汇编实验报告

邱日 学号19215116

实验一 加密与解密

1.实验目的

加密与解密

2.实验内容

题目：编写一个加密程序，实现：将‘449255’保存在数据段NUM中；将这6个数字按照加密规则进行加密后的结果存入到数据段MIMA中。其中加密规则为：'0123456789'分别对应"6435072198"。

3.实验要求

（1）上机实验前，仔细复习课本有关知识；

（2）独立完成实验，画出流程图并上交实验报告；

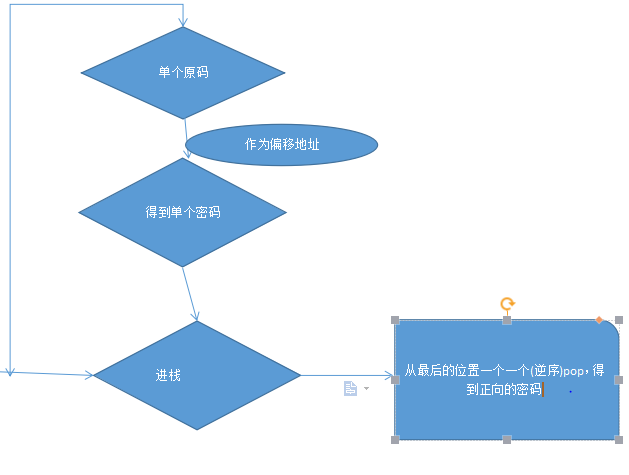
4.实验步骤

1．）算法分析

本次实验可分为三步，逐步细化，最后得到本题的解答

1. 只显示不存储
2. 显示之后逆序存储
3. 基于上步，把结果正放，得到最终解答
4. 优化

流程图



1. ）算法设计

1.用NUM里的数作为寻址时候的偏移地址，得到密码，先把密码输出在显示屏上

2.在输出时用一个栈push,保存密码，然后再MIMA里pop,存好密码（但它是逆序）

3.将上步存好的密码再通过栈逆序存放，得到正向的密码序列

4.优化：其实没有必要用两次栈，只要栈pop的时候是从最后的地址逆过来pop,就能得到正向的序列，由此可以简化代码，另外用伪指令equ可使代码更有通用性，除此，在debug的过程中，我了解到pop不能和byte ptr[bx]一起用，pop一次要pop出一个字，故用ax作为中介，传给相应的地址。

3.）编写代码，调试分析

（a）只显示不存储

小解：（只显示，不存储）

data segment

BIAO db 06h,04h,03h,05h,00h,07h,02h,01h,09h,08h ;存放密码对应表

db 00h,01h,02h,03h,04h,05h,06h,07h,08h,09h ;这两行是对应规则

NUM db 04h,04h,09h,02h,05h,05h ;原码

MIMA db 00h,00h,00h,00h,00h,00h ;密码，初始默认是0

data ends

assume cs:code,ds:data,ss:stack

stack segment

db 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0

stack ends

code segment

start: mov ax,stack

mov ss,ax ;栈段

mov sp,20h ;栈顶指针

mov ax,data

mov ds,ax ;代码段

mov cx,6 ;密码有6位

mov bx,0014h ;数据段放的密码规则共20个数，从他们后面开始

s: mov al,[bx] ;把NUM的数存到al

mov ah,00h ;把ah清零

mov di,ax ;di设为NUM里的那个数（将在下面的程序中得到细化）

mov ax,[di] ;用NUM里的密码来寻址，找到对应的码，放到ax

;;;;;;;;;;;;;

mov dx,ax

mov ax,0200h

add dx,30h ;变为ASCII码，显示

int 21h

;;;;;;;;;;;;;;

inc bx

loop s

mov ax,4c00h

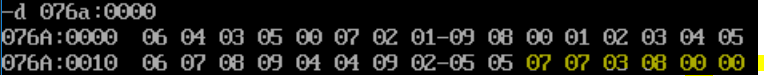
int 21h

code ends

end start



(b)显示之后逆序存储

再解：（逆序存储了） 

data segment

BIAO db 06h,04h,03h,05h,00h,07h,02h,01h,09h,08h

db 00h,01h,02h,03h,04h,05h,06h,07h,08h,09h

NUM db 04h,04h,09h,02h,05h,05h

MIMA db 00h,00h,00h,00h,00h,00h

data ends

assume cs:code,ds:data,ss:stack

stack segment

db 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0

stack ends

code segment

start: mov ax,stack

mov ss,ax

mov sp,20h

mov ax,data

mov ds,ax

mov cx,6

mov bx,0014h

s: mov al,[bx]

mov ah,00h

mov di,ax

mov ax,[di]

;;;;;;;;;;;;;这段是显示密码

mov dx,ax

mov ax,0200h

push dx ;将得到的密码进栈

add dx,30h

int 21h

;;;;;;;;;;;;;;

inc bx

loop s

mov bx,001ah

mov cx,6

sb: pop [bx] ;将栈内的密码出栈

inc bx ;但栈后进先出，是反着放的（在下一个程序中改进）

loop sb

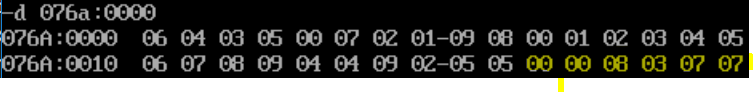
mov ax,4c00h

int 21h

code ends

end start

(c)基于上步，把结果正放，得到解答

再来解（最终解答）（存储加显示）：

data segment

BIAO db 06h,04h,03h,05h,00h,07h,02h,01h,09h,08h

db 00h,01h,02h,03h,04h,05h,06h,07h,08h,09h

NUM db 04h,04h,09h,02h,05h,05h

MIMA db 00h,00h,00h,00h,00h,00h

data ends

assume cs:code,ds:data,ss:stack

stack segment

db 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0

stack ends

code segment

start: mov ax,stack

mov ss,ax

mov sp,20h

mov ax,data

mov ds,ax

mov cx,6

mov bx,0014h

s: mov al,[bx]

mov ah,00h

mov di,ax

mov ax,[di]

;;;;;;;;;;;;;

mov dx,ax

mov ax,0200h

push dx

add dx,30h

int 21h

;;;;;;;;;;;;;;

inc bx

loop s

mov bx,001ah

mov cx,6

sb: pop [bx]

inc bx

loop sb

;以上代码将密码逆着放好了，以下做一个逆序的工作，得到正向的代码

mov bx,001ah

mov cx,6 ;把这6个密码全部进栈

ml1: push [bx]

inc bx

loop ml1

mov bx,001ah

mov cx,6

ml2: pop [bx] ;再把栈中的密码出栈，得到正向的码

inc bx

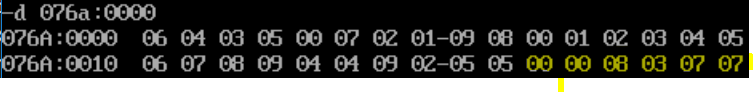
loop ml2

mov ax,4c00h

int 21h

code ends

end start



优化：其实没有必要用两次栈，只要栈pop的时候是从最后的地址逆过来pop,就能得到正向的序列，由此可以简化代码，另外用伪指令equ可使代码更有通用性，除此，在debug的过程中，我了解到pop不能和byte ptr[bx]一起用，pop一次要pop出一个字，故用ax作为中介，传给相应的地址。

本题最终解答（具有存储和显示两个功能）：

data segment

BIAO db 06h,04h,03h,05h,00h,07h,02h,01h,09h,08h

db 00h,01h,02h,03h,04h,05h,06h,07h,08h,09h

NUM db 04h,04h,09h,02h,05h,05h

N equ $-NUM

MIMA db 00h,00h,00h,00h,00h,00h

data ends

assume cs:code,ds:data,ss:stack

stack segment

db 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0

stack ends

code segment

start: mov ax,stack

mov ss,ax

mov sp,20h

mov ax,data

mov ds,ax

mov cx,N

mov bx,0014h

s: mov al,[bx]

mov ah,00h

mov di,ax

mov ax,[di]

;;;;;;;;;;;;;这段是显示密码

mov dx,ax

mov ax,0200h

push dx ;将得到的密码进栈

add dx,30h

int 21h

;;;;;;;;;;;;;;

inc bx

loop s

mov bx,0014h

add bx,2\*N-1

mov cx,N

sb: pop ax

mov [bx],al ;将栈内的密码出栈

dec bx

loop sb

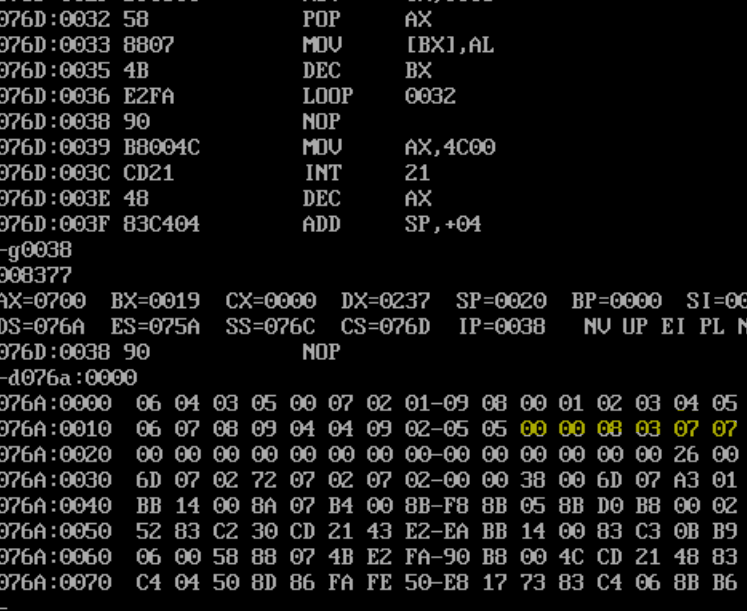
nop

mov ax,4c00h

int 21h

code ends

end start



实验二 显示6的阶乘结果（注：课上只是把6！求出，但没有输出到屏幕，

**这次把它补充完整）**

1.实验目的

（a）求6的阶乘

(b)把结果720转化成ASCII码，输出到屏幕

2.实验内容

题目：编写一个过程求6！，结果放在DX:AX中，同时在屏幕中显示。

3.实验要求

（1）上机实验前，仔细复习课本有关知识；

（2）独立完成实验，画出流程图并上交实验报告；

4.实验所用知识

A)除法指令

div 指令

div是除法指令，使用div作除法的时候：

除数：8位或16位，在寄存器或内存单元中

被除数：（默认）放在AX 或 DX和AX中

结果：运算 8位 16位

商 AL AX

余数 AH DX

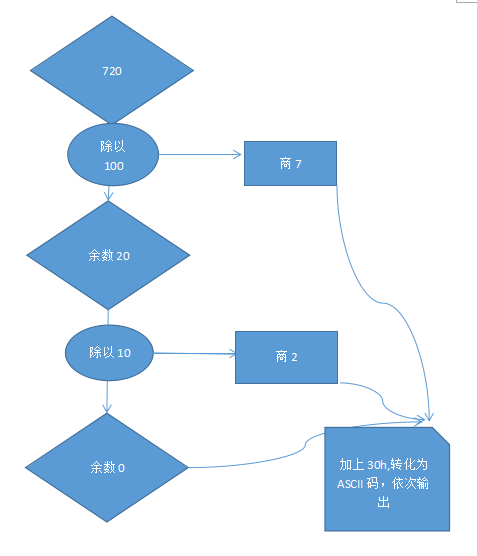
5.实验步骤

1．）算法分析

求阶乘，用循环，ax保存结果，bx从6开始，依次递减，与ax相乘，得到结果

显示，先要得到结果转化成10进制后的每一位，然后加30h得到ASCII码，然后用DOS的2号功能输出到屏幕。

流程图



1. ）算法设计

结果要得到转化成10进制后的每一位，可以这样 720/100=7

余数20/10=2 余数0 分别得到7，2，0

3.）编写代码调试分析

解：

data segment

db 00h,00h,00h,00h

data ends

assume cs:code,ds:data,ss:stack

stack segment

db 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0

stack ends

code segment

start:

mov ax,stack

mov ss,ax

mov sp,0020h ;设栈顶指针

mov ax,1 ;ax最终要放结果，为了方便阶乘计算，初始化为1

mov cx,6 ;求6的阶乘，循环6次

mov bx,6 ;bx从6开始

s: mul bx ;bx从6开始，每次与ax相乘，然后自减

dec bx

loop s ;循环，求解

mov dx,ax

mov dx,0000h

;;;;;;;;;求得ax值为720d,即02D0h,下面把结果输出

mov bx,100

div bx ;除完，dx放余数20，ax放商7

push dx ;720/100=7余20 把余数20给push

mov bx,0

mov bl,al ;除完后把商7存进bx

mov ax,0200h

mov dx,bx

add dx,30h ;加30h,转化成ASCII码，输出

int 21h

pop dx ;把余数20pop到dx,再移到ax

mov ax,dx

mov dx,0

mov bx,10

div bx ;ax里的20除以10，商2放dx,余数0放al

mov dh,0

mov dl,ah

push dx ;把余数0push

mov dl,al ;准备显示商2

mov ax,0200h

add dx,30h ;ASCII

int 21h ;显示2

pop dx ;把余数0显示

mov ax,0200h

add dx,30h

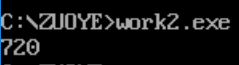
int 21h

mov ax,4c00h

int 21h

code ends

end start



实验三

1.实验目的

2.实验内容

题目：从键盘输入一个长度为N的字符串，存放在A1单元，找出其中所有的小写字母并存放到以A2单元开始的存储区中。统计出小写字母的个数，存放到SL单元中。

3.实验要求

（1）上机实验前，仔细复习课本有关知识；

（2）独立完成实验，画出流程图并上交实验报告；

4.实验步骤

1．）算法分析

（A）大小写转换的问题

就ASCII码的二进制形式来看，除第5位（位数从0开始计算）外，大写字母和小写字母的其他各位都一样。大写字母ASCII码的第5位（位数从0开始计算）为0，小写字母的第5位为1。这样，我们就有了方法： 一个字母，我们不管它原来是大写还是小写：我们将它的第5位置0，它就必将变为大写字母；将它的第5 位置1，它就必将变为小写字母。

（B）如何置1或置0

用什么方法将一个数据中的某一位置0还是置1？用or和and指令。

Test指令的用法：

**“测试”指令TEST**

TEST OPRD1，OPRD2

操作：

执行“与”运算，但运算的结果不送回目标地址，只是影响标志位。

应用：

常用于测试某些位的状态

例：

TEST AL，02H；若AL中D1位为1，则ZF=0，否则ZF=1

TEST AX，8000H；若AX中最高位为1，则ZF=0，否则ZF=1

1. ）算法设计
2. 分为两步，先在数据段定义好字符串内容
3. 加入键盘输入功能

考虑用DOS的0a功能

AH=0aH (ax=0a00)

Ds:DX=字节缓存区首地址

3.）编写代码调试分析

DATA SEGMENT

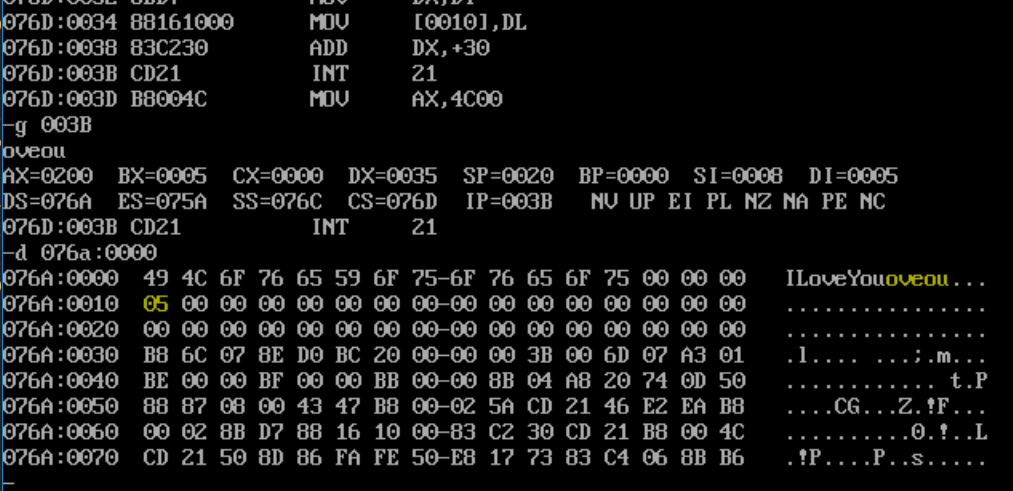
A1 DB ?

N EQU $-A1

A2 DB N DUP(?)

SL DB ?

DATA ENDS

解：

data segment

a1 db 'ILoveYou';定义字符串

N EQU $-a1 ;N为字符长度

a2 db N dup(?)

SL db ? ;放小写字母

data ends

assume cs:code,ds:data,ss:stack

stack segment

db 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0

stack ends

code segment

start:

mov ax,stack

mov ss,ax

mov sp,20h ;设置栈顶指针

mov ax,data

mov ds,ax ;设置数据段

mov cx,N ;字符有几个，就循环几次

mov si,0

mov di,0

mov bx,0

haha:

mov ax,[si] ;a1从0号单元开始放

test al,00100000b ;看它第三位是不是1执行“与”运算，但运算的结果不送回目标 ;地址，只是影响标志位

jz bushow ;第三位如果是1，那就是大写字母，就不显示

push ax ;如果是第三位不是1，那就是小写字母，把它进栈

mov a2[bx],al ;并且把这个小写字母送进相应地址

inc bx ;小写字母个数加一，地址向后挪

inc di

mov ax,0200h

pop dx

int 21h

bushow: inc si

loop haha

mov ax,0200h

mov dx,di

mov SL,dl

add dx,30h

int 21h

mov ax,4c00h

int 21h

code ends

end start

解：键盘输入

data segment

a1 db 'ILoveYou'

N EQU $-a1

a2 db N dup(?)

SL db ?

data ends

assume cs:code,ds:data,ss:stack

stack segment

db 32 dup(0)

stack ends

code segment

start:

mov ax,stack

mov ss,ax

mov sp,20h

mov ax,data

mov ds,ax

mov ax,0a00h

mov dx,0

int 21h

mov cx,N

mov si,0

mov di,0

mov bx,0

haha:

mov ax,[si]

test al,00100000b

jz bushow

push ax

mov a2[bx],al

inc bx

inc di

mov ax,0200h

pop dx

int 21h

bushow: inc si

loop haha

mov ax,0200h

mov dx,di

mov SL,dl

add dx,30h

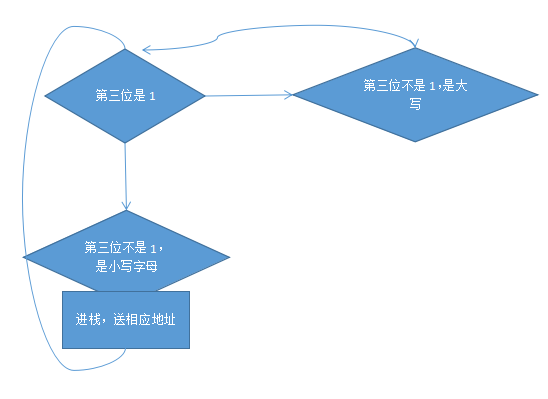
int 21h

mov ax,4c00h

int 21h

code ends

end start

流程图

实验四 二进制数逆序

1.实验目的

逆序二进制数

2.实验内容

题目：编写一个完整程序，将DB1中的二进制数按位倒序排列存入DB2中，即DB1第0位送DB2第7位，DB1第1位送DB2第6位，依次类推

DATA SEGMENT

DB1 DB XX;任一字节数

DB2 DB?

DATA ENDS

3.实验要求

（1）上机实验前，仔细复习课本有关知识；

（2）独立完成实验，画出流程图并上交实验报告；

4.实验步骤

1．）算法分析

可以考虑用移位指令，通过CF位来传递0和1

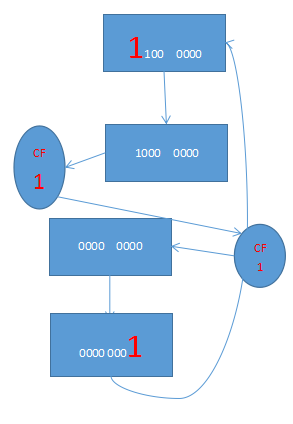
shl逻辑左移指令，功能为：

（1）将一个寄存器或内存单元中的数据向左移位；

（2）将最后移出的一位写入CF中；

（3）最低位用0补充。

2．）算法设计

流程图

3.）编写代码，调试分析

再解：

data segment

buf1 db 11000000b

buf2 db 00000000b

data ends

assume cs:code,ds:data

code segment

start: mov ax,data

mov ds,ax

mov ah,buf1

mov dh,buf2

mov cx,8

yi:clc

shl ah,1 ;shl逻辑左移指令

rcr dh,1 ;**带进位的循环移位（右移）**

loop yi

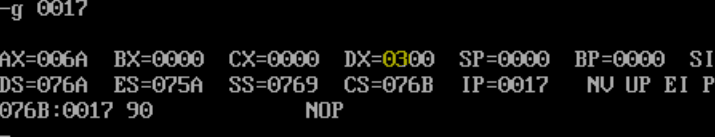
nop

mov ax,4c00h

int 21h

code ends

end start



移位指令总结：

1. Shl(shift logic left) ;逻辑左移
2. Sal(shift arithmetic left) ;算术左移
3. Shr(shift logic right) ;逻辑右移
4. Sar(shift arithmetic right) ;算术右移
5. Rol(rotat left) ;循环左移
6. Ror(rotat right) ;循环右移
7. Rcl(rotat left with carry) ;带进位循环左移
8. Ror(rotat right right wth carry) ;带进位循环右移

共添加了两个功能

新功能1，把多个数进行逆序：

data segment

dba db 00h,01h,02h,03h,04h,05h,06h,07h

dbb db 0,0,0,0,0,0,0,0

data ends

stack segment

db 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0

stack ends

assume ds:data,cs:code,ss:stack

code segment

start: mov ax,stack

mov ss,ax

mov sp,20h

mov ax,data

mov ds,ax

mov bx,0

mov cx,8

s1: push [bx]

inc bx

loop s1

mov bx,8

mov cx,8

s2:

pop [bx]

inc bx

loop s2

mov ax,4c00h

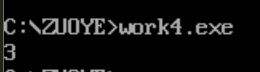
int 21h

code ends

end start



再解：



data segment

db1 db 11000000b

db2 db 00000000b

data ends

assume cs:code,ds:data

code segment

start:

mov ax,data

mov ds,ax

mov dx,0

mov bl,00000001b

mov bh,10000000b

mov ax,0

mov cx,8

s: mov al,byte ptr ds:[0]

or al,bl

cmp al,byte ptr ds:[0]

jnz ok

or ah,bh

ok:shl bl,1

shr bh,1

loop s

mov byte ptr ds:[1],ah

mov dl,ah

mov ax,0200h

add dx,30h

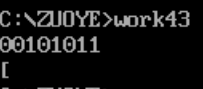
int 21h

mov ax,4c00h

int 21h

code ends

end start

新功能2：把移位后的结果打印出来

data segment

db1 db 11010100b

db2 db 00000000b

data ends

assume cs:code,ds:data,ss:stack

stack segment

db 32 dup(0)

stack ends

code segment

show0:

push ax

push dx

mov ax,0200h

mov dx,30h

int 21h

pop dx

pop ax

ret

show1:

push ax

push dx

mov ax,0200h

mov dx,31h

int 21h

pop dx

pop ax

ret

showhuiche:

push ax

push dx

mov ax,0200h

mov dx,0ah

int 21h

pop dx

pop ax

ret

start:

mov ax,stack

mov ss,ax

mov sp,32

mov ax,data

mov ds,ax

mov dx,0

mov bl,00000001b

mov bh,10000000b

mov ax,0

mov cx,8

s: mov al,byte ptr ds:[0]

or al,bl

cmp al,byte ptr ds:[0]

jnz ok

or ah,bh

call show1

jmp buxie0

ok:

call show0

buxie0:

shl bl,1

shr bh,1

loop s

call showhuiche

mov byte ptr ds:[1],ah

mov dl,ah

mov ax,0200h

add dx,30h

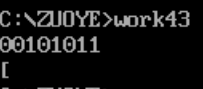
int 21h

mov ax,4c00h

int 21h

code ends

end start



实验五 求绝对值之差

1.实验目的

2.实验内容

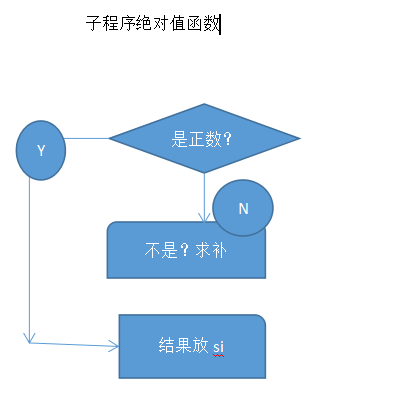
题目：在BUF1和BUF2两个数据区中，各定义有5个带符号的数据，试编制一完整的源程序，求它们对应项的绝对值之差，并将差值存入以SUB为首地址的数据区中。

