汇编语言期中复习资料

> 版本：2023-11-10
>
> 最近更新日期：2023-11-11

考试题型与提纲

题型：

1. 选择题（5×2’=10‘）
2. 填空题（6×2’=12‘）
3. 简答题（4×7’=28‘）
4. 程序分析与填空题（4×7’=28‘）
5. 程序设计题（2×7’=14‘）
6. 论述题（1×8’=8‘）

复习提纲：

1. 第一章：

   1.2 计算机中的数据的表示

2. 第二章：

   2.2 存储器

   2.3 中央处理器中的寄存器

   2.5 32位80×86的工作模式

3. 第四章：

   全部

4. 第五章：

   除了串操作指令，其他都是

5. 第六章：

   6.1 伪指令

6. 第七章：

   7.1 分支程序设计

   7.2 循环程序设计

7. 第八章：

   8.1 子程序结构

   8.2 子程序的参数传递

说明：

- 有删除线部分不做考点
- 加粗部分为知识点的核心（用于快速浏览）
- 每章课后习题部分的任务栏用于是否确认复习

第一章 汇编语言基础知识

> 重点：
>
> ①汇编语言与机器语言的关系
>
> ②汇编语言的组成、编译、实现步骤

一、机器语言与汇编语言

1. 计算机程序是由各种程序设计语言根据编程规则实现的，计算机程序设计语言经历了从低级到高级的发展，通常分为三类：**机器语言**、**汇编语言**、**高级语言**。

   相互转换：高级语言 ==> 汇编语言 < == > 机器语言

   汇编语言介绍：

   ①面向**机器**的符号语言。

   ②用**助记符**来表示指令和数据变量

   ③**要翻译**成**机器语言**程序才可以由计算机执行（汇编指令与机器指令一一对应）

   ④被称为**第二代语言**。

2. 汇编语言的组成：

   ①**汇编指令**：机器码的**助记符**，有对应的机器码，它是汇编语言的核心。

   ②**伪指令**：没有对应的机器码，由**编译器执行**，计算机并不执行。

   ③**其他符号**：如+、-、*、/等，由编译器识别，**没有对应的机器码**。

3. 汇编语言的优缺点：

   - **优点**：

     ①**直接控制**计算机硬件部件，程序效率高，执行速度快，内存占用小

     ②可以编写在“**时间**”和“**空间**”两方面最有效的程序

     ③可以**与高级语言配合使用**（c/c++），应用广泛

     ④语言**直接、简洁**

   - **缺点**：

     ①面向机器的低级语言，**可移植性差**

     ②编写繁琐，调试、维护、交流和**移植困难**

     ③需要熟悉计算机硬件系统、**考虑许多细节**

     ④受指令的**限制**

4. 汇编的定义：汇编语言源程序翻译成机器语言程序的过程。

5. 汇编程序与汇编源程序的区别：

   - 汇编程序：**特指将汇编源程序翻译成机器语言程序（目标程序）的编译程序**；主要功能：检查源程序；检测出源程序中语法错误，并给出出错信息；产生源程序的目标程序，并可给出列表文件；展开宏指令。
   - 汇编源程序：用**汇编语言编写**的程序。

6. 学习汇编语言的原因：

   ①**汇编语言程序是由符号指令写成的，本质上还是机器语言，与具体机型的硬件结构密切相关，**可直接、有效地控制计算机硬件，运行速度快，程序短小精悍，占用内存容量少。

②在某些**特定应用场合**更能发挥作用，如**实时控制系统**，需要对硬件设置直接进行数据的输入/输出和控制，如在**嵌入式系统和智能化仪器的开发**中，需要更好地利用有限的硬软件资源，发挥硬件的功能。

③学习汇编语言是从根本上认识和理解计算机工作过程的最好方法，通过汇编语言指令，可以清楚地看到程序在计算机中如何一步步执行，有利于**更深入理解计算机的工作原理和特点。**

④汇编语言把软件和硬件紧密地结合在一起，起到**连接硬件和软件的桥梁作用。**

二、课后习题

- [ ] 1-1. 什么是机器语言？什么是汇编语言？简述汇编语言的特点。
  - 机器语言就是用二进制编码组成的机器指令的集合和一组使用机器指令的规则。
  - 汇编语言是对机器指令中的操作码用英文单词的缩写描述，对操作数用标号、变量、常量描述。
  - 汇编语言的特点：

    ①与机器有关：移植性差，但可直接控制硬件

    ②程序效率高

    ③局限性：受指令的限制，如考虑存储单元等

    ④调试困难
- [ ] 1-2. 汇编程序与汇编源程序的区别是什么？

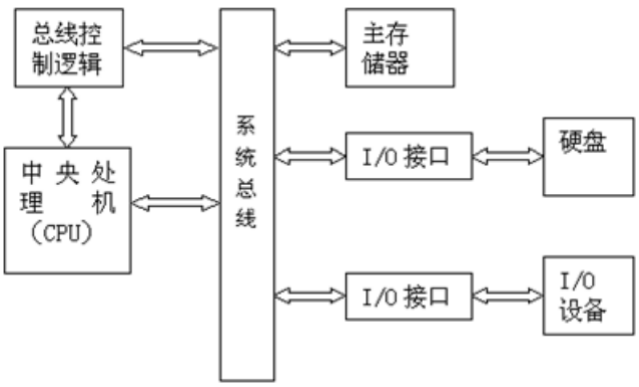
  汇编源程序是指用汇编语言编写的程序，而汇编程序特指将汇编源程序汇编成目标文件的编译程序。

第二章 计算机基本原理

> 重点：
>
> ①计算机系统组成
>
> ②寄存器组成与作用

~~一、计算机系统组成~~

1. 计算机的基本工作原理是**存储程序**和**程序控制**，称为冯诺依曼原理。
   冯诺依曼结构：主要由微处理器芯片构成的**中央处理器**、**存储器**和**输入/输出**子系统三大部分组成，用**系统总线连接**在一起。



2. 中央处理器：主要包括运算器和控制器。**运算器执行指令**，**控制器**负责计算机的**控制**，负责从主存储器取指令，对指令进行译码，发出访问主存储器或I/O设备接口的控制信号，完成程序的要求。
3. 存储器：是计算机记忆部件，以二进制形式**存放程序和数据。**
4. 输入/输出：包括大容量**存储器（硬盘）**和其他**外设**，如显示器、键盘、打印机、鼠标等。
5. 系统总线：**连接CPU、主存储器和I/O子系统**三大部分，用以完成各部分的数据交换。系统总线**包括数据总线、地址总线和控制总线。**数据总线负责传送数据，地址总线负责指示主存地址或I/O接口地址，控制总线负责总线的动作，如时间、方向、状态。

二、存储器

地址计算公式：段地址×16+偏移地址=物理地址

三、中央处理器中的寄存器

1. 通用寄存器
    - **AX**（accumulate register）：**累加器**，运算较多时使用这个寄存器，有些指令规定必须使用它。
    - **BX**（based register）：**基址寄存器**，存放数据和内存的起始偏移地址。
    - **CX**（count register）：**计数寄存器**，存放数据和存放重复操作的次数。
    - **DX**（data registered）：**数据寄存器**，存放数据和存放32位数据的高16位。
2. 地址寄存器
    - **SP**（stack point）：**堆栈指针**，存放堆栈栈顶的偏移地址。
    - **BP**（base point）：**基址指针**，存放内存中数据的偏移地址。
    - **SI**（source index）：**源变址寄存器**，存放内存中源数据区的偏移地址；指在某些指令作用下它可以自动地递增或递减其中的值。
    - **DI**（destination index）：**目的变址寄存器**，存放内存中目的数据区的偏移地址，并在某些指令作用下自动地递增或递减其中的值。
3. 段寄存器
    - **CS**（code segment）：**代码段寄存器**，存放当前正在执行的程序段的段基址。
    - **SS**（stack segment）：**堆栈段寄存器**，存放堆栈段的段基址。
    - **DS**（data segment）：**数据段寄存器**，存放数据段的段基址。
    - **ES**（extra segment）：**附加段寄存器**，存放另一个数据段的段基址。
4. 指令指针寄存器
    - **IP**：**指令指针寄存器**，存放即将执行指令的偏移地址。
5. 标志寄存器
    - FLAGS：存放CPU的两类状态。
    - 状态标志：**CF（进位）**、PF（奇偶）、AF（辅助进位）、**ZF（零）、SF（符号）和OF（溢出）**
    - 控制标志：TF（陷阱）、IF（中断）和DF（方向）
    - 标志位的符号说明：

标志名	标志为1	标志为0
OF 溢出（是/否）	OV（overflow-溢出）	NV（not overflow-没溢出）

标志名	标志为1	标志为0
DF 方向（减/增）	DN（down-下方）	UP（up-上方）
IF 中断（允许/不允许）	EI（enable interrupt-允许中断）	DI（disabled interrupt-不允许中断）
SF 符号（负/正）	NG（negative-负）	PL（plus-正）
ZF 零 （是/否）	ZR（zero-为零）	NZ（not zero-不为0）
AF 辅助进位（有/无）	AC（auxiliary carry-有辅助进位）	NA（not auxiliary carry-无辅助进位）
PF 奇偶（偶/奇）	PE（parity even-偶）	PO（parity odd-奇）
CF 进位（有/无）	CY（carried-有进位）	NC（not carried-没进位）

OF：溢出标志。OF=1表示有两个有符号数的运算结果超出了可以表示的范围，结果是错误的；OF=0表示没有溢出，结果正确。进行无符号数运算时也会产生新的OF标志（CPU不知道处理对象是否为有符号数），此时程序员可以不关心OF标志。// 因为计算机只计算二进制，所谓的有符号和无符号运算都是看程序员自己觉得的，对于有符号数程序员会利用补码定义负数

