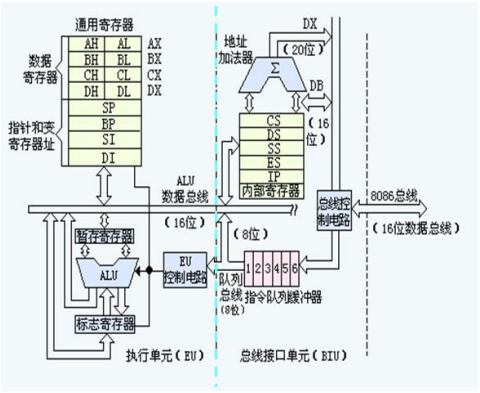
2^{16}=64KB=65535 2^{20}=1MB 2^{24}=16MB 2^{32}=4GB 2^{46}=64TB

第一章 微机系统导论

1.1 微型计算机系统（MCS）包括硬件和软件；硬件包括微型计算机（MC）、外围设备和电源；微型计算机（MC）包括微处理器（MP）、存储器、I/O 设备和系统总线；微处理器（MP）包括算术逻辑单元（ALU）、控制单元（CU）和寄存器阵列（RA）；系统总线包括**地址总线(AB)、数据总线(DB)和控制总线(CB)**

系统总线：实现 CPU、I/O 设备、和存储器之间相互连接的总线

1.2 微处理器(算术逻辑单元(ALU):操作数来自累加器 A 和内部数据总线;控制部件:指令寄存器 IR,指令译码器 ID,可编程逻辑阵列 PLA;内部寄存器: 累加器 A,数据寄存器 DR,程序计数器 PC,地址寄存器 AR,标志寄存器 F);微型计算机(主机);根据总线结构组织方式的不同,可以将总线结构分为:单总线结构,双总线结构,双重总线结构,分为地址总线,数据总线和控制总线,分别用于传输地址,数据和控制信息.面向系统的总线结构);微型计算机系统(硬盘显示器是外设)



算术逻辑部件（ALU）：执行算术和逻辑操作以及循环移位等。参加运算的两个操作数，一般来自累加器 A(Accumulator) 和内部数据总线,可以是数据寄存器 DR(Data Register)中的内容，也可以是寄存器阵列 RA 中某个寄存器的内容。运算结果送回累加器 A 暂存。

控制部件：产生一定的时序，控制指令所规定的操作的执行。

(1) 指令寄存器 IR(Instruction Register)存放从存储器取出的将要执行的指令。

(2) 指令译码器 ID(Instruction Decoder)对指令寄存器 IR 中的指令进行译码，确定该指令应执行什么操作。

(3) 可编程逻辑阵列 PLA(Programmable Logic Array)产生取指令和执行指令所需的各种微操作控制信号。

内部寄存器：

累加器 A：累加器是用得最频繁的一个寄存器。在进行算术逻辑运算时，具有双重功能：运算前，用来保存一个操作数；运算后，用来保存结果。

数据寄存器 DR：数据寄存器 DR 用来暂存数据或指令。 从存储器读出时，若读出的是指令，经 DR 暂存的指令通过内部数据总线送到指令寄存器 IR;若读出的是数据, 则通过内部数据总线送到有关的寄存器或运算器。向存储器写入数据时,数据是经数据寄存器 DR，再经数据总线 DB 写入存储器的。

程序计数器 PC(Program Counter)：程序计数器 PC 中存放着正待取出的指令的地址。根据 PC 中的指令地址，准备从存储器中取出将要执行的指令。通常，程序按顺序逐条执行。任何时刻，PC 都指示微处理器要取的下一个字节或下一条指令(对单字节指令而言)所在的地址。因此,PC 具有自动加 1 的功能。

地址寄存器 AR(Address Register)：地址寄存器 AR 用来存放正要取出的指令的地址或操作数的地址。在取指令时，将 PC 中存放的指令地址送到 AR，根据此地址从存储器中取出指令。在取操作数时，将操作数地址通过内部数据总线送到 AR，再根据此地址从存储器中取出操作数；在向存储器存入数据时，也要先将待写入数据的地址送到 AR，再根据此地址向存储器写入数据

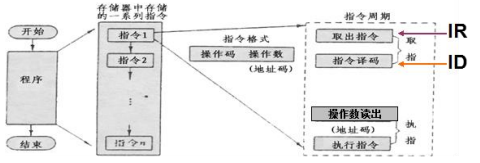
标志寄存器 F(Flag Register)：标志寄存器 F 用来寄存执行指令时所产生的结果或状态的标志信号。关于标志位的具体设置与功能将视微处理器的型号而异。根据检测有关的标志位是 0 或 1,可以按不同条件决定程序的流向。