3.实验要求

（1）上机实验前，仔细复习课本有关知识；

（2）独立完成实验，画出流程图并上交实验报告；

子函数流程图



4.实验步骤

1．）算法分析

可以设计一个子函数，专门用来求绝对值

1. ）算法设计

方法一（写一个求绝对值的子函数）

1）用si来给子程序传参数，子函数absolute的功能就是对si里的数进行处理，若是正，就原封不动送还si，若是负，就求补，仍送回si

方法二，（不写子函数）通过si，di的值来寻址

3.）编写代码调试分析

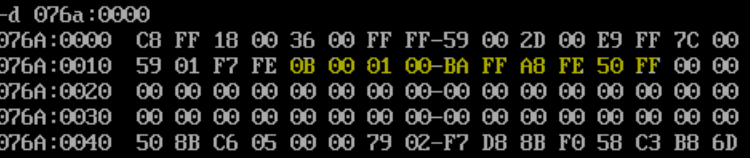
DATA SEGMENT

BUF1 DW -56,24,54,-1,89

BUF2 DW 45,-23,124,345,-265

SUM DW 10DUP(0)

DATA ENDS



解：

data segment

buf1 dw -56,24,54,-1,89

buf2 dw 45,-23,124,345,-265

result dw 10 dup(0)

data ends

assume cs:code,ds:data,ss:stack ;定义栈段

stack segment

db 0,0,0,0 ,0,0,0,0, 0,0,0,0 ,0,0,0,0

stack ends

code segment

absolute: push ax ;求绝对值的子函数

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;该子程序功能是吧si里的数若正则还是本身，若负则求补，仍送回si

mov ax,si

add ax,0 ;为了用jns指令，必须之前要运算

jns zhen

neg ax ;如果是负数，就不跳转，就求补

zhen :mov si,ax ;如果ax里的数是正就直接把他送回si

;;;;;;;;;;;;;;;;;;;;;;;;;;;;

pop ax ;恢复好

Ret ;返回

start:mov ax,stack

mov ss,ax

mov ax,data

mov ds,ax ;数据段设为ds

mov bx,0

mov si,0

mov di,0

mov cx,5;有5对数，循环5次

s: mov ax,word ptr [bx].00h ;buf1的数们偏移地址从0开始，每个数占一个字节

mov si,ax ;把数送si

call absolute ;调用子程序求绝对值

mov ax,si ;buf1的绝对值存ax

mov dx,word ptr [bx].0ah;buf2的数们偏移地址从0ah开始，每个数占一个字节

mov si,dx

call absolute;求绝对值

mov dx,si ;buf2的绝对值存dx

sub ax,dx ;两绝对值相减

push ax ;结果进栈

pop word ptr result[bx] ;结果放result里面

add bx,2

loop s

nop

mov ax,4c00h

int 21h

code ends

end start

另解：

assume cs:code,ds:data

data segment

buf1 dw -56,24,54,-1,89

buf2 dw 45,-23,124,345,-265

sum dw 10 dup(0)

data ends

code segment

start:mov ax,data

mov ds,ax ;数据段

mov cx,5 ;一共5个数循环5次

lea si,buf1 ;si存放buf1的偏移地址

lea di,buf2 ;di存放buf2的偏移地址

lea bp,sum ;bp存放sum的偏移地址

next3:

mov bx,ds:[si]

add bx,0

jns next1

neg bx

next1:

mov dx,ds:[di]

add dx,0 ;用jns选择跳转指令之前一定先要计算

jns next2

neg dx ;如果是正数，就跳转，不求补，如果是负数，就求补

next2:

sub bx,dx ; 求绝对值之差

mov ds:[bp],bx ; 结果送ds:[bp]

add si,2 ;由于定义的是一个字（dw），所以si要移两位

add di,2 ;由于是字，di要移动两位

add bp,2 ;由于是字

loop next3

nop

mov ax,4c00h

int 21h

code ends

end start

1. 总结
2. 为方便在debug里调试，可以在程序mov 4c00h之前写一个空指令nop,到debug时先U，看编译后的代码，找到nop所在地址，然后G这个地址，即可运行本程序，另外用d076a:0000来查看数据段的内容。
3. 用jns这类选择跳转指令之前一定要先计算，add dx,0 并不多余

实验六 冒泡排序

1. 实验目的

以前的实验做得不好，再做一遍

2.实验内容

题目：冒泡排序

3.实验要求

（1）上机实验前，仔细复习课本有关知识；

（2）独立完成实验，画出流程图并上交实验报告；

4.实验步骤

1．）算法分析

2．）算法设计

4)所用知识

1）Dup

dup是一个操作符，在汇编语言中同db、dw、dd 等一样，也是由编译器识别处理的符号。它是和db、dw、dd 等数据定义伪指令配合使用的，用来进行数据的重复。dup示例db 3 dup (0) 定义了3个字节，它们的值都是0，相当于 db 0,0,0

2）两层循环如何避免冲突

我们应该在每次开始内层循环的时候，将外层循环的cx中的数值保存起来，在执行外层循环的loop指令前，再恢复外层循环的cx数值。 我们可以用寄存器dx来临时保存cx中的数值。

1. 【bx】的方法来寻址
2. 共N个数，需要比较N次

3.）编写代码调试分析

data segment

shuzu db 08h,02h,09h,04h,05h,0ah,01h,03h,07h,06h ;10个乱序的数

N EQU $-shuzu ;N为这些数的长度

xulie db 10 dup(0)

data ends

assume cs:code,ds:data

code segment

start:

mov ax,data

mov ds,ax

mov bx,0

mov cx,N-1;共N个数，需要比较N次

daxunhuan: mov dx,cx

mov bx,0

xiaos :

mov al,[bx] ;前一个数

inc bx ;指向下一个数

cmp al,[bx] ;比较前后两个数（其中前一个数已经暂时放在al里）

jna zhenchang

xchg al,[bx] ;如果前一个数比后一个数大，就交换

zhenchang:

mov [bx-1],al ;把暂时放在al里的数又放回原来的前一个位置

loop xiaos

mov cx,dx

loop daxunhuan

nop

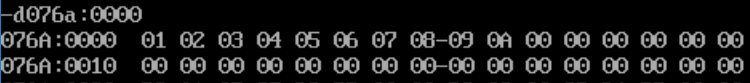
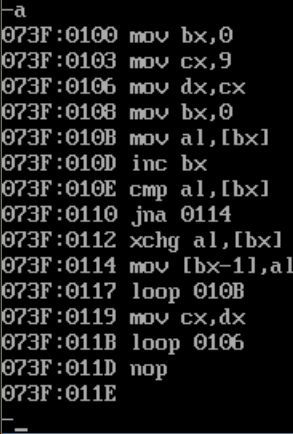
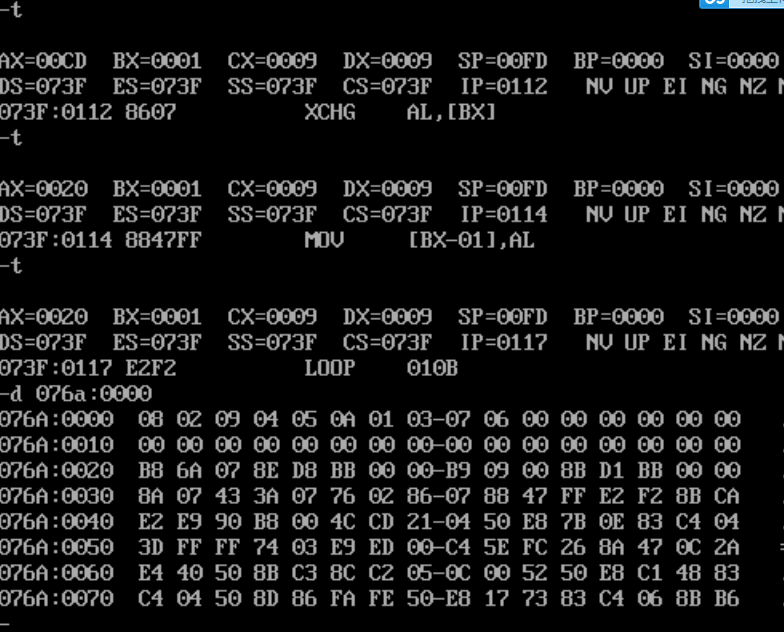
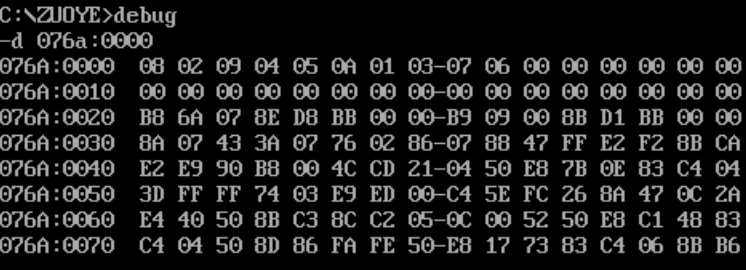
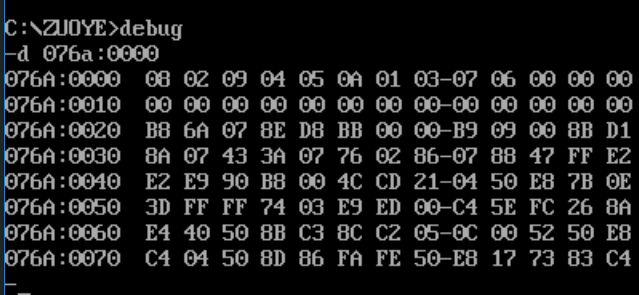
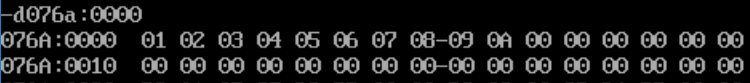
mov ax,4c00h

int 21h

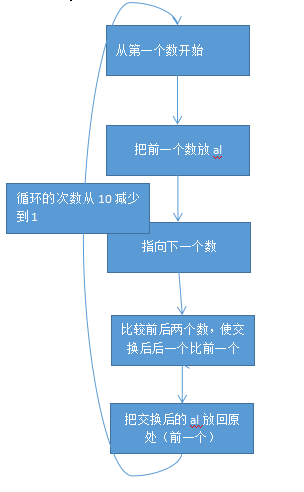
code ends

end start

实验结果：



流程图



**《邱日汇编语言六讲》**

**邱日 编著**

南京农业大学 计科151 邱日2016.12.26

**80X86汇编语言程序设计 20舍209**

**[19215116@njau.edu.cn](mailto:19215116@njau.edu.cn)**

**[3103204417@qq.com](mailto:3103204417@qq.com)**

**目 录**

第1章------------ 预备知识

第2章----------- 寻址方式

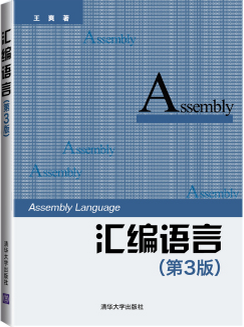
第3章----------- 指令系统

第4章---------- 汇编语言程序格式

第5章---------- 基本汇编语言程序设计

第6章---------- 案例分析

**前言**

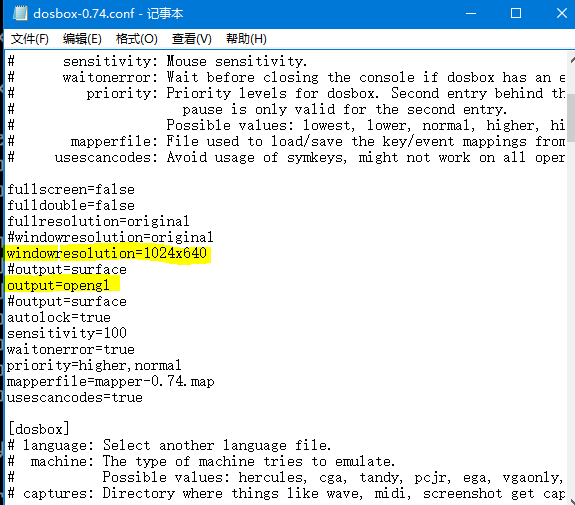


学习汇编，必须要看王爽，必须要看王爽，必须要看王爽，重要的事情说三遍。-----邱日

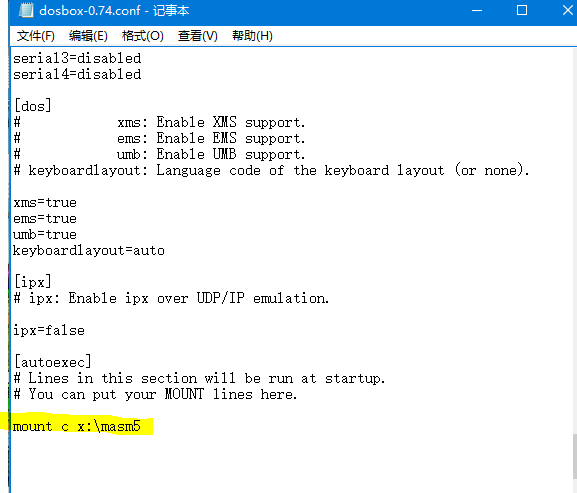
第一章：基础知识

1. DOSBOX的使用

（1）如何调整Dosbox窗口大小



（2）如何挂载



（2）Dosbox常用命令

1. CD choose directory
2. DIR directory
3. REN rename
4. DEL delete
5. MD make directory
6. COPY
7. TYPE 显示文件内容
8. HELP HELP DIR
9. 小知识

（1）存储容量：

1Byte(字节)=8bit(比特)

1B=8b

1KB=2^10B=1024B（Byte）

K ----->M ------>G-------> T ×1024

（2）进制转换：

1Word=2B=16b 1Byte=8bit

2 Binary

8 Octal

10 Decimal

16 Hexadecimal

1. 段地址和偏移地址：

段地址×16+偏移地址=物理地址

乘以16即在16进制的情况下左移4位

M：N M×16+N

2000：1F60

20000

+ 1F60

\_\_\_\_\_\_\_

21F60

1. 寄存器

16位寄存器max: 2^16-1=65535B=64KB

8086的16位通用寄存器：

AX BX CX DX SI DI BP SP

1.8086通用寄存器简介：

分类：

（1）数据寄存器：

AX Accumulator (累加寄存器) AH (High 8) + AL(Low 8)=AX (16)

BX Base(基址寄存器) BH (High 8) + BL(Low 8)=BX (16)

CX Count(计数寄存器) CH (High 8) + CL(Low 8)=CX(16)

DX Data(数据寄存器) DH (High 8) + DL(Low 8)=DX(16)

（2）变址（Index）寄存器：

SI Source Index(源变址寄存器)

DI Destination Index(目的变址寄存器)

（3）指针（Pointer）寄存器：

SP Stack Pointer(堆栈指针)

BP Base Pointer(基址指针寄存器)

功能简介：

AX用的最多，计算传数据

BX放存储器地址

CX循环串操作时计数（隐含）

DX放16位数据或外设端口地址

SI 和DI用于存储器操作时地址

SP放栈顶的偏移地址，不作他用

BP数据在栈段的基地址

SP和BP与SS段寄存器联合使用

2.8086段寄存器简介

段（Segment）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)

3.8086控制寄存器

控制寄存器：

IP Instruction Pointer指令指针寄存器和CS联用 CS:IP

Flags 即PSW （Program Status Word）(状态寄存器)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|  |  |  |  | OF | DF | IF | TF | SF | ZF |  | AF |  | PF |  | CF |

OF Overflow Flag 溢出标志

DF Direction Flag 方向标志

IF Interrupt Flag 中断标志

TF Trap Flag 单步跟踪标志 陷阱标志

SF Sign Flag 符号标志

ZF Zero Flag 零标志

AF Auxiliary Carry Flag(辅助进位标志)

PF Parity Flag 奇偶标志

CF Carry Flag 进位标志

1. Debug.exe的使用

Debug

1. U unassemble
2. D dump 显示内存
3. G go
4. R register
5. T trace
6. P proceed
7. A assemble
8. Q quit
9. N name
10. L load
11. W write
12. H hex
13. M move
14. ? 查看

**80X86汇编语言程序设计**

**邱日**

**[19215116@njau.edu.cn](mailto:19215116@njau.edu.cn)**

**[3103204417@qq.com](mailto:3103204417@qq.com)**

**20舍209**

课程介绍

* **专业基础课、必修课**
* **计算机组成原理的一个部分**
* **软件开发的一个组成部分**
* **一种低级语言的程序设计**
* **高级语言程序设计的扩展**
* **先修课**
* **高级语言程序设计**
* **后续课**
* **微机原理及接口技术**
* **计算机组成原理**
* **……**

使用教材

王爽著（第3版）

教学内容

**第1章 预备知识**

第2章 寻址方式

第3章 指令系统

第4章 汇编语言程序格式

第5章 基本汇编语言程序设计

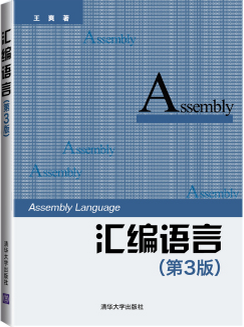
教学要求

* **课堂上注意听讲、重要的是理解**
* 部分补充例题等内容需要笔记
* 课堂外一定要看书复习或预习、完成作业
* 充分利用上机时间、多编程练习

考核要求

* **考试成绩：70％**
* 平时成绩：30％（作业、上课表现、考勤等）

**第1章   
 预备知识**



主要内容：

* **计算机系统概述**
* 数据存储方式
* 数据表示
* 8086寄存器组
* 存储器编址
* 什么是汇编语言
* 汇编语言程序举例

**§1.1 计算机系统概述**

* **硬件（Hardware）**
* 中央处理单元CPU
* 控制器、运算器、寄存器
* 存储器
* 主存储器：RAM和ROM
* 辅助存储器：磁盘、光盘、U盘
* 外部设备
* 输入设备和输出设备
* 软件（Software）
* 系统软件
* 应用软件

程序员能看到的硬件

* **中央处理单元 CPU（Intel 80x86）**
* **对汇编语言程序员，最关心其中的寄存器**
* **存储器（主存储器）**
* **呈现给汇编语言程序员的，是存储器地址**
* **外部设备（接口电路）**
* **汇编语言程序员看到的是端口（I/O地址）**

寄存器（Register）

* **寄存器是CPU内部的高速存储单元**
* **它们为处理器提供各种操作所需要的数据或地址等信息**
* **汇编语言程序采用它们各自的符号名**
* **16位Intel 8086/80286 CPU中有**
* **AX BX CX DX**
* **SI DI BP SP**
* **32位80386/80486/Pentium系列 CPU中有**
* **EAX EBX ECX EDX**
* **ESI EDI EBP ESP**

存储器地址（Address）

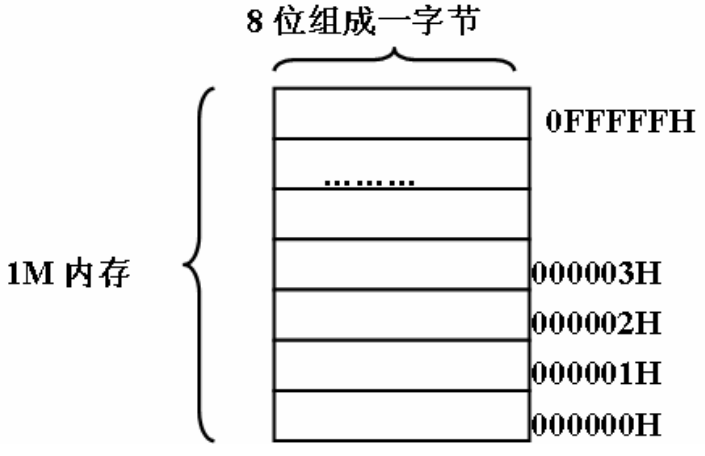
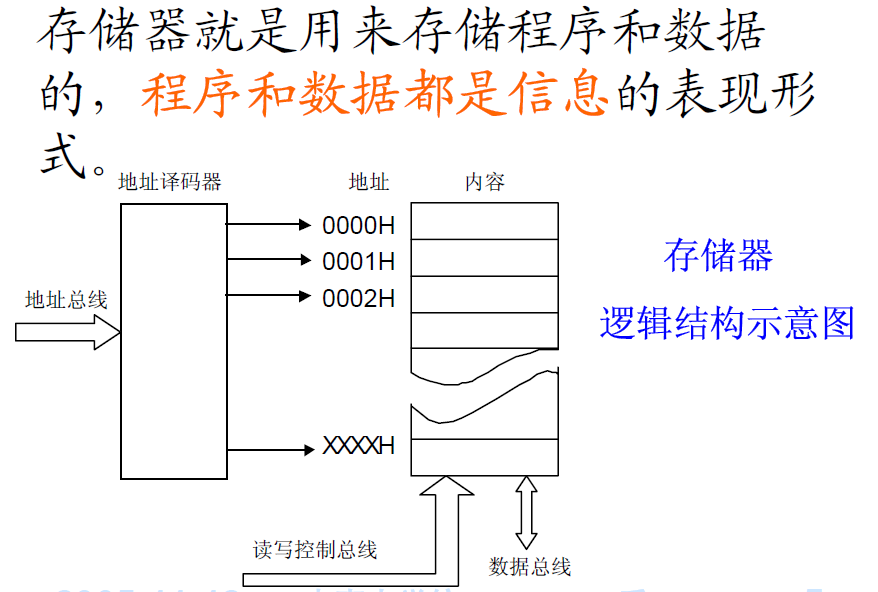
* **存储器是由大量存储单元组成，需要用编号区别每个单元：编号＝地址**
* **存储器地址是存储器中存储单元的编号**
* **每个存储单元存放一个字节量的数据**
* **一个字节B（Byte）＝8个二进制位b（bit）**
* **采用十六进制数来表达地址**
* **Intel 8086具有1兆字节（1MB）存储器容量**
* **存储器地址表示为：00000H ～ FFFFFH**
* **其中大写H（或小写h）表示是十六进制数**

端口（Port）

* **I/O接口电路由接口寄存器组成，需要用编号区别各个寄存器：编号＝地址**
* I/O地址是接口电路中寄存器的编号
* 端口是I/O地址的通俗说法
* 系统通过这些端口与外设进行通信
* 采用十六进制数来表达端口
* Intel 8086支持64K个8位端口
* I/O地址可以表示为：0000H ～ FFFFH

**§1.2 数据存储方式**

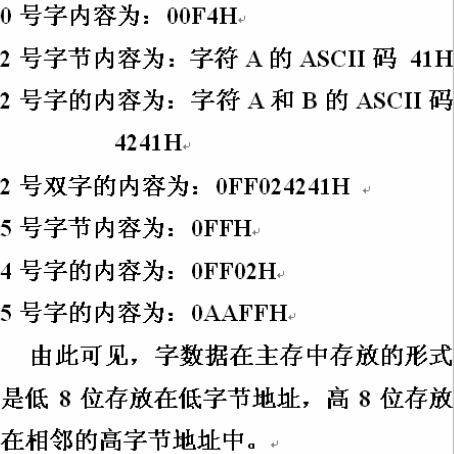
* **字：由2个字节组成，字地址由两字节中地址较小的一个即低字节的地址决定。存放的方式是低8位存放低字节，高8位存放高字节。**
* 双字：双字的地址也由四个字节中的最低地址确定
* **问题：一个字数据该怎样存入计算机主存呢? 双字数据呢？  
  高-高 低-低  
  字：要占有连续的两个字节. 16位中，低8位存放在低地址字节，高8位存放在相邻的高地址字节中。  
  双字：32位中，低16位存放在低地址字，高16位存放在相邻的高地址字地址中。**



**§1.3 数据表示**

* **数据在计算机中的表示形式**
* 进制与编码

1.3.1 数据在计算机中的表示形式



真值与机器数

* **真值：现实中真实的数值**
* 机器数：计算机中用0和1数码组合表达的数值
* 无符号数：只表达0和正整数的定点整数
* 有符号数：表达负整数、0和正整数的定点整数
* 符号位需要占用一个位
* 常用机器数的最高位
* 0表示正数、1表示负数
* 定点数：固定小数点的位置表达数值的机器数
* 定点整数：将小数点固定在机器数的最右侧表达的整数
* 定点小数：将小数点固定在机器数的最左侧表达的小数
* 浮点数：小数点浮动表达的实数