SF：符号标志。SF=1表示**运算结果的最高位**为“1“。对于有符号数，在溢出标志OF=0时，SF=1表示运算结果为负，SF=0表示运算结果非负。OF=1时，由于结果是错误的，所以符号位也和正确值相反。例如：两个负数相加产生溢出，此时SF=0。对于无符号数运算，SF无意义（但是可以看出结果的大小规模）。

ZF：零标志。ZF=1表示运算结果为零，**减法运算后**结果为零意味着两个参加运算的数**大小相等**；ZF=0，表示运算结果非零。

CF：进位/借位标志。CF=1 表示两个无符号数的加法运算有进位，或者是减法运算有借位，需要对它们的高位进行补充处理；CF=0 表示没有产生进位或借位。同样，进行有符号数运算时也会产生新的CF标志，此时程序员可以不关心CF标志。

DF：方向标志。DF=0时，每次执行字符串指令后，源或目的地址指针用加法自动修改地址；DF=1时用减法来修改地址。它用来控制地址的变化方向。

IF：终端允许标志。IF=1表示允许处理器响应可屏蔽中断请求信号，称为开中断，IF=0表示不允许处理器响应可屏蔽中断请求信号，称为关中断。

AF：辅助进位标志。在进行字操作时，低字节向高字节进位时，AF=1，否则为0。一般用于两个BCD数运算后调整结果用，对其他数的运算没有意义。

PF：奇偶标志 。PF=1表示运算结果的低8位中偶数个”1“；PF=0表示有奇数个”1“。可以用来奇偶校验。

6. 符号地址
    - **DB,DW** 定义的存储单元名字，默认对应为数据段的偏移地址。
    - **EQU** 定义的符号地址没有默认的数据段，等价于一个数值变量。
7. 溢出标志 OF 与进位标志 CF 的作用以及两者之间的区别。

    处理器对两个操作数进行运算时，按照无符号数求得结果，按如下规则设置OF，CF

    - OF是溢出标志，表示两个有符号数运算结果**是否超出可表示的范围。**
    - CF是进位/借位标志，表示两个无符号数运算结果**有进位或借位**，需要对它们的高位进行补充处理。

如果将参加运算的操作数认为是**无符号数注意是否进位**；认为是**有符号数注意是否溢出。**

四、32位80×86CPU的工作模式

1. 工作模式：①实模式②保护模式③虚拟8086模式
2. 执行步骤：
    - 从CS:IP指向的内存单元读取指令，读取的指令进入**指令队列缓冲器**；
    - IP=IP+所读取指令的长度，从而指向下一条指令；
    - 执行指令，转到步骤（1），重复这个过程。

五、课后习题

- [ ] 2-1. 简述计算机系统组成。

  计算机由中央处理器CPU、存储器、输入系统和输出系统组成，由系统总线连接在一起。CPU包括运算器和控制器，运算器执行指令、控制器负责计算机的控制。存储器是计算机的记忆部件，以二进制形式存放程序和数据。输入输出系统包括大容量存储器，如硬盘。以及其他外设，如鼠标、键盘、显示器等。

- [ ] 2-2. 简述16位机的各类寄存器的主要作用。

  见本章第三节CPU中的寄存器

- [x] 2-3. 写出每条汇编指令执行后的相关寄存器的值。

  Mov ax, 1345H ax = 1345H

  Mov ah, 24H ax = 2445H

  Mov al, 45H ax = 2445H

  Mov bx, 3412H bx = 3412H

  Mov al, bh ax = 2434H

  Mov ah, bl ax = 1234H

- [x] 2-4. 实模式下，写出段地址和偏移地址为1234:2002、1430:0042、FF00:0FFF的物理地址。

  1234:2002 => 12340H + 2002H = 14342H

  1430:0042 => 14300H + 0042H = 14342H

  FF00:0FFF => FF000H + 0FFFH = FFFFFH

- [x] 2-5. 下列各数均为十进制数，请采用8位二进制补码运算，并回答标志寄存器FLAGS中CF和OF的值，运算结果所代表的十进制数是多少？如果用16位二进制补码运算，其结果所代表的十进制数是多少？FLAGS中CF和OF的值呢？

    1. 86+69 // 题目是85+69 我直接将错就错
       $86=0101\ 0110_2,69=0100\ 0101_2$
       $86+69=0101\ 0110_2+0100\ 0101_2=1001\ 1011_2$

       补码转换$1001\ 1011_2\ => -0110\ 0101_2$此时十进制数为$-101$
       因为溢出OF=1，没有进位CF=0，最高位为1故符号位SF=1，计算结果不为零ZF=0
       16位下：$005BH$十进制数为155
       没有溢出OF=0，没有进位CF=0，最高位为0故符号位SF=0，计算结果不为零ZF=0

2. 86+(-69)
   $86=0101\ 0110_2,-69=-0100\ 0101_2=1011\ 1011_2$
   $86+(-69)=0101\ 0110_2+1011\ 1011_2=1\ 0001\ 0001_2$
   补码转换$0001\ 0001_2=>0001\ 0001_2$此时十进制数为$17$
   因为没溢出OF=0，有进位CF=1，最高位为0故符号位SF=0，计算结果不为零ZF=0
   16位下：$0011H$十进制为17
   因为没溢出OF=0，没进位CF=0，最高位为0故符号位SF=0，计算结果不为零ZF=0

3. 86-(-69) // 加法看进位，减法看借位

   $86=0101\ 0110_2,-69=-0100\ 0101_2=1011\ 1011_2$
   $86-(-69)=>$（无符号） 86-187=0101\ 0110_2-1011\ 1011_2=0101\ 0110_2 + 0100\ 0101_2=1001\ 1011_2$
   补码转换$1001\ 1011_2\ => -0110\ 0101_2$此时十进制数为$-101$
   因为溢出OF=1，有借位CF=1，最高位为1故符号位SF=1，计算结果不为零ZF=0
   16位下：$005BH$十进制数为155
   没有溢出OF=0，有借位CF=1，最高位为0故符号位SF=0，计算结果不为零ZF=0

4. 86-(69)
   $86=0101\ 0110_2,69=0100\ 0101_2$
   $86-69=0101\ 0110_2-0100\ 0101_2=0101\ 0110_2+1011\ 1011_2=1\ 0001\ 0001_2$

   补码转换$0001\ 0001_2=>0001\ 0001_2$此时十进制数为$17$
   因为没溢出OF=0，无借位CF=0，最高位为0故符号位SF=0，计算结果不为零ZF=0
   16位下：$0011H$十进制数为17
   没有溢出OF=0，没有借位CF=0，最高位为0故符号位SF=0，计算结果不为零ZF=0

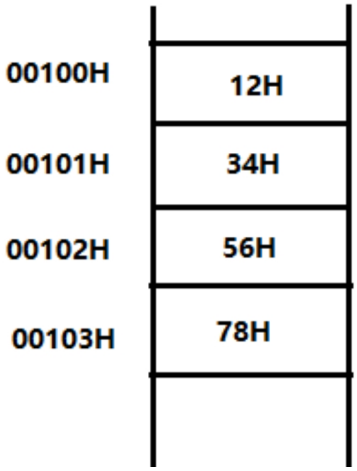
   总结：CF与SF按照结果得出，即如果计算得出结果多出一位就为进位，如果计算结果与实际结果符号不相同就为溢出，符号位就根据最高位得出；两个数相减，最高位不足减向前借位；算术左移，数据的最高位和符号位不同时溢出。

- [x] 2-6. 给定段地址为0001H，仅通过变化偏移地址寻址，CPU的寻址范围从__到__
  答：偏移地址16位从0000H~FFFFH
  故寻址从00010H+0000H ~ 00010H+FFFFH 即00010H到1000FH

- [x] 2-7. 有一数据存放在内存20000H单元中，现给定段地址为SA，若想从偏移地址寻到此单元，则SA应满足的条件是：最小为__，最大为__。
  答：偏移地址16位从0000H~FFFFH
  寻址到20000H，**最小**为(20000H-FFFFH=10001H)**除10H向上取整**得到**1001H**，则**最大**为(20000H-0000H)**除10H向下取整**得到**2000H**。

- [x] 2-8. 已知8086系统某存储单元物理地址为：52506H，你认为段基址的最大值、最小是分别是多少？8086微机最多可以有多少个不同的段基址？
  答：同上计算，最大：52506H/10H向下取整得到5250H，最小：(52506H-FFFFH)/10H向上取整得到4251H。
  段地址16位，故最多有$2^{16}=65536$个段基址。

- [x] 2-9. 从物理地址为00100H开始到00103H单元中顺序存放的数据为：12H、34H、56H，78H。请画出数据存放示意图，并回答以下问题：



1. 写出地址00101H字节单元的内容；
   答：34H
2. 写出地址00102H字节单元的内容
   答：56H，若为字单元则7856H（**小端：低字节低地址，高字节高地址**）

第三章 汇编语言程序实例及上机操作

> 重点：
>
> ①编写代码步骤
>
> ②常用的DOS系统功能调用

一、程序实例的上机步骤

1. 编辑-建立asm源程序文件：edit xx.asm
2. 汇编-产生obj二进制目标文件：masm xx.asm
3. 连接-产生exe可执行文件：link xx.asm
4. 列表文件生成：列表文件报告了汇编过程中产生的源程序和机器语言清单、指令和变量的偏移地址等。如果希望生成列表文件则使用MASM xx xx xx，即给出3个xx作为命令参数即可生成.LST文件，.LST文件是普通文本文件可使用edit直接编辑。
5. 程序运行：xx.exe 或 xx
6. 程序调试：debug xx.exe
   - R命令：查看和改变CPU寄存器的内容

