>



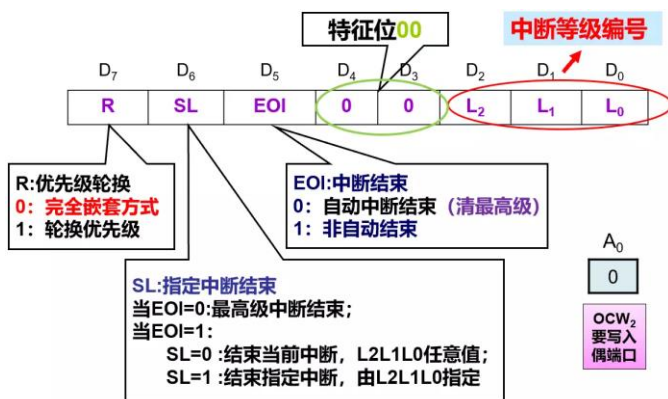
例: 要使中断源 IR3 开放, 其余被屏蔽;

```
IN AL, 21H ; 回读 21H 端口的内容
AND AL, 11110111B
OUT 21H, AL
```

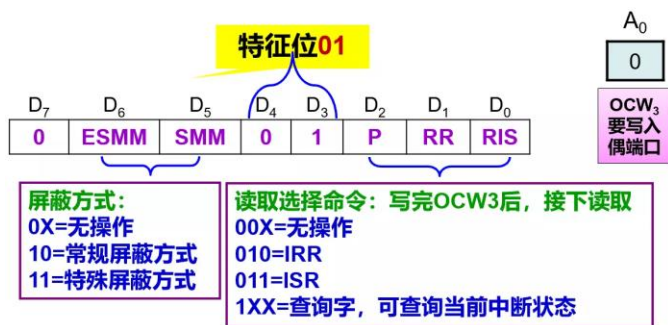
## OCW2: 进行中断结束和优先级轮换排队操作

用来发中断结束 EOI 命令

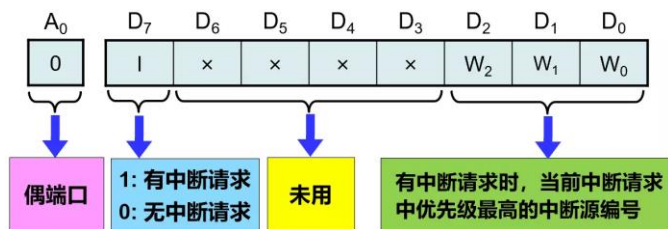
```
out8 M8259, 20h ; 主片最高级非自动结束
out8 S8259, 62h ; 从片指定 IR2 结束
```



## OCW3: 设定屏蔽方式和读取状态



## 8259 查询字的格式



## ※ 多通道 A/D 转换器 ADC0809

外部特性: 命令直接发到和  $\overline{CS}$  连接的端口

此处是 AD

```
out8 AD, 0 ; 进行假写操作, 使 AD0809 启动转换
in8 buf[di], AD ; 读取 AD0809 转换的数据
```

## ※ D/A 转换器 DAC0832

外部特性: 命令直接发到和  $\overline{CS}$  连接的端口

- 直通方式:  $LE1=LE2=1$ , 数据直接进 DA 转换器
- 单缓冲方式:  $LE1$  或  $LE2=1$ , 一个始终直通一个缓存状态
- 双缓冲方式: 两个寄存器都缓冲

$\overline{ILE}$  (输入锁存允许)、 $\overline{CS}$  (片选) 和  $\overline{WR_1}$  (写信号 1) 组合控制第一级缓冲器的锁存

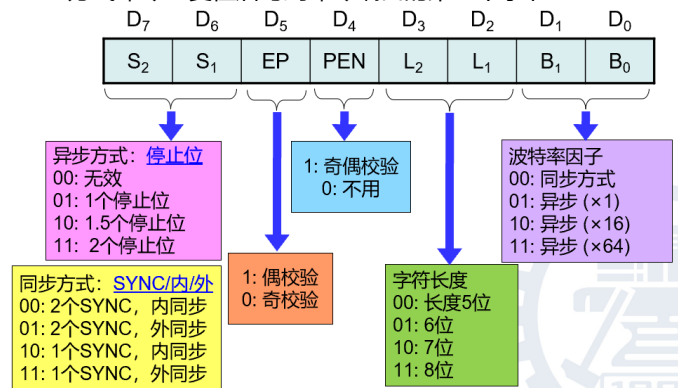
$\overline{WR_2}$  (写信号 2) 和  $\overline{XFER}$  (传递控制) 两个信号组合控制第二级缓冲寄存器的锁存

例: 软件程序, 指定 8255 的 B 口控制 DAC 的转换

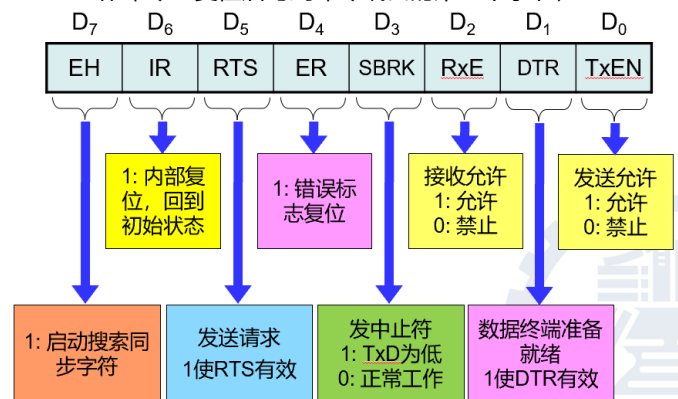
```
MOV DX, 301H ; 8255A 的 B 口地址
MOV AL, 00010000B ; 置 DAC0832 为直通工作方式
; ILE 置为 1, CS/WR1/WR2/XFER 均置为 0
OUT DX, AL
```