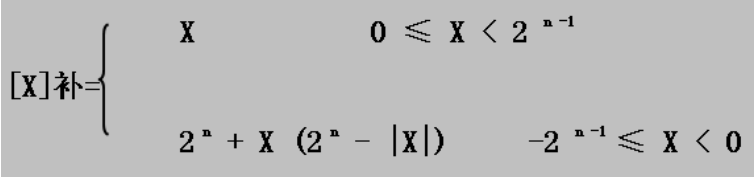
**无符号数表示范围**

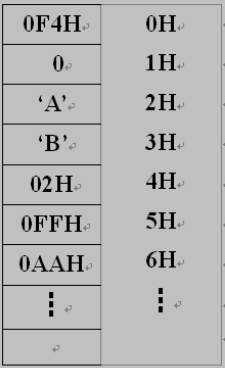
**有符号数表示范围**

（1）数值数据的表示形式

* **对于有符号数，一律采用二进制补码形式**

此处不是乘号，仅表示“或者”



14

补码

* **补码表示的几个特点：  
  1.所有正数的补码表示最左(高)位为0，其二进制补码表示为本身；  
  2.所有负数的补码表示最左(高)位为1，其补码表示为：(原码)除符号位保持不变外，其它位取反加1。  
  例如：设n=8，  
  [50]补 = [00110010B]补 = 00110010B  
  [-50]补 = [-00110010B]补 =11001110B**

**3.一个二进制补码的最高位向左延伸S位，所得到的仍 是此数的补码表示。**

（2）字符数据的表示形式

* **键盘输入的字母和数字、显示器显示的文字等都是字符信息。  
  西文：常用的将字符与2进制数对应起来的编码方法是美国信息标准交换代码ASCII码。  
  汉字： GB2312编码（1980）、GBK编码（1990）**
* 区位码表：分94区\*94位，包括一级汉字（最常用汉字
* ，按拼音排序）和二级汉字（稍常用汉字，按部首排
* 序），均为简体汉字，共约6700个。

ASCII码（美国标准信息交换码）

* **标准ASCII码用7位二进制编码，有128个**
* 不可显示的控制字符
* 前32个和最后一个编码
* 回车CR：0DH 换行LF：0AH 响铃BEL：07H
* 可显示和打印的字符：20H后的94个编码
* 数码0～9：30H～39H
* 大写字母A～Z：41H～5AH
* 小写字母a～z：61H～7AH
* 空格：20H
* 扩展ASCII码：最高D7位为1，表达制表符

1.3.2 进制与编码

* **二进制**
* 十六进制
* 进制转换
* BCD码

二进制

* **便于计算机存储及物理实现**
* 特点：逢二进一，由0和1两个数码组成，基数为2，各个位权以2k表示
* 二进制数：
* anan-1…a1a0.b1b2…bm＝
* an×2n＋an-1×2n-1＋…＋a1×21＋a0×20
* ＋b1×2-1＋b2×2-2＋…＋bm×2-m
* 其中ai，bj非0即1

十六进制

* **用于表达二进制数，相互转换简单**
* 基数16，逢16进位，位权为16k，16个数码：
* 0，1，2，3，4，5，6，7，8，9
* A，B，C，D，E，F
* 十六进制数：
* anan-1…a1a0.b1b2…bm＝
* an×16n＋an-1×16n-1＋…＋a1×161＋ a0×160
* ＋b1×16-1＋b2×16-2＋…＋bm×16-m
* 其中ai，bj是0～F中的一个数码

十进制整数转换为二或十六进制数

* **整数部分转换：用除法**
* 十进制数整数部分不断除以基数2或16，并记下余数，直到商为0为止
* 由最后一个余数起逆向取各个余数，则为转换成的二进制和十六进制数
* 126＝01111110B
* 126＝7EH

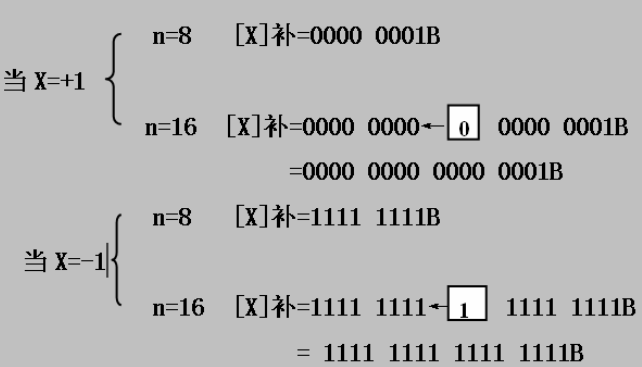
十进制小数转换为二或十六进制数

* **小数部分转换：用乘法**
* 分别乘以各自的基数，记录整数部分，直到小数部分为0为止
* 0.8125＝0.1101B
* 0.8125＝0.DH
* 小数转换会发生总是无法乘到为0的情况
* 可选取一定位数（精度）
* 将产生无法避免的转换误差

二或十六进制数转换为十进制数

* **方法：按权展开**
* 二进制数转换为十进制数
* 0011.1010B
* ＝1×21＋1×20＋1×2-1＋0×2-2＋1×2-3
* ＝3.625
* 十六进制数转换为十进制数
* 1.2H
* ＝1×160＋2×16－1
* ＝1.125





BCD码（Binary Coded Decimal）

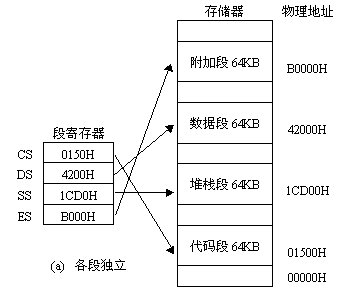
* **二进制编码的十进制数**
* **一个十进制数位0～9用4位二进制编码来表示**
* **常用8421 码：低10个4位二进制编码表示**
* **压缩BCD码：一个字节表达两位BCD码**
* **0001 0000 =10 ,1000 1001 =89**
* **非压缩BCD码：一个字节表达一位BCD码（低4位表达数值，高4位常设置为0）**
* **0000 0000 = 0 ,0000 0001 = 1 ,0000 0010 = 2**
* **BCD码很直观**
* **BCD码：0100 1001 0111 1000.0001 0100 1001**
* **十进制真值：4978.149**

§1.4 存储器编址

* **存储器的编址**
* **实地址模式下的存储器地址变换**

1.4.1 存储器的编址

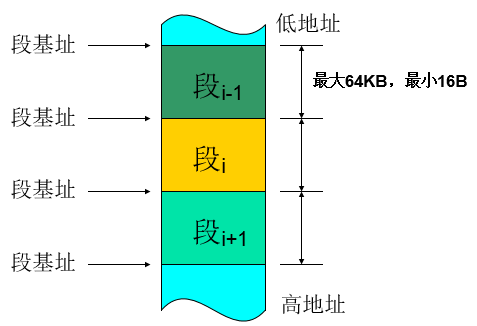
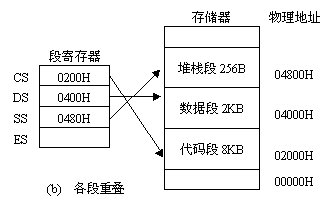
* **8086 CPU有20条地址线**
* **最大可寻址空间为220＝1MB**
* **物理地址范围从00000H～FFFFFH**
* **8086 将1MB空间分成许多逻辑段（Segment）**
* **这样，一个存储单元除具有一个唯一的物理地址外，还具有多个逻辑地址。**
* **物理地址：内存单元在整个内存空间中的惟一的20位地址00000H～FFFFFH**
* **逻辑地址：段基地址 : 段内偏移地址**
* **段地址说明逻辑段在主存中的起始位置**
* **8086规定段地址必须是模16地址：xxxx0H**
* **省略低4位0000B，段地址就可以用16位数据表示，就能用16位段寄存器表达段地址**
* **偏移地址说明主存单元距离段起始位置的偏移量**
* **每段不超过64KB，偏移地址也可用16位数据表示**
* **8086对逻辑段要求：**
* 段地址低4位均为0
* 每段最大不超过64KB
* 8086对逻辑段并不要求：
* 必须是64KB
* 各段之间完全分开（即可以重叠）



**1MB空间最多能分成多少个段？**

* 每隔16个存储单元就可以开始一个段
* 所以1MB最多可以有：
* 220÷16＝216＝64K 个段
* 1MB空间最少能分成多少个段？
* 每隔64K个存储单元开始一个段
* 所以1MB最少可以有：
* 220÷216＝16 个段

1.4.2 地址变换



**段首地址**

**物理地址=段基地址×16+偏移地址**

**例：已知 CS=1055H,DS=250AH,ES=2EF0H,SS=8FF0H  
,画出各段在内存中的分布。**

CS=1055H

* 段首地址=10550H
* DS=250AH
* 段首地址=250A0H
* ES=2EF0H
* SS=8FF0H
* **例：设某操作数存放在数据段，DS=250AH，数据所在单元的偏移地址=0204H。则该操作数所在单元的物理地址为：**
* **250AH ×16+0204H = 252A4H**

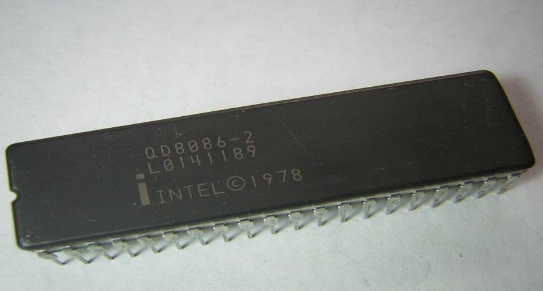
§1.5 8086寄存器组

* **8086微处理器**
* **8086寄存器组**

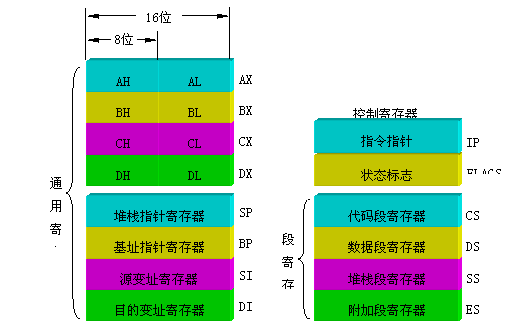
1.5.1 8086微处理器

* **微处理器是微机的硬件核心，主要包含指令执行的运算和控制部件，还有多种寄存器**
* 对程序员来说，微处理器抽象为以名称存取的寄存器
* 8086可访问主存空间为1MB，20条地址线，16条数据线。
* 内部结构有两个功能模块，完成一条指令的取指和执行功能
* 总线接口单元BIU：负责读取指令和操作数
* 执行单元EU ：负责指令译码和执行

8086的内部结构



1.5.2 8086寄存器组



通用寄存器

* **8086的16位通用寄存器是：**
* **AX BX CX DX**
* **SI DI BP SP**
* **其中前4个数据寄存器都还可以分成高8位和低8位两个独立的寄存器**
* **8086的8位通用寄存器是：**
* **AH BH CH DH**
* **AL BL CL DL**
* **对其中某8位的操作，并不影响另外对应8位的数据**

数据寄存器

* **数据寄存器用来存放计算的结果和操作数，也可以存放地址**
* **每个寄存器又有它们各自的专用目的**
* **AX－－累加器，使用频度最高，用于算术、逻辑运算以及与外设传送信息等；**
* **BX－－基址寄存器，常用做存放存储器地址；**
* **CX－－计数器，作为循环和串操作等指令中的隐含计数器；**
* **DX－－数据寄存器，常用来存放双字长数据的高16位，或存放外设端口地址。**

变址和指针寄存器

* **变址寄存器常用于存储器寻址时提供地址**
* **SI是源变址寄存器**
* **DI是目的变址寄存器**
* **指针寄存器用于寻址内存堆栈内的数据**
* **SP为堆栈指针寄存器，指示栈顶的偏移地址，不能再用于其他目的，具有专用目的**
* **BP为基址指针寄存器，表示数据在堆栈段中的基地址**
* **SI和DI在串操作指令有特殊用法**
* **SP和BP寄存器与SS段寄存器联合使用确定堆栈段中的存储单元地址**

段寄存器

* **8086有4个16位段寄存器，每个段寄存器确定一个逻辑段的起始地址，每种逻辑段均有各自的用途**
* **CS（Code Segment）**
* **指明代码段的起始地址**
* **SS（Stack Segment）**
* **指明堆栈段的起始地址**
* **DS（Data Segment）**
* **指明数据段的起始地址**
* **ES（Extra Segment）**
* **指明附加段的起始地址**
* **（1）代码段**
* **代码段用来存放程序的指令序列**
* **代码段寄存器CS存放代码段的段地址**
* **指令指针寄存器IP指示下条指令的偏移地址**
* **处理器利用CS:IP取得下一条要执行的指令**
* **（2）堆栈段**
* **堆栈段确定堆栈所在的主存区域**
* **堆栈段寄存器SS存放堆栈段的段地址**
* **堆栈指针寄存器SP指示堆栈栈顶的偏移地址**
* **处理器利用SS:SP操作堆栈顶的数据**
* **（3）数据段**
* **数据段存放运行程序所用的数据**
* **数据段寄存器DS存放数据段的段地址**
* **各种主存寻址方式（有效地址EA）得到存储器中操作数的偏移地址**
* **处理器利用DS:EA存取数据段中的数据**
* **（4）附加段**
* **附加段是附加的数据段，也用于数据的保存：**
* **附加段寄存器ES存放附加段的段地址**
* **各种主存寻址方式（有效地址EA）得到存储器中操作数的偏移地址**
* **处理器利用ES:EA存取附加段中的数据**
* **串操作指令将附加段作为其目的操作数的存放区域**

如何分配各个逻辑段

* **程序的指令序列必须安排在代码段**
* **程序使用的堆栈一定在堆栈段**
* **程序中的数据默认是安排在数据段，也经常安排在附加段，尤其是串操作的目的区必须是附加段**
* **数据的存放比较灵活，实际上可以存放在任何一种逻辑段中**

段超越前缀指令

* **没有指明时，一般的数据访问在DS段；使用BP访问主存，则在SS段**
* **默认的情况允许改变，需要使用段超越前缀指令；8086指令系统中有4个：**
* **CS: ；代码段超越，使用代码段的数据**
* **SS: ；堆栈段超越，使用堆栈段的数据**
* **DS: ；数据段超越，使用数据段的数据**
* **ES: ；附加段超越，使用附加段的数据**
* **没有段超越的指令实例：**
* **MOV AX,[2000H] ；AX←DS:[2000H]**
* **；从默认的DS数据段取出数据**
* **采用段超越前缀的指令实例：**
* **MOV AX,ES:[2000H] ；AX←ES:[2000H]**
* **；从指定的ES附加段取出数据**

段寄存器的使用规定

|  |  |  |  |
| --- | --- | --- | --- |
| 访问存储器的方式 | 默认 | 可超越 | 偏移地址 |
| 取指令 | CS | 无 | IP |
| 堆栈操作 | SS | 无 | SP |
| 一般数据访问 | DS | CS ES SS | 有效地址EA |
| BP基址的寻址方式 | SS | CS ES DS | 有效地址EA |
| 串操作的源操作数 | DS | CS ES SS | SI |
| 串操作的目的操作数 | ES | 无 | DI |

指令指针IP

* **指令指针寄存器IP，指示代码段中指令的偏移地址**
* **它与代码段寄存器CS联用，确定下一条指令的物理地址**
* **计算机通过CS : IP寄存器来控制指令序列的执行流程**
* **IP寄存器是一个专用寄存器**

标志寄存器

**CF 进位标志（Carry Flag）：有进/借位（字节运算时时第7位，字运算为第15位）为1，无进借位为0。另循环指令也使CF=1。**

AF 辅助进位标志（Auxiliary Carry Flag）：低4位向高位有进/借位（即第3位向第4位进位）为1，否则AF=0。

PF 奇偶标志（Parity Flag）：运算结果若低8位（对于16位数也是检查低8位）所含1的个数为偶数，则PF=1，否则PF=0。

ZF 全零标志（Zero Flag）：当运算结果使有效位数的各位全为零时ZF=1，否则ZF=0。

SF 符号标志（Sign Flag）：当运算结果为负时SF=1，否则SF=0。SF的值就是有符号数的最高位(符号位)。

OF 溢出标志（Overflow Flag）：当运算结果超出了机器所能表示的范围时，则OF=1，表示溢出，否则OF=0。

状态标志

**DF 方向标志（Direction Flag）：如果DF为0，则串操作过程中地址会不断增值，反之，如果为1，则会不断减值。**

IF 中断允许标志（Interrupt Enable Flag）：如果为0，则CPU不能响应可屏蔽中断请求，反之，可接受可屏蔽中断请求。

TF 单步标志（Trap Flag）：如果为1，则CPU按跟踪方式执行命令。

控制标志

**5.OF溢出标志位**

**3AH + 7CH＝B6H，产生溢出：OF = 1**

**AAH + 7CH＝（1）26H，没有溢出：OF = 0**

* **“溢出”和“进位”有所不同**
* **溢出标志OF和进位标志CF是两个意义不同的标志**
* **进位标志表示无符号数运算结果是否超出范围，超出范围后加上进位或借位运算结果仍然正确；**
* **溢出标志表示有符号数运算结果是否超出范围，超出范围后运算结果不正确。**
* **“溢出”和“进位”的对比**

**例：3AH + 7CH＝B6H**

**无符号数运算： 58＋124＝182 范围内，无进位**

**有符号数运算： 58＋124＝182 范围外，有溢出**

* **溢出和进位的运用**
* **处理器对两个操作数进行运算时，按照无符号数求得结果，并相应设置进位标志CF；同时，根据是否超出有符号数的范围设置溢出标志OF**
* **应该利用哪个标志，则由程序员来决定。也就是说，如果将参加运算的操作数认为是无符号数，就应该关心进位；认为是有符号数，则要注意是否溢出**
* **“溢出”的判断**
* **判断运算结果是否溢出的简单规则：**
* **只有当两个相同符号数相加（减法运算转换为加法运算），而运算结果的符号与原数据符号相反时，产生溢出；因为，此时的运算结果显然不正确**
* **其他情况下（例如，两个不同符号数相加），则不会产生溢出**
* **例：设n=8 X1=-0101 0111B(-87),**
* **X2=-0011 0101B(-53)，求[X1+X2]补**
* **解：[X1+X2]补 = [X1]补+[X2]补  
   = 1010 1001B+1100 1011B  
   = 1 0111 0100B(116) > 0**
* **相加后，原本8位的二进制数变成了9位。由于一个字节只能**
* **是8位，多出的1位就被丢掉了，丢失了符号位使得结果成为**
* **一正数，这种情况称为溢出**
* **例：设n=8，X1=-0100 0010B(-66)，  
  X2=0110 1101B(109)，求[X1]补-[X2]补。  
  [X1]补-[X2]补 = [X1]补+[-X2]补  
  = 1011 1110B + 1001 0011B  
  = 1 0101 0001B (81)>0  
  在用补码作减运算时，负数减正数，一定为负数（结果），而结果为正说明产生了溢出**

§1.6 什么是汇编语言

* **计算机的程序设计语言**
* **什么是汇编语言**
* **汇编语言和高级语言的比较**
* **汇编语言的特点**
* **汇编语言的应用场合**

1.6.1 计算机的程序设计语言

* **机器语言（Machine Language）**
* **B8 64 00 05 00 01**
* **汇编语言（Assembly Language）**
* **mov ax,100**
* **;取得一个数据100（MOV是传送指令）**
* **add ax,256**
* **;实现100+256（ADD是加法指令）**
* **高级语言（High-level Language）**
* **100＋256**

1.6.2 什么是汇编语言

* **以助记符形式表示计算机指令**
* **助记符（mnemonic）是便于人们记忆、并能描述指令功能和指令操作数的符号**
* **助记符是表明指令功能的英语单词或其缩写**
* **汇编格式指令以及使用它们编写程序的规则就形成汇编语言（Assembly Language）**
* **汇编语言程序：用汇编语言书写的程序**
* **汇编程序：将汇编语言程序“汇编”成机器代码目标模块的程序**

1.6.3 汇编语言的特点

* **汇编语言的优点：**
* **直接控制计算机硬件部件**
* **编写“时间”和“空间”两方面最有效程序**
* **汇编语言的缺点：**
* **与处理器密切有关**
* **需要熟悉计算机硬件系统、考虑许多细节**
* **编写繁琐，调试、维护、交流和移植困难**

1.6.4 汇编语言和高级语言的比较

* **汇编语言与处理器密切相关**
* **汇编语言程序的通用性、可移植性较差**
* **高级语言与具体计算机无关**
* **高级语言程序是标准化语言，可在多种计算机上编译**
* **后执行**
* **汇编语言功能有限、涉及硬件细节**
* **程序编写比较繁琐，调试比较困难**
* **高级语言提供了强大的功能，不必关心琐碎问题**
* **类似自然语言的语法，易于掌握和应用**
* **汇编语言本质上就是机器语言**
* **可以直接、有效地控制计算机硬件**
* **易于产生速度快、容量小的高效率目标程序**
* **高级语言不针对具体计算机系统**
* **不易直接控制计算机的各种操作**
* **目标程序比较庞大、运行速度较慢**

**1.6.5 汇编语言混合编程**

* **汇编语言的优点使得它在程序设计中占有重要的位置，不可被取代**
* **汇编语言的缺点使得人们主要采用高级语言进行程序开发工作**
* **有时需要采用高级语言和汇编语言混合编程，互相取长补短，更好地解决实际问题**

1.6.6 汇编语言的应用场合

* **程序要具有较快的执行时间，或者只能占用较小的存储容量**
* 程序与计算机硬件密切相关，程序要直接、有效地控制硬件
* 大型软件需要提高性能、优化处理的部分
* 没有合适的高级语言、或只能采用汇编语言的时候
* 分析具体系统尤其是该系统的低层软件、加密解密软件、分析和防治计算机病毒等等

1.7 汇编语言相关的软件

* **系统软件：DOS平台**
* MS-DOS 6.22实地址方式
* Windows的MS-DOS模拟环境
* 应用软件：开发汇编语言程序涉及
* 文本编辑器
* 汇编程序
* 连接程序
* 调试程序
* 集成化开发环境

1.7.1文本编辑器（Editor）

* **文本编辑器用于编辑无任何格式的文档**
* 程序设计要采用文本编辑器编写源程序
* 常见的文本编辑软件有很多，如
* MS-DOS的EDIT全屏幕编辑器
* Windows的Notepad计事本
* 程序开发系统中的程序编辑器
* Turbo C
* Visual Studio
* MASM的PWB