- D命令：查看当前地址存放数据的信息。
- E命令：改写内存中的内容。
- Q命令（退出命令）：退出debug。
- U命令（反汇编命令）：用于把机器语言反汇编为汇编语言，便于用户查看程序。
- G命令（运行程序命令）：G[=起始地址][中止地址] 从起始地址运行到中止地址。如：G=0代表从0处执行，G=1:12代表执行1-12
- T命令（跟踪程序命令）：T[=起始地址][**指令条数**] 从起始地址后运行n条指令。如：T代表当前地址继续运行一条指令，T=0代表从0处执行一条暂停。
- P命令（单步执行程序命令）：P。一次执行完这个系统例行程序。如：P直接执行完int 21h调用。

说明：对于INT指令不能使用T命令进行跟踪。因为INT指令实质上就是调用一个系统例行程序，T命令使程序进入一个陌生的系统程序之中；对于P命令可以一次执行完这个系统例行程序，回到用户程序中。T和P命令的区别**可以理解为类似于C语言中调试过程中的步入和跳过。**

二、常用的DOS系统功能调用（int 21h）

1. DOS系统功能INT 21H调用的方法

   ①调用的功能号**存入AH寄存器**

   ②如必要，设置该调用功能的入口参数（调用参数）

   ③**执行INT 21H**指令

   ④如必要，按规定取得出口参数（返回参数）

2. 部分常用的DOS系统调用

功能号	说明	功能	示例
1号功能	键盘输入单个字符并回显	等待从键盘输入一个字符，将该字符的ASCII码送入AL中，并送屏幕显示	MOV AH, 1 INT 21H

2号功能	显示单个字符	在当前光标位置显示字符（注意：执行后AL寄存器的值被修改为DL的值。原因：读入存放在DL的一个字符，将其送至向标准输出，然后将最后输出的那个字符存**放在AL后返回**）	MOV AH, 2
			MOV DL, 'A'
			INT 21H

7号功能	键盘输入无回显	等待从键盘输入一个字符，将该字符的ASCII码送入AL中，但是屏幕无显示	MOV AH, 7 INT 21H

功能号	说明	功能	示例
9号功能	显示字符串	显示由DS:DX所指向的以'$'结束的字符串str（注意：执行后AL寄存器的值被修改为'$'的ASCII码24H）	MOV AH, 9 LEA DX, STR INT 21H
0A号功能	键盘输入到缓冲区	DS:DX=BUF首地址即缓冲区首地址，BUF缓冲区第1个字节是缓冲区大小（含回车键）。第2个字节是实际输入的字节数（不含回车键），当以回车键输入结束时系统自动存入的。从第3个字节开始存放输入的内容（含回车键）。	MOV AH, 0AH LEA DX, BUF INT 21H
4C号功能	结束程序并返回DOS	结束程序并返回操作系统	MOV AH, 4CH INT 21H

三、课后习题

- [ ] 3-1. 写出从汇编语言源程序的建立到产生可执行文件的步骤和上级操作命令。
①用编辑程序EDIT建立.ASM源文件
②用汇编程序MASM把.ASM源文件汇编成.OBJ文件
③用连接程序LINK把.OBJ文件转换成.EXE文件
④在DOS下直接运行.EXE文件或在DEBUG下调试该.EXE文件

- [ ] 3-2. 列表文件.LST是在什么阶段产生的？列表文件.LST中有哪些内容？
①.LST在汇编得到.OBJ的同时得到

②列表文件报告了汇编过程中产生的很多有价值的参考信息。主要包括源程序和机器语言清单、指令和变量的偏移地址等

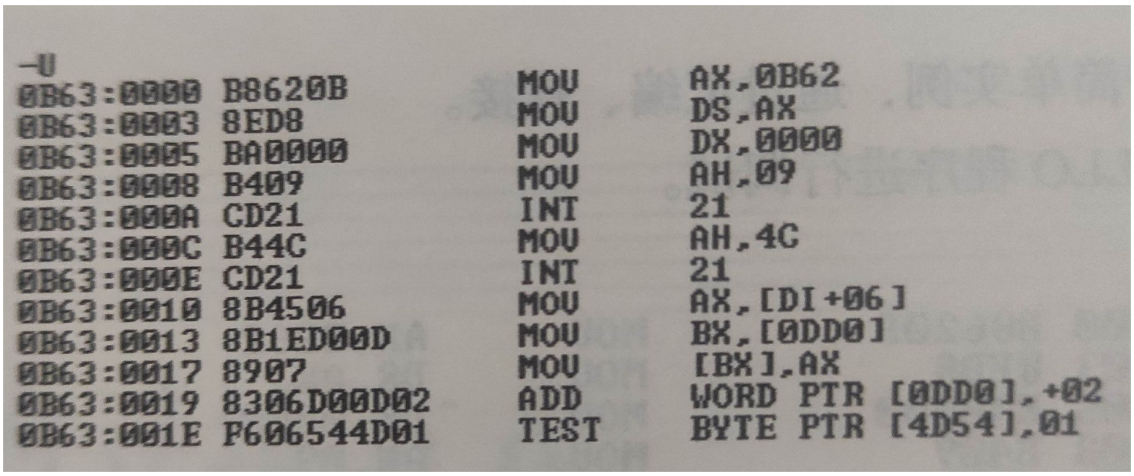
- [ ] 3-3. 写出定义一个代码段的语句，段名为MYPRG。

```
MYPRG segment
    assume cs:MYPRG
start
MYPRG ends
end start
```

- [ ] 3-4. 程序中用什么语句来结束程序的执行？用什么语句来表示程序的结束和指出程序执行的开始？
①程序用4C号功能结束程序的执行

②程序用`end start`来表示程序的结束和指出程序执行的开始

- [x] 3-7. 下图为Debug调入的可执行程序，回答以下问题



（1）程序的起始物理地址是多少？结束地址是多少？

起始物理地址是0B63:0000=B630H，结束地址是0B63:0010-1=B63FH（段大小一定是16的倍数）

（2）CS寄存器的值是什么？

0B63H

（3）程序的功能是什么？

显示0B62:0000存储的字符串

（4）写出查看DS:0处内容的Debug命令。

-d ds:0

（5）设置断点执行程序，两处的INT21指令执行后有什么结果？

第一处是显示一个字符串，第二处是结束程序

（6）如果要运行这个程序应该用什么Debug命令？

-g

（7）Debug命令T=0 4之后，寄存器AX、DS、DX的值是多少？

AX：0962H，DS：0B62H，DX：0000H

第四章 操作数的寻址方式

> 重点：
>
> ①寻址方式以及与之匹配的寄存器

一、立即寻址方式

1. 定义：**操作数（立即数）就在指令中**，紧跟在操作码之后，操作数作为指令的一部分存放在代码段。
2. 注意：
   - 执行时**无需去内存取数**，因此称为立即数。
   - 主要**用于寄存器赋初值。**
   - 立即数**只能作为源操作数**，并且长度与类型和目的操作数一致。
3. 示例：

```
MOV AL, 06H
MOV AX, 12AFH
```

二、寄存器寻址方式

1. 定义：**操作数就是寄存器中的值。**指令中给出寄存器名。
2. 注意：
   - 寄存器和立即数寻址方式都与主存无关。
3. 示例：

```
MOV AX, BX
```

三、直接寻址方式

1. 定义：操作数的**有效地址EA就在指令**中（变量/常量/立即数），默认段地址=(DS)。
2. 用途：用于直接指定一个**变量作为操作数。**
3. 注意：对于直接寻址方式而言，必须使用前缀“DS:”指出该单元再数据段中，如：DS:[2000H]代表一个数据段的存储单元，其偏移地址为2000H。如果**没写前缀**"DS:"，则系统在**用MASM汇编**时就认为2000H**是立即数而不是偏移地址。**而在**Debug的A命令**输入指令，则**不需要加前缀**，系统**默认为数据段。**
4. 操作数的物理地址=（DS）*10H+EA
5. 示例：

```
;存储器读操作
MOV AX, DS:[2000H]
;存储器写操作
MOV DS:[4000H], AX
;符号地址
VALUE DW 5678H
MOV AX,VALUE ;或
MOV AX,[VALUE]
;段前缀
VALUE EQU 1000H
MOV AX, DS: [VALUE]
MOV AX, ES: [VALUE]
```

四、寄存器间接寻址方式

1. 定义：操作数的**有效地址EA就在寄存器**中。
2. 理解：寄存器间接寻址方式与寄存器寻址方式不同，它不是把寄存器的内容作为操作数，而是把**寄存器的内容作为操作数的地址**，而操作数还是在内存中，故称为间接寻址。
3. 用途：用寄存器间接指向一个内存单元，寄存器的值不同指向的内存单元的地址就不同，常用于**循环程序**中。
4. 物理地址计算方式：

- 物理地址=10H x (DS) + (BX)：MOV AX,[BX]
- 物理地址=10H x (DS) + (SI)：MOV AX,[SI]
- 物理地址=10H x (DS) + (DI)：MOV AX,[DI]
- 物理地址=10H x (SS) + (BP)：MOV AX,[BP]

5. 注意：
   - **只允许使用BX、BP、SI和DI寄存器。**
   - 若有效地址用**SI、DI或BX指定**，则默认**段寄存器为DS**；若为**BP**，则默认**段寄存器为SS。**
6. 示例：

```
MOV AX, [BX]    ;默认DS寄存器作为段地址
MOV DX, [BP]    ;默认SS寄存器作为段地址
MOV ES:[DI], AX    ;默认ES寄存器作为段地址
```

