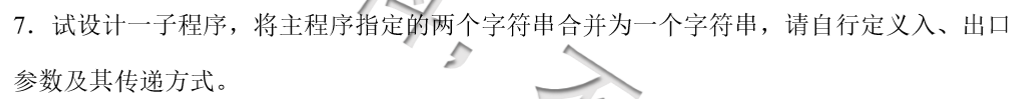
|  |  |  |
| --- | --- | --- |
| 指令用法 | 指令含义 | 指令注意事项 |
| MOV AL, [1000H] | 将DS:1000H中的数据取出并放入AL中 |  |
| MOV DS:[1000H], BX | 将BX中的数据放入DS:1000H中 |  |
| MOV VAR1+2, AX | 将AX的值放到VAR1所在的地址加上2的位置处 |  |
| MOV AL, [BP]100H | 以BP寄存器为基址，偏移量为100H的值传给AL |  |
| MOV BYTE PTR [SI]200H, 30H | 将立即数30H放到SI所在的地址+200H的偏移处 | BYTE PTR指明内存单元类型为字节类型，在无寄存器寻址的指令中需要指明内存单元的类型 |
| MOV AX, 30H  MOV VAR1, 59H  XCHG VAR1, AX | 将VAR1的值和AX的值进行交换 | XCHG指令不能直接使用立即数，也不能使用段寄存器 |
| PUSH SRC | 将SP的值减2使之指向一个空的栈顶，再将源操作数送至栈顶保存 | 操作数必须为16为的寄存器或内存单元 |
| POP DEST | 将栈顶字单元中的数据传送至目的操作数保存，降后将SP+2 | 操作数必须为16为的寄存器或内存单元，遵循后进先出 |
|  | | |
| PUSHF | 标志压栈指令，将16位标志寄存器的数据进行保存 |  |
| POPF | 标志出栈指令 |  |
| 假设[0100H]处数据为80H  MOV SI, [0100H]  LEA DI, [0100H]  SI的值为80H，DI的值为0100H | 装入有效地址指令  将源操作数的有效地址传送至目的操作数保存 | 源操作数只能是内存单元。MOV指令访问源操作数的数据，而LEA指令获取源操作数的有效地址 |
| LDS DEST,SRC  LES DEST, SRC  Eg.  LDS AX，（SI +20 ）  其中（20050H）=1234H，（20052H）=5678H，AX=0000H，DS=2000H，SI=0030H，  执行的结果为：AX=1234H，DS=5678H | 装入地址指针指令  将地址处的数据放入DEST中随后将地址＋2处的数值放入DS或ES中 |  |
| ADC DEST, SRC | 带进位加法指令  SRC+DEST+CF => DEST CF指令来源于上一次运算的结果 | 注意不能为 ADC VAR1, VAR2.必须有一个是寄存器，不能两个同时是内存单元 |
| SBB DEST, SRC | 带结尾减法指令  DEST – SRC -CF => DEST  CF指令来源于上一次运算的结果 |  |
| NEG DEST | 求相反数指令 | 只能处理16位的，要想处理更高位的需要按如下方式处理：  NEG AV  MOV AX, 0  SBB AX, AV+2  MOV VA+2, AX |
| CMP DEST, SRC | 比较指令  采用减法的方式对两个数进行比较但是不保存结果。 | 在运算过程中按照无符号数的思维运算。如果DEST<SRC的话则CF=1，如果DEST>SRC的话则CF=0.  如果OF=SF并且ZF=0则被减数大于减数;如果OF=SF并且ZF=1则被减数等于减数；如果OF!=SF则被减数小于减数 |
| XOR DEST, SRC | 逻辑异或指令  将指定的二进制位取反 | 无法直接对[SI]进行运算，即XOR [SI], 01H是不被允许的 |
|  |  |  |
| TEST DEST，SRC | 测试指令  类似于AND,CMP,SUB指令 | 不关注结果，对指定的单个二进制位进行分离，通过ZF标志判断所分离的二进制位的取值，从而实现跳转或循环 |
| SAL DEST, COUNT  SAR DEST, COUNT | 算术左/右移指令 | 若COUNT的值大于1，则必须借助：  MOV CL, 10H  SAL DEST, CL  完成。  当COUNT=1时OF才能被检测出来；若COUNT>1时OF无法判断  当做有符号数进行运算。看最高位的符号是否发生改变发生溢出 |
|  | | |
| SHL DEST, COUNT | 逻辑左移指令 | 将每一个移除的最高位放在CF中直到COUNT=0 |
|  | | |
| SHL DEST, COUNT | 逻辑右移指令 | 将每一个移除的最低位放在CF中直到COUNT=0  当做无符号数进行运算 |
| ROL DEST, COUNT  ROR DEST, COUNT | 循环左/右移指令 |  |
|  | | |
| RCL DEST, COUNT  RCR DEST, COUNT | 带进位的循环左/右移 |  |
| ADD BYTE PTR [0100H], AL |  |  |
| ADC VAR1, VAR2 |  |  |
| END/ENDS | 结束指令  ENDS伪指令用于结束段定义，与一个SEGMENT伪指令对应使用。  END指令为当程序运行到这里是表示整个源程序结束 |  |
| NUM1 DB 10H  NUM2 DB 20H  MOV AX, NUM2-MUN1 |  | 此处的NUM2-NUM1表示的是NUM2和NUM1两个变量在段内偏移量之差。故不允许加法 |