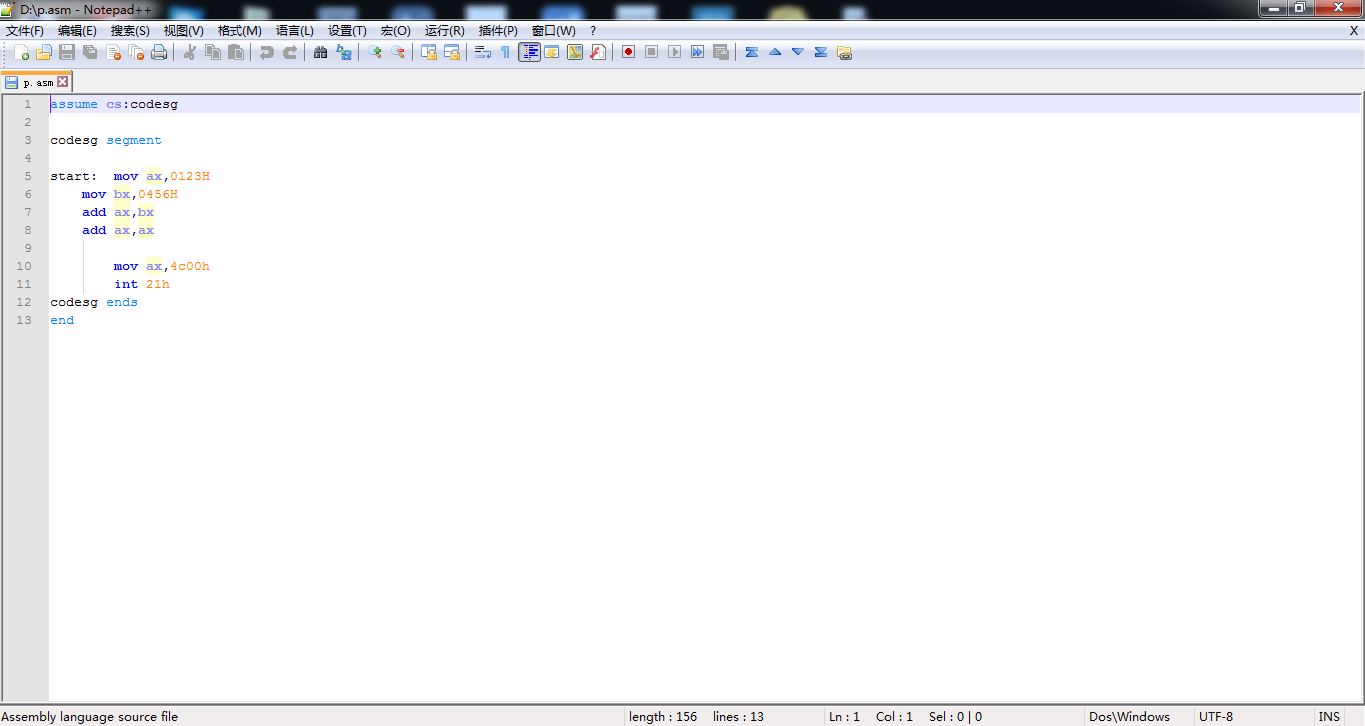
1.7.2 汇编程序（Assembler）

* **汇编程序将汇编语言源程序翻译（称为“汇编”）成机器代码目标模块**
* 微软的MASM 6.15
* MASM的最后一个独立版本MASM 6.11
* 可免费升级为MASM 6.14（支持SSE）
* Visual C++中有MASM 6.15（支持SSE2）
* Visual C++.NET 2003有MASM 7.10
* Visual C++.NET 2005的MASM支持Penium 4的SSE3指令系统，同时有ML64.EXE程序用于支持64位指令系统

幻灯片90

1.7.3 连接程序（Linker）

* **连接程序将汇编后的目标模块转换为可执行程序**
* 每个程序开发环境都有连接程序
* 连接程序的文件名通常是：LINK.EXE
* 32位Windows连接程序不同于16位DOS连接程序

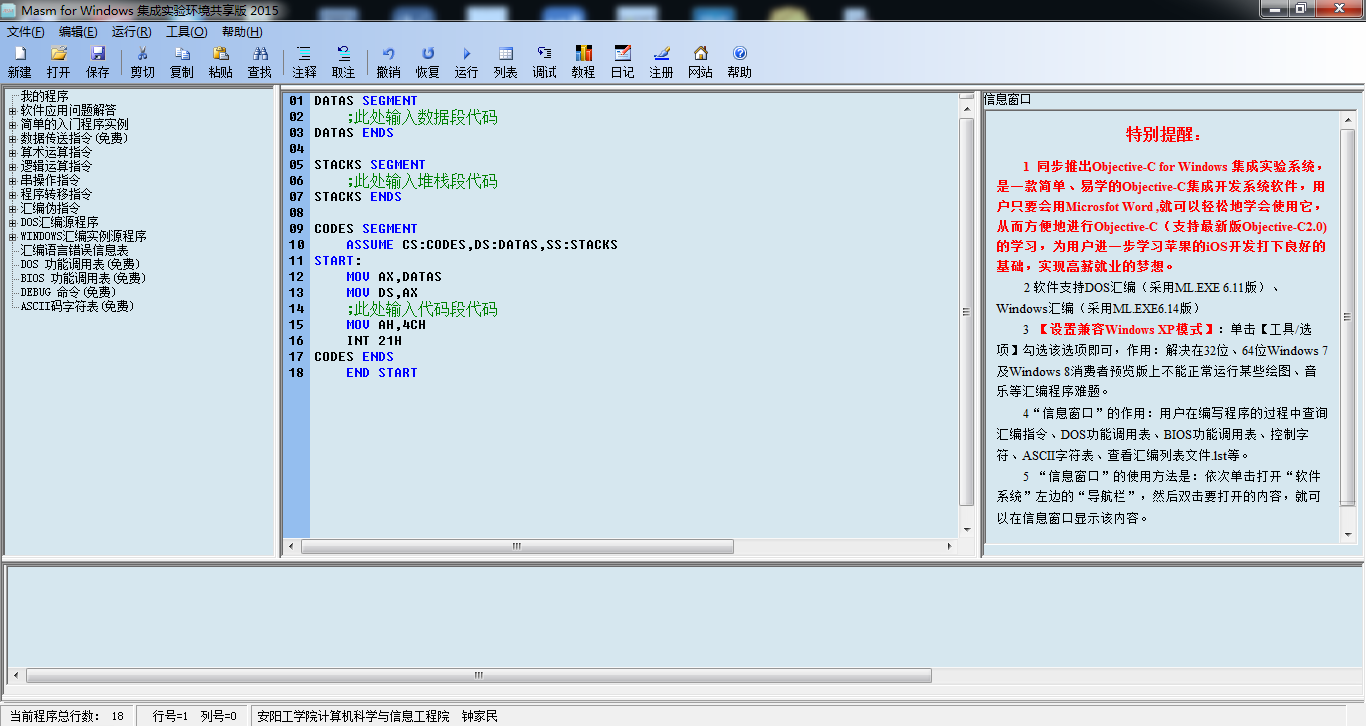
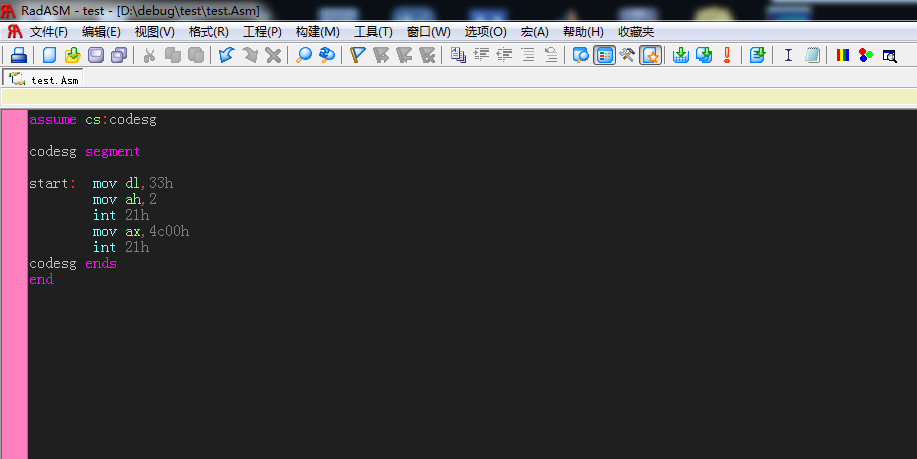


1.7.4 调试程序（Debugger）

* **调试程序进行程序排错、分析等**
* **MASM的CodeView**
* **DOS的DEBUG程序**
* **还有Turbo Debugger等**

1.7.5 集成化开发环境

* **进行程序设计使用的各种软件的有机集合，有文本编辑器，有语言翻译程序，有连接程序，还组合有调试程序等**
* **大型的程序设计项目往往要借助这种集成开发环境，也就是软件开发工具（包）**
* **MASM提供程序员工作平台PWB**
* **微软的Visual Studio开发系统**
* **国内学者开发的“MASM集成开发环境”**
* **国外著名工具：radasm**



常用命令总结

指令

1. mov (move) ;传送
2. Push(push onto the stack) ;进栈
3. Pop(pop from the stack) ;出栈
4. Xchg(exchange) ;交换
5. Lea(load effective address) ;有效地址送寄存器
6. Lds(load ds with pointer) ;指针送寄存器和ds
7. Les(load es with pointer) ;指针送寄存器和es
8. Lahf(load ah with flags) ;标志送ah
9. Sahf(store ah into flags) ;ah送标志寄存器
10. Pushf(push flags) ;标志入栈
11. Popf(pop flags) ;标志出栈
12. Cbw(convert byte to word) ;byte比特->word字
13. Cwd(convert word to double word) ;word字->double word双字
14. Add(add) ;加
15. Adc(add with carry) ;带进位加法
16. Inc(increment) ;加1
17. Sub(substract) ;减
18. Sbb(substract with borrow) ;带进位的减法
19. Dec(decrement) ;减1
20. Neg(negate) ;求补
21. Cmp(compare) ;比较
22. Mul(unsigned mulutiple);无符号乘法
23. Imul(signed mulutiple) ;带符号乘法
24. Div(unsigned divide) ;无符号除法
25. Idiv(signed divide) ;带符号除法
26. Daa(decimal adjust for addition)
27. Das(decimal adjust for subtraction)
28. Aaa(ASCII adjust for addition)
29. Aas(ASCII adjust for subtraction)
30. Aam(ASCII adjust for mulutiplication)
31. Aad(ASCII adjust for division)
32. And(and) ;与
33. Or(or) ;或
34. Not(not) ;非
35. Xor(xor) ;异或
36. Test(test) ;测试
37. Shl(shift logic left) ;逻辑左移
38. Sal(shift arithmetic left) ;算术左移
39. Shr(shift logic right) ;逻辑右移
40. Sar(shift arithmetic right) ;算术右移
41. Rol(rotat left) ;循环左移
42. Ror(rotat right) ;循环右移
43. Rcl(rotat left with carry) ;带进位循环左移
44. Ror(rotat right right wth carry) ;带进位循环右移
45. Movs(mov string)
46. Cmps(campare string)
47. Scans(scan string)
48. Stos(store)
49. Lods(load from string)
50. Rep(repeat)
51. Repe/repz(repeat while equal /zero)
52. Repne/repnz(repeat while not equal /not zero)
53. Cld (clear direction flag)
54. Std(set direction flag)
55. Cmpsb(byte)
56. Cmpsw(word)
57. Cmps
58. scansB(byte)
59. Scansw(word)
60. Scans
61. Lodsb(byte)
62. Lodsw(word)
63. Lods
64. Jmp
65. Loop(loop)
66. Loopz/loope(loop while zero ,or equal)
67. Loopnz/loopne(loop while not zero,or not equal)
68. Call(call)
69. Ret(return)
70. Int(interrupt)
71. Iret(return from interrupt)
72. Clc(clear carry)
73. Cmc(complement carry)
74. Stc(set carry)
75. Cld(clear direction)
76. Std(set direction)
77. Cli(clear interrupt)
78. Sti(set interrupt)
79. Nop(no operation)
80. Hlt(halt)
81. Wait(wait)
82. Lock(lock)
83. ：寻址方式

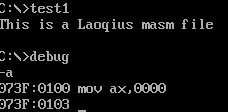
一．Debug的使用：

练习使用Debug调试程序

1. 汇编命令

A（Assemble）

功能：在相应的地址处存入所写的汇编语句汇编后的机器码



直接-a,地址默认为CS：0100

也可指定地址如 -a CS:69



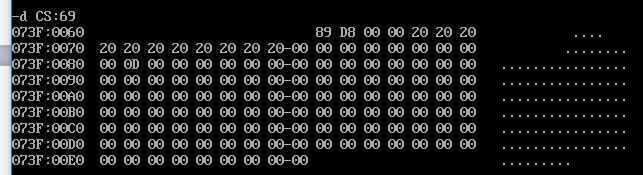
D(Dump)

功能：

默认显示DS段的内容

用法:如d69:250

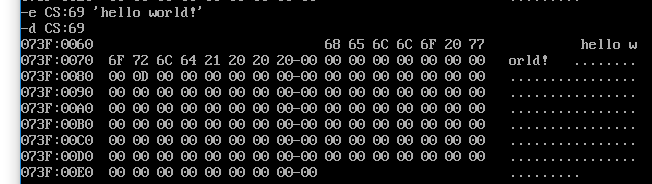
显示DS:69到DS:250的内容

如d CS:69显示CS段69H处的内容

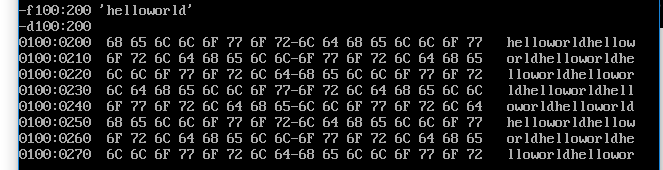
E(Enter)

功能修改存储单元内容

例：用e命令修改CS:69的内容为字符串hello world!并用d命令查看



F(Fill)

功能：填写内存单元

G（go）

功能：执行程序命令

默认从CS：IP开始执行

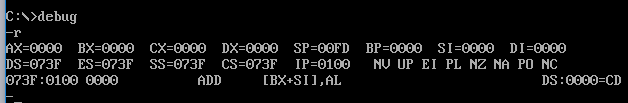
-g100：200

q(Quit)

退出debug

R(register)

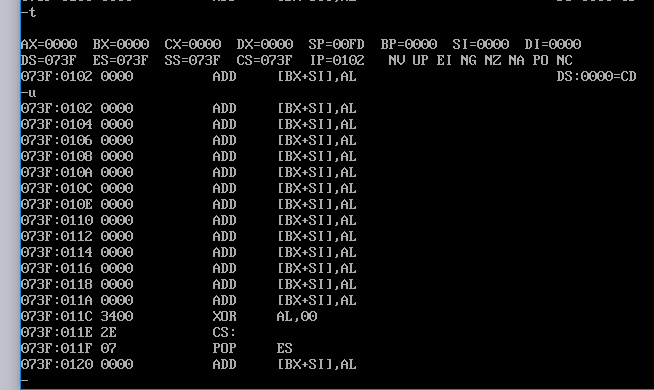
查看改变寄存器的内容



T(Trace)

逐条追踪

U(Unassemble)

反汇编

二．几种不同的寻址方式：

1.Immediate addressing :(立即寻址)

2.register addressing :(寄存器寻址)

3.direct addressing :(直接寻址)

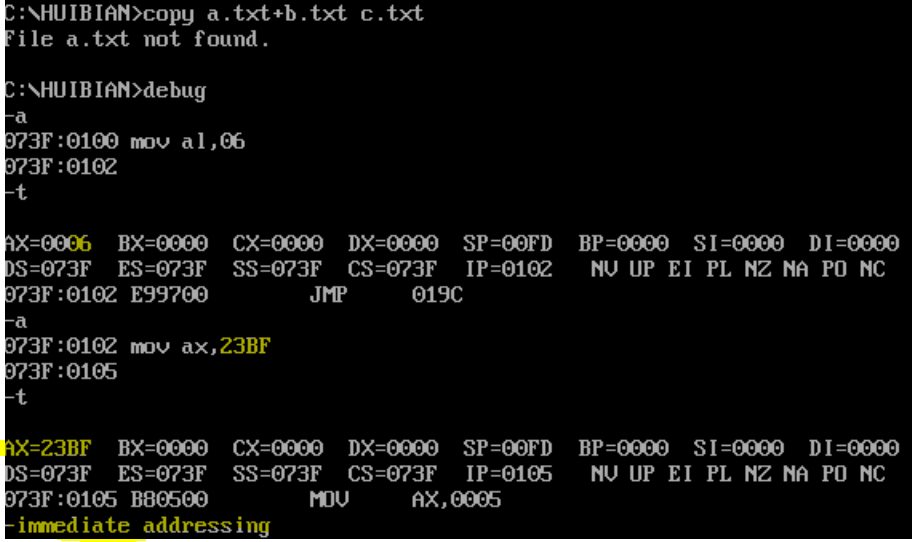
4.register indirect addressing :(寄存器间接寻址)

5.register relative addressing :(寄存器相对寻址)

6.relative based indexed addressing:(相对基址变址寻址)

详解：

1. Immediate addressing :(立即寻址)



操作数立即从指令中得到

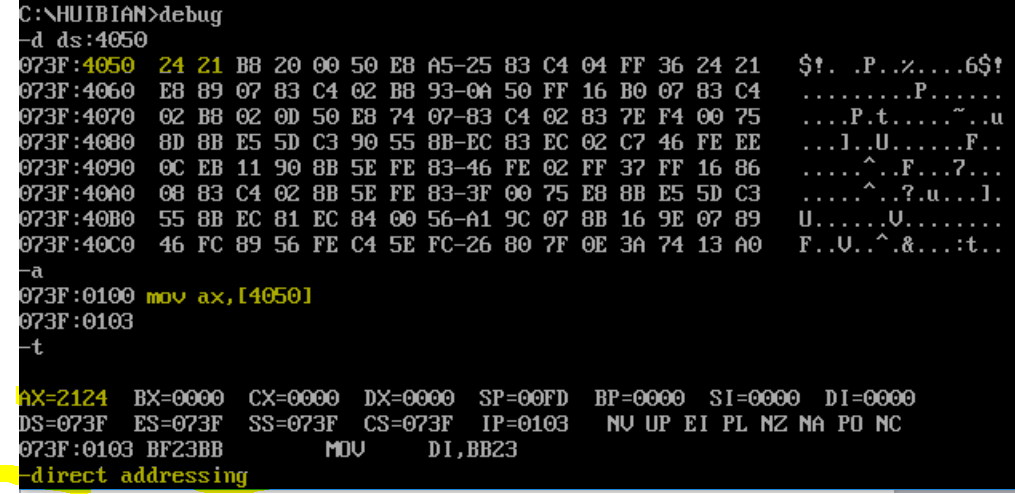
1. register addressing :(寄存器寻址)



两个操作数都为寄存器

1. direct addressing :(直接寻址)

地址直接在指令里给出（方括号里是数，默认DS作为段地址）



1. register indirect addressing :(寄存器间接寻址)

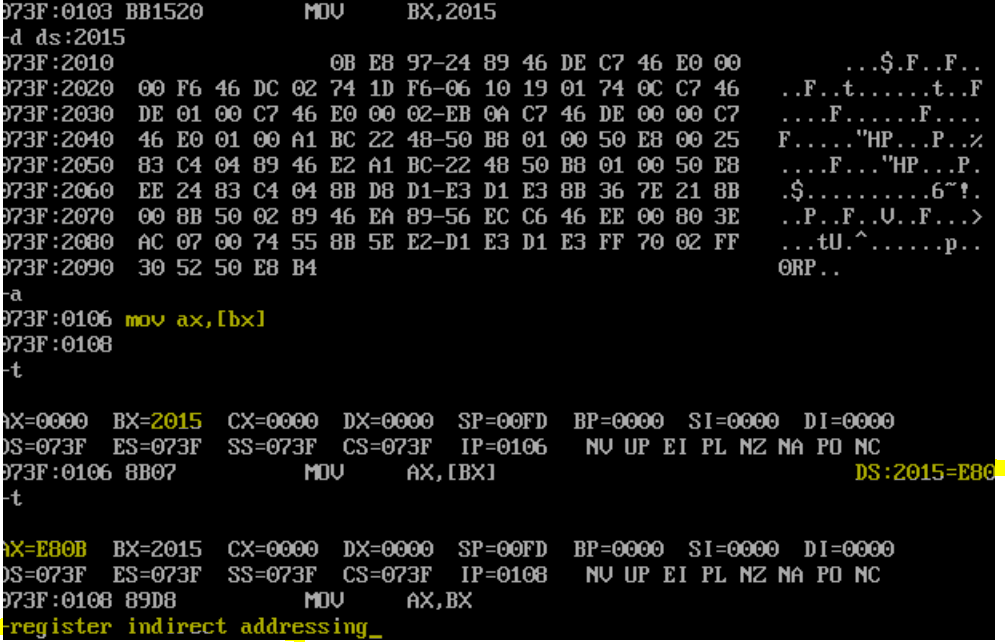
方括号里的东西不直接给你，而是存放在寄存器的里面

即【BX或BP或SI或DI】

Mov ax,[bx]

Moc dx,[bp]

Mov es:[di],ax



5.register relative addressing :(寄存器相对寻址)

6.relative based indexed addressing:(相对基址变址寻址)

段寄存器就是提供段地址的。

8086CPU有4个段寄存器：

CS、DS、SS、ES

1. ：指令系统

汇编语言由以下3类组成：

1、汇编指令（机器码的助记符）

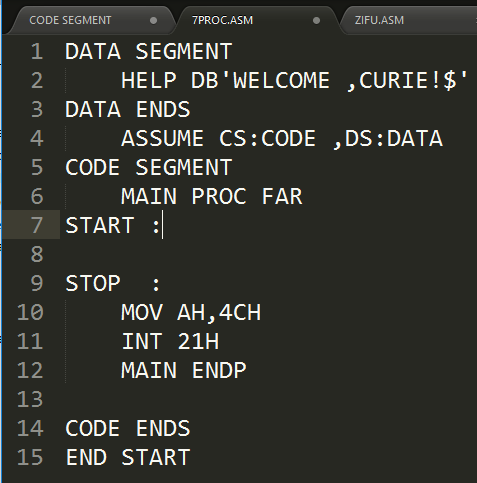
2、伪指令 （由编译器执行）

3、其它符号（由编译器识别）

汇编语言的核心是汇编指令，它决定了汇编语言的特性。

1. ：
2. ：

写汇编代码的一般格式：



自己写的一些具有参考价值的简单程序

**80X86汇编语言程序设计**

**邱日**

**[19215116@njau.edu.cn](mailto:19215116@njau.edu.cn)**

**[3103204417@qq.com](mailto:3103204417@qq.com)**

**20舍209**

**第2章   
寻址方式**

指令组成

**[标号]: 指令助记符 [目的操作数],[源操作数];[注释]**

**寻找操作数的过程就是操作数的寻址**

* **每种指令的操作码：**
* **用一个唯一的助记符表示（指令功能的英文缩写）**
* **对应着机器指令的一个二进制编码**
* **指令中的操作数：**
* **可以是一个具体的数值**
* **可以是存放数据的寄存器**
* **或指明数据在主存位置的存储器地址**
* **操作数可能的来源或存放处：**
* **由指令直接给出**
* **寄存器**
* **内存单元**
* **寻找操作数所在地址的方法可以有三种大类型**
* **指令直接给出的方式**
* **存放于寄存器中的寻址方式**
* **存放于存储器中的寻址方式**

指令中的操作数

**立即数**

**寄存器**

**存储器**

表征参加操作的数据本身

表征数据存放的地址

立即数操作数

* 立即数本身是参加操作的数据，可以是8位或16位，只能作为源操作数。
* 例： MOV AX，1234H
* MOV BL，22H
* 立即数无法作为目标操作数
* 立即数可以是10进制或者2进制的数,但指令运行后CPU自动把该数转换为16进制数,然后完成给定功能的操作，其中，二进制在数字后面加B区分。
* MOV AL,57 ；指令执行后将AL 39H
* MOV AL,01010111B ；指令执行后将AL 57H
* 立即数可以是无符号或带符号数，其数值应在可取值范围内。

寄存器操作数

* **参加运算的数存放在指令给出的寄存器中，可以是16位或8位。**
* **例：**
* **MOV AX，BX**
* **MOV DL，CH**

存储器操作数

* **参加运算的数存放在存储器的某一个或某两个单元中。**
* **表现形式：[ ]**

**立即数或寄存器**

**[ ]中的内容是存放所寻找数据的单元的偏移地址**

幻灯片10

* **例：MOV AX，[1200H]**
* **MOV AL，[1200H]**

2.1 立即寻址

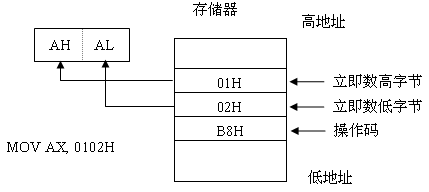
**立即寻址仅适合于源操作数**

* **指令中的源操作数是立即数，即源操作数是参加操作的数据本身**
* **例：MOV AX，1200H**

**MOV AX,0102H ；AX←0102H**

**2.2 寄存器寻址**

* **参加操作的操作数在CPU的通用寄存器中。**
* AH、AL、BH、BL、CH、CL、DH、DL
* AX、BX、CX、DX、SI、DI、BP、SP
* CS、DS、SS、ES
* 例：MOV AX，BX



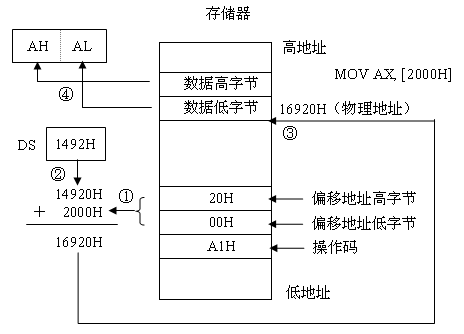
2.3 存储器寻址方式

* **指令中给出操作数的主存地址信息（偏移地址，称之为有效地址EA），而段地址在默认的或用段超越前缀指定的段寄存器中**
* **8086设计了多种存储器寻址方式**
* **1、直接寻址方式**
* **2、寄存器间接寻址方式**
* **3、寄存器相对寻址方式**
* **4、基址变址寻址方式**
* **5、相对基址变址寻址方式**