五、寄存器相对寻址方式

1. 定义：操作数的有效地址**EA**是一个**寄存器**和**位移量**（变量/常量/立即数）**之和。**
2. 用途：特别适合用于**访问一维数组**，**寄存器可作为数组下标**（或数组元素的位置），利用修改寄存器的值来定位数组中的每个元素。
3. 物理地址计算方式：
   - 物理地址=10H x (DS) + (BX) + 8(16)位位移量：MOV AX,[BX+00H]
   - 物理地址=10H x (DS) + (SI) + 8(16)位位移量：MOV AX,[SI+0000H]
   - 物理地址=10H x (DS) + (DI) + 8(16)位位移量：MOV AX,[DI+0000H]
   - 物理地址=10H x (SS) + (BP) + 8(16)位位移量：MOV AX,[BP+00H]
4. 注意：
   - 只允许使用BX、BP、SI和DI寄存器
   - 若有效地址用SI、DI或BX指定，则默认段寄存器为DS；若为BP，则默认段寄存器为SS
5. 示例：

```
MOV AX，ARRAY[BX]
MOV AX，[ARRAY][BX]
MOV AX，[ARRAY+BX]
MOV AL，[BX].ARRAY    ;这四者的表现结果一致，只是表达形式不同
MOV AL，[BX+1234H]
```

六、基址变址寻址方式

1. 定义：操作数的有效地址**EA**是一个**基址寄存器**和一个**变址寄存器**的内容**之和**，基址寄存器为BX和BP，变址寄存器为SI和DI。
2. 用途：可用于**一维数组的处理**，**数组的首地址**可放在**基址寄存器**，利用修改**变址寄存器**的内容来**定位数组中的各个元素。**由于基址寄存器和变址寄存器都可以修改，所以访问数组中的各个元素更加灵活。
3. 物理地址计算方式：
   - 物理地址=10H x (DS) + (BX) + (DI)：MOV AX,[BX+DI]
   - 物理地址=10H x (DS) + (BX) + (SI)：MOV AX,[BX+SI]
   - 物理地址=10H x (SS) + (BP) + (SI)：MOV AX,[BP+SI]

- 物理地址=10H x (SS) + (BP) + (DI)：MOV AX,[BP+DI]

4. 注意：
   - 若有效地址用SI、DI或BX指定，则默认段寄存器为DS；若为BP，则默认段寄存器为SS
5. 示例：

```
MOV AX，[BX] [SI]  ;默认DS寄存器作段地址
MOV AX，[BP] [DI]  ;默认SS寄存器作段地址
MOV AX，ES:[BX] [DI]  ;指定ES寄存器作段地址
MOV DX，[BP] [SI]  ;默认SS寄存器作段地址
MOV [BX+DI]，CX  ;默认DS寄存器作段地址
MOV [BP+SI]，AL  ;默认SS寄存器作段地址
```

七、相对基址变址寻址方式

1. 定义：操作数的有效地址**EA**是一个**基址寄存器**和一个**变址寄存器**以及一个**位移量**（变量/常量/立即数）**之和**，基址寄存器为BX和BP，变址寄存器为SI和DI。
2. 用途：可用于**二维数组的处理**，数组的首地址为ARRAY，**基址寄存器指向**数组的**行**，**变址寄存器**指向**该行的某个元素。**利用修改基址寄存器和变址寄存器的内容方便地访问数组中的各个元素。
3. 物理地址计算方式：
   - 物理地址=10H x (DS) + (BX) + (DI) + 8(16)位位移量：MOV AX,[BX+DI+00H]
   - 物理地址=10H x (DS) + (BX) + (SI) + 8(16)位位移量：MOV AX,[BX+SI+0000H]
   - 物理地址=10H x (SS) + (BP) + (SI) + 8(16)位位移量：MOV AX,[BP+SI+0000H]
   - 物理地址=10H x (SS) + (BP) + (DI) + 8(16)位位移量：MOV AX,[BP+DI+00H]
4. 注意：
   - 若有效地址用SI、DI或BX指定，则默认段寄存器为DS；若为BP，则默认段寄存器为SS
5. 示例：

```
MOV AX，MASK[BX][SI]  ;默认DS寄存器作段地址
MOV AX，[MASK+BX+SI]  ;默认DS寄存器作段地址
MOV AX，[BX+SI].MASK  ;默认DS寄存器作段地址
```

八、课后习题

- [ ] 4-2. 指出以下指令的寻址方式，array是变量。

```
(1)MOV  AX,9    ;   立即数寻址
(2)MOV  BYTE PTR[BX],9  ;   寄存器间接寻址
(3)MOV  BX,[DI] ;   变址寻址
(4)MOV  AX,BX   ;   寄存器寻址
(5)MOV  [SI+BX],9   ;   基址变址寻址
(6)MOV  ARRAY[BX],CX    ;   相对基址寻址
(7)MOV  AX,ARRAY+9  ;   直接寻址
(8)MOV  AX,ARRAY[BX+DI] ;   相对基址变址寻址
```

- [ ] 4-3. 假定(DS)=1200H，(SS)=4400H，(BX)=463DH，(BP)=2006H，(SI)=6A00H，位移量D=4524H，以AX寄存器为目的操作数，试写出以下各种寻址方式下的传送指令，并确定源操作数的有效地址EA和物理地址。

```
(1)立即寻址：MOV AX,D  => AX = 4524H
(2)直接寻址：MOV AX,DS:[D]   => AX = 12000H+4524H = 16524H
(3)使用BX的寄存器寻址：MOV   AX,BX   => AX = 463DH
(4)使用BX的间接寻址：MOV    AX,[BX] => AX = 12000H+463DH = 1663DH
(5)使用BP的寄存器相对寻址：MOV AX,[BP+D] => AX = 44000H+2006H+4524H = 4A52AH
(6)基址变址寻址：MOV   AX,[BX+SI]  => AX = 12000H+463DH+6A00H = 1D03DH
(7)相对基址变址寻址：MOV AX,[BX+SI+D]    => AX =
12000H+463DH+6A00H+4524H=21561H
```

第五章 常用指令系统

> 重点：
>
> ①所有常用指令
>
> ②程序题

一、指令格式与要求

1. 格式：[标号:] 指令助记符 [操作数] [;注释]
2. 双指令操作数指令要求：
   - 源操作数与目的操作数的**长度必须明确且一致**，即必须同时为8位或16位。
   - 目的操作数与源操作数**不能同为存储器**，不允许在两个存储单元之间直接传送数据。
   - **目的操作数不能为CS或IP**，因为CS:IP指向的是当前要执行的指令所在的地址。
   - **目的操作数不可以是立即数**。
3. 标识符构成规则：
   - 字符个数为1-31。
   - 第一个字符必须为字母、"?"、"@"或者"_"中的一种。
   - 第二个字符开始可以是字母、数字、"?"、"@"或者"_"。

二、数据传送指令

1. 通用数据传送指令
   - MOV 传送指令
     - 格式：MOV DST, SRC
     - 操作：(DST) <- (SRC)，将源操作数传送到目的操作数，其中DST表示目的操作数。SRC表示源操作数
     - 示例：

```
;段地址寄存器的传送
MOV AX, DATA_SEG
MOV DS, AX
;段地址寄存器须通过寄存器得到段地址，不能直接由符号地址、段地址、立即数得到。原因：数据寄存器和数据总线相连，不直接和段寄存器相连，所以写入数据需要通过数据寄存器
以下指令不合法：
MOV DS, DATA_SEG
MOV DS, ES
MOV DS, 1234
MOV CS, AX
```

```
;传送变量
MOV BX, TABLE    ;TABLE为16位变量
以下指令不合法：
MOV BL, TABLE    ;TABLE为16位变量，操作数长度不一致
MOV [BX], TABLE     ;两个操作数不能同为内存单元
```

```
;传送地址
MOV BX, OFFSET TABLE
OFFSET 为偏移地址属性操作符，通常是把变量TABLE的偏移地址送给BX
以下指令不合法：
MOV BL, OFFSET TABLE    ;    有效地址总是16位
```

- PUSH 进栈指令
  - 格式：PUSH SRC
  - 操作：(SP) <- (SP) - 2，((SP)+1,(SP)) <- (SRC)
  - 该条指令表示将源操作数压入堆栈，**目的操作数地址由SS:SP指定**，指令无需给出。
  - SP总是指向栈顶，即大地址。每次操作先将栈顶指针SP减2，再存入16位源操作数。
- POP 出栈指令
  - 格式：POP DST
  - 操作：(DST) <- ((SP)+1,(SP))，(SP) <- (SP) + 2
  - 将栈顶元素弹到16位目的操作数，同时栈顶指针加2
  - 示例：

```
MOV BX, 1234H
PUSH BX
POP AX  ;AX = 1234H
主存的堆栈操作 ，单操作数 合法
PUSH [2016H]
POP [2016H]
```

- XCHG 交换指令
  - 格式：XCHG OPR1，OPR2
  - 操作：交换OPR1，OPR2位置
  - 示例：

```
XCHG AX, BX
XCHG AX, [BX]
XCHG AX, VAR
以下不合法:
XCHG AX, 5   ;操作数不能为立即数
XCHG [BX], VAR   ;操作数不能同为内存单元
XCHG AX, BH ;操作数长度要一致
```

2. 累加器专用传送指令

输入/输出(I/O)端口是CPU与外设传送数据的接口，单独编制，不属于内存。端口地址范围为0000~FFFFH。这组指令**只限于AX,AL**，根据端口号的长度，有长格式和短格式两种。

- IN(input)　输入
  - 长格式：IN AL, PORT(字节)，IN AX, PORT(字)
  - 短格式：IN AL, DX(字节)，IN AX, DX(字)
  - 功能：将端口的数据读入到AL/AX中
  - 当**端口号为0100H~FFFFH**时，必须使用**短格式。**
  - 示例：

```
IN  AL, 61H
IN  AX, 61H
MOV DX, 0110H
IN  AL, DX
IN  AX, DX
以下不合法:
IN  AL, 1010H    ;端口超出了FFH
IN  AX, [DX]     ;端口地址不能加[]
```

- OUT(output)　输出
  - 长格式：OUT PORT, AL(字节)，OUT PORT, AX(字)
  - 短格式：OUT DX, AL(字节)，OUT DX, AX(字)
  - 功能：将数据传送到端口
  - 当端口号为0100H~FFFFH时，必须使用短格式
  - 示例：

```
OUT 61H, AL
OUT 61H, AX
MOV DX, 1234H
OUT DX, AL
OUT DX, AH
```