存储器：是微机中的存储和记忆部件，用来存放用二进制代码形式表示的数据和程序。

字节（byte）：通常将 8 位二进制码作为一个字节。

字（word）：通常将两个字节也就是 16 位称为一个字。

字长：表示计算机数据总线上一次能处理的信息的位数即位长，并由此而定义是多少位的计算机,如 1 位机，4 位机、8 位机、16 位机、32 位机等。



1.3 冯.诺依曼型数字计算机工作原理:计算机系统由运算器,控制器,存储器,输入和输出设备组成.指令组成:操作码+操作数

第三章 8086/8088 微处理器及系统

3.1.1 内部结构组成：总线接口单元（BIU）和执行单元（EU）

BIU：4 个 16 位的段寄存器（CS、DS、SS、ES）和 16 位指令指针 IP

3.13 EU：16 位 ALU、16 位标志寄存器 F、数据暂存寄存器、16 位通用寄存器

数据寄存器：AX、BX、CX、DX；AX：用作累加器；BX：多用作基址寄存器；CX：多作为计数器；DX：多用作辅助累加器

指针寄存器 SP 堆栈指针寄存器、BP 基址指针寄存器；变址寄存器 SI 源变址寄存器、DI 目的变址寄存）。

EU 控制电路

状态标志位：

CF：产生进位或者借位时为 1

PF：结果低 8 位中含有偶数个 1 时为 1

AF：D3 向 D4 进位或者借位时为 1

ZF：结果为 0 时为 1

SF：与结果的最高位相同

OF：有符号数加减法溢出时为 1

总线周期：4 个周期组成，在 T3 后可插于 1 个或多个等待状态 TW，CPU 发出

READY 信号后进入 T4。地址信息：T1；状态信息：T3

规则字：存放字的低字节从偶地址开始存放，可一个周期完成存取

8086 中低位库（512K）与数据总线 D7~D0 相连，该库中每个地址为偶地址；高位库与 D15~D8 相连，该库中每个地址为奇地址逻辑地址：段地址和偏移地址

3.1.5 T1 为地址周期.CPU 通过地址/数据(或地址/状态)复用总线发出地址信息,指示要寻址的存储器单元或者 I/O 的地址.T2 为缓冲周期.例如,在总线读周期,CPU 在 T2 撤销低 16 位地址信号,使该组信号线置为高阻态,准备接收存储器或 I/O 的数据.T3 为数据周期. 数据出现在复用总线的低 16 位上.如果外设或存储器没有准备好,CPU 会在 T3 周期后插入等待周期 TW.T4 总线周期结束.

3.2.1 8086 控制信号 **BHE*/S7**:数据总线高 8 位使能和状态复用信号,在总线周期 T1 状态,BHE*有效,表示数据线上高 8 位数据是有效的,在 T2～T4 状态,BHE*/S7 输出状态信息 S7.S7 在 8086 中未做实际定义.**RD***:读信号,输出,三态,低电平有效,RD*信号有效,表示 CPU 执行一个对存储器或 I/O 端口的读操作,在一个读操作的总线周期中,RD*在 T2～T3 状态中有效,为低电平.**READY**:准备好信号,输入, 高电平有效.CPU 在每个总线周期的 T3 状态对 READY 进行采样.当 READY 信号有效时表示存储器或 I/O 准备好发送或接收数据.CPU 在 T3 采样到 READY 为低电平以后,便在 T3 之后插入 Tw,延长读写周期,使 CPU 能和较慢速度的存储器或 I/O 接口相匹配.**TEST***:测试信号,输入,低电平有效.TEST*信号和 WAIT 指令结合起来使用,在 CPU 执行 WAIT 指令时,CPU 便一直处于空转状态,进行等待.只有当 8086 检测到 TEST*信号有效时,才结束等待状态,继续执行 WAIT 之后的指令. **NMI**:非屏蔽中断请求,输入,上升沿有效.NMI 不受中断允许标志的影响.当 CPU 检测到 NMI 有一个上升沿触发的信号以后,CPU 执行完当前指令便响应中断类型号为 2 的非屏蔽中断请求.**INTR**:可屏蔽中断请求,输入,高电平有效.如果 INTR 信号有效,当 CPU 的中断允许标志 IF=1 时,CPU 结束当前指令后,响应 INTR 中断.**RESET**:复位信号,输入,高电平有效.复位信号有效时,CPU 结束当前操作并对标志寄存器 FLAG,IP,DS,SS,ES 及指令队列清零,并将 CS 设置为 FFFFH.当复位信号撤除时,(即电平由高变低时)CPU 从 FFFF0H 开始执行程序.**CLK**:时钟信号,输入.为 CPU 和总线控制逻辑提供定时.要求时钟信号的占空比为 33%.