**2.3.1 直接寻址**

* **指令中直接给出操作数的偏移地址**
* **例：MOV AX，[1200H]**
* **直接寻址方式下，操作数的段地址默认为数据段，但允许段重设，即由指令定义段。**
* **例：MOV AX，ES：[1200H]**

**MOV AX,[2000H] ；AX←DS:[2000H]**



**2.3.2 寄存器间接寻址**

* **参与操作的操作数存放在内存中，其偏移地址为指令中的寄存器的内容。**

**例：MOV AX，[BX]**

**设BX=1200H**

**代码段**

* **由寄存器间接给出操作数的偏移地址；**
* **存放偏移地址的寄存器称为间址寄存器，它们是：BX，BP，SI，DI**
* **操作数的段地址（数据处于哪个段）取决于选择哪一个间址寄存器：**
* **BX，SI，DI**
* **BP**

**基址寻址：**

**称该间址寄存器为基址寄存器BX，BP**

**变址寻址：**

**称该间址寄存器为变址寄存器SI，DI**

**2.3.3 寄存器间接相对寻址**

* **操作数的偏移地址为寄存器的内容加上一个位移量**
* **例：**
* **MOV AX，[BX+DATA]**
* **设：DS=2000H，BX=0220H，DATA=05H**
* **则：AX=[20225H]**

2.3.4 基址-变址寻址

* **操作数的偏移地址为**
* **一个基址寄存器的内容 + 一个变址寄存器的内容；**
* **操作数的段地址由选择的基址寄存器决定**
* **基址寄存器为BX，默认在数据段**
* **基址寄存器为BP，默认在堆栈段**
* **基址变址寻址方式与相对寻址方式一样，主要用于一维数组操作。**

**例：执行下列指令：**

**MOV SI，1100H**

**MOV BX，SI**

**MOV AX，[SI+BX]**

**例：判断以下指令是否正确：**

**MOV AX，[BX][BP]; 错误，不能同时出现两个基址寄存器**

**MOV AX，[SI][DI]; 错误，不能同时出现两个变址寄存器**

**3.3.6 基址-变址-相对寻址**

**2.3.5 基址-变址-相对寻址**

* **操作数的偏移地址为：**
* **基址寄存器内容+变址寄存器内容+位移量**
* **或：BX/BP（基址寄存器）＋SI/DI（变址寄存器）＋位移量**
* **操作数的段地址由选择的基址寄存器（BX对应DS，BP对应SS）决定。**
* **基址变址相对寻址方式主要用于二维表格操作。**

不允许将两个基址寄存器或两个变址寄存器组合在一起寻址，即指令中不允许同时出现两个基址寄存器或两个变址寄存器

**例：执行以下程序段：**

**MOV DI，1100H**

**MOV BP，DI**

**MOV AL，[BP][DI]5**

**例： 判断以下指令是否正确：**

**MOV AX，DATA[SI][DI]；非法**

**MOV AX，[BX][BP]DATA；非法**

**2.4 隐含寻址**

* **指令中隐含了一个或两个操作数的地址，即操作数在默认的地址中。**
* **例：**
* **MUL BL**
* **指令执行：**
* **AL×BL**

寄存器操作数的表达

* **r8——任意一个8位通用寄存器**
* **AH AL BH BL CH CL DH DL**
* **r16——任意一个16位通用寄存器**
* **AX BX CX DX SI DI BP SP**
* **reg——代表r8或r16**
* **seg——段寄存器**
* **CS DS ES SS**

存储器操作数的表达

* **m8——一个8位存储器操作数单元（所有主存寻址方式）**
* **m16——一个16位存储器操作数单元（所有主存寻址方式）**
* **mem——代表m8或m16**

立即数的表达

* **i8——一个8位立即数**
* **i16——一个16位立即数**
* **imm——代表i8或i16**
* **dest——目的操作数**
* **src——源操作数**

**80X86汇编语言程序设计**

**邱日**

**[19215116@njau.edu.cn](mailto:19215116@njau.edu.cn)**

**[3103204417@qq.com](mailto:3103204417@qq.com)**

**20舍209**

**Di 第三章 指令系统**

3.1 指令与指令系统

3.2 IA-32指令系统

3.3 本章小结

**3.1 指令与指令系统**

* **指令**
* **控制计算机完成某种操作的命令**
* **指令系统**
* **处理器所能识别的所有指令的集合**
* **指令的兼容性：**
* **同一系列机的指令都是兼容的。**
* **16位8086指令系统是Intel 80x86系列微处理器指令系统的基础**

**零操作数指令： 操作码**

**单操作数指令： 操作码 操作数**

**双操作数指令： 操作码 操作数，操作数**

**多操作数指令： 三操作数及以上**

**指令**

**长度**

**指令的执行速度**

* **指令的字长影响指令的执行速度**
* **对不同的操作数，指令执行的时间不同：**

**立即数**

**存储器**

**寄存器**

**3.2 IA-32指令系统**

**数据传送**

**算术运算**

**逻辑运算和移位**

**串操作**

**程序控制**

**处理器控制**

**按功能划分为六大类：**

**3.2.1 数据传送类指令**

* **通用数据传送**
* **输入输出**
* **地址传送**
* **标志位操作**

**3.2.1.1 通用数据传送类指令**

**一般数据传送指令**

**堆栈操作指令**

**交换指令**

**查表转换指令**

**字位扩展指令**

* **特点：**
* **该类指令的执行对标志位不产生影响**

**（1）一般数据传送指令**

* **一般数据传送指令： MOV**
* **格式：**
* **MOV dest，src**
* **操作：**
* **src**
* **例：**
* **MOV AL，BL**

**dest**

* **注意点：**
* **两操作数字长必须相同；**
* **两操作数不允许同时为存储器操作数；两操作数不允许同时为段寄存器；在源操作数是立即数时，目标操作数不能是段寄存器；**
* **IP和CS不作为目标操作数，FLAGS一般也不作为操作数在指令中出现。**

MOV CS,3000H ；不正确

MOV AX,3000H ;

MOV CS,AX ；不正确

MOV SS,1000H ；不正确

MOV DI,1000H

MOV SS,DI ；正确

* **例1**
* **判断下列指令的正确性：**
* **MOV AL，BX；错误，操作数类型不同**
* **MOV AX，[SI]05H；正确**
* **MOV [BX][BP]，BX；错误，不能将两个基址寄存器组合一起寻址**
* **MOV DS，1000H；错误，立即数不能给段寄存器赋值**
* **MOV [1200]，[SI]；错误，不能用1条指令完成两个存储单元之间的数据传送**
* **例2例**
* **将立即数2AH送入内存数据段1000H开始的100个单元中。**
* **题目分析：**
* **确定首地址**
* **确定数据长度**
* **写一次数据**
* **修改单元地址**
* **修改长度值**
* **判断写完否？**
* **未完继续写入，否则结束**
* ┇
* ┇

**程序段：**

**MOV DI，1000H**

**MOV CX，64H**

**MOV AL，2AH**

**AGAIN：MOV [DI]，AL**

**INC DI ；DI+1**

**DEC CX ；CX-1**

**JNZ AGAIN ；CX≠0则继续**

**HLT**

幻灯片15

**上段程序在代码段中的存放形式**

* **设CS=109EH，IP=0100H，则各条指令在代码段中的存放地址如下：**
* **CS : IP 机器指令 汇编指令**
* **109E：0100 B80010 MOV DI，1000H**
* **109E：0103 . MOV CX，64H**
* **109E：0105 . MOV AL，2AH**
* **109E：0107 . MOV [DI]，AL**
* **109E：0109 INC DI**
* **109E：010A DEC CX**
* **109E：010B JNZ 0107H**
* **109E：010D HLT**
* **数据段中的分布**

**送上2AH后数据段中相应存储单元的内容改变如下：**

**DS：1000 2A 2A 2A 2A 2A 2A 2A 2A-2A 2A 2A 2A 2A 2A 2A 2A**

**DS：1010 2A 2A 2A 2A 2A 2A 2A 2A-2A 2A 2A 2A 2A 2A 2A 2A**

**DS：1020 2A 2A 2A 2A 2A 2A 2A 2A-2A 2A 2A 2A 2A 2A 2A 2A**

**DS：1030 2A 2A 2A 2A 2A 2A 2A 2A-2A 2A 2A 2A 2A 2A 2A 2A**

**DS：1040 2A 2A 2A 2A 2A 2A 2A 2A-2A 2A 2A 2A 2A 2A 2A 2A**

**DS：1050 2A 2A 2A 2A 2A 2A 2A 2A-2A 2A 2A 2A 2A 2A 2A 2A**

**DS：1060 2A 2A 2A 2A 00 00 00 00 00 00 00 00 00 00 00 00**

**偏移地址[DI]**

**（2）堆栈操作指令**

* **压栈指令 PUSH**
* **格式: PUSH OPRD**
* **出栈指令 POP**
* **格式: POP OPRD**

OPRD只能是16位寄存器或存储器操作数（两单元）

* **压栈指令 PUSH**
* **指令执行过程：**
* **SP - 2 → SP**
* **操作数高字节 → SP+1**
* **操作数低字节 → SP**

**堆栈段**

**SP**

**低8位**

**高8位**

**SP**

* **压栈指令的操作**

**设AX=1234H，SP=1200H**

**执行 PUSH AX 指令后堆栈区的状态：**

**入栈前**

**入栈后**

* **出栈指令POP**
* **指令执行过程：**
* **SP**
* **SP+1**
* **SP ← SP+2**

**操作数低字节**

**操作数高字节**

* **出栈指令的操作**

**执行 POP AX**

**出栈前**

**出栈后**

* **堆栈操作指令说明**
* 指令的操作数必须是16位的；
* 操作数可以是寄存器或存储器两单元，但不能是立即数；
* 不能从栈顶弹出一个字给CS（允许PUSH CS）；
* PUSH和POP指令在程序中一般成对出现；
* PUSH指令的操作方向是从高地址向低地址，而POP指令的操作正好相反。
* **例：**

**MOV AX，9000H**

**MOV SS，AX**

**MOV SP，E200H**

**MOV DX，38FFH**

**PUSH DX**

**PUSH AX**

**POP DX**

**POP AX**

**如此，会使DX和AX的内容互换**

**（3）交换指令**

* **格式：**
* **XCHG REG，MEM/REG**
* **注：**
* **两操作数必须有一个是寄存器操作数**
* **不允许使用段寄存器**
* **两个操作数字长要一致。**
* **例：**
* **XCHG AX，BX**
* **XCHG [2000]，CL**

**（4）查表指令**

* **格式：**
* **XLAT；（AL） （（BX）+（AL））**
* **或：XLAT src\_table ；**
* **（src\_table表示要查找的表的首地址）**
* **说明：**
* **用BX的内容代表表格首地址，AL内容为表内位移量，BX+AL得到要查找元素的偏移地址**
* **表格长度最大不能超过256个字节**
* **操作：**
* **将BX+AL所指单元的内容送AL**
* **例：**

**数据段中存放有一**

**张ASCII码转换表，**

**设首地址为2000H，**

**现欲查出表中第11**

**个代码的ASCII码**

* **可用如下指令实现：**
* **MOV BX，2000H ；BX←表首地址**
* **MOV AL，0BH ；AL←序号**
* **XLAT ；查表转换**
* **执行后：AL = 42H**
* **还可用其他方法实现，如：**
* **MOV BX，2000H**
* **MOV AL，[BX+0BH]**

**（5）字位扩展指令**

* **将有符号数的符号位扩展到高位；**
* **指令为零操作数指令，采用隐含寻址，隐含的操作数为AX及AX，DX**
* **无符号数的扩展规则为在高位补0**
* **字节到字的扩展指令**
* **格式：**
* **CBW**
* **操作：**
* **将AL内容扩展到AX**
* **规则：**
* **若最高位=1，则执行后AH=FFH**
* **若最高位=0，则执行后AH=00H**
* **字到双字的扩展指令**
* **格式：**
* **CWD**
* **操作：**
* **将AX内容扩展到DX AX，高位存放在DX中，隐含了操作数AX。**
* **规则：**
* **若最高位=1，则执行后DX=FFFFH**
* **若最高位=0，则执行后DX=0000H**

**例：**

* **判断以下指令执行结果：**
* **MOV AL，44H**
* **CBW**
* **MOV AX，0AFDEH**
* **CWD**
* **MOV AL，86H**
* **CBW**

**3.4.1.2 输入输出指令**

* **掌握：**
* **指令的格式及操作**
* **指令的两种寻址方式**
* **指令对操作数的要求**
* **专门面向I/O端口操作的指令**
* **指令格式：**
* **输入指令： IN acc，PORT**
* **输出指令 ：OUT PORT，acc**

**端口地址**

* **根据端口地址码的长度，指令具有两种不同的端口地址表现形式。**
* **直接地址**
* **端口地址为8位时，指令中直接给出8位端口地址（立即数方式）；**
* **寻址256个端口。**
* **间接地址**
* **端口地址为16位时，指令中的端口地址必须由DX指定；**
* **可寻址64K个端口。**
* **例：**

**IN AX，80H**

**MOV DX，2400H**

**IN AL，DX**

**OUT 35H ，AX**

**OUT AX，35H**

**3.4.1.3 地址传送指令**

**取偏移地址指令LEA**

**\*LDS指令**

**\*LES指令**

**（1）取偏移地址指令LEA**

* **操作：**
* **将变量的16位偏移地址取出送目标寄存器**
* **当程序中用符号地址表示内存偏移地址时，须使用该指令。**
* **格式：**
* **LEA REG，MEM**
* **指令要求：**
* **源操作数MEM必须是一个存储器操作数，目标操作数必须是16位通用寄存器，一般是间址寄存器（BX，BP，SI，DI）。**
* **例1：**
* **设（BX）=1000H，（DS）=6000H，（61050H）=33H，（61051H）=44H。比较以下两条指令的执行结果。**
* **LEA BX，[BX+50H]**
* **MOV BX，[BX+1]**
* **则，第1条指令执行后，（BX）=？，第2条指令执行后，（BX）=？。**
* **例2：**
* **将数据段中首地址为MEM1 的50个字节的数据传送到同一逻辑段首地址为MEM2的区域存放。编写相应的程序段 。**

**LEA SI，MEM1**

**LEA DI，MEM2**

**MOV CL，50**

**NEXT： MOV AL，[SI]**

**MOV [DI]，AL**

**INC SI**

**INC DI**

**DEC CL**

**JNZ NEXT**

**HLT**

**（2）LDS指令**

* **LDS reg16，mem32；**
* **（reg16） ((mem32)+1:(mem32))**
* **(DS) ((mem32)+3:(mem32)+2)**
* **其中，源操作数mem32位存储器操作数，给出的是内存中4个连续的存储单元的逻辑地址。目标操作数reg16必须是BX、BP、SI、DI等4个间址寄存器之一。**
* **该指令用于把存储器mem32中存放的一个32位远地址指针（包括偏移地址）送到reg16和DS。4个存储单元的前两个单元的内容作为偏移地址送reg16，后两个单元的内容作为段地址装入段寄存器DS**
* **例1：**
* **设（DS）=6000H，内存地址为60348H开始的4个单元中存放了一个32位的远指针98011H（该地址存放的内容是3412H），以下指令将该指针装入DS：SI中。**
* **LDS SI，[0348H]**
* **MOV AX，[SI]**
* **指令执行后：（SI）=8011H，（DS）=9000H，（AX）=3412H**
* **（3）LES指令**
* **这条指令的格式及功能与LDS指令非常类似，不同的是，两个高地址单元中给出的段地址不是送往DS，而是送到ES。**
* **例如：将上例中的LDS指令改为LES指令，则指令执行后：**
* **（SI）=8011H，（ES）=9000H，而DS内容没有改变**

**3.4.1.4 标志位操作指令**

**LAHF**

**SAHF**

**PUSHF**

**POPF**

**（1）LAHF，SAHF**

* **LAHF**
* **操作：**
* **将FLAGS的低8位的SF，ZF，AF，PF，和CF装入AH对应位置，8位中其余3位为无效位**

**执行与LAHF相反的操作**

幻灯片46

**（2）PUSHF，POPF**

* **针对FLAGS的堆栈操作指令**
* **将标志寄存器压栈或从堆栈弹出**
* **PUSHF:(指令本身不影响标志位)**
* **[SP-1] （FLAGSH）**
* **[SP-2] （FLAGSL）**
* **（SP） （SP）-2**
* **POPF:(指令影响标志位)**
* **（FLAGSL） [SP]**
* **（FLAGSH） [SP+1]**
* **（SP） （SP）+2**

**3.4.2 算术运算类指令**

* **加法运算指令**
* **减法运算指令**
* **乘法指令**
* **除法指令**

**算术运算指令的执行大多对状态标志位会产生影响**

**3.4.2.1 加法指令**

**普通加法指令ADD**

**带进位的加法指令ADC**

**加1指令INC**

**加法指令对操作数的要求与MOV指令相同**

**（1）ADD指令**

* **格式：**
* **ADD OPRD1，OPRD2**
* **操作：**
* **OPRD1+OPRD2**

**ADD指令的执行对全部6个状态标志位都产生影响**

注意：源操作数和目标操作数不能同时为存储器操作数；

不能对段寄存器进行运算。

幻灯片50

* **例： MOV AL，78H**
* **ADD AL，99H**
* **指令执行后6个状态标志位的状态**

幻灯片51

**（2）ADC指令**

* **指令格式、对操作数的要求、对标志位的影响与ADD指令完全一样**
* 指令的操作：
* OPRD1+OPRD2+CF OPRD1
* ADC指令多用于多字节数相加，使用前要先将CF清零。



**例：求两个20B数的和**

**LEA SI，M1**

**LEA DI，M2**

**MOV CX，20**

**CLC ；使CF=0**

**NEXT ： MOV AL，[SI]**

**ADC [DI]，AL**

**INC SI**

**INC DI**

**DEC CX**

**JNZ NEXT**

**HLT**

**（3）INC指令**

**不能是段寄存器或立即数**

**格式：**

**INC OPRD**

**操作：**

**OPRD+1 OPRD**

**常用于在程序中修改地址指针**

INC指令不影响CF标志位，但对其它5个状态标志AF、OF、PF、SF及ZF会产生影响。

幻灯片54

**3.4.2.2 减法指令**

**普通减法指令SUB**

**考虑借位的减法指令SBB**

**减1指令DEC**

**比较指令CMP**

**求补指令NEG**

**减法指令对操作数的要求与对应的加法指令相同**

**（1）SUB指令**

* **格式：**
* **SUB OPRD1，OPRD2**
* **操作：**
* **OPRD1- OPRD2 OPRD1**
* **对操作数的要求以及对标志位的影响与ADD指令相同**

**（2）SBB指令**

* **指令格式、对操作数的要求、对标志位的影响与SUB指令完全一样。**
* **主要用于多字节减法运算。**
* **指令的操作：**
* **OPRD1- OPRD2- CF OPRD1**

**（3）DEC指令**

* **格式：**
* **DEC OPRD**
* **操作：**
* **OPRD - 1 OPRD**
* **指令对操作数的要求与INC相同（不能是段寄存器或立即数）**
* **指令常用于在程序中修改计数值**
* **例：**

**MOV BL，2**

**NEXT1： MOV CX，0FFFFH**

**NEXT2： DEC CX**

**JNZ NEXT2 ; ZF=0转NEXT2**

**DEC BL**

**JNZ NEXT1 ; ZF=0转NEXT1**

**HLT ; 暂停执行**

**（4）NEG指令（求补指令）**

**8/16位寄存器或存储器操作**

* **格式：**
* **NEG OPRD**
* **操作：**
* **0 - OPRD OPRD**

**用0减去操作数，相当于对该操作数求补码**

NEG指令对6个状态标志位均有影响，但要注意以下两点：

①执行NEG指令后，一般情况都会使CF为1，除非给定操作数为0，CF才会为0。

②当指定的操作数为80H（-128）或为8000H（-32768），则执行NEG指令后，结果不变，但OF置1，其它情况下OF均值0

**（5）CMP指令**

* **格式：**
* **CMP OPRD1，OPRD2**
* **操作：**
* **OPRD1- OPRD2**
* 用途：用于比较两个数的大小，可作为条件转移指令转移的条件
* 指令执行的结果不影响目标操作数，仅影响标志位！
* 指令对操作数的要求及对标志位的影响与SUB指令相同
* 两个无符号数的比较
* CMP AX，BX
* 若 AX > BX CF=0
* 若 AX < BX CF=1

根据ZF状态判断。如果ZF=1，则两个操作数相等，否则不相等。

不等关系

相等关系

* 两个带符号数的比较
* CMP AX，BX
* 两个数的大小由OF和SF共同决定
* OF和SF状态相同 AX > BX
* OF和SF状态不同 AX < BX

**LEA BX，MAX**

**LEA SI，BUF**

**MOV CL，20**

**MOV AL，[SI]**

**NEXT：INC SI**

**CMP AL，[SI]**

**JNC GOON ；CF=0转移**

**XCHG [SI]，AL**

**GOON：DEC CL**

**JNZ NEXT**

**MOV [BX]，AL**

**HLT**

**3.4.2.3 乘法指令**

**无符号的乘法指令MUL**

**带符号的乘法指令IMUL**

* **注意点：**
* **乘法指令采用隐含寻址，隐含的是存放被乘数的累加器AL或AX及存放结果的AX，DX.**

**（1）无符号数乘法指令**

* **格式：**
* **MUL OPRD**
* **操作：**
* **OPRD为字节数**
* **OPRD为16位数**
* **注意：两操作数字长必须相等，且不能为立即数**

**（2）带符号数乘法指令**

* **格式：**
* **IMUL OPRD**
* **操作：**
* **OPRD为字节数**
* **OPRD为16位数**

DX:高16位，AX:低16位

幻灯片67

* **说明：**
* **IMUL指令在格式上和功能上都与MUL指令类似，只是有如下区别：**
* **要求两乘数都须为有符号数**
* **若乘积的高半部分是低半部分的符号位的扩展，则CF=OF=0，否则CF=OF=1**
* **指令中给出的源操作数应满足带符号数的表示范围**
* **【例】**
* **设（AL）=FEH，（CL）=11H，求AL与CL的乘积。**
* **若将两个寄存器中内容看做无符号数，则应使用指令：**
* **MUL CL**
* **指令执行后（AX）=10DEH，因AH中的结果不为零，故CF=OF=1**
* **若将两操作数看做有符号数，则二者相乘应采用有符号数的乘法指令，即：**
* **IMUL CL**
* **指令执行后（AX）=FFDEH=-34，则AH中内容为AL中的符号扩展，故CF=OF=0.**

**3.4.2.4 除法指令**

* 无符号除法指令：DIV
* 有符号除法指令：IDIV

16位/8位 → 8位商

32位/16位→16位商

* 进行除法时：
* 对被除数、商及余数存放有如下规定：
* 被除数 商 余数
* 字节除法 AX AL AH
* 字除法 DX:AX AX DX

除法指令规定被除数的字长必须为除数字长的2倍。如果被除数字长不够，就要使用字位扩展指令来扩展其位数。

无符号除法指令和有符号除法指令对6个标志位均无影响。

除数为16位或8位寄存器或者内存单元

幻灯片70

**（1）无符号数除法指令**

* **格式**
* **DIV OPRD；OPRD的长度确定除法类型**
* **操作：字节操作 (AL)  (AX)/(OPRD)的商**
* **(AH)  (AX)/(OPRD)的余数**
* **字操作 (AX)  (DX, AX)/(OPRD)的商**
* **(DX)  (DX, AX)/(OPRD)的余数**
* **注：**
* **若除数为零或AL中商大于FFH,(或AX中商大于FFFFH)，则CPU产生一个类型0的内部中断。**

**【1】**

**DIV BL ；（AX）除以（BL），商=（AL），余数=（AH）**

**DIV WORD PTR[SI]；**

**；（DX）:(AX)除以SI和SI+1所指向单元的内容，**

**；商=（AX），余数=（DX）**

**【2】**

**MOV AX，7FA2H ；（AX）=7FA2H**

**MOV BX，03DDH ；（BX）=03DDH**

**CWD ；（DX）:(AX)=00007FA2H**

**DIV BX ；商=（AX）=0021H，**

**；余数=（DX）=0025H**

**（2）有符号数除法指令**

这条指令除要求操作数为有符号数外，在格式和功能上都和DIV指令类似。

例：

IDIV CX ；DX和AX中的32位数除以（CX），

；商=（AX），余数=（DX）

IDIV BYTE PTR[BX]；（AX）除以BX所指单元中的内容，

；商=（AL），余数=（AH）

IDIV指令的结果，商和余数均为带符号数，且余数符号与被除数符号相同。如-26除以+4，可得到两种结果，一是商=-6，余数=-2；另一种是商=-7，余数=+2。两种结果都正确，但是，按照8086指令系统规定，会取第一种结果。