- XLAT　换码指令
  - 格式：XLAT
  - 操作：AL <- (BX+AL)
  - 功能：把**BX+AL的值**作为有效地址，取出其中的一个字节**送AL**
  - 示例：

```
DATA SEGMENT
    ORG 0100H   ; 初始地址
    STRING  DB  'abcdefg'
DATA ENDS

CODE    SEGMENTS
    ASSUME  CS:CODE,DS:DATA
START:
    MOV AX,DATA
    MOV DS,AX
    MOV BX,100H
    MOV AL,4
    XLAT
    INT 21H
    MOV AH,4CH
    INT 21H
CODE ENDS
END START
; 将段偏移地址BX+AL = 104的数据'e' ASCII码65H传输到了AL
```

3. 地址传送指令
   - LEA（Load Effective Address） 有效地址送寄存器
     - 格式：LEA REG, SRC
     - 操作：REG <- SRC
     - 功能：把源操作数的**有效地址EA送到指定的寄存器。**可以理解为C语言中的**取地址&**
     - 示例：

```
LEA BX, TABLE
MOV BX, OFFSET TABLE ;两者等价

LEA BX, [2016H]
MOV BX, OFFSET [2016H]  ;执行后 BX为2016H
```

   - LDS 指针送寄存器和DS
     - 格式：LDS REG, SRC
     - 操作：REG <- (SRC)，DS <- (SRC+2)
     - 功能：把源操作数SRC所指向的内存单元中**2个字**送到指定的**寄存器REG和DS**
     - 示例：

```
LDS DI, [BX]
若DS=2000H，BX=0400H，(2000:0400)=1234H，(2000:0402)=5678H
则执行后，DI=1234H，DS=5678H
```

   - LES 指针送寄存器和ES
4. 标志寄存器传送指令
   - LAHF（Load AH with Flags） 标志送AH寄存器
   - SAHF （Store AH into Flags） AH送标志寄存器
   - PUSHF（Push Flags） 标志入栈
   - POPF（Pop Flags） 标志出栈

三、算术运算指令

1. 类型扩展

   目的：**解决部分指令需要双操作数**时，使得操作数的长度一致。

   ①CBW 字节扩展成字 （将AL扩展到AX，AL为正数AH变为00，AL为负数，AH变为FF）

   ②CWD 字扩展成双字（将AX扩展到DX,AX，AX为正数时DX变为0000，AX为负数时，DX变为FFFF）

2. 加法指令
   - ADD
     - 格式：ADD DST, SRC
     - 操作：(DST) <- (DST)+(SRC)
     - **ADD影响OF（溢出）,CF（进位）标志位。对于有符号数，两个正数相加变负数，两个负数相加变正数说明溢出。对于无符号数，加法产生进位。**
   - ADC 带进位加法
     - 格式：ADC DST, SRC
   - INC 加1指令
     - 格式：INC OPR
   - 注意：除 **INC 不影响CF**，其他都影响状态标志位。

3. 减法指令
   - SUB
     - **SUB影响OF（溢出），CF（借位）。对于有符号数，正数减负数变为负数，负数减正数变为正数说明溢出。对于无符号数，小的减大的产生借位。**
   - SBB 带借位减法
   - DEC 减1指令
   - NEG 求补码
     - 格式：NEG ORP
   - CMP 比较大小
     - 格式：CMP DST, SRC
     - 功能：计算DST - SRC 但不影响DST结果，只影响借位CF标志
   - 注意：除 **DEC 不影响CF**，它们都影响状态标志位。

4. 乘法指令
   - MUL 无符号数乘法
     - 格式：MUL SRC
     - 操作：**SRC与AX/AL进行相乘**
   - IMUL 有符号数乘法

5. 除法指令
   - DIV 无符号数除法
     - 格式：DIV SRC
     - 操作：

       ①SRC为字节时，**(AL) <- (AX)/(SRC)的商，(AH) <- (AX)/(SRC)的余数**

       ②SRC为字时，(AL) <- (DX,AX)/(SRC)的商，(AH) <- (DX,AX)/(SRC)的余数

- IDIV 有符号数除法
  - **余数与被除数符号一致。**
  - **商的符号**是两个操作数符号的**异或。**
  - 注意：乘法不会产生溢出，但是**除法会产生溢出**，16位/8位得到8位、32位/16位得到16位的商，若商超过对应位则会溢出。**除法溢出会直接退出程序。**

6. BCD码的十进制调整指令
   - DAA 加法的十进制调整指令
     - 作用：以AL为目的操作数，在执行加法以后，使用DAA指令可以把AL的和调整为正确的BCD码格式。
     - 原理：**①如果AL低4位>9，或AF=1（辅助进位），则AL=AL+6。②如果AL高4位>9，或CF=1，则AL=AL+60H，CF=1**
     - 示例：

```
MOV AL, 88H
MOV BL, 89H
ADD AL, BL  ;AL=11H,AF=1,CF=1
DAA ;AL=77H   88+89=177,CF=1
;这里是两个条件都符合，所以加了66
```

   - DAS 减法的十进制调整指令
     - 以AL为目的操作数，在执行减法以后，使用DAS指令可以把AL的差调整为正确的BCD码格式。
     - 原理：**①如果AL低4位>9，或AF=1，则AL=AL-6，AF=1。②如果AL高4位>9，或CF=1，则AL=AL-60H，CF=1。**
     - 示例：

```
MOV AL,93
MOV BL,65
SUB AL,BL  ;AL=93-65=2D,AF=1,CF=0
ADS ;AL=28, CF=0
```

四、逻辑与位移指令

1. 逻辑指令
   - AND 与
   - OR 或
   - NOT 非
   - XOR 异或
   - TEST 测试指令

     测试操作X与Y不存储结果，只根据结果影响标志位（比如符号位SF）。

```
;测试AL寄存器中的数，如果是负数则转到标号NEXT去执行。如AL=86H。
MOV AL, 86H
TEST AL, 80H ;TEST指令的结果将标志寄存器的SF位置1
JS NEXT ;因此指令会跳转到标号NEXT去继续执行
```

2. 移位指令

   格式：寄存器 寄存器/立即数, 位数

   - SHL 逻辑左移：高位推给CF，低位补零
   - SAL 算术左移：高位推给CF，低位补零
   - SHR 逻辑右移：**高位补零**，低位推给CF
   - SAR 算术右移：**高位补原符号位**，低位推给CF
   - ROL 循环左移：最高位送CF和最低位
   - ROR 循环右移：最低位送CF和最高位
   - RCL 带进位循环左移：最高位送CF，**而后将原CF位送最低位**
   - RCR 带进位循环右移：最低位送CF，**而后将原CF位送最高位**

五、串操作指令

1. 原因：
   - 串操作指令每次处理的是一个字节或字，因此需要重复执行串操作指令才能处理完一个数据串。
   - **数据传送是从内存到内存。**
2. 串传送指令与REP联用时要做好哪些准备工作?
   - 源串首（末）地址送DS：SI；
   - 目的串首（末）地址送ES：DI；
   - 串长送CX；
   - 设置方向标志（CLD/STD）
3. 其他串操作指令（详细见书P84）
   - MOVS（Move String）串操作指令

     格式：MOVSB ES:[DI], DS:[SI]

     作用：将源串SRC传送到目的串DST中
   - CMPS（CoMPare String）串比较指令
   - SCAS（SCAn String）串扫描指令
   - STOS（STOre in to String）存入串指令
   - LODS（LOaD from String）从串取指令

六、程序转移指令

1. **无条件转移指令**与程序的可重新定位
   - 定义：无条件转移到指令指定的地址去执行程序。
   - 根据目标地址分类：
     - 段内转移：转移的目标地址和本跳跳转指令在同一个代码段（NEAR PTR）
     - 段间转移：转移的目标地址和本条跳转指令不在同一个代码段（FAR PTR）
   - 根据目标地址是否在跳转指令中直接给出分类：

        - 直接转移：转移的目标地址在跳转指令中**直接给出（立即数）**
        - 间接转移：转移的目标地址在跳转指令中通过**其他方式间接给出**
    - 段内直接转移
        - 格式: JMP NEAR PTR OPR
        - 操作: IP←IP+16位位移量
        - 注意：
            - NEAR PTR为目标地址OPR的属性说明，表明是一个近（段内）跳转，**通常可以省略。**
            - **位移量是带符号数**（相对于原地址），IP的值可能减小（程序向后跳），也可能增加（程序向前跳）。
            - 程序的**重新定位并不影响程序的正确执行。**
    - 段内间接转移
        - 格式: JMP WORD PTR OPR
            - 操作: IP←（EA）
            - 注意：可以使用**除立即数以外的任何一种寻址方式。**
    - 段间直接转移（跨段直接远转移）
        - 格式: JMP FAR PTR OPR
            - 操作：
                - IP←OPR的偏移地址
                - CS←OPR所在段的段地址
            - **OPR可采取符号地址（标号）**
    - 段间间接转移
        - 格式: JMP DWORD PTR OPR
            - 操作：
                - **IP←（EA）**
                - **CS←（EA+2）**
            - 注意：可以使用**除立即数和寄存器方式以外的任何一种寻址方式。**

2. **条件转移指令**
    - 定义：条件转移指令根据上一条指令所设置的标志位来判别测试条件，从而决定程序转向。
    - 注意：
        - 通常在使用条件转移指令之前，应有一条能产生**标志位**的前导指令，如CMP指令。
        - 汇编指令格式中，转向地址由**标号**表示。
        - 所有的条件转移指令都**不影响标志位。**
    - 分类：
        - 根据单个条件标志的设置情况转移