## 通用同步/异步接收发送器 8251A

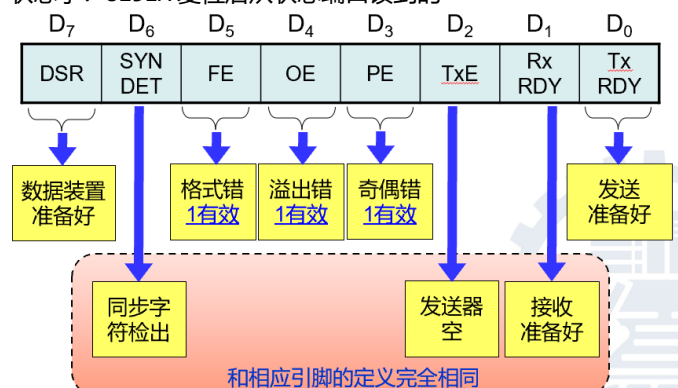
8251 方式命令: 复位后写到命令端口的第一个字节



8251 工作命令: 复位后写到命令端口的第二个字节;



状态字: 8251A 复位后从状态端口读到的



\* 后三章串行、PCI、和 USB 没记, 有需要翻书和复习题最后



逻辑地址和物理地址

逻辑地址	内容	物理地址
段地址：偏移地址		
1000：0000H	10011111	10000H
1000：0001H	00100110	10001H
1000：0002H	01001000	10002H
1000：0003H	10000011	10003H
1000：0004H	01011100	10004H
1000：0005H	10100010	10005H

8086 通用寄存器的专门用途

寄存器	用 法
AX	字乘法，字除法，字I/O
AL	字节乘法，字节除法，字节I/O，十进制算术运算
AH	字节乘法，字节除法，DOS中断功能号
BX	存储器指针
CX	串操作或循环控制计数器
CL	移位计数器
DX	字乘法，字除法，间接I/O
SI	存储器指针(串操作中的源指针)
DI	存储器指针(串操作中的目的指针)
BP	存储器指针(存储堆栈指针)
SP	堆栈指针

有符号数在 8086 中默认采用补码，最高位表示符号：正数用 0，负数用 1；负数的写法：其绝对值的二进制取反加一（即正数求补码）

1111 1111B   ;-1  
0000 0001B   ;1  
1111 1011B   ;-5

附录 1：十进制的 ASCII 码表

0FFH 或 255 是 res

000: null	032: spa	064: @	096: '
001: ☺	033: !	065: A	097: a
002: ☹	034: "	066: B	098: b
003: ♥	035: #	067: C	099: c
004: ♦	036: \$	068: D	100: d
005: ♣	037: %	069: E	101: e
006: ♠	038: &	070: F	102: f
007: beep	039: ' &	071: G	103: g
008: back	040: (	072: H	104: h
009: tab	041: )	073: I	105: i
010: newl	042: *	074: J	106: j
011: ♂	043: +	075: K	107: k
012: ♀	044: ,	076: L	108: l
013: cret	045: -	077: M	109: m
014: ♪	046: .	078: N	110: n
015: ☼	047: /	079: O	111: o
016: ▼	048: 0	080: P	112: p
017: ▲	049: 1	081: Q	113: q
018: ↑	050: 2	082: R	114: r
019: ↓	051: 3	083: S	115: s
020: ☸	052: 4	084: T	116: t
021: §	053: 5	085: U	117: u
022: ⇄	054: 6	086: V	118: v
023: ⇅	055: 7	087: W	119: w
024: ⇆	056: 8	088: X	120: x
025: ↗	057: 9	089: Y	121: y
026: →	058: :	090: Z	122: z
027: ←	059: ;	091: [	123: {
028: ↖	060: <	092: \	124:
029: ↗	061: =	093: ]	125: }
030: ↘	062: >	094: ^	126: ~
031: ▲	063: ?	095: _	127: Δ

附录 2：16 进制的 ASCII 码表

00: null	20: spa	40: @	60: '
01: ☺	21: !	41: A	61: a
02: ☹	22: "	42: B	62: b
03: ♥	23: #	43: C	63: c
04: ♦	24: \$	44: D	64: d
05: ♣	25: %	45: E	65: e
06: ♠	26: &	46: F	66: f
07: beep	27: ' &	47: G	67: g
08: back	28: (	48: H	68: h
09: tab	29: )	49: I	69: i
0A: newl	2A: *	4A: J	6A: j
0B: ♂	2B: +	4B: K	6B: k
0C: ♀	2C: ,	4C: L	6C: l
0D: cret	2D: -	4D: M	6D: m
0E: ♪	2E: .	4E: N	6E: n
0F: ☼	2F: /	4F: O	6F: o
10: ▼	30: 0	50: P	70: p
11: ▲	31: 1	51: Q	71: q
12: ↑	32: 2	52: R	72: r
13: ↓	33: 3	53: S	73: s
14: ☸	34: 4	54: T	74: t
15: §	35: 5	55: U	75: u
16: ⇄	36: 6	56: V	76: v
17: ⇅	37: 7	57: W	77: w
18: ⇆	38: 8	58: X	78: x
19: ↗	39: 9	59: Y	79: y
1A: →	3A: :	5A: Z	7A: z
1B: ←	3B: ;	5B: [	7B: {
1C: ↖	3C: <	5C: \	7C:
1D: ↗	3D: =	5D: ]	7D: }
1E: ↘	3E: >	5E: ^	7E: ~
1F: ▲	3F: ?	5F: _	7F: Δ