**3.4.2.5 BCD码调整指令**

* **BCD码**
* **BCD码：用四个二进制位表示一个十进制数字；最常用的是8421 BCD码。**
* **压缩BCD码：每一个10进制位用4位二进制表示，一个字节可以表示2个10进制数。**
* **如：56的压缩型8421 BCD码是01010110；**
* **非压缩BCD码：每一个10进制位用1个字节二进制表示。其中，高4位为0，低4位存放对应的二进制数。**
* **如：5的非压缩型BCD码是0000 0101，56的非压缩型BCD码是00000101 00000110。**

1. 真值 8 64
2. 二进制编码 08H 40H
3. 压缩BCD码 08H 64H
4. 非压缩BCD码 08H 0604H

* BCD码运算调整指令
* 加法10进制调整指令：DAA,AAA
* 减法10进制调整指令：DAS,AAS
* 乘法10进制调整指令：AAM
* 除法10进制调整指令：AAD
* 注：
* BCD调整指令不能单独使用，必须跟在算术运算指令之后（除法例外）。
* 均为隐含寻址方式，隐含的操作数是AL或AL、AH。

BCD码本质上是十进制数，即应遵循逢十进一的规则。而计算机是按二进制（十六进制）进行运算，并未按十进制规则进行运算。

* **（1）压缩BCD码加法十进制调整指令DAA**
* **DAA用于对两个压缩BCD码相加之后的和（结果必须存入AL）进行调整，产生正确的压缩BCD码。**
* **调整方法：**
* **若（AL）中低4位>9或AF=1，则（AL）+06H →（AL），并使AF=1**
* **若（AL）中高4位>9或CF=1，则（AL）+60H →（AL），并使CF=1**
* 例1：编程用BCD数计算48+27=?
* MOV AL，48H
* ADD AL，27H
* DAA
* 则ADD结果为：01101111。而BCD数48H+27H应该等于75H，但ADD运算结果为6FH，结果不正确，应该采用DAA指令进行调整。
* 因为低4位（1111）>9,故进行加6调整：01101111+00000110=01110101
* 可以看出，调整后：（AL）=75H，AF=1，CF=0，结果正确。
* （2）非压缩BCD码加法十进制调整指令AAA
* AAA指令用于对两个非压缩BCD数相加之和（结果必须存入AL）进行调整，形成一个正确的非压缩BCD码，调整后的结果低位存入AL，高位存入AH。
* 具体步骤为：
* 若（AL）中低4位>9或AF=1，则（AL）+6,（AH）+1，AF=1；
* 屏蔽掉（AL）高4位，即（AL）←（AL）∩0FH；
* CF←AF
* **例1：**
* **用非压缩BCD码计算9+4=？**
* **MOV AL，09H ；BCD数9**
* **MOV BL，04H ；BCD数4**
* **ADD AL，BL ；（AL）=09H+04H=0DH**
* **AAA ；（AL）=0DH+06H=03H（高4位清零）**
* **；（AF）=1；（CF）=1**
* **例2：**
* **计算8＋9 0000 1000**
* **+0000 1001**
* **0001 0001 ＝ 11**
* **结果应为17，而计算机相加为11，原因在于运算过程中，如**
* **遇到低4位往高4位产生进位时（此时AF=1）是按逢十六进一**
* **的规则，但BCD码要求逢十进一，因此只要产生进位，个位就**
* **会少6，这就要进行加6调正。**

**这个1代表了16，而实际上仅应为10。**

**实际上当低4位的结果＞9(即A～F之间)时，也应进行加6调整 (原因是逢十没有进位，故用加6的方法强行产生进位) 。**

**如对上例的结果进行加6：**

**0001 0001 11**

**+ 0000 0110 6**

**0001 0111 17**

* （3）压缩BCD码减法十进制调整指令DAS
* DAS用于对两个压缩BCD码相减后的结果（存入AL）进行调整，产生正确的压缩BCD码。对标志位的影响与DAA指令相同。调整方法如下：
* 若（AL）中低4位>9或AF=1，则（AL）-06H，AF=1
* 若（AL）中高4位>9或CF=1，则（AL）-60H，CF=1
* （3）非压缩BCD码减法十进制调整AAS
* AAS用于对两个非压缩BCD码相减后的结果（存入AL）进行调整，产生正确的非压缩BCD码，其低位存入AL，高位存入AH。对标志位的影响与AAA指令相同。调整方法如下：
* 若（AL）中低4位>9或AF=1，则（AL）-06H，（AH）-1，AF=1
* 屏蔽掉（AL）高4位，即（AL）←（AL）∩0FH；
* CF←AF

（4）乘法的十进制调整指令AAM

AAM是非压缩BCD码乘法的十进制调整指令。对两个非压缩BCD码数相乘的结果（存入AX）进行调整，以得到正确的结果。具体步骤：

（AH）←（AL）/0AH

（AL）←（AL）%0AH

即把AL寄存器的内容除以OAH，商存入AH，余数存入AL。

* **说明：**
* **AAM的实质是把AL中的二进制数转换为十进制数，所以对于不超过99的二进制数，只用一条AAM指令即可实现二-十进制转换**
* **AAM指令影响PF、SF和ZF标志位**
* **执行AAM指令前须有一条MUL指令（BCD码总视为无符号数）将两个非压缩BCD码相乘，结果放入AL，然后用AAM指令进行调整。**

**例：**

* **用非压缩BCD码计算7\*9=？**
* **MOV AL，07H ；（AL）=07H，即非压缩BCD数7**
* **MOV BL，09H ；（AL）=09H，即非压缩BCD数9**
* **MUL BL ；（AX）=07H\*09H=0603H**
* **AAM ；（AX）=0603H，即非压缩BCD数63，SF=0，**
* **； ZF=0，PF=1**

（5）除法的十进制调整指令AAD

在两个非压缩BCD码相除之前，先用一条AAD指令进行调整，然后再用DIV指令。具体步骤：

（AL）←（AH）\*A+（AL）

（AH）←0

即把AX中的非压缩BCD码（十位数放AH，个位数放AL）调整为二进制数，并将结果放入AL中。AAD的操作实质上是把AX中的两位十进制数转换为二进制数，所以对于不超过99的十进制数，只用一条AAD指令即可实现十-二进制转换。

AAD指令影响PF、SF和ZF标志位。

* **例：**
* **计算23/4=？**
* **MOV AX，0203H ；（AX）=0203H，即非压缩BCD数23**
* **MOV BL，4 ；（BL）=04H，即非压缩BCD数4**
* **AAD ；（AX）=02H\*0AH+03H=0017H**
* **DIV BL ；（AH）=03H，（AL）=05H，即商5余3**
* **执行完AAD后，（AH）=0，（AL）=17；再执行DIV指令后，（**
* **AH）=03H，（AL）=05H**

**3.4.3 逻辑运算和移位指令**

* 逻辑运算
* and，or，not，xor，test
* 移位操作
* 非循环移位，循环移位
* 说明
* 逻辑运算指令对操作数的 要求大多与MOV指令相同。
* 除“非”运算指令 外，其余逻辑运算指令的执行都会使标志位OF=CF=0,AF值不定，并对SF，PF和ZF有影响。

**3.4.3.1 逻辑运算指令**

**（1）“与”指令AND：**

* **格式：**
* **AND OPRD1，OPRD2**
* **操作：**
* **两操作数相“与”，结果送目标地址。**
* 说明：
* 源操作数OPRD2可以是寄存器、存储器或立即数，但目标操作数OPRD1只能是寄存器或存储器
* **“与”指令的应用**
* **实现两操作数按位相与的运算**
* **AND BL，[SI]**
* **使目标操作数的某些位不变，某些位清零**
* **AND AL，0FH**
* **在操作数不变的情况下使CF和OF清零**
* **AND AX，AX**
* **例**
* **从地址为3F8H端口中读入一个字节数，如果该数bit1位为1，则可从38FH端口将DATA为首地址的一个字输出，否则就不能进行数据传送。编写相应的程序段。**

**LEA SI，DATA**

**MOV DX，3F8H**

**WATT：IN AL，DX**

**AND AL，02H**

**JZ WATT ；ZF=1转移**

**MOV DX，38FH**

**MOV AX，[SI]**

**OUT DX，AX**

**（2）“或”运算指令OR**

* **格式：**
* **OR OPRD1，OPRD2**
* **操作：**
* **两操作数相“或”，结果送目标地址**
* 说明：
* 对操作数的要求以及对标志位的影响和AND指令一样

**“或”指令的应用**

* **实现两操作数相“或”的运算**
* **OR AX，[DI]**
* **使某些位不变，某些位置“1”**
* **OR CL，0FH**
* **在不改变操作数的情况下使OF=CF=0**
* **OR AX，AX**

**【例】奇偶校验**

**OR AL，AL**

**JPE GOON**

**OR AL，80H**

**GOON：….**

**PF=1转移**

**（3）“非”运算指令**

* 格式：
* NOT OPRD
* 操作：
* 操作数按位取反再送回原地址
* 操作数为8位或16位的寄存器或者存储器操作数，但不能是立即数
* 注：
* 指令中的操作数不能是立即数
* 指令的执行对标志位无影响
* 例：NOT BYTE PTR[BX]

**（4）“异或”运算指令**

* 格式：
* XOR OPRD1，OPRD2
* 操作：
* 两操作数相“异或”，结果送目标地址
* 例：
* XOR BL，80H
* XOR AX，AX

某一操作数和自身相“异或”，结果为0，在程序中常利用这一特性，使某寄存器清零

幻灯片99

**（5）“测试”指令TEST**

TEST OPRD1，OPRD2

* 操作：
* 执行“与”运算，但运算的结果不送回目标地址，只是影响标志位。
* 应用：
* 常用于测试某些位的状态
* 例：
* TEST AL，02H；若AL中D1位为1，则ZF=0，否则ZF=1
* TEST AX，8000H；若AX中最高位为1，则ZF=0，否则ZF=1
* TEST AX，AX；该条语句，影响ZF标志位,可判断AX是否为0
* 如果AX的内容为空，那么执行该语句后ZF标志位为1
* 如果AX的内容不为空，那么执行该语句后ZF标志位为0

**3.4.3.2 移位指令**

**非循环移位指令**

**循环移位指令**

**（1）非循环移位指令**

* **逻辑左移SHL**

1. **算术左移SAL**

* **逻辑右移SHR**

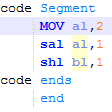
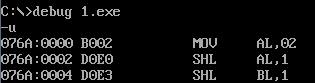
**算术右移SAR**

* 左移可实现乘法运算

右移可实现除法运算

* **算术左移和逻辑左移**
* **算术左移指 令：**
* **SAL OPRD，1**
* **SAL OPRD，CL**
* **逻辑左移指 令：**
* **SHL OPRD，1**
* **SHL OPRD，CL**
* 在移动次数为1的情况下，若移位之后，操作数的最高位与CF标志位的值不相等，则溢出标志位OF=1，否则OF=0.此外，指令还影响标志位PF、SF和ZF。
* SHL指令和SAL指令的区别是：逻辑左移指令SHL将操作数视为无符号数，而算术左移指令SAL将操作数视为有符号数。OF=1对SHL指令不表示左移后溢出，而对SAL指令表示移位后超出了符号数的表示范围。

注意：SAL指令不能在Debug中测试，但可以在MASM中编译，因为汇编中将SAL自动转换为SHL执行的，这点可以通过对MASM生成的EXE文件反编译进行验证。



参考资料：浅谈8086汇编语言中SAL指令的应用研究，冯文超

【例】

MOV AL，41H

SHL AL，1

执行结果：（AL）=82H，CF=0，OF=1.若视82H位无符号数，则它没有溢出（82H<FFH）；若视为有符号数，则溢出了（82H>7FH），因为移位后正数变成了负数。

幻灯片107

* **逻辑右移SHR**
* **格式：**
* **SHR OPRD，1**
* **SHR OPRD，CL**

**无符号数的右移**

* 它将目标操作数视为无符号数，其操作是将目的操作数顺序向右移1位或CL指定的位数，每右移1位，右边的最低位移入标志位CF，而在左边的最高位补0。
* SHR指令也影响标志位CF和OF。如果移动次数为1，且移位之后新的最高位和次高位不相等，则标志位OF=1，否则OF=0。若移位次数不为1，则OF状态不定。
* **例1：**

**MOV AL，68H**

**MOV CL，2**

**SHR AL，CL**

**CF**

**AL**

**【例】将1000H开始存放的4个压缩BCD码转换为ASCII码存放在3000H开始的单元中去。**

**MOV SI,1000H**

**MOV DI,3000H**

**MOV CX,4**

**Next：MOV AL,[SI]**

**MOV BL,AL**

**AND AL,0FH**

**OR AL,30H**

**MOV [DI],AL**

**INC DI**

**MOV AL,BL**

**PUSH CX**

**MOV CL,4**

**SHR AL,CL**

**OR AL,30H**

**MOV [DI],AL**

**INC DI**

**INC SI**

**POP CX**

**DEC CX**

**JNZ Next**

**HLT**

* **算术右移**

**有符号数的右移**

* **格式：**
* **SAR OPRD，1**
* **SAR OPRD，CL**
* SAR指令将目标操作数视为有符号数，格式与SHR相同。指令的操作是将目标操作数顺序向右移1位或CL指定的位数，操作数最低位移入标志位CF。
* 与SHR指令的区别是：算术右移时最高位不是补零，而是保持不变。
* 对标志位CF、OF、PF、SF和ZF有影响，但使AF值不定。

SAR和SHR的区别

汇编语言中SAR和SHR指令都是右移指令，SAR是算数右移指令（shift arithmetic right），而SHR是逻辑右移指令（shift logical right）。

两者的区别在于SAR右移时保留操作数的符号，即用符号位来补足，而SHR右移时总是用0来补足。

例：10000000算数右移1位是11000000，而逻辑右移1位是01000000。

* **不带进位的循环移位**
* **带进位的循环移位**

**左移 ROL**

**右移 ROR**

**左移 RCL**

**右移 RCR**

**指令格式、对操作数的要求与非循环移位指令相同**

左移 ROL

右移 ROR

**不带进位的循环移位**

1. 把目的地址中的数据循环左移CL次，每次从最高位（最左）移出的数据位都补充到最低位（最右），最后从最高位（最左）移出的数据位保存到CF标志位。
2. 把目的地址中的数据循右移CL次，每次从最低位（最右）移出的数据位都补充到最高位（最左），最后从最低位（最右）移出的数据位保存到CF标志位。

ROL和ROR指令影响CF和OF标志位

幻灯片115

左移 RCL

右移 RCR

**带进位的循环移位**

RCL和RCR指令影响CF和OF标志位

幻灯片116

**3.4.4 串操作指令**

* 针对数据块或字符串的操作；
* 可实现存储器到存储器的数据传送；
* 待操作的数据串称为源串，目标地址称为目标串。
* 源串一般存放在DS段，偏移地址由SI指定，允许段重设；
* 目标串必须在ES段，偏移地址由DI指定；指令自动修改地址指针，修改方向由DF决定。
* DF=0
* DF=1
* 数据块长度值由CX指定
* 可增加自动重复前缀以实现自动修改CX内容。

增地址方向

减地址方向

* **重复前缀**
* **无条件重复**

**REP**

**条件重复**

* **REPE 相等重复**
* **REPZ 为零重复**
* **REPNE 不相等重复**
* **REPNZ 不为零重复**
* **串操作指令流程(以传送操作为例)**

**设串长度**

**修改串长度值**

* **串操作指令**
* **串传送 MOVS**
* **串比较 CMPS**
* **串扫描 SCAS**
* **串装入 LODS**
* **串送存 STOS**

**（1）串传送指令**

* **格式：**
* **MOVS OPRD1，OPRD2 ；多用在需要段超越的场合**
* **MOVSB ；隐含了操作数地址，源串和目标串地址取默认值**
* **MOVSW ；与MOVSB类似，差别是该指令一次传送一个字**
* **串传送指令常与无条件重复前缀连用**

串传送指令允许进行内存单元到内存单元的数据传送，解决了MOV指令不能直接在内存单元之间传送数据的限制。

**【例】设变量ADDR1和ADDR2为字类型时，下面两条指令是等效的。  
MOVS ADDR1, ADDR2**

**MOVSW**

**上述第一条指令中的操作数仅指出了指令为字操作类型，实际目标操作数与源操作数的逻辑地址由ES:DI和DS:SI指出**

**其中第二，三种格式明确地注明是传送字节或字，第一种格式则应在操作数中表明是字还是字节操作，例如：**

**MOVS ES：BYTE PTR[DI]，DS：[SI]**

**是字节操作，例如：**

**MOVS ES：BYTE PTR[DI]，DS：[SI]**

**【例】对比用MOV指令和MOVS指令实现将200个字节数据从内存的一个区域送到另一个区域的程序段。**

**LEA SI，MEM1**

**LEA DI，MEM2**

**MOV CX，200**

**CLD**

**REP MOVSB**

**HLT**

**（2）串比较指令**

* 格式：
* CMPS OPRD1，OPRD2
* CMPSB
* CMPSW
* 串比较指令常与条件重复前缀连用，指令的执行不改变操作数，仅影响标志位。
* 前缀的操作对标志位不影响

将源串地址中的字节（或字）与目标串地址中的字节（或字）相比较，但比较（相减）结果不送回目标串地址中，而是反映在标志位上。

幻灯片124

【例】测试200个字节数据是否传送正确：

*LEA SI，MEM1*

*LEA DI，MEM2*

*MOV CX，200*

*CLD*

*REPE CMPSB*

*TEST CX，00FFH*

JZ STOP

……

STOP：HLT

幻灯片125

**（3）串扫描指令**

* **格式：**
* **SCAS OPRD**
* **SCASB**
* **SCASW**
* **执行与CMPS指令相似的操作，只是这里的源操作数是AX或AL。**
* **常用于在指定存储区域中寻找某个关键字。**

**【例】 要求从段首元素的偏移地址为0100H的一个字符串（字符个数为256）中找出指定的字符（如＄）。可用REPNZ SCASB指令实现。程序如下：**

**CLD 　　　　 ; DF＝0，地址自动递增**

**MOV 　DI, 0100H 　; 目标串首元素的偏移地址**

**MOV 　CX, 256 　　; 设置计数器初始值**

**MOV 　AL, ‘＄’　; 设关键字**

**REPNZ SCASB 　　　; 找关键字，若未找到则重复查找**

幻灯片127

**（4）串装入指令**

* **格式：**
* **LODS OPRD**
* **LODSB**
* **LODSW**
* **操作：**
* **对字节：**
* **对 字：**

串装入指令不影响标志位，且一般不带重复前缀

* 用于将内存某个区域的数据串依次装入累加器，以便显示或输出到接口。
* LODS指令一般不加重复前缀。

**（5）串存储指令**

* **格式：**
* **STOS OPRD**
* **STOSB**
* **STOSW**
* **操作：**
* **对字节： AL**
* **对 字： AX**
* 常用于将内存某个区域置同样的值
* 将待送存的数据放入AL（字节数）或AX（字数据）
* 确定操作方向（增地址/减地址）和区域大小（串长度值）
* 使用串存储指令+无条件重复前缀，实现数据传送

**【例】 将字符$送入附加段中偏移地址为0100H的连续五个单元中。**

**CLD ; DF＝0，地址自动递增**

**MOV CX, 5 ; 设置计数器**

**MOV DI, 0100H ; 目标串首元素的偏移地址**

**MOV AL, ‘$’ ; 设置关键字**

**REP STOSB ; 连续将‘$’写入相应存储单元中**

* 例
* 将内存某单元清零。设计思想如下：

**3.4.5 程序控制指令**

* **转移指令**
* **循环控制**
* **过程调用**
* **中断控制**

**程序控制类指令与程序执行方向**

* **程序控制类指令的本质是：控制程序的执行方向**
* **决定程序执行方向的因素：CS，IP**
* **控制程序执行方向的方法：**
* **修改CS和IP，则程序转向另一个代码段执行；**
* **仅修改IP，则程序将改变当前的执行顺序，转向本代码段内其它某处执行。**

**3.4.5.1 转移指令**

**无条件转移指令**

**无条件转移到目标地址，执行新的指令**

**有条件转移指令**

**在具备一定条件的情况下转移到目标地址**

**无条件转移指令**

**原则上可实现在整个内存空间的转移**

幻灯片135

**（1）段内相对转移**

* **转移的目标地址由指令直接给出**
* 格式：
* JMP Label



地址标号

**下一条要执行指令的偏移地址=当前IP+位移量**

【例】

**……**

MOV AX，BX

JMP SHORT NEXT

AND CL，0FH

……

NEXT：OR CL，7FH

**（2）段内间接转移**

* **段内间接转移**
* 转移的目标地址存放在某个16位寄存器或存储器
* 的某两个单元中
* 【例】
* JMP BX
* 若：BX=1200H
* 则：转移的目标地址=1200H



**【例】**

**JMP WORD PTR[BX]**

**设：BX=1200H**

**BX=1200**

**数据段**

**【例】**

**2000:0100 MOV AX,1200H**

**2000:0103 JMP NEXT**

**┅**

**2000:0120 NEXT: MOV BX,1200H**

**JMP BX**

**┅**

**2000:1200**

**（3）段间直接转移**

* **段内直接转移**
* **转移的目标地址由指令直接给出。指令操作码后的连续两个字作为立即地址，低字作为偏移地址送入IP，高字作为段地址送入CS。**
* **格式：**
* **JMP FAR Label**

1. **Label必须用一个32位数表达，叫做32位远指针，它就是逻辑地址。**

**【例】JMP 8000H：1200H**

幻灯片142

**（4）段间间接转移**

**转移的目标地址由指令中的32位操作数给出**

**【例】JMP DWORD PTR[BX]**

**代码段2**

**指令码**

**指令执行前：（DS）=3000H，（BX）=3000H，（33000H）=0BH，（33001H）=20H，（33002H）=10H，（33003H）=80H**

**则指令执行后，（IP）=200BH，（CS）=8010H。**

**转移后的目标地址=8210BH**

**【例】**

* **MOV SI，1122H**
* **MOV WORD PTR[SI]，0120H**
* **ADD SI，2**
* **MOV WORD PTR[SI]，0122H**
* **JMP DWORD PTR[SI-2]**
* **JMP WORD PTR[SI]**

幻灯片144

**条件转移指令**

* **在满足一定条件下，程序转移到目标地址继续执行**
* **条件转移指令均为段内短转移，即转移范围为：**
* **-128------+127**

幻灯片145

**四种转移条件**

**（1）以单个状态标志作为转移条件**

**（2）以CX的值为0作为转移条件**

**（3）以两个无符号数比较的结果作为转移条件**

**（4）以两个有符号数比较的结果作为转移条件**

**test al,80h ；测试最高位**

**jz next0 ；D7＝0（ZF＝1），转移**

**mov ah,0ffh ；D7＝1，顺序执行**

**jmp done ；无条件转向**

**next0: mov ah,0**

**done: ...**

**test al,80h ；测试最高位**

**jnz next1 ；D7＝1（ZF＝0），转移**

**mov ah,0 ；D7＝0，顺序执行**

**jmp done ；无条件转向**

**next1: mov ah,0ffh**

**done: ...**

**【例】：JS/JNS指令**

**；计算|X－Y|（绝对值）**

**；X和Y为存放于X单元和Y单元的16位操作数**

**；结果存入result**

**mov ax,X**

**sub ax,Y**

**jns nonneg**

**neg ax ；neg是求补指令**

**nonneg:mov result,ax**

**【例】：JO/JNO指令**

**计算X－Y；**

**；X和Y为存放于X单元和Y单元的16位操作数**

**；若溢出，则转移到overflow处理**

**mov ax,X**

**sub ax,Y**

**jo overflow**

**... ；无溢出，结果正确**

**overflow: ... ；有溢出处理**

**【例】JC/JNC指令**

**记录BX中1的个数**

**xor al,al；AL＝0，CF＝0**

**again: test bx,0ffffh ；等价于**

**；cmp bx,0**

**je next**

**shl bx,1**

**jnc again**

**inc al**

**jmp again**

**next: ...；AL保存1的个数**

**xor al,al；AL＝0，CF＝0**

**again: cmp bx,0**

**jz next**

**shl bx,1；也可使用**

**；shr bx,1**

**adc al,0**

**jmp again**

**next: ...；AL保存1的个数**

**3.4.5.2 循环控制指令**

* **循环范围：**
* **以当前IP为中心的-128～+127范围内循环。**
* **循环次数由CX寄存器指定。**
* **循环指令：**

**无条件循环指令**

**LOOP**

**\*LOOPZ**

**\*LOOPNZ**

**条件循环指令**

**（1）无条件循环指令**

* **格式：**
* **LOOP LABEL**
* **循环条件：**
* **CX ≠ 0**
* **操作：**
* **DEC CX**
* **JNZ 符号地址**