指令	说明	测试条件
JZ / JE	结果为零/相等则转移	ZF=1,则转移
JNZ / JNE	结果不为零/不相等则转移	ZF=0,则转移
JS / JNS	结果为负/不为负则转移	SF=1/0,则转移

指令	说明	测试条件
JO / JNO	结果溢出/不溢出则转移	OF=1/0,则转移
JC / JB / JNAE	结果进位/低于(below)/不高于(above)等于则转移	CF=1,则转移
JNC / JNB / JAE	结果不进位/不低于/高于等于则转移	CF=0,则转移
JP / JPE	奇偶位为1则转移/偶数个1则转移	PF=1,则转移
JNP / JPO	奇偶位为0则转移/奇数个1则转移	PF=0,则转移

- 测试CX寄存器的值为0则转移
  - 指令格式：JCXZ　OPR　　；CX寄存器为零则转移
  - 测试条件：CX=0,则转移。
- 比较两个无符号数,根据结果转移(Below/Above)

指令	说明	测试条件
JC / JB / JNAE	进位位为1/低于/不高于等于则转移	CF=1,则转移
JNC / JNB / JAE	进位位为0/不低于/高于等于/则转移	CF=0,则转移
JBE / JNA	结果低于等于/不高于则转移	CF|ZF=1,即CF与ZF的或为1,则转移
JNBE / JA	结果不低于等于/高于则转移	CF|ZF=0,即CF与ZF的或为0,则转移

- 比较两个有符号数, 根据结果转移(Lower/Greater)

指令	说明	测试条件
JL / JNGE	小于/不大于等于则转移	SF^OF=1,即SF与OF的异或为1,则转移
JNL / JGE	不小于/大于等于则转移	SF^OF=0,即SF与OF的异或为0,则转移
JLE / JNG	小于等于/不大于则转移	(SF^OF)|ZF=1,即SF与OF的异或为1或者ZF=1,则转移
JNLE / JG	不小于等于/大于则转移	(SF^OF)|ZF=0,即SF与OF的异或为0且ZF=0,则转移

3. 循环指令

指令	说明	测试条件
LOOP	循环	CX ≠ 0，则循环

指令	说明	测试条件
LOOPZ / LOOPE	为零或相等时循环	ZF=1 AND CX≠0，则循环
LOOPNZ / LOOPNE	不为零或不相等时循环	ZF=0 AND CX≠0，则循环

*说明：相同操作**首先CX寄存器减1**，然后根据测试条件决定是否转移*

示例5.47：在首地址为MESS长为19字节的字符串中查找'空格'（20H）字符，如找到则继续执行，否则转标号NO。用循环指令实现程序的循环。

```
    MOV    AL,20H
    MOV    CX,19
    MOV    DI,-1
LK:
    INC    DI
    CMP    AL,MESS[DI]
    LOOPNE  LK    ; 当CX≠0且ZF=0时跳转

    JNZ    NO     ; 当ZF=0时跳转
YES: …
    JMP EXIT
NO: …
EXIT:
    MOV AH,4CH
    INT 21H…
;当LOOPNE LK结束时有两种可能，即CX=0或者ZF=1（即没有找到或找到了），而这两种可能对应的结果是不相同的，因此在LOOPNE下方紧跟着一条JNZ的跳转指令，用以区分这两种情况。需特别注意CX=0与 ZF=1同时成立的情况，即比较的最后一个字符是相同的，所以区分以上两种情况时，需优先判断ZF=1是否成立。
```

七、课后习题

- [ ] 5-3. CMP 和 TEST 指令与其他指令的不同之处在于什么？它们通常都紧跟着跳转指令，用在什么场合?

  CMP 和 TEST指令分别执行减法和逻辑与操作，但不回送结果，只影响标志位，通常用于判断两个数的大小或者用于判断操作数的某些位是1或0。

- [ ] 5-5. 设V是变量，指出下列错误的指令，说出错误原因并修改。

```
MOV AX, [DX]     ;   DX不能做内存指针，寄存器间接寻址方式有：BX、SI、DI、BP可以
MOV DS, DATA     ;   段不能直接送DS，需要先送AX，再由AX送DS
MOV CS, AX   ;   CS不能修改
MOV AX, DL   ;   数据类型不一致
PUSH     AL   ;   必须为16位，应改成AX 不能WORD PTR AL
ADD [BX],    [DI]     ;    不能同问内存
LEA [BX],    V    ;   不能对内存直接写立即数 要改成BX
MOV [DX],    OFFSET  V    ;    DX不能做内存指针，去掉方括号
MOV [SI],    2    ;   立即数位数不确定，需要加PTR运算符
MUL BX, CX   ;   MUL 为单操作数    默认目的寄存器为AX、DX 或 AX
DIV 5    ;   立即数不能作为除数，需要先存入另一寄存器
MOV BYTE[SI],    AX   ;    类型不一致或用了保留字做标识符
MOV AX, [SI+DI] ;   只能基址变址，需要将SI、DI其中一个改为BP/BX
```

```
SHR AX, 4    ;    移位次数非1时，只能用CL计数
CMP 6,  AX   ;    立即数不能出现在目的地址
MOV [FFFF], AX   ;    FFFF会被当做标识符，需要加前缀0或后缀H
MOV AX, BX + 4   ;    如果相对基址，加方括号
JMP FAR PRO ;    需加PTR运算符
```

- [ ] 5-6. 在数据段定义了ARRAY数组，其中依次存储了4个字数据，根据以下要求把第4个字送AX寄存器。

（1）直接寻址

```
MOV AX, ARRAY+6
```

（2）使用BX的间接寻址

```
MOV BX, OFFSET [ARRAY+6]
MOV AX, [BX]
```

（3）使用BX和ARRAY的寄存器相对寻址

```
MOV BX, 0006H
MOV AX, ARRAY[BX]
```

（5）基址变址寻址

```
MOV SI, 0006H
LEA BX, ARRAY
MOV AX, [BX][SI]
```

（6）MOV以外的其他指令

```
SUB AX, AX
ADD AX, ARRAY+6
```

- [ ] 5-17. 在下列程序段的括号中分别填入如下指令，程序执行完后，AX、CX的内容是什么?

```
(1)LOOP L1
(2)LOOPE L1
(3)LOOPNZ L1
MOV AX, 6
MOV CX, 3
L1: ROL AX, CL
TEST AL, 3
(      )
```

第六章 伪指令与源程序格式

> 重点：
>
> ①常用伪指令
>
> ②存储、内存分配相关
>
> ③数据定义
>
> ④计算内存偏移量

一、伪指令

1. 指令与伪指令的区别

	指令	伪指令
有对应的机器语言代码	是	否
处理/执行	cpu执行（硬件）	汇编程序处理（软件）
出现位置	代码段	数据段代码段均可
功能	告诉CPU做什么	对汇编过程进行控制，定义数据变量和程序结构。仅告诉汇编程序对其后面的指令语句和伪指令的操作数如何处理。伪指令不产生对应的机器目标代码。

2. 段定义伪指令
    - 段定义伪指令格式：

```
segment_name SEGMENT    ;段名
... ; 段定义的内容
segment_name ENDS
```

    - ASSUME伪指令格式：
        - ASSUME register_name : segment_name，register_name : segment_name
        - 注意：
            - 只是指定把**某个段分配给哪个段寄存器**，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，通常用两条MOV指令完成这一操作，但是**代码段不需要这样做**，它在**程序初始化的时候完成。**
3. 程序开始和结束伪指令
    - 格式：END [标号]
    - 注意：**只有主程序开始时的标号xxx:才能用end xxx**，表示主程序的范围(类似main函数)，其它标号仅起标记程序段地址的作用。

4. 数据定义与存储器单元分配伪指令
    - 格式：[变量名] 伪指令 N个操作数 [;注释]
    - 变量定义：实质上就是某一个或几个存储单元。
    - 变量的三种属性：
        - 段属性（指明存在于哪个段中）
          变量（名字）对应存储单元的段地址
        - 偏移属性（指明段内的位置）
          变量（名字）对应存储单元的偏移地址
        - 类型属性（指明每个变量占用的字节数）

类型	说明
DB(1字节)	用来定义字节，其后的每个操作数都占用1个字节，8位
DW(2字节)	用来定义字，其后的每个操作数都占用1个字，16位
DD(4字节)	用来定义双字，其后的每个操作数都占用2个字，32位
DF(6字节)	用来定义6个字节的字，其后的每个操作数都占用48位
DQ(8字节)	用来定义4个字，其后的每个操作数都占用4个字，64位
DT(10字节)	用来定义10个字节，其后的每个操作数都占用10个字节，80位

    - 注意：
        - 程序中默认的数据为**十进制数。**
        - 当数据**第一位不是数字**时，应在**前面加0**，避免被误认为是变量或常量。
        - **负数均为补码**形式存放。
        - 字符串用' '括起来。
        - **'?'**表示**只分配存储单元，不存入数值。**（实际上一般以0填充）
    - DUP 复制伪指令
        - 格式：count DUP (operand, ..., operand)
        - 操作：将括号中的操作数重复count次，count既可以是正数也可以是表达式，**DUP操作可嵌套。**
        - 示例：

```
ARRAY DB 2 DUP(1,3,2 DUP(4,5)) ; 循环2份DUP1的内容，DUP1内（1,3，循环两份DUP2的内容（4,5））
;即1 3 4 5 4 5 1 3 4 5 4 5
```

5. 类型属性操作符

   访问内存变量要指定地址，同时指定访问长度进行匹配，**为避免出现两个长度不匹配**的操作数使用类型属性操作符，进行访问。

    - WORD PTR ; 字类型（PTR是强制转换的意思）
    - BYTE PTR ; 字节类型

```
OPER1 DB 3, 4 ; 03 04  这里每隔一个就是一个单位
OPER2 DW 5678H, 9 ; 78 56 09 00   这里每隔一个代表高低位然后是单位

MOV AX, WORD PTR OPER1 ; 从OPER1取1个字 即 AX = 0403H
MOV BL, BYTE PTR OPER2 ; 从OPER2取1个字节 即 BL = 78H
MOV BYTE PTR[DI], 0 ; 将常数0送到DS:[DI]一个字节单元
```