最小工作方式:系统中只有一个微处理器,所有的总线控制信号都直接由 8086/8088 产生. 最大工作模式:最大模式系统总线控制信号由 CPU 和总线控制器(如 8288)联合产生,多用于包含了两个或者两个以上处理器的系统,如包含了协处理器 8087.

3.3.1 物理地址是由 8086 的地址引线送出的 20 位地址码。这 20 位地址码送到存储器经过译码，最终选定一个存储单元进行读/写。物理地址可写成 5 位的十六进制数。

偏移地址是相对于某段首地址的段内偏移量，用 16 位二进制代码表示，写成 4 位十六进制数，例如：004AH。

逻辑地址是在程序中对存储器地址的一种表示方法，由某段的段地址和段内偏移地址组成。写成：段地址 ： 偏移地址

例如：2000H：0080H

物理地址 PA：段寄存器十六进制后添 0+偏移量（EA）

有效地址 EA=基地址（BX/BP）+变地址（SI/DI）+位移量 D（0/8/16）

3.3.3 堆栈：SS 堆栈段的段基址，SP 栈顶偏移地址；压栈时放入 SP-1 和 SP-2 两个地址 ,出栈时弹出 SP 和 SP+1 两个地址

3.4.2 寻址方式

固定寻址：例：AAA 规定被调整的数总位于 AL 中

立即数寻址：MOV AX，‘AB’；送 BA 的 ASCII 码

1 立即数只能用于源操作数 SRC 字段

2 SRC 和 DST 的字长一致 MOV AH, 3064H

寄存器寻址

1 字节寄存器只有 AH AL BH BL CH CL DH DL

2 SRC 和 DST 的字长一致 MOV AH, BX ×

存储器寻址：

直接寻址：

1 隐含的段为数据段

2 如不在数据段中，可以用段跨越前缀 MOV AX, ES:[2000H]

3 操作数地址可以由变量（符号地址）表示

COUNT DW 1000H

MOV AX, COUNT

Eg：MOV AX,ES:[1680H]；赋两个字节

MOV AX,NUM；赋符号地址 NUM 内容

基址寻址：操作数的有效地址由基址寄存器(BX/BP)和指令中的偏移量给出（只允许）

偏移地址 = (BX)/(BP)+8 位/16 位偏移量

MOV AX, [BX] PA=16d×(DS)+(BX)

MOV AX, ES: [BX] PA=16d×(ES)+(BX)

MOV AX, [BP] PA=16d×(SS)+(BP)

MOV AL , [BX+1000H] PA=16d×(DS)+(BX+1000H)

BX，数据段 DS；只要含有 BP，堆栈段 SS

变址寻址：操作数的有效地址 EA 由变址寄存器(SI/DI)和指令中的偏移量给出，数据段 DS

偏移地址 = (SI)/(DI)+8 位/16 位偏移量

基址加变址寻址：必须用 BYTE PTR、WORD PTR 或 DWORD PTR 伪指令以确定类型

$$\text{地址}=\begin{cases}(BX)\\(BP)\end{cases}+\begin{cases}(SI)\\(DI)\end{cases}+\begin{cases}8\text{位}\\16\text{位}\end{cases}\text{位移量}$$

例：MOV AX,[BP][DI]

串操作指令寻址（数据串）。

I/O 端口寻址： IN/OUT

直接端口寻址：端口号在指令中以 8 位立即数方式直接给出，其范围是 00-FFH

间接端口寻址：端口号在 DX 寄存器中给出，其范围是 0000-FFFFH。

转移类指令寻址：段内/段间 + 直接/间接 转移

JMP 指令

数据传送指令

(1) 通用数据传送指令 (2)目标地址传送指令

MOV	传送	LEA	有效地址送寄存器
PUSH	进栈	LDS	指针送寄存器和 DS
POP	出栈	LES	指针送寄存器和 ES
XCHG	交换		
XLAT	换码		

(3)标志寄存器传送指令 (4) I/O 数据传送

LAHF	标志送 AH	IN	输入
SAHF	AH 送标志寄存器	OUT	输出
PUSHF	标志进栈		
POPF	标志出栈		

（1）通用数据传送指令

MOV d,s[MOV ES,2000H×; MOV ES, DS×]

●目标和源操作数不能同时用存储器寻址方式（双操作数指令全部适应）

●SRC 和 DST 的字长一致。 MOV BX, AL ×

● 目标操作数不能是 CS，不能用立即数方式。 MOV CS, AX ×

● 立即数不能直接送到段寄存器。 MOV ES, 2000H ×

● 目标和源操作数不允许同为段寄存器。 MOV ES, DS ×

● MOV 指令不影响标志位.