**（2）条件循环指令**

* **格式：**
* **LOOPZ LABEL**
* **LOOPE LABEL**
* **循环条件：**
* **CX ≠ 0 ZF=1**
* **格式：**
* **LOOPNZ LABEL**
* **LOOPNE LABEL**
* **循环条件：**
* **CX ≠ 0 ZF=0**

***功能：CX－1→CX，若CX≠0且ZF=1则转移至目标，否则停止循环(CX=0或ZF=0)。***

不影响标志位

**LOOPxx　　目标地址**

* LOOP退出循环条件是（CX）= 0
* LOOPZ和LOOPNZ提供了提前结束循环的可能，不一定要等到(CX)= 0才退出循环：
* 在串中查找字符，不相等时继续查找，使用指令LOOPNZ。
* 比较两串时，当有字符不等就可退出，使用指令LOOPZ。

**mov cx,count ；设置循环次数**

**mov si,offset string**

**xor bx,bx ；bx＝0，记录空格数**

**jcxz done ；如果长度为0，退出**

**mov al,20h**

**again: cmp al,es:[si]**

**jnz next ；ZF＝0非空格，转移**

**inc bx ；ZF=1是空格，个数加1**

**next: inc si**

**loop again**

**；字符个数减1，不为0继续循环**

**done: hlt**

**dec cx**

**jnz again**

3.4.5.3 过程调用和返回

* **子过程（程序）是完成特定功能的一段程序**
* **当主程序（调用程序）需要执行这个功能时，采用CALL调用指令转移到该子程序的起始处执行**
* **当运行完子程序功能后，采用RET返回指令回到主程序继续执行**
* **调用指令的执行过程**
* **保护断点；**
* **将调用指令的下一条指令的地址（断点）压入堆栈。**
* **获取子过程的入口地址：子过程第1条指令的偏移地址**
* **执行子过程，含相应参数的保存及恢复；**
* **恢复断点，返回原程序。将断点偏移地址由堆栈弹出。**
* **CALL指令需要保存返回地址：**
* **段内调用——入栈偏移地址IP**
* **SP←SP－2，SS:[SP]←IP**
* **段间调用——入栈偏移地址IP和段地址CS**
* **SP←SP－2，SS:[SP]←IP**
* **SP←SP－2，SS:[SP]←CS**

**CALL OPRD**

**段内直接调用**

**段内间接调用**

* **段内调用**
* **段间调用**

**过程调用**

**段间直接调用**

**段间间接调用**

**（1）段内调用**

* **被调用程序与调用程序在同一代码段**
* **调用前只需保护断点的偏移地址**
* **格式：**
* **CALL NEAR PROC**
* **执行过程：**

**被调用程序**

**调用程序**

* **将断点的偏移地址压入堆栈**
* **根据过程名找子程序入口**

**【例】**

**（1）CALL NEAR PROC**

**（2）CALL WORD PTR[SI]**

**（2）段间调用**

* **子过程与原调用程序不在同一代码段**
* **断点保护时的压栈顺序：先将断点的CS压栈，再压入IP。**

调用前需保护断点的段基地址和偏移地址

**【例】**

* **格式：**
* **CALL FAR PROC**
* **格式例：**
* **CALL 3000H：2100H （直接调用）**
* **CALL DWORD PTR[SI]（间接调用）**

**（3）返回指令**

* **根据段内和段间、有无参数，分成4种类型**
* **RET ；无参数段内返回**
* **RET i16 ；有参数段内返回**
* **RET ；无参数段间返回**
* **RET i16 ；有参数段间返回**
* **需要弹出CALL指令压入堆栈的返回地址**
* **段内返回——出栈偏移地址IP**
* **IP←SS:[SP]， SP←SP＋2**
* **段间返回——出栈偏移地址IP和段地址CS**
* **IP←SS:[SP]，SP←SP＋2**
* **CS←SS:[SP]，SP←SP＋2**

**【例】子程序**

**；主程序**

**mov al,0fh ；提供参数AL**

**call htoasc ；调用子程序**

**...**

**；子程序：将AL低4位的一位16进制数转换成ASCII码**

**htoasc: and al,0fh ；只取al的低4位**

**or al,30h ；al高4位变成3**

**cmp al,39h ；是0～9，还是0Ah～0Fh**

**jbe htoend**

**add al,7 ；是0Ah～0Fh，加上7**

**htoend: ret ；子程序返回**

3.4.5.4 中断指令

* **中断**
* **中断源**
* **中断的类型**
* **中断指令**
* **中断与过程调用**
* **中断是随机事件或异常事件引起，调用则是事先已在程序中安排好；**
* **响应中断请求不仅要保护断点地址，还要保护FLAGS内容；**
* **调用指令在指令中直接给出子程序入口地址，中断指令只给出中断向量码，入口地址则在向量码指向的内存单元中。**

**（1）中断指令**

* **格式：**
* **INT n**
* **说明：**
* **nх4**
* **中断指令的执行过程**
* **将FLAGS压入堆栈；**
* **将INT指令的下一条指令的CS、IP压栈；**
* **由n×4得到存放中断向量的地址；**
* **将中断向量（中断服务程序入口地址）送CS和IP寄存器；**
* **转入中断服务程序。**

**中断指令的执行过程**

* **例**

**执行程序段：**

**CS IP**

**┇**

**6200H:0110H INT 21H**

**6200H:0112H MOV AX，BX**

**┇**

**（2）溢出中断指令**

* **格式：**
* **INTO**
* **若OF=1,则启动一个类型为4的中断过程,给出一个出错标志,如果OF=0,不做任何操作。**
* **INTO指令通常安排在有符号数加减运算指令之后。**

**相当于**

**INT 4**

**（3）中断返回指令**

* **格式：**
* **IRET**
* **中断服务程序的最后一条指令，负责**

**恢复断点**

**恢复标志寄存器内容**

**（6）处理器控制指令**

**对标志位的操作**

**与外部设备的同步**

**结束语**

* **掌握：**
* **指令的格式及意义；**
* **指令对操作数的要求及对标志位的影响；**
* **指令的应用。**

幻灯片1

**第4章  
汇编语言程序设计**

主要内容

* **汇编语言源程序的结构**
* **汇编语言语句格式**
* **伪指令**
* **功能调用**
* **汇编语言程序设计**

幻灯片3

**§4.1 汇编语言源程序**

* **了解：**
* **汇编语言源程序的结构**
* **汇编语言语句类型及格式**

一、汇编语言源程序结构

**1. 汇编语言源程序与汇编程序**

* **汇编语言源程序**
* **汇编程序**

**2. 汇编语言程序设计与执行过程**

3. 汇编语言源程序结构

**数据段名 SEGMENT**

**…**

**数据段名 ENDS**

**附加段名 SEGMENT**

**…**

**附加段名 ENDS**

**堆栈段名 SEGMENT**

**…**

**堆栈段名 ENDS**

**代码段名 SEGMENT**

**…**

**代码段名 ENDS**

**END**

一个完整源程序结构

**DSEG SEGMENT**

**DATA1 DB 1，2， 3 DUP（？）**

**DATA2 DW 1234H**

**DSEG ENDS**

**ESEG SEGMENT**

**DB 20 DUP（？）**

**ESEG ENDS**

**SSEG SEGMENT STACK ‘STACK’**

**DB 200 DUP（？）**

**SSEG ENDS**

**CSEG SEGMENT**

**ASSUME CS：CSEG，DS：DSEG，ES：ESEG，SS：SSEG**

**START：MOV AX，DSEG**

**MOV DS，AX**

**MOV AX，ESEG**

**MOV ES，AX**

**MOV AX，SSEG**

**MOV SS，AX**

**┇**

**CSEG ENDS**

**END START**

**段寄存器初始化**

**——将段地址送相应的段寄存器**

**源程序**

**代码**

**【例】编写一个两个字相加的程序**

**DSEG SEGMENT ；定义数据段**

**DATA1 DW 1234H ；定义被加数**

**DATA2 DW 5678H ；定义加数**

**DSEG ENDS ；数据段结束**

**ESEG SEGMENT ；定义附加段**

**SUM DW 2 DUP（？） ；定义存放结果区**

**ESEG ENDS ；附加段结束**

**CSEG SEGMENT ；定义代码段**

* **；下面的语句说明程序中定义的各段分别用哪个段寄存器寻址**
* **ASSUME CS：CSEG，DS：DSEG，ES：ESEG**
* **START： MOV AX，DSEG ；START为程序开始执行的启动标号**
* **MOV DS，AX ；初始化DS**
* **MOV AX，ESEG**
* **MOV ES，AX ；初始化ES**
* **LEA SI，SUM ；存放结果的偏移地址送SI**
* **MOV AX，DATA1 ；取被加数**
* **ADD AX，DATA2 ；两数相加**
* **MOV ES：[SI]，AX ；和送附加段的SUM单元中**
* **HLT**
* **CSEG ENDS ；代码段结束**
* **END START ；源程序结束**

1. **汇编语言语句类型及格式**

**1. 汇编语言语句类型**

**CPU执行的语句，能够生成目标代码**

**CPU不执行，而由汇编程序执行的语句，不生成目标代码**

**2. 汇编语言语句格式**

**指令性语句：**

**[标号：] [前缀] 助记符 [操作数]，[操作数] [ ；注释]**

**操作码**

**指令的符号地址**

**标号后要有冒号**

**注释前加分号**

**标号有三种属性：段、偏移量和类型。**

**① 标号的段属性是定义标号在程序段的段地址，当程序中引用一个标号时，该标号的段值应在CS寄存器中。**

**② 标号的偏移量属性表示标号所在段的起始地址到定义该标号的地址之间的字节数。偏移量是一个16位无符号数。**

**③ 标号的类型属性有两种：NEAR和FAR。前一种标号可以在段内被引用；后一种标号可以在其它段被引用。**

指示性语句格式

**[名字] 伪指令助记符 操作数 [，操作数，…] [ ；注释]**

**指示性语句中至少有一个操作数**

**变量的符号地址**

**其后不加冒号**

**3. 操作数**

**寄存器**

**存储器单元**

**常量**

**变量或标号**

**表达式**

常 量

* **数字常量:以字母A～F开头的十六进制数，前面要用0表达，以避免与其他符号混淆。**
* **字符串常量:用单引号引起的字符或字符串**
* **例：‘A’**
* **MOV AL，’A’**
* **例：’ABCD’**

**汇编时被译成对应的ASCII**

**码41H，42H，43H，44H**

常量并不被分配内存空间，因为存储在内存空间中的内容是可变的

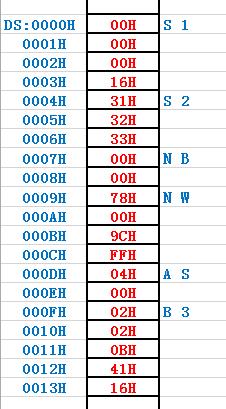
幻灯片19

变 量

* **代表内存中的数据区，程序中视为存储器操作数**
* **变量的属性：**
* **变量所在段的段地址**
* **变量单元地址与段首地址之**
* **间的位移量。**
* **字节型、字型和双字型**

幻灯片20

**变量空间分配：变量与常量不同，汇编程序会给变量被分配存储空间，EQU定义的常量则不占用存储空间。**



**例1：某个数据段为：**

**DATA SEGMENT**

**S1 DB 0,?,?,22**

**S2 DB ‘123’**

**COUNT EQU $－S1**

**NB DB 2 DUP(?)**

**NW DW 120,-100**

**AS DW S2 ;用地址表达式定义变量**

**B3 DB 2 DUP(2),1 UP(11,’A’),16H**

**DATA ENDS**

表达式

**算术运算**

**逻辑运算**

**\*关系运算**

**取值运算和属性运算**

**其它运算**

算术运算和逻辑运算符

* **算术运算符**
* **+，-，\*，/，MOD**
* **逻辑运算符**
* **AND，OR，NOT，XOR**
* **例：**
* **MOV AL，8 AND 4**
* **MOV AL，8+4-1**

取值运算符

* **用于分析存储器操作数的属性**
* **获取变量的属性值**

**取得其后变量或标号的偏移地址**

**取得其后变量或标号的段地址**

**OFFSET**

**TYPE 取变量的类型**

**LENGTH 取所定义存储区的长度**

**SIZE 取所定义存储区的字节数**

* **Type 返回值：**
* **（1）对于变量有3种。**
* **1：字节型； 2：字型；4：双字型;**
* **（2）对于标号有2种：**
* **－1：NEAR（段内）**
* **－2：FAR（段间）**

取值运算符例

**MOV AX，SEG DATA**

**MOV DS，AX**

**MOV BX，OFFSET DATA**

**LEA BX，DATA**

取值运算符例

* **若BUFFER存储区用如下伪指令定义：**
* **BUFFER DW 200 DUP(0)**
* **则：**
* **TYPE BUFFER 等于2**
* **LENGTH BUFFER 等于200**
* **SIZE BUFFER 等于400**

属性运算符

* **用于指定其后存储器操作数的类型**
* **运算符：PTR**
* **例：**
* **MOV BYTR PTR[BX]，12H**

其它运算符

* **方括号：**
* **[ ]**
* **段重设符**
* **段寄存器名：[ ]**

上述运算符和操作符构成表达式时的优先级：

（）、[ ]、< >、LENGTH、SIZE、WIDTH、MASK；PTR、OFFSTE、SEG、TYPE、THIS、段操作符；HIGH、LOW；

\*、/、MOD、SHL、SHR；

+、-；

EQ、NE、LT、LE、GT、GE；

NOT；

AND、OR、XOR；

SHORT

**从左到右顺序运算**

**§4.2 伪指令**

* **掌握：**
* **伪指令的格式及实现的操作**
* **伪指令的应用**

伪指令

* **由汇编程序执行的“指令系统”**
* **作用：**
* **定义变量；**
* **分配存储区**
* **定义逻辑段；**
* **指示程序开始和结束；**
* **定义过程等。**

常用伪指令

**数据定义伪指令**

**符号定义伪指令**

**段定义伪指令**

**结束伪指令**

**过程定义伪指令**

**宏命令伪指令**

一、数据定义伪指令

* **用于定义数据区中变量的类型及大小**
* **格式：**
* **[变量名] 伪指令助记符 操作数，… ；[注释]**

**定义变量值及区域大小**

**符号地址**

**定义变量类型**

幻灯片34

**1. 数据定义伪指令助记符**

* **DB 定义的变量为字节型**
* **DW 定义的变量为字类型（双字节）**
* **DD 定义的变量为双字型（4字节）**
* **DQ 定义的变量为4字型（8字节）**
* **DT 定义的变量为10字节型**

幻灯片35

数据定义伪指令例

* **DATA1 DB 11H，22H，33H，44H**
* **DATA2 DW 11H，22H，3344H**
* **DATA3 DD 11H\*2，22H，33445566H**

**以上变量在内存中的存放形式**

**数据定义伪指令例\_变量在内存中的分布**

数据定义伪指令的几点说明

* **伪指令的性质决定所定义变量的类型；**
* **定义字符串必须用DB伪指令**
* **例：**
* **DATA1 DB ‘ABCD’，66H**

**2. 重复操作符**

* **作用：**
* **为一个数据区的各单元设置相同的初值**
* **目的：**
* **常用于声明一个数据区**
* **格式：**
* **[变量名] 伪指令助记符 n DUP（初值，…）**
* **例：**
* **M1 DB 10 DUP（0）**

**3. “？”的作用**

* **表示随机值，用于预留存储空间**
* **MEM1 DB 34H，’A’，？**
* **DW 20 DUP（？）**

**随机数**

**占1个字节单元**

**预留40个字节单元，每单元为随机值**

数据定义伪指令例

* **M1 DB ‘How are you?’**
* **M2 DW 3 DUP(11H)，3344H**
* **DB 4 DUP（？）**
* **M3 DB 3 DUP（22H，11H，？）**

数据定义伪指令例

二、符号定义伪指令

* **格式：**
* **符号名 EQU 表达式 等价于 符号名 = 表达式**
* **操作：**
* **用符号名取代后边的表达式，不可重新定义**
* **例：**
* **CONSTANT EQU 100**
* **VAR EQU 30H+99H**

**EQU说明的表达式不占用内存空间**

三、段定义伪指令

* **说明逻辑段的起始和结束；**
* **说明不同程序模块中同类逻辑段之间的联系形态**

段定义伪指令格式

**段名 SEGMENT [定位类型] [组合类型] [’类别’]**

**┇**

**段名 ENDS**

**说明不同模块中同名段的组和连接方式**

**说明逻辑段的起点**

1. 段名 SEGMENT
2. ┇
3. 段名 ENDS

只需要掌握这种形式

定位类型

* **PARA：段的起点从节边界开始（16个字节为1节）**
* **段的起始地址应为：XXXX0H（被16整除）**
* **BYTE：段的起点从存储器任何地址开始（字节边**
* **界开始）**
* **WORD：段的起点从偶地址开始（字边界开始）**
* **PAGE：段的起点从页边界开始（256个字节为1页）**
* **段的起始地址应为：XXX00H（被256整除）**

组合类型

* **与其它模块中的同名段在满足定位类型的前提**
* **下具有的组合方式：**
* **NONE： 不组合**
* **PUBLIC： 依次连接（顺序由LINK程序确定）**
* **COMMON： 覆盖连接**
* **STACK： 堆栈段的依次连接**
* **AT 表达式：段定义在表达式值为段基的节边界**
* **MEMORY： 相应段在同名段的最高地址处。**

类别

* **不同模块连接时将相同类别的段放在连续的内存区域中**

段定义伪指令例

**DATA SEGMENT**

**MEM1 DB 11H，22H**

**MEM2 DB ‘Hello！’**

**MEM3 DW 2 DUP（？）**

**DATA ENDS**

四、设定段寄存器伪指令

* **说明所定义逻辑段的性质**
* **说明定义的各段分别用哪一个段寄存器寻址**
* **格式：**
* **ASSUME 段寄存器名:段名[，段寄存器名:段名，…]**

五、结束伪指令

* **表示源程序结束**
* **格式：**
* **END [标号]**

六、过程定义伪指令

* **用于定义一个过程体**
* **格式：**
* **过程名 PROC [ NEAR / FAR ]**
* **┇**
* **RET**
* **过程名 ENDP**

**过程入口的符号地址**

**P157例**

过程定义及调用例

* **定义延时子程序**
* **DELAY PROC**
* **PUSH BX**
* **PUSH CX**
* **MOV BL，2**
* **NEXT： MOV CX，4167**
* **W10M： LOOP W10M**
* **DEC BL**
* **JNZ NEXT**
* **POP CX**
* **POP BX**
* **RET**
* **DELAY ENDP**
* **调用延时子程序：**
* **CALL DELAY**

七、宏命令伪指令

**宏**

* **源程序中由汇编程序识别的具有独**
* **立功能的一段程序代码**
* **格式：**
* **宏命令名 MACRO <形式参数>**
* **┇**
* **┇**
* **ENDM**

八、其它伪指令

* **段内程序代码或变量的起始偏移地址**
* **格式：**
* **ORG 表达式**
* **例：**
* **ORG 2000H**

**ORG**

**计算值为非负常数**

ORG可置于代码段、数据段的任何地方

无ORG时则从偏移地址为0000H开始存放代码和数据

【例】给汇编地址计数器（指针）＄赋值

DATA SEGMENT

ORG 10 ；置＄值为10

VAR1 DW 100H，200H

ORG ＄+5 ；置＄的值为14+5，即为19

VAR2 DB 1，2， ＄+1，＄+2

N EQU ＄-VAR2 ；=23

DATA ENDS

注：$表示当前汇编地址指针的值。

其它伪指令

* **NAME-------为目标程序设定一个名字**
* **格式：**
* **NAME 模块名**
* **TITLE-------为程序清单指定打印标题**
* **格式：**
* **TITLE 标题名**
* **.8086-------汇编程序将在8086／8088方式下操作**
* **.386-------汇编程序将在80386方式下操作**

**§4.3 功能调用**

**DOS 功能调用**

**BIOS功能调用**

**DOS 功能调用**

* **包含多个子功能的功能包；**
* **用软中断指令调用，中断类型码固定为21H；**
* **各子功能采用功能号来区分。**

**DOS 功能调用**

**设备管理**

**目录管理**

**文件管理**

**其它**

**DOS 功能调用**

* **调用格式：**
* **MOV AH，功能号**
* **<置相应参数>**
* **INT 21H**

一、单字符输入

* 键盘输入单字符（1号调用）
* 格式：MOV AH，1
* INT 21H
* 功能：等待键盘输入，输入字符的ASCII码送入AL寄存
* 器，并在显示器上显示该字符。

**例：程序中有时需要用户对提示做出应答。**

**GET\_KEY: MOV AH,1 ;等待键入字符**

**INT 21H ;结果在AL中**

**CMP AL,’Y’ ;是’Y’?**

**JZ YES ;是,转YES**

**CMP AL,’N’ ;是’N’?**

**JZ NO ;是,转NO**

**无入口参数；**

**☞执行时系统等待键盘输入，若键入Ctrl-Break，则退出;**

* **功能号7和8也可接收键盘输入的单字符，但无回显。**

单字符输入例

**GET\_KEY: MOV AH,1**

**INT 21H**

**CMP AL,’Y’**

**JZ YES**

**CMP AL,’N’**

**JZ NO**

**JMP GET\_KEY**

**YES: ┇**

**NO: ┇**

二、字符串输入

* **注意问题：**
* **调用格式**
* **字符输入缓冲区的定义**

**1. 调用格式**

* **功能号OAH**
* **字符串在内存中的存放地址**
* **INT 21H**

幻灯片65

**2.键盘输入字符串（0A号调用）**

**格式：MOV AH，0AH**

**LEA DX, <字符串缓冲区首地址>**

**INT 21H**

**功能：等待键盘输入，输入字符串的ASCII码送内存单元。**

1. 注意：**先定义一个缓冲区；最后输入回车键以示字符串输入结束：未填满，剩余填零；超过缓冲区容量，自动丢失并响铃。**

**N1: 缓冲区长度(最大键入字符数)， N2: 实际键入的字符数(不包括回车符)**

**例：设在数据段定义键盘缓冲区如下：**

STR1 DB 10,?,10 DUP(?)

**；程序段：**

LEA DX，STR1

MOV AH，0AH

输入字符串程序段

* **DAT1 DB 20，？，20 DUP（？）**
* **┇**
* **LEA DX，DAT1**
* **MOV AH，0AH**
* **INT 21H**

输入缓冲区

**定义后的输入缓冲区初始状态：**

三、单字符显示输出

* **功能号O2H**
* **待输出字符**
* **INT 21H**
* **将DL寄存器中的字符在显示器上显示，如果要退出按〈CTRL〉+〈BREAK〉**

单字符显示输出例

* **MOV AH，02**
* **MOV DL，41H**
* **INT 21H**

四、字符串输出显示

* **功能号O9H**
* **待输出字符串的偏移地址**
* **INT 21H**

**被显示的字符串必须以‘$’结束,且所显示的内容不应出现非可见的ASCII码**

字符串输出显示例

**DATA SEGMENT**

**MESS1 DB ‘Input String:’, 0DH，0AH,’$’**

**DATA ENDS**

**CODE SEGMENT**

**┇**

**MOV AH，09**

**MOV DX，OFFSET MESS1**

**INT 21H**

**┇**

五、返回操作系统（4CH号调用）

**格式：MOV AH，4CH**

**INT 21H**

**功能：终止当前程序并返回调用程序(操作系统)。**

1. DATA SEGMENT
2. STR DB 0DH，0AH，‘Example of string display！＄’
3. DATA ENDS
4. ；定义了数据段，0DH回车符ASCII码、0AH换行符ASCII码

**【例】编写程序完成在屏幕上显示字符串“Example of string display！”**

【例】编写程序完成从键盘上输入一字符串到输入缓冲区，然后将输入的字符串在显示器上以相反的顺序显示。

**§4.4源程序两种格式书写**

* **第一种格式从MASM 5.0开始支持**
* **简化段定义格式**
* **第二种格式MASM 5.0以前就具有**
* **完整段定义格式**

Hello, Everybody !