6. THIS操作符和LABEL伪操作
   - 原因：一个变量可以定义成不**同的访问类型**（比如word byte）方便访问，为此引入了THIS和LABEL。
   - 理解：可以理解为C语言中的指针，只不过这个"指针"比较特殊，它每次进行访问的范围有限，即指定的访问类型就是它每次访问的范围。
   - 说明：**THIS运算符**一般与**等值运算符EQU连用**，用来定义一个变量或者标号的类型属性，**所定义的变量或者标号的段基址和偏移量与紧随其后的变量或者标号相同。**
   - 格式：

     ```
     [name=]THIS type ; buf = THIS WORD
     name LABEL type ; VALUE LABEL BYTE
     两者只是指定一个type 长度的访问方式，地址和下一个存储单元地址相同。
     ```

   - 示例：

     ```
     BUF = THIS WORD
     DAT DB 8,9 ; (BUF,DAT)08 09 // 只是BUF访问的是用WORD
     OPR_B LABEL BYTE
     OPR_W DW 4 DUP(2) ; (OPR_B,OPR_W)02 00 02 00 02 00 02 00 // OPR_B访问这组数据的时候是用byte
     ```

7. 表达式赋值伪指令"EQU"和"="
   - 作用：可以用赋值伪操作给表达式赋予一个常量或名字
   - 格式：①表达式名 EQU 表达式②表达式名 = 表达式
   - 注意：
     - 表达式中的**变量或标号**，必须**先定义后引用。**
     - 伪指令"EQU"和"="都**不能为符号分配存储单元**，因此所定义的符号没有段、偏移量和类型等属性。
     - EQU伪操作中的表达式名是不允许重复定义的，而**"="伪操作则允许重复定义。**

```
VALUE = 54
VALUE = VALUE + 89
以下不允许
VALUE EQU 54
VALUE EQU VALUE + 54
```

8. 汇编地址计数器$与定位伪指令
   - 地址计数器$
     - 定义：$用在伪操作的参数段时，表示的是**地址计数器的值**，即$表示当前**正在汇编的指令的偏移地址。**

- 示例：

```
;假定$初值为0
ARRAY DW 3,$+7,7
COU = $
NEW DW COU
;内存模型为：03 00 09 00 07 00 06 00 (COU = 6，符号变量不占存储，但是NEW占存储)
```

- ORG 伪操作
  - 作用：ORG用来**设置当前地址计数器$的值**，可以理解为定位了一个开始地址。
  - 示例：

```
ORG 0H  ;   代表$ = 0 ，同时当前开始存储的地址为0H
DB 3    ;
ORG 4
BUFF    DB  6
ORG $ + 6
VAL DB  9
;内存模型为：03 -- -- -- 06 -- -- -- -- -- -- 09
```

- EVEN伪操作
  - 作用：使下一个变量或指令开始于偶数地址，方便16位变量。
- ALIGN伪操作
  - 作用：使下一个变量的地址从4的倍数开始，方便多字变量。

9. 基数控制伪指令
   - 二进制：0101 0101B
   - 十进制：默认数字均为十进制，23D
   - 十六进制：0ABCD 9876H ；要求：第一个字符必须是0~9，如果为A~F则在前面加上数字0。
10. 过程定义伪指令
    - 定义：子程序又称过程，**可以把一个程序写成一个过程或多个过程。**
    - 格式：

      过程名 PROC [属性] ; 过程开始
      ... ; 过程体
      过程名 ENDP ; 过程结束
    - 注意：
      - 80x86中调用过程指令是CALL，从过程返回的指令是RET。
      - 过程名是标识符，起到标号的作用，是子程序入口的符号地址。
      - 过程的属性可以是FAR或NEAR类型。NEAR为近，是段内调用。FAR类型为远，是跨段调用，缺省时为NEAR。
    - 示例：

```
DATA SEGMENT    ;定义数据段data
    STRING DB 'HELLO,WORLD$'
DATA ENDS

CODE SEGMENT    ;定义代码段code
```

```
    ASSUME  CS:CODE,DS:DATA
    MAIN PROC FAR    ;定义过程main
    MOV AX,DATA
    MOV DS,AX
    LEA DX,STRING
    MOV AH,9
    INT 21H
    MOV AH,4CH
    INT 21H
    MAIN ENDP
CODE ENDS
END MAIN     ;汇编结束，程序起始点main
```

~~二、语句格式~~

1. 格式：

   [name] operation operand [; comment] // [名字] 操作 操作数 [; 注释]

2. 名字项和操作项

   说明：下面各项只能符号变量或常数直接计算，不能寄存器直接运算需要用对应的指令。

   - 名字项
     - 组成：字母、数字和专用字符（?,·,@,_,$）。
     - 注意：
       - 除数字外，所有字符都可以放在源语句的第一个位置。
       - 名字中如果用到·，则必须是第一个字符。
       - 不能与保留字相同。
       - 不能重复定义。
   - 操作项
     - 定义：是一个操作码的助记符，它可以是**指令**、**伪指令**或**宏指令名**。
     - 注意：表达式在**汇编阶段**起作用，只有正确的表达式才能通过汇编。
     - 分类：
       - 算术操作符：+、-、*、/、MOD
       - 逻辑与逻辑移位操作符：AND、OR、NOT、XOR、SHL、SHR
       - 关系操作符：EQ、NE、LT、GT、LE、GE
       - 数值回送操作符：TYPE、LENGTH、SIZE、OFFSET、SEG等

~~三、EXE文件与COM文件~~

具体见书P112

四、课后习题

第七章 分支与循环程序设计

> 重点：
>
> ①主要用于程序分析与填空题
>
> ②或者用于程序设计题

一、程序设计的一般步骤

1. 分析问题，确定算法和数据结构。
2. 根据算法绘制程序流程图。
3. 根据流程图编写程序。
4. 上机调试程序。

二、分支程序设计

1. 单分支程序
   - 例题：双字长数存放在DX和AX寄存器中(高位在DX)，求该数的绝对值(用16位指令)。
   - 算法分析：
     - 双字长数高字在DX中，低字在AX中；
     - 判该数的正负，为正数（最高位为0），该数不处理；为负数，就对该数求补（即反码加1）。
   - 代码：

```
code  segment
    assume  cs:code
start:
    test   dx,8000h          ;测试数的正负，等价于dx最高位与1
    jz     exit              ;不为负数就退出

    not    ax                ;求补，反码加一
    not    dx
    add    ax,1
    adc    dx,0
exit:
    mov    ah,4ch
    int    21h
code  ends
    end    start
```

2. 复合分支程序
   - 例题：从键盘输入一位十六进制数，并将其转换为十进制数输出显示。

$$Y=\begin{cases}X & (0\text{-}9)\ 30H\leqslant X\leqslant 39H\\ 3100H+(X-11H) & (A\text{-}F)\ 41H\leqslant X\leqslant 46H\\ 3100H+(X-31H) & (a\text{-}f)\ 61H\leqslant X\leqslant 66H\end{cases}$$

- 算法分析：
  - 为数字0～9（ASCII码30～39H），无需处理，直接输出；
  - 为大写字母A～F（ASCII码41～46H），先输出31H（1），再输出该数ASCII码-11H；
  - 为小写字母a～f（ASCII码61～66H），先输出31H，再输出该数ASCII码-31H；
  - 该数不为0～9、A～F、a～f，是非法字符，应退出程序或输出错误信息。
- 代码：

```
code segment
    assume cs:code
start:
    mov   ah,1                  ;键盘输入至AL
    int   21h

    cmp   al,30h
    jl    exit                      ;非法输入

    cmp   al,39h
    jle   dig                       ;输入是数字0～9

    cmp   al,41h
    jl    exit                      ;非法输入

    cmp   al,46h
    jle   print                     ;输入是大写A～F

    cmp   al,61h
    jl    exit                      ;非法输入

    cmp   al,66h
    jg    exit                    ;非法输入

    sub al,31h
    jmp   out1                  ;输入是小写a～f
print: sub   al,11h
out1: mov   dl,31h              ;输出字符1
    push  ax                    ;暂存AX
    mov   ah,2
    int   21h                       ;2号中断在int 21H指令子程序中改写了
AL=DL！！！※
    pop   ax                    ;恢复AX
dig: mov dl,al                ;输出个位
    mov   ah,2
    int   21h
exit: mov ah,4ch            ;程序终止并退出
    int 21h
code ends
    end start
```

3. 多分支程序

- 例题：根据键盘输入的一位数字(1～4)，使程序转移到4个不同的分支中去，以显示键盘输入的数字。
- 算法分析：
  - 从键盘输入一个数1～4
  - 建立一个分支向量表branch，集中存放四个分支的偏移地址；
  - 每个偏移地址位16位，占用2个单元；
  - 四个分支的偏移地址在转移地址表的地址是：转移地址表首址+输入数字（0～3）×2；
  - 用间接寻址方式转向对应分支。
- 代码：

```
code    segment
    assume  cs:code,ds:code
start:
    mov   ax,code           ;ds=cs
    mov   ds,ax

    mov   ah,7             ;键盘输入无回显
    int   21h

    cmp   al,31h
    jl    exit                    ;非法输入
    cmp   al,34h
    jg    exit                   ;非法输入

    mov   dl,al             ;放入dl,待显示

    mov   bl,al
    sub   bl,31h             ;转换ascii码为数值

    shl   bl,1                   ;(bl)×2,指向分支向量表中某地址
    mov   bh,0

    jmp   branch[bx]        ;转向分支
r1:    mov   ah,2
    int   21h                       ;显示键盘输入的数字DL
    jmp   exit
r2:    mov   ah,2
    int   21h
    jmp   exit
r3:    mov   ah,2
    int   21h
    jmp   exit
r4:    mov   ah,2
    int   21h
    jmp   exit
exit: mov   ah,4ch            ;程序终止并退出
    int   21h

branch dw r1 ;变量，存放各个分支首地址（注意：变量名和变量值定义必须在同一行）
    dw    r2
    dw    r3
    dw    r4
code    ends
```

```
    end   start
```