例：MOV [SI],IP, ×指令指针 IP 不能传送

PUSH s/POP d 低到低，高到高

● PUSH 和 POP 指令只能是字操作，因此存取字数据后，SP 的修改必须是 + 2 或 - 2；

● PUSH 和 POP 指令不能使用立即数寻址方式；

● POP 指令的 dst 不允许是 CS 寄存器；

●不影响标志位。

XCHG dst, src; 交换指令（exchange）

一个操作数必须在寄存器上；不允许使用段寄存器

(2) 目标地址传送指令

LEA reg, src ;有效地址送寄存器，执行操作:(reg)←offset of src

LEA 指令把源操作数的有效地址送到指定的寄存器 ,这个有效地址是由 src 选定的一种存储器寻址方式确定的。

LDS reg, src ; 指针送寄存器和 DS 执行操作: (reg)←(src) ;(DS)← (src+2)

LES reg, src ; 指针送寄存器和 ES 执行操作: (reg)←(src) ; (ES)←(src+2)

LDS 和 LES 指令把偏移地址送寄存器，段地址送 DS 或 ES。这个偏移地址和段地址（也称地址指针）是由 src 指定的两个相继字单元提供的。

(3) 标志寄存器传送指令

LAHF 标志寄存器的低字节送 AH，执行操作: (AH) ← (FLAGS)0-7

SAHF AH 送标志寄存器低字节，执行操作: (FLAGS) 0-7 ← (AH)

PUSHF 标志进栈，执行操作: (SP) ← (SP) - 2

((SP)+1,(SP)) ← (FLAGS)0-15

POPF 标志出栈，执行操作: (FLAGS) 0-15 ← ((SP)+1,(SP))
(SP) ← (SP)+2

- LAHF 和 SAHF 指令隐含的操作寄存器是 AH 和 FLAGS
- LAHF 和 PUSHF 不影响标志位，SAHF 和 POPF 则由装入的值来确定标志位的值。

(4) I/O 数据传送

IN 累加器, 端口号(port)； 输入指令 (input) , port≤0FFH
执行操作: (AL) ← (port) 传送字节
或 (AX) ← (port+1,port) 传送字
IN 累加器, DX; 输入指令, DX 中的 port 在 0 和 0FFFFH 之间
执行操作: (AL) ← ((DX)) 传送字节
或 (AX) ← ((DX)+1,(DX)) 传送字
OUT 端口号(port), 累加器 ；输出指令 (output) , port≤0F
执行操作: (port) ← (AL) 传送字节
或 (port+1,port) ← (AX) 传送字
OUT DX, 累加器 ；输出指令, DX 中的 port 在 0 和 0FFFFH 之间
执行操作: ((DX)) ← (AL) 传送字节
或 ((DX)+1,(DX)) ← (AX) 传送字

- 只限于在 AL 或 AX 与 I/O 端口之间传送信息
- 不影响标志位

算术运算指令

(1) 加法指令

ADD d,s 加法 目标不能为立即数，不能同时为存储器
ADC d,s 带进位加 d=d+s+CF
INC d 加 1 不能为立即数

(2) 减法指令

SUB d,s 减法
SBB d,s 带借位减 d=d-s-CF
DEC d 减 1
NEG d 求补 将目标操作数当做有符号数处理;
CMP d,s 比较

(3) 乘法指令

MUL s 无符号数乘法
执行操作 : 字节操作: (AX) ← (AL)×(src)
字操作: (DX, AX) ← (AX)×(src)
注意 :

- AL(AX)为隐含的被乘数寄存器
- AX(DX,AX)为隐含的乘积寄存器
- SRC 不能为立即数
- 除 CF 和 OF 外，对其它状态标志位无意义

IMUL s 带符号数乘法
(4) 除法指令
DIV src 无符号数除法 执行操作 :
字节操作: (AL) ← (AX) / src 的商
(AH) ← (AX) / src 的余数
字操作: (AX) ← (DX, AX) / src 的商
(DX) ← (DX, AX) / src 的余数
IDIV src 带符号数除法 (signed divide)
IDIV 指令执行的操作与 DIV 相同，但操作数必须是带符号数，商和余数也均为带符号数，而且余数的符号与被除数的符号相同。

- AX(DX,AX)为隐含的被除数寄存器
- AL(AX)为隐含的商数寄存器
- AH(DX)为隐含的余数寄存器
- SRC 不能为立即数
- 对所有状态标志位均无定义