**;lt301a.asm（文件名）**

**.model small ;定义程序的存储模式**

**.stack ;定义堆栈段**

**.data ;定义数据段**

**string db ’Hello, Everybody !’,0dh,0ah,’$’**

**;在数据段定义要显示的字符串**

**.code ;定义代码段**

**.startup ;程序起始点，建立DS、SS**

**mov dx,offset string ;指定字符串**

**mov ah,9**

**int 21h ;利用功能调用显示信息**

**.exit 0 ;程序结束点，返回DOS**

**end ;汇编结束**

**简化段定义格式**

**;SampleA.ASM**

**.model small**

**.stack**

**.data**

**... ;在数据段定义数据**

**.code**

**.startup**

**... ;在代码段填入指令序列**

**.exit 0**

**... ;子程序代码**

**end**

**简化段定义格式  
 MASM 6.x支持**

**;SampleC.ASM**

**.model small**

**.stack**

**.data**

**... ;在数据段定义数据**

**.code**

**start：mov ax,@data**

**mov ds,ax**

**... ;在代码段填入指令序列**

**mov ax,4c00h**

**int 21h**

**... ;子程序代码**

**end start**

**简化段定义格式  
 MASM 5.x支持**

**;lt301b.asm（文件名）**

**stack segment stack ;定义堆栈段**

**dw 512 dup(?)**

**;堆栈段有512字（1024字节）空间**

**stack ends ;堆栈段结束**

**data segment ;定义数据段**

**string db ’Hello, Everybody !’,0dh,0ah,’$’**

**data ends**

**code segment ’code’ ;定义代码段**

**assume cs:code,ds:data,ss:stack**

**start: mov ax,data ;建立DS段地址**

**mov ds,ax**

**完整段定义格式**

**mov dx,offset string**

**mov ah,9**

**int 21h**

**mov ax,4c00h**

**int 21h ;利用功能调用返回DOS**

**code ends ;代码段结束**

**end start**

**;汇编结束，同时指明程序起始点**

**完整段定义格式**

**;SampleB.ASM**

**stack segment stack**

**dw 512 dup(?)**

**stack ends**

**data segment**

**... ;在数据段定义数据**

**data ends**

**code segment ’code’**

**assume cs:code,ds:data,ss:stack**

**start: mov ax,data**

**mov ds,ax**

**... ;在代码段填入指令序列**

**mov ax,4c00h**

**int 21h**

**... ;子程序代码**

**code ends**

**end start**

**完整段定义格式  
 MASM 5.x支持**

**§4.5 汇编语言程序设计**

* **设计步骤：**
* **根据实际问题抽象出数学模型**
* **确定算法**
* **画程序流程图**
* **分配内存工作单元和寄存器**
* **程序编码**
* **调试**

汇编语言程序结构

* **顺序结构**
* **循环结构**
* **先判定型**
* **后判定型**
* **分支结构**
* **单分支**
* **多分支**

**顺序结构程序**

**什么是顺序结构程序？**

**【例】已知某班学生的英语成绩按学号（从1开始）从小到大的顺序排列在TAB表中，要查的学生的学号放在变量NO中，查表结果放在变量ENGLISH中。试编写程序实现成绩查询。**

**分析：**

**1，建立数据表，BX  TAB偏移地址**

**2，学号NO 值送AL**

**3、查表：AL [BX+AL]**

**5、返回操作系统，使用DOS功能调用**

**顺序结构程序**

思考：

**程序运行时改为有键盘输入学号，会改程序吗？**

**分析：1、建立数据表，BX  TAB偏移地址**

2、键盘输入学号送AL，使用DOS功能调用

3、将键盘输入的字符转换为数字

4、查表：AL [BX+AL]

5、返回操作系统，使用DOS功能调用

**MOV AH,01H**

INT 21H ;键盘输入的数据送AL

**分支结构程序**

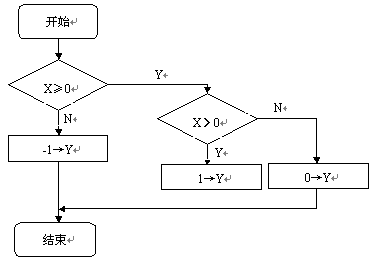
分支结构程序也称条件结构，根据不同条件转移到不同程序段执行相关分支程序。有两个分支和多个分支。

**【例】：编写计算下面函数值的程序：**

**1 ;X>0**

**Y= 0 ;X=0**

**-1 ;X<0**



**分支结构程序**

**【例】 统计BUF+1开始的N个字符组成的字符串中数字、大写字母、和其它字符的个数，并存入字符串后面的三个单元中。**

**BUF+1**

**BUF+2**

**分析：**

**（1）数字ASCII码在30H~39H之间**

**大写字母ASCII码在41H~5AH之间**

**48H**

**（2）CX计数，初值 为BUF 单元的内容**

**（3）BX作指针初值为1**

**（4）DH，DL存放数字和字母的个数**

**（5）其它字符的个数=N-DH-DL**

幻灯片94

**分支结构程序**

流程图如下：

**CX N ；字符个数**

**DH、DL 0；数字及字母个数**

**BX 1；指针**

**循环结构程序**

一、 循环程序的结构

**循环结构是指重复执行某公共程序段若干次，直到满足条件，才结束循环操作。有单重循环和多重循环。构成：**

**1．初始化部分；2．循环体部分；3．循环控制部分**

**当型循环：当条件满足时，执行循环体，否则退出循环**

**直到型循环：先执行循环体，再判条件是否满足。若不满足则执行循环体，否则退出循环**

**循环结构程序**

二、单重循环程序设计

**常用循环控制方法有：**

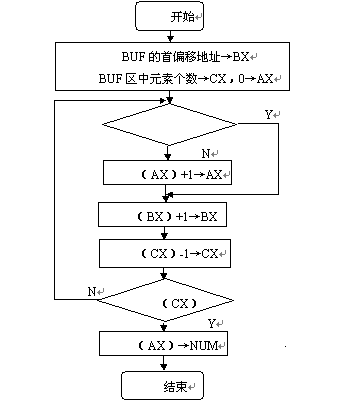
（1）计数器控制：循环次数已知的情况；

（2）条件控制：循环次数未知，循环过程中某个特定条件是否满足；

（3）开关变量控制：根据开关变量的值确定进入下一个循环体，有循环条件来控制整个循环的执行。

（4）逻辑尺控制：以某一存储单元中某位是1或0去执行不同的循环体。

幻灯片97



本章注意点

* **完整的汇编语言源程序结构**
* **定义逻辑段，说明段的含义，初始化段寄存器**
* **伪指令**
* **数据定义方式**
* **字符及字符串的输入和显示输出**
* **字符输入缓冲区的定义，输出字符串的定义**
* **源程序的编写**
* **几种结构（顺序、循环、分枝等）**

附录.自己平时编写的一些有参考价值的程序

1. 编写一个加密程序，实现：将‘449255’保存在数据段NUM中；将这6个数字按照加密规则进行加密后的结果存入到数据段MIMA中。其中加密规则为：'0123456789'分别对应"6435072198"。

注：分成三次，逐步将程序细化，最后一个为本题的解答

小解：（只显示，不存储）

data segment

BIAO db 06h,04h,03h,05h,00h,07h,02h,01h,09h,08h

db 00h,01h,02h,03h,04h,05h,06h,07h,08h,09h

NUM db 04h,04h,09h,02h,05h,05h

MIMA db 00h,00h,00h,00h,00h,00h

data ends

assume cs:code,ds:data,ss:stack

stack segment

db 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0

stack ends

code segment

start: mov ax,stack

mov ss,ax

mov sp,20h

mov ax,data

mov ds,ax

mov cx,6

mov bx,0014h

s: mov al,[bx]

mov ah,00h

mov di,ax

mov ax,[di]

;;;;;;;;;;;;;

mov dx,ax

mov ax,0200h

add dx,30h

int 21h

;;;;;;;;;;;;;;

inc bx

loop s

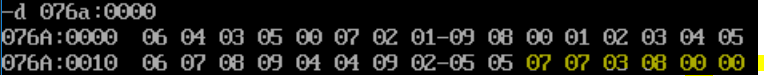
mov ax,4c00h

int 21h

code ends

end start



再解：（逆序存储了） 

data segment

BIAO db 06h,04h,03h,05h,00h,07h,02h,01h,09h,08h

db 00h,01h,02h,03h,04h,05h,06h,07h,08h,09h

NUM db 04h,04h,09h,02h,05h,05h

MIMA db 00h,00h,00h,00h,00h,00h

data ends

assume cs:code,ds:data,ss:stack

stack segment

db 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0

stack ends

code segment

start: mov ax,stack

mov ss,ax

mov sp,20h

mov ax,data

mov ds,ax

mov cx,6

mov bx,0014h

s: mov al,[bx]

mov ah,00h

mov di,ax

mov ax,[di]

;;;;;;;;;;;;;

mov dx,ax

mov ax,0200h

push dx

add dx,30h

int 21h

;;;;;;;;;;;;;;

inc bx

loop s

mov bx,001ah

mov cx,6

sb: pop [bx]

inc bx

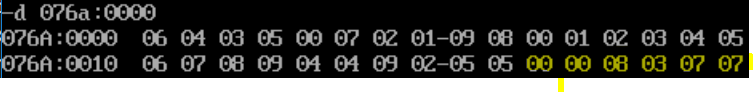
loop sb

mov ax,4c00h

int 21h

code ends

end start

再来解（最终解答）（存储加显示）：

data segment

BIAO db 06h,04h,03h,05h,00h,07h,02h,01h,09h,08h

db 00h,01h,02h,03h,04h,05h,06h,07h,08h,09h

NUM db 04h,04h,09h,02h,05h,05h

MIMA db 00h,00h,00h,00h,00h,00h

data ends

assume cs:code,ds:data,ss:stack

stack segment

db 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0

stack ends

code segment

start: mov ax,stack

mov ss,ax

mov sp,20h

mov ax,data

mov ds,ax

mov cx,6

mov bx,0014h

s: mov al,[bx]

mov ah,00h

mov di,ax

mov ax,[di]

;;;;;;;;;;;;;

mov dx,ax

mov ax,0200h

push dx

add dx,30h

int 21h

;;;;;;;;;;;;;;

inc bx

loop s

mov bx,001ah

mov cx,6

sb: pop [bx]

inc bx

loop sb

mov bx,001ah

mov cx,6

ml1: push [bx]

inc bx

loop ml1

mov bx,001ah

mov cx,6

ml2: pop [bx]

inc bx

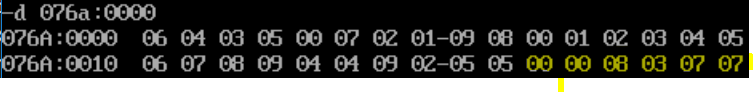
loop ml2

mov ax,4c00h

int 21h

code ends

end start

2、编写一个过程求6！，结果放在DX:AX中，同时在屏幕中显示。

解：

data segment

db 00h,00h,00h,00h

data ends

assume cs:code,ds:data,ss:stack

stack segment

db 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0

stack ends

code segment

start:

mov ax,stack

mov ss,ax

mov sp,0020h

mov ax,1

mov cx,6

mov bx,6

s: mul bx

dec bx

loop s

mov dx,ax

mov dx,0000h

mov bx,100

div bx

push dx

mov bx,0

mov bl,al

mov ax,0200h

mov dx,bx

add dx,30h

int 21h

pop dx

mov ax,dx

mov dx,0

mov bx,10

div bx

mov dh,0

mov dl,ah

push dx

mov dl,al

mov ax,0200h

add dx,30h

int 21h

pop dx

mov ax,0200h

add dx,30h

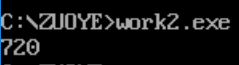
int 21h

mov ax,4c00h

int 21h

code ends

end start



3、从键盘输入一个长度为N的字符串，存放在A1单元，找出其中所有的小写字母并存放到以A2单元开始的存储区中。统计出小写字母的个数，存放到SL单元中。

DATA SEGMENT

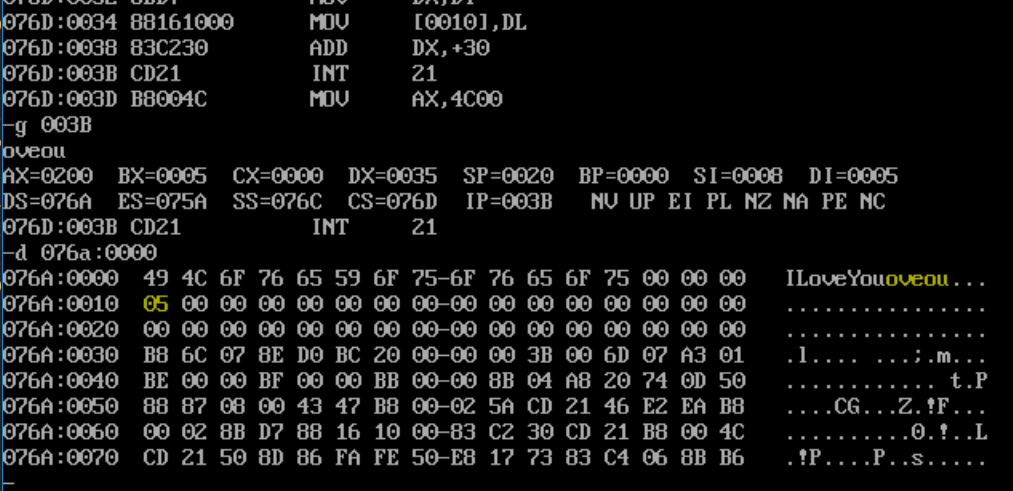
A1 DB ?

N EQU $-A1

A2 DB N DUP(?)

SL DB ?

DATA ENDS

解：

data segment

a1 db 'ILoveYou'

N EQU $-a1

a2 db N dup(?)

SL db ?

data ends

assume cs:code,ds:data,ss:stack

stack segment

db 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0

stack ends

code segment

start:

mov ax,stack

mov ss,ax

mov sp,20h

mov ax,data

mov ds,ax

mov cx,N

mov si,0

mov di,0

mov bx,0

haha:

mov ax,[si]

test al,00100000b

jz bushow

push ax

mov a2[bx],al

inc bx

inc di

mov ax,0200h

pop dx

int 21h

bushow: inc si

loop haha

mov ax,0200h

mov dx,di

mov SL,dl

add dx,30h

int 21h

mov ax,4c00h

int 21h

code ends

end start

4、编写一个完整程序，将DB1中的二进制数按位倒序排列存入DB2中，即DB1第0位送DB2第7位，DB1第1位送DB2第6位，依次类推

DATA SEGMENT

DB1 DB XX;任一字节数

DB2 DB?

DATA ENDS

解：

data segment

dba db 00h,01h,02h,03h,04h,05h,06h,07h

dbb db 0,0,0,0,0,0,0,0

data ends

stack segment

db 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0

stack ends

assume ds:data,cs:code,ss:stack

code segment

start: mov ax,stack

mov ss,ax

mov sp,20h

mov ax,data

mov ds,ax

mov bx,0

mov cx,8

s1: push [bx]

inc bx

loop s1

mov bx,8

mov cx,8

s2:

pop [bx]

inc bx

loop s2

mov ax,4c00h

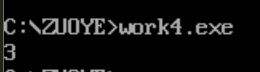
int 21h

code ends

end start



再解：



data segment

db1 db 11000000b

db2 db 00000000b

data ends

assume cs:code,ds:data

code segment

start:

mov ax,data

mov ds,ax

mov dx,0

mov bl,00000001b

mov bh,10000000b

mov ax,0

mov cx,8

s: mov al,byte ptr ds:[0]

or al,bl

cmp al,byte ptr ds:[0]

jnz ok

or ah,bh

ok:shl bl,1

shr bh,1

loop s

mov byte ptr ds:[1],ah

mov dl,ah

mov ax,0200h

add dx,30h

int 21h

mov ax,4c00h

int 21h

code ends

end start

再解：

data segment

buf1 db 11000000b

buf2 db 00000000b

data ends

assume cs:code,ds:data

code segment

start: mov ax,data

mov ds,ax

mov ah,buf1

mov dh,buf2

mov cx,8

yi:clc

shl ah,1

rcr dh,1

loop yi

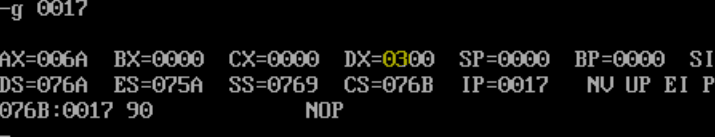
nop

mov ax,4c00h

int 21h

code ends

end start



新功能：把移位后的结果打印出来

data segment

db1 db 11010100b

db2 db 00000000b

data ends

assume cs:code,ds:data,ss:stack

stack segment

db 32 dup(0)

stack ends

code segment

show0:

push ax

push dx

mov ax,0200h

mov dx,30h

int 21h

pop dx

pop ax

ret

show1:

push ax

push dx

mov ax,0200h

mov dx,31h

int 21h

pop dx

pop ax

ret

showhuiche:

push ax

push dx

mov ax,0200h

mov dx,0ah

int 21h

pop dx

pop ax

ret

start:

mov ax,stack

mov ss,ax

mov sp,32

mov ax,data

mov ds,ax

mov dx,0

mov bl,00000001b

mov bh,10000000b

mov ax,0

mov cx,8

s: mov al,byte ptr ds:[0]

or al,bl

cmp al,byte ptr ds:[0]

jnz ok

or ah,bh

call show1

jmp buxie0

ok:

call show0

buxie0:

shl bl,1

shr bh,1

loop s

call showhuiche

mov byte ptr ds:[1],ah

mov dl,ah

mov ax,0200h

add dx,30h

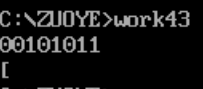
int 21h

mov ax,4c00h

int 21h

code ends

end start



5、在BUF1和BUF2两个数据区中，各定义有5个带符号的数据，试编制一完整的源程序，求它们对应项的绝对值之差，并将差值存入以SUB为首地址的数据区中。

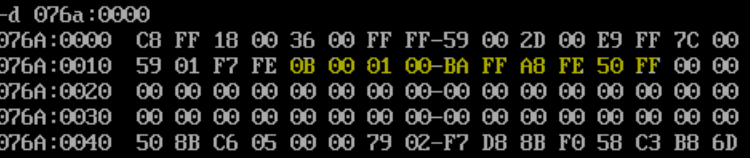
DATA SEGMENT

BUF1 DW -56,24,54,-1,89

BUF2 DW 45,-23,124,345,-265

SUM DW 10DUP(0)

DATA ENDS



解：

data segment

buf1 dw -56,24,54,-1,89

buf2 dw 45,-23,124,345,-265

result dw 10 dup(0)

data ends

assume cs:code,ds:data,ss:stack

stack segment

db 0,0,0,0 ,0,0,0,0, 0,0,0,0 ,0,0,0,0

stack ends

code segment

absolute: push ax

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

mov ax,si

add ax,0

jns zhen

neg ax

zhen :mov si,ax

;;;;;;;;;;;;;;;;;;;;;;;;;;;;

pop ax

ret

start:mov ax,stack

mov ss,ax

mov ax,data

mov ds,ax

mov bx,0

mov si,0

mov di,0

mov cx,5

s: mov ax,word ptr [bx].00h

mov si,ax

call absolute

mov ax,si

mov dx,word ptr [bx].0ah

mov si,dx

call absolute

mov dx,si

sub ax,dx

push ax

pop word ptr result[bx]

add bx,2

loop s

nop

mov ax,4c00h

int 21h

code ends

end start

解：

assume cs:code,ds:data

data segment

buf1 dw -56,24,54,-1,89

buf2 dw 45,-23,124,345,-265

sum dw 10 dup(0)

data ends

code segment

start:mov ax,data

mov ds,ax

mov cx,5

lea si,buf1

lea di,buf2

lea bp,sum

next3:

mov bx,ds:[si]

add bx,0

jns next1

neg bx

next1:

mov dx,ds:[di]

add dx,0

jns next2

neg dx

next2:

sub bx,dx

mov ds:[bp],bx

add si,2

add di,2

add bp,2

loop next3

nop

mov ax,4c00h

int 21h

code ends

end start

冒泡排序：

解：

data segment

shuzu db 08h,02h,09h,04h,05h,0ah,01h,03h,07h,06h

N EQU $-shuzu

xulie db 10 dup(0)

data ends

assume cs:code,ds:data

code segment

start:

mov ax,data

mov ds,ax

mov bx,0

mov cx,N-1

daxunhuan: mov dx,cx

mov bx,0

xiaos :

mov al,[bx]

inc bx

cmp al,[bx]

jna zhenchang

xchg al,[bx]

zhenchang:

mov [bx-1],al

loop xiaos

mov cx,dx

loop daxunhuan

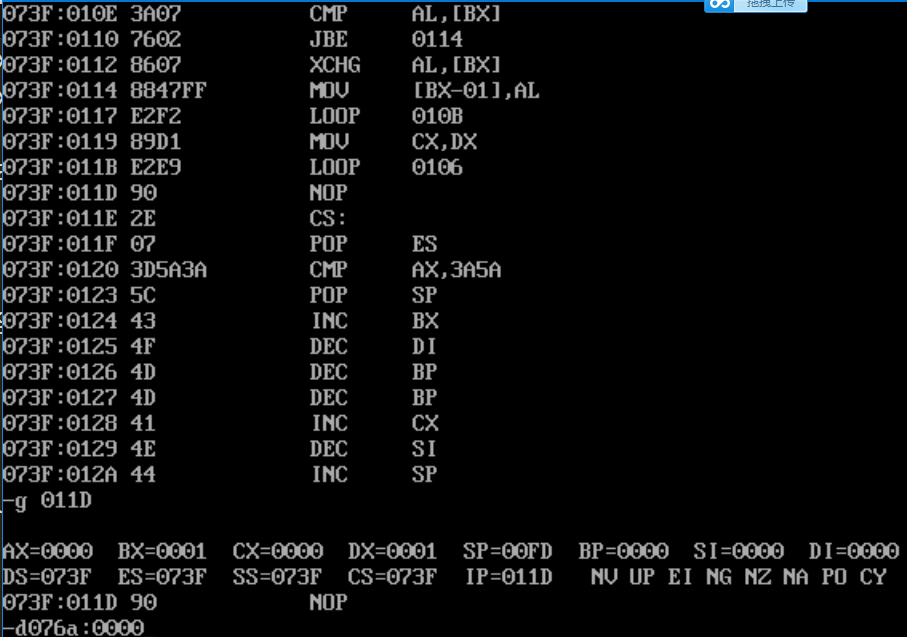
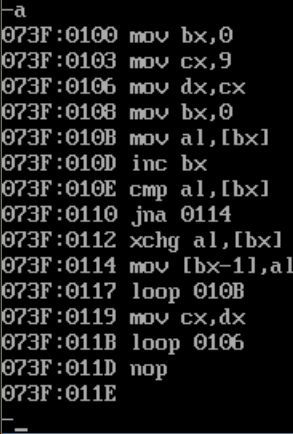
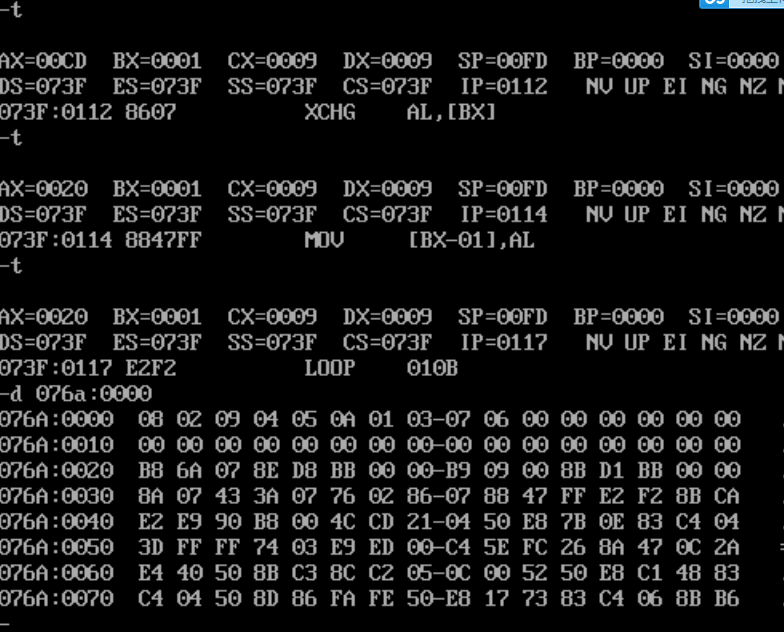