三、循环程序设计

1. 计数循环程序
   - 例题：把BX中的二进制数用十六进制显示.（设BX=123AH）。
   - 算法分析：
     - 屏幕显示字符用2号DOS功能调用，DL=输出字符。
     - BX寄存器每4位表示一位十六进制数位：BX=123AH=0001 0010 0011 1010
     - BX=1 2 3 A H,
       1–>31H 2–>32H 3–>33H A–>41H
     - BX从左到右循环移位，每移4位，就把要显示的4位二进制位移到最右边。
     - 取出最右边的4位，加上30H，转换成8位ASCII字符码。
     - 当8位二进制数大于39H时，应再加上7。（41H-3AH=7H）
   - 代码：

```
code    segment
    assume  cs:code
start:  mov   cx,4
shift:
    rol   bx,1              ;连续循环左移4位
    rol   bx,1
    rol   bx,1
    rol   bx,1
    mov   al,bl
    and   al,0fh                ;取最右4位
    add   al,30h            ;转为ascii
    cmp   al,39h
    jle   dig                ;是0～9则转dig

    add   al,7               ;是A～F
 dig: mov   dl,al
     mov   ah,2
     int   21h
     loop  shift

     mov   ah,4ch
     int   21h
code    ends
    end   start
```

2. 条件循环程序
   - 例题：从键盘输入8位二进制数作为逻辑尺。再输入一个英文字母，根据逻辑尺当前的最高位标志显示输出该字母的相邻字符，标志位为0则显示其前趋字符，标志位为1则显示其后继字符。显示相邻字符后，逻辑尺循环左移一位，再接收下一个字母的输入，并依据逻辑尺显示相邻字符，直到回车键结束程序。
   - 算法分析：
     - 循环次数已知，但每次循环所做的操作不同；
     - 设置逻辑尺，循环中依据逻辑尺中的标志位选择操作。

- 代码：

```
code    segment
    assume  cs:code
start:
    mov   bx,0              ;初始化，BL存储逻辑尺
    mov   cx,8
inlog:
    mov   ah,1               ;键盘输入0/1逻辑尺
    int   21h

    cmp   al,30h
    jb    exit                  ;非法输入
    cmp   al,31h
    ja    exit                  ;非法输入

    sub   al,30h             ;输入是0/1，减'0'
    shl   bl,1                  ;逻辑左移
    add   bl,al
    loop  inlog

    mov   ah,2
    mov   dl,10            ;输出换行
    int   21h
inchr:
    mov   ah,1             ;键盘输入字母
    int   21h

    cmp   al,13
    je    exit                ;回车键

    mov   dl,al          ;dl存储输入的字母
    rol   bl,1           ;循环左移，OPR整体向左移一位，最高位移出，同时将其送CF标志位和最低位
    jnc   k30                ;cf=0 则转k30

    inc   dl             ;后继
    jmp   putc
k30: dec   dl            ;前驱
putc: mov   ah,2
    int   21h
    jmp   inchr

exit: mov   ah,4ch               ;程序终止并退出
    int   21h
code    ends
    end   start
```

3. 条件计数循环程序

- 例题：设置键盘缓冲区为16个字节，从键盘输入一串字符，然后再从键盘输入一个单个字符，查找这个字符是否在字符串中出现，如果找到，显示该字符串，否则显示'NOT FOUND'。
- 算法分析：

- 使用DOS系统功能调用10号功能实现键盘缓冲区输入，
- 使用1号功能实现单个字符输入，
- 使用9号功能实现字符串显示输出。
- 程序采用循环结构实现查找，最大计数值为实际输入的字符个数。

- 代码：

```
data    segment
    buffer   db   16,?,16 dup(?),13,10,'$' ;包括回车符的最大输入字符串长度为16，13回车10换行
    inputs   db   13,10,'input string:$' ;回车换行符加提示字符串
    getc     db   13,10,'input char:$'   ;回车换行符加提示字符串
    output   db   13,10,'not found$'     ;回车换行符加提示字符串
data    ends

code    segment
    assume  cs:code,ds:data
start:
    mov   ax,data              ;ds赋值
    mov   ds,ax

    lea   dx,inputs            ;信息提示输入串
    mov   ah,9
    int   21h

    lea   dx,buffer            ;键盘输入串到缓冲区
    mov   ah,10
    int   21h

    lea   dx,getc             ;信息提示输入字符
    mov   ah,9
    int   21h

    mov   ah,1               ;输入字符到al
    int   21h

    mov   bl,al               ;输入要查找的字符保存到bl
    lea   di,buffer+1         ;di作为指针指向缓冲区，第一次循环时指向输入字符串第一个字符
    mov   cl,buffer+1         ;cx设置计数值为实际字符串长度
    mov   ch,0               ;cx高位置为0
seek:
    inc   di
    cmp   bl,[di]
    loopne   seek                ;未完且没找到,转seek继续循环

    jne   nof                     ;没找到,转nof输出‘not found’

    mov   dl,10                  ;输出换行
    mov   ah,2
    int   21h

    lea   dx,buffer+2             ;指向缓冲区,输出字符串
    mov   ah,9
    int   21h
    jmp   exit
```

```
nof: lea   dx,output
    mov   ah,9
    int   21h
exit: mov   ah,4ch
    int   21h
code    ends
    end   start
```

4. 多重循环程序
   - 例题：显示输出20H～7EH的ASCII码字符表，每行16个字符。
   - 算法分析：
     - 20H～7EH的ASCII字符共有95个，需6行显示。（8*16-1-32=95）（‘空格’=20H）
     - 程序需两重循环，内循环输出每行16个字符，循环计数初值为16，程序终止条件是显示最后一个字符。
   - 代码：

```
code    segment
assume  cs:code
    ;常量
    k=16
    first=20h
    last=7eh
start:
    mov   dx,first              ;从第一个开始
a10:
    mov   cx,k                 ;每行16个
a20:
    mov   ah,2
    int   21h                 ;显示字符

    cmp   dl,last               ;是最后一个则退出
    je    exit

    push  dx                    ;暂存dx
    mov   dl,20h                ;空2格
    int   21h
    int   21h

    pop   dx                    ;恢复dx
    add   dx,1
    loop  a20                   ;进入内循环

    push  dx                     ;暂存dx

    mov   dl,13                ;回车
    int   21h
    mov   dl,10                ;换行
    int   21h

    pop   dx                       ;恢复dx
    loop  a10                     ;进入外循环
exit:
    mov   ah,4ch
```

```
    int   21h
code    ends
    end   start
```

四、课后习题

第八章 子程序设计

一、子程序结构

1. 子程序调用指令原理：

   主程序通过调用**指令（CALL）启动子程序**执行。该指令执行时，首先把它**下一条指令的地址（返回地址）压入堆栈**保存。再把**子程序的入口地址置入IP（CS）寄存器**，以便实现转移。子程序执行完毕后，用**返回指令（RET）回到主程序**，返回程序把堆栈里保存的返回地址送回IP（CS）寄存器实现程序的返回。

2. CALL调用指令
   - 作用：用在主程序中实现子程序的调用
   - 功能：
     - 入栈返回地址
     - 转移到目标地址
   - 注意：CALL分为**段内调用（近调用）和段间调用（远调用）**，目标地址支持相对寻址、直接寻址或间接寻址。

3. RET调用指令
   - 作用：用在子程序结束实现返回主程序
   - 功能：
     - 弹出返回地址
     - 转移到返回地址
   - 注意：调用、返回有**段内near和段间far的区别**，但是CALL和RET指令助记符没有区别。

4. 子程序设计调用
   - 注意点：
     - RET指令返回主程序，CALL指令调用子程序
     - 利用过程定义，获得**子程序名和调用属性**
     - 压入和弹出操作要成对使用，**保持堆栈平衡**
     - 子程序开始保护寄存器，返回前提前恢复
     - 安排在**代码段的主程序之外**
     - 子程序**允许嵌套和递归**
   - 子程序需要使用下面的结构：

```
name PROC
    ...;要执行的代码
name ENDP
```

说明：MASM会根据存储模型等信息确定子程序的远近调用，并相应产生调用、返回指令

- 示例：

```
include io32.inc
.data
    msg byte 'Hello World!',13,10,0
.code

HW PROC
    push eax
    mov  eax,offset msg
    call dispmsg
    pop  eax
    RET
HW ENDP

start:
    call HW
    exit 0
end start
```

二、子程序的参数传递

1. 寄存器传递参数
   - 注意点：
     - 把参数存于约定的寄存器
     - 少量数据直接传递参数
     - 大量数据只能传递地址
     - 带有出口参数的寄存器不能保护和恢复
     - 带有入口参数的寄存器可以保护、也可以不保护，但最好能保持一致
2. 共享变量传递参数
   - 说明：对应于高级语言中的全局变量
   - 注意：
     - 子程序和主程序使用同一个变量名存取数据
     - 如果变量定义和使用不在同一个程序模块中，需要利用public、extern声明
     - 共享变量传递参数、子程序的通用性较差
     - 特别适合在多个程序段间、尤其在不同的程序模块间传递数据
3. 堆栈传递参数
   - 注意：
     - 主程序将入口压入堆栈
     - 子程序从堆栈中取出参数
     - 采用堆栈传递参数是程式化的
     - 子程序设置EBP等于当前ESP
     - 利用EBP相对寻址访问堆栈中的参数
     - 出口参数通常不使用堆栈

三、课后习题