●除法指令要求字操作时，被除数必须为 32 位，除数是 16 位，商和余数是 16 位的
●字节操作时，被除数必须为 16 位，除数是 8 位，得到的商和余数是 8 位的。
●除数为 0 或商溢出等错误，由系统直接转入 0 型中断来处理。商溢出，是指被除数高一半的绝对值大于除数的绝对值时，商超出了 16 位的表示范围 (字操作) 或 8 位的表示范围 (字节操作)。
(5) 符号扩展指令
CBW 字节扩展为字
执行操作 : (AH) = 00H 当 (AL) 的最高有效位为 0 时
(AH) = FFH 当 (AL) 的最高有效位为 1 时
CWD 字扩展为双字
执行操作 : (DX) =0000H 当 (AX) 的最高有效位为 0 时
(DX) =FFFFH 当 (AX) 的最高有效位为 1 时
这是两条无操作数指令，进行符号扩展的操作数必须存放在 AL 寄存器或 AX 寄存器中。这两条符号扩展指令都不影响状态标志位。
●除法指令要求字操作时，被除数必须为 32 位；字节操作时，被除数必须为 16 位。可以用上述两条指令对带符号数除法的被除数进行扩展。
(6) 十进制调整指令[将 ASCII 码表示的数转换成 BCD 码只要把 ASCII 码的高 4 位清零]
压缩的 BCD 码调整指令 :
DAA 加法的十进制调整 DAS 减法的十进制调整 (在 AX 中操作，不必写操作数)
非压缩 BCD 码调整 :
AAA 加法的 ASCII 调整，AAS 减法 (影响 AF,CF)
(AL) ←把 AL 中的 ASCII 码结果调整为非压缩的 BCD 格式
(AH) ← (AH) +/-调整产生的进/借位值
AAM 乘法，AAD 除法的 ASCII 调整(AX, 影响 SF、ZF 和 PF)
●先加减乘除后调整，不必写操作数

逻辑运算和移位循环指令

(1) 逻辑运算指令

AND dst,src ；逻辑与 (logic and)
OR dst,src ；逻辑或 (logic or)
NOT opr ；逻辑非 (logic not)
XOR dst,src ；异或 (exclusive or)
TEST opr1,opr2 ；测试 (test)

(2) 移位指令和循环移位指令 :

算术移位 :带符号数进行移位，其右移操作在移位过程中必须保持符号不变；
逻辑移位 : 对无符号数移位，总是用 0 来填补已空出的位置。
循环移位 : 指令是将操作数首尾相接进行移位，它分为不带进位位和带进位位循环移位。指令中的目的操作数 dst 可以是除立即数外的任何寻址方式。
移位指令
SHL dst,cnt ; 逻辑左移
执行操作: 最低位用 0 来补充 ,最高位移入 CF
SHR dst,cnt ; 逻辑右移
执行操作:最高位用 0 来补充 ,最低位移入 CF
SAL dst,cnt ; 算术左移
执行操作:最低位用 0 来补充 ,最高位移入 CF
SAR dst,cnt ; 算术右移
执行操作 符号位值补充最高位,最低位移入 CF
ROL dst,cnt ; 循环左移
执行操作:最高位移入 CF 和最低位
ROR dst,cnt ; 循环右移
执行操作:最低位移入 CF 和最高位
RCL dst,cnt ; 带进位循环左移
执行操作:CF 移入最低位，最高位移入 CF
RCR dst,cnt ; 带进位循环右移
执行操作:CF 移入最高位，最低位移入 CF

(1) 串处理指令

MOVS 目标串，源串；串传送
MOVSB (目标串，源串); 字节传送
MOVSW (目标串，源串);字传送
执行操作:
(ES:DI)←(DS:SI);将源串中元素传送到对应的目标串元素中,
(SI)←(SI)±1 (字节) 或±2 (字) ;自动修改指针 SI/DI，使之指向下一个元素
(DI)←(DI)±1 (字节) 或±2 (字)
CMPSB / CMPSW (目标串,源串) ;串比较
执行操作 :
(DS:SI) - (ES:DI) ；源串中元素减去目标串中对应元素，不回送结果，只根据比较的结果设置状态标志位
(SI)←(SI)±1 (字节) 或±2 (字) ;自动修改指针 SI/DI，使之指向下一个元素
(DI)←(DI)±1 (字节) 或±2 (字)
SCASB / SCASW (目标串); 串搜索
执行操作 :
(AL) - (ES:DI)或(AX) - (ES:DI);AX 或者 AL 中的关键字减去目标串元素，不传送结果，只根据搜索比较的结果设置状态标志位
(DI)←(DI)±1 (字节) 或±2 (字) ;自动修改指针 DI，使之指向下一个元素
LODSB / LODSW (源串)； 读串
执行操作 :
(AL)或(AX)←(DS:SI) ；将源串所指元素取入 AX/AL 寄存器中
(SI)←(SI)±1 (字节) 或±2 (字) ; 自动修改指针 SI，使之指向下一个元素
STOSB / STOSW (目标串) ; 写串
执行操作 :
(ES:DI)←(AL)或(AX) 将 AX/AL 寄存器中的内容写入目标串所指元素中
(DI)←(DI)±1 (字节) 或±2 (字) ; 自动修改指针 DI，使之指向下一个元素