| JMP 标号 | 段内直接转移 |  |
| JMP [BX]100H | 段内间接转移  将逻辑地址为DS:BX+100H的字类型内存单元中的16位数据作为目标地址的偏移量并送至IP进行保存 |  |
| JMP FAR PTR 标号 | 段内长距离转移指令 |  |
| JMP NEAR PTR 标号 | 段间近距离转移指令,16位内 |  |
| JMP SHORT 标号 | 段内端转移指令，8位以内 |  |
| JMP AX  JMP BX | IP=寄存器中的值 | MOV AX, 0123H  MOV DS:[0], AX  JMP WORD PTR DS:[0]  则IP=0123H。上述指令等价于：  MOV AX, 0123H  JMP AX |
| JMP WORD PTR 标号 | 段内转移  从内存单元地址处开始存放一个字，是转移的目的偏移地址 | MOV AX, 0123H  MOV DS:[0], AX  JMP WORD PTR DS:[0]  则IP=0123H。 |
| JMP DWORD PTR 标号 | 段间转移  从内存单元地址处开始存放着两个字，高地址处的字是转移的目的段地址。低地址处是转移的目的的偏移地址 | MOV AX, 0123H  MOV DS:[0], AX  MOV WORD PTR DS:[2], 0  JMP DWORD PTR DS:[0]  则CS=0, IP=0123H |
| **注意：一切源程序指令不允许出现: JMP 2000:1000这样的** | | |
| JC | CF=1是程序转移至目的地址  CF=0时顺序执行 |  |
| JNC | CF=1时顺序执行  CF=0时程序转移至目的地址 |  |
| JO/JNO | OF=1或OF=0 |  |
| JS/JNS | SF=1或SF=0 |  |
| JZ/JE | ZF=1转移，ZF=0顺序 |  |
| JNZ/JNE | ZF=1顺序，ZF=0转移 |  |
| JP/JPE | PF=1转移，PF=0顺序 |  |
| JCZX 标号 | 如果CX中的值为0，则进行条件转移 | 转移为短转移 |
| CMP 对象1， 对象2 |  | 是比较指令，功能相当于减法指令，不保存计算结果  程序执行后将对标志寄存器产生影响 |
| CALL PROC\_NAME  CALL NEAR PTR PROC\_NAME  CALL FAR PTR PROC\_NAME | 段内直接调用指令  SP=SP-2,SP指向空栈顶  SP=IP,CALL指令后面一条指令的偏移量送入栈顶保存  IP=PROC\_NAME，将PROC\_NAME的符号位偏移量作为子程序入口地址的偏移量送入IP保存，使程序流程向子程序入口转移 |  |
| CALL BX  CALL 0100H[BX]  CALL 0020H[BX][SI] | 段内间接调用指令 |  |
| RET  RETN | 不带参数的段内返回指令  IP=SP  SP=SP+2 |  |
|  | | |
| RET N  RETN N | 带参数的段内返回指令  IP=SP  SP=SP+2  SP=SP+N从栈顶直接出栈N/2个字，用于清除主程序在调用子程序前通过堆栈传递的N/2个入口参数 |  |
| CALL PROC\_NAME  CALL FAR PTR PROC\_NAME | SP=SP-2指向空栈顶  SP=IP，CALL指令后面一条指令的偏移量送入栈顶保存  SP=SP-2指向空栈顶  SP=CS，CALL指令后面一条指令的段基址送入栈顶保存 |  |
| CALL WORD PTR 内存地址 | 相当于  PUSH IP  JMP WORD PTR 内存单元地址 |  |
| CALL DWORD PTR 内存地址 | 相当于  PUSH CS  PUSH IP  JMP DWORD PTR |  |
| CALL 双字内存单元的逻辑地址 | 段间间接调用指令 |  |
| RET  RETF  RET N  RETF N | 不带/带参数的段间返回指令 |  |
| **子程序调用事例** | | |
| 代码1：  DATA SEGMENT  VAR1 DW 5482H, 669EH, 02C7H, 14B2H, 0C254H  VAR2 DW 8C2BH, 0C24H, 0AB12H, 357AH, 41A5H  LEN EQU $-VAR2  SUM DW LEN DUP(0)  OVR DB ？  DATA ENDS  STACK1 SEGMENT STACK  DW 40H DUNP(0)  STACK1 ENDS  CODE SEGMENT  ASSUME CS:CODE, DS:DATA, SS:STACK1  BEGIN:MOV AX, DATA  MOV DS, AX  LEA SI, VAR1  LEA DI, VAR2  MOV CX, LEN  LEA BX, SUM  CALL LONGADD  MOV OVR DL  MOV AX, 4CH  INT 21H  LONGADD PROC  PUSHF  PUSH AX  CLC  L1:MOV AX, [SI]  ADC AX, [DI]  PUSHF  MOV [BX], AX  ADD SI, 2  ADD DI, 2  ADD BX, 2  POPF  LOOP L1    JO L2  MOV DL, 0  JMP L3  L2: MOV DL, 0FFH  L3:POP AX  POPF  RET  LONGADD ENDP  CODE ENDS  END BEGIN | | |