(2) 串重复前缀

REP 重复执行串指令，(CX)=重复次数
① (CX)=0 时，串指令执行完毕，否则执行② ~ ④
② (CX)←(CX) - 1
③ 执行串指令 (MOVS 或 STOS)
④ 重复执行①
REPE / REPZ 相等/为零时重复执行串指令，(CX)=比较/扫描的次数
① (CX)=0 或 ZF=0 时，结束执行串指令，否则继续② ~ ④
② (CX)←(CX) - 1
③ 执行串指令 (CMPS 或 SCAS)
④ 重复执行①
REPNE / REPNZ 不等/不为零时重复执行串指令，(CX)=比较/扫描的次数
① (CX)=0 或 ZF=1，结束执行串指令，否则继续② ~ ④
② (CX)←(CX) - 1
③ 执行串指令 (CMPS 或 SCAS)
④ 重复执行①
例题 CLD;DF=0,地址自动递增 MOV CX, 100;串的长度 MOV SI, 2500H\MOV DI, 1400H\REP MOVSB;重复传送直到 CX=0

5.程序控制

JMP a(目标标号)/CALL a/RET 弹出值;从过程返回/JA a; 高于/JAE a/JB a;低于(无符号数)/JBE a/JG a;大于/JGE a/JL a;小于 /JLE a/JC a;进位/JNC a/JE(JZ) a;等于,结果为 0/JNE(JNZ) a/LOOP a;循环,CX 减 1 后判断;INT a;IRET;中断返回段内短 JMP SHORT LABLE;段内近(缺省) JMP LABLE;段内间接 JMP reg\JMP WORD PTR OPR;段间直接 JMP FAR PTR label;段间间接 JMP DWORD PTR [SI];所有条件转移都是短转移
例题段内直接间接转移
设 DS=2000h,[21020h]=34h,[21021h]=12h 程序:MOV BX,1000h\JMP BX;转向 CS:1000h\JMP WORD PTR [BX+20h];程序转向 CS:1234h 例题带立即数返回 CODE SEGMENT\MAIN PROC FAR\...\PUSH AX\PUSH BX\CALL SUB\...\RET\MAIN ENDP\SUB PROC NEAR\....\RET 4\SUB ENDP\CODE ENDS

6.处理器控制指令

CLD DF=0 正向/STD/CLI 禁止可屏蔽中断 IF=0/STI/HLT 暂停/NOP 空操作(3 周期)

(1) 标志位处理指令

CLC CF 置 0
STC CF 置 1
CMC CF 求反
CLD DF 置 0
STD DF 置 1
CLI IF 置 0
STI IF 置 1

(2) 同步控制指令

WAIT 等待
ESC 交权
LOCK 封锁

(3) 其它

NOP 空操作
HLT 暂停

例题 CLD;设置方向标志位,DF=0,地址自动递增\MOV CX,100;设置计数值\MOV SI,6180H;设置源变址寄存器\MOV DI,2000H;设置目的变址寄存器\REP MOVSB;实现数的移位传送\MOV CX,100;重新设置计数值\MOV DI,2000H;设置目的变址寄存器\S1: SCASB;检索出等于 AL 中的字符单元\JNZ S2;不相等,跳转到 S2\MOV BYTE PTR[DI-1],20H;如果相等,将该单元值换成空格符\S2: LOOP S1;继续检索,如果还存在其它相等字符进行替换\HLT

第四章 汇编语言程序设计

DB：1 字节；**DW**：2 字节；**DD**：4 字节；**DQ**：8 字节；**DT**：10 字节例 1 :(DW：16 位偏移地址，DD：16 位基址和 16 位偏移地址)

```
FOO SEGMENT AT 55H                ;段基址为 55H
ZERO DB 0
ONE DW ONE                        ;内容为 0001H
TWO DD TWO                        ;内容为 0055,0003H
FOUR DW FOUR+5                    ;内容为 7+5=12
SIX          DW ZERO-TWO          ;内容为 0-3=-3
ATE          DB 5*6               ;内容为 30
FOO ENDS 例 2：STR1 DB 'AB'；将 A 放低字节；STR2 DW 'AB'；反之
```

4.2.1

每个段都有一个名字(叫段名),以 SEGMENT 作为段的开始,以 ENDS 作为段的结束,这两者(伪指令)前面都要冠以相同的名字。段可以从性质上分为代码段、堆栈段、数据段和附加段 4 种,但代码段与堆栈段是不可少的,数据段与附加段可根据需要设置。