| 代码2：  DATA SEGMENT  VAR1 DW 5482H, 669EH, 02C7H, 14B2H, 0C254H  VAR2 DW 8C2BH, 0C24H, 0AB12H, 357AH, 41A5H  LEN EQU $-VAR2  SUM DW LEN DUP(0)  OVR DB ？  DATA ENDS  STACK1 SEGMENT STACK  DW 40H DUP(0)  STACK1 ENDS  CODE SEGMENT  ASSUME DS:DATA, CS:CODE, SS:STACK1  BEGIN:MOV AX, DATA  MOV DS, AX  LEA SI, VAR1  PUSH SI  MOV SI, LEN  PUSH SI  LEA SI, SUM  PUSH SI  CALL LONGADD  POP DX  MOV OVR, DL  MOV AH, 4CH  INT 21H  LONGADD PROC  PUSH BP  MOV BP, SP  PUSHF  PUSH AX  PUSH BX  PUSH CX  PUSH SI  PUSH DI  MOV BX, [BP+4]  MOV CX, [BP+6]  MOV DI, [BP+8]  MOV SI, [BP+10]  CLC  L1:MOV AX, SI  ADC AX, [DI]  PUSHF  ADD SI, 2  ADD DI, 2  ADD BX, 2  POPF  LOOP L1  JO L2  MOV [BP=10], 0  JMP L3  L2: MOV [BP+10], 0FFH  L3: POP DI  POP SI  POP CX  POP BX  POP AX  POPF  POP BP  RET 6  LONGADD ENDP  CODE ENDS  END BEGIN | | |
| MOV AH, 01H  INT 21H | 单个字符输入 |  |
| MOV AH, 02H  INT 21H | 单个字符输出 |  |
| MOV AH, 0AH  MOV DX, 缓冲区起始偏移量  INT 21H | 字符串输入 |  |
| MOV AH, 09H  MOV DX,字符串起始偏移量  INT 21H |  | 字符串必须使用‘$’结尾 |
|  | | |
| DATA SEGMENT  MAXLEN DB 20  INPTLEN DB ?  STR1 DB 21 DUP(0)  STR2 DB 21 DUP(0)  DATA ENDS  STACK1 SEGMENT STACK  DW 40H DUP(0)  STACK1 ENDS  CODE SEGMENT  ASSUME CS:CODE,DS:DATA,SS:STACK  BEGIN:MOV AX, DATA  MOV DS, AX  LEA DX, MAXLEN  MOV AH, 0AH  INT 21H  XOR CX, CX  MOV CL, INPTLEN  LEA SI, STR1  LEA DI, STR2  LOP1:  CMP [SI], 30H  JAE L1  JMP L2  L1: CMP [SI], 39H  JA L2  JMP L3  L2: MOV AL, [SI]  MOV [DI], AL  INC DI    L3: INC SI  LOOP LOP1  MOV [DI],’$’  CALL NEWLINE  LEA DX, STR2  MOV AH, 09H  INT 21H  MOV AH, 4CH  INT 21H  NEWLINE PROC  PUSH AX  PUSH DX  MOV DL, 0DH  MOV AH, 02H  INT 21H  MOV DI, 0AH  MOV AH, 02H  INT 21H  POP DX  POP AX  RET  NEWLINE ENDP  CODE ENDS  END BEGIN | | |
|  | | |
| STACK1 SEGMENT STACK  DW 40H DUP(0)  STACK1 ENDS  CODE SEGEMENT  ASSUME CS:CODE, DS:DATA, SS:STACK  BEGIN:CALL NEWLINE  MOV AH, 01H  INT 21H  CMP AL, 0DH  JNZ L1  L1: CMP AL, ‘A’  JAE L3  JMP L4  L3: CMP AL, ‘Z’  JBE L5  JMP L4  L5: SUB AL, 20H  NEWLINE PROC  PUSH AX  PUSH DX  MOV DL, 0DH  MOV AH, 02H  INT 21H  POP DX  POP AX  RET  NEWLINE ENDP | | |
|  |  |  |



DATA SEGMENT

STRING STR1 “hello”