4.2.2

1.指令语句的格式 :
[标号:] [前缀] 指令助记符 [操作数表] [;注释]
伪指令语句的格式 :
[名字] 伪指令助记符 [参数表] [；注释]
标号
1) NEAR (近)
●表示本标号只能被标号所在段内的转移和调用指令访问 (即段内转移)
2) FAR (远)
●表示本标号可以被其他段 (不是标号所在段) 的转移和调用指令访问 (即段间转移)，也可为段内转移。

3. 操作数

- 当基址寄存器为 BP 时，相对应的段寄存器为 SS (堆栈段寄存器)。
- 对串操作，源串起始地址为 DS:SI，允许使用段超越前缀修改源串段地址。目的串起始地址为 ES:DI，禁止用段超越前缀修改目的串地址 ES。
- 可以采用“段超越”前缀，来改变段寄存器。

ADD AL, SS:[DI+3]

运算符

算术运算符 : + 、 - 、*、/、MOD、SHR、SHL
逻辑运算符 : AND、OR、XOR、NOT
关系运算符 : EQ(=),NE(≠),LT(<),GT(>),LE(≤),GE(≥),当关系成立的时候，结果为 FFFFH，否则为 0.
“变量”和“标号”的区别
变量指向的是数据段或者附加段中的某个数据项 ,标号指向的是代码段某条指令
变量的类型是数据项存取单位的大小，标号的类型是 NEAR 或 FAR
地址表达式的运算规则 :
1) 变量或者标号加上或者减去某个结果为整数的数值表达式，其结果仍为变量或者标号，指向某存储单元。
2) 同一段内的变量或者标号可以相减，其结果不是地址，而是一个数值，该数值表示两者间相距的字节数。
合成运算符(Synthetic operators) : PTR,THIS
PTR:指定操作数的类型属性，它优先于隐含的类型属性。其格式为 :
类型 PTR 变量[± 常数表达式]
其中类型可以是 BYTE、WORD、DWORD、FWORD、QWORD 或 TBYTE，这样变量的类型就可以指定了。
THIS:建立指定类型的存储器地址或者操作数，但不分配存储区。

```
例 :  DATAB  EQU THIS BYTE
      DATAW  DW ?
```

1. DB 是唯一能定义字符串的伪操作，字符串用 ' ' 括起来，串中的每个字符占用一个字节，在存储器内用相应的 ASCII 码表示。
2.当 $ 用在伪指令的参数字段时，它所表示的是地址计数器的当前值
操作数为地址表达式 : 数据定义时，出现在地址表达式中的变量表示该变量的偏移地址 ,用根据偏移地址计算出的地址表达式的值初始化相应的存储单元。
格式 1 : DW 地址表达式
功能 : 利用该地址表达式的计算值初始化相应的存储字
格式 2: DD 地址表达式
功能 : 地址表达式计算结果对应的段地址(高位字)和偏移地址 (低位字) 来初始化相应的两个存储字。

```
名字  EQU  表达式
名字  =    表达式
变量名/标号名 LABEL 类型
```

1. EQU 伪操作不允许重复定义

```
TMP     EQU 5
TMP     EQU TMP+1       ×
```

2.在 EQU 语句的表达式中，如果有变量或标号的表达式，则在该语句前应该先给出它们的定义。

伪指令语句

```
段名  SEGMENT  [定位类型] [组合类型] [ '类别' ]
  .
  .
  .
段名  ENDS
```

注意：

（1）SEGMENT 和 ENDS 成对出现

（2）段名由用户命名。

（3）段体：对于数据段、附加段和堆栈段来说，段体内一般是存储单元的定义、分配等伪指令语句；对于代码段中则主要是指令及伪指令语句。

（4）定位类型：表示该段对起始边界地址的要求：

BYTE 起始地址 = XXXXXXXXXXXXXXXX，即字节型;

WORD 起始地址 = XXXXXXXXXXXXXXX0，即字型;

PARA 起始地址 = XXXXXXXXXXXX0000，即节型;

PAGE 起始地址 = XXXXXXXXX0000000，即页型。

其中PARA为缺省值，即如果省略“定位类型”，则汇编程序按PARA处理。

组合类型	说 明
NONE(缺省)	该段为私有段，连接时将不与其它模块中的同名段合并
PUBLIC	该段连接时将与其它同名段连接在一起，连接次序由连接命令指定
COMMON	该段在连接时与其它同名段有相同的起始地址，所以会产生覆盖
AT表达式	可以用于指定段地址，段地址=表达式的值，其值必为16位,但AT不能用来指定代码段
MEMORY	该段在连接时被放在所有段的最后（地址最高部分），如果几个段均为该类型，则认为第一个为MEMORY类型，其余为COMMON类型
STACK	将多个同名堆栈段连接在一起，SP设置在第一个堆栈段的开始，SS和SP一起指向堆栈段的开始位置

ASSUME 伪指令：就是建立段和段寄存器关系的伪指令

格式：ASSUME 段寄存器名：段名，...

例： ASSUME CS: SEGA, DS: SEGB, SS:NOTHING

*** ASSUME CS:SEGA** 表示 CS 被设定为以 SEGA 为段名的代码段的段地址寄存器

注意：

1. 其中段寄存器名必须是 CS、DS、ES 和 SS 中的一个，而段名必须是由 SEGMENT 定义的段名。

2. 由于 ASSUME 伪指令只是指定某个段分配给哪一个段寄存器，它并不能把段地址装入段寄存器中，所以在代码段中，还必须把段地址装入相应的段寄存器中

3. 在程序中不需要用指令装入代码段的段地址，因为在程序初始化时，装入程序已将代码段的段地址装入 CS 寄存器了

ORG 伪指令：

ORG 伪指令用来表示起始的偏移地址，紧接着 ORG 的数值就是偏移地址的起始值。ORG 伪操作常用在数据段指定数据的存储地址，有时也用来指定代码段的起始地址。

***ORG** 用来指出其后的程序段或数据块存放的起始地址的偏移量

SEG：段地址；**OFFSET**：偏移地址；**SIZE**：数据字节总数；**LENGTH** 数据项总数

4. **过程定义伪指令**

格式：

```
过程名  PROC  Attribute
.....
过程名  ENDP
```

过程名：为一个标识符，是子程序入口的符号地址，与标号的作用相同。

属性：

（1）如果调用程序和过程在同一个代码段中，则使用 NEAR 属性。

（2）如果调用程序和过程不在同一个代码段中，则使用 FAR 属性。

分类：

（1）外部过程：调用该过程的主程序与该过程不在同一源文件中。此时需要在主程序文件中说明该过程(设过程名为 PROCD)：

```
EXTRN PROCD: FAR
```

同时在定义该过程的程序文件中说明该过程可以被其它程序文件调用：

```
PUBLIC PROCD
```

（2）内部过程：调用该过程的主程序和该过程在同一个源文件中。内部过程又分为段内过程和段外过程。

INT n 中断指令（interrupt），n 为中断类型号

执行操作：① 入栈保存 FLAGS：(SP) ← (SP) - 2，((SP)) ← (FLAGS)

② 入栈保存返回地址：(SP) ← (SP) - 2，((SP)) ← (CS)

(SP) ← (SP) - 2，((SP)) ← (IP)

③ 转中断处理程序：(IP) ← (n×4)

(CS) ← (n×4 + 2)

INTO 溢出则中断（中断类型为 4），相当于 INT 4

执行操作：若 OF=1（有溢出），则：

① 入栈保存 FLAGS：(SP) ← (SP) - 2，((SP)) ← (FLAGS)

② 入栈保存返回地址：(SP) ← (SP) - 2，((SP)) ← (CS)

(SP) ← (SP) - 2，((SP)) ← (IP)

③ 转中断处理程序：(IP) ← (4×4)= (10H)

(CS) ← (4×4 + 2)= (12H)

注意：

●根据中断类型号（n×4）到中断向量表中取得中断例行程序的入口地址，分别将对应的内容送给 IP 和 CS，可以以实现调用中断例行程序的功能。隐含的中断类型号为 3，即 INT 等价于 INT 3

● INT 指令（包括 INTO）执行后，把 IF 和 TF 置 0，但不影响其它标志位。

IRET 中断返回指令（return from interrupt）

执行操作：

① 返回地址出栈：(IP) ← ((SP))，(SP) ← (SP) + 2

(CS) ← ((SP))，(SP) ← (SP) + 2

② FLAGS 出栈：(FLAGS) ← ((SP))，(SP) ← (SP) + 2

调用 DOS 或 BIOS 中断功能时，有以下几个基本步骤：

1、 将调用参数装入指定的寄存器中；

2、 如需功能号，把它装入 AH；

3、 按中断号调用 DOS 或 BIOS 中断；

4、 检查返回参数是否正确。

注：中断向量分配情况

0～1FH, 80H～F0H 为 BIOS 中断向量号

20H～3FH 为 DOS 中断向量号

40H～7FH 为用户备用

功能	矢量号	功能号	备注
返回 DOS	21H	4CH	
设置中断向量	21H	25H	DS=段地址；DX=偏移地址；AL=中断类型码
键盘输入一字符并	21H	01H	AL=输入字符的 ASCII 码
显示			
字符串输入	21H	0AH	DS:DX 指向输入缓冲区，输入缓冲区第一个单元设置字符串最大输入长度；输入缓冲区第二个单元为实际输入字符串长度
显示字符	21H	02H	DL=显示字符的 ASCII 码
显示以'$'结尾的字符串	21H	09H	DS:DX 指向字符串的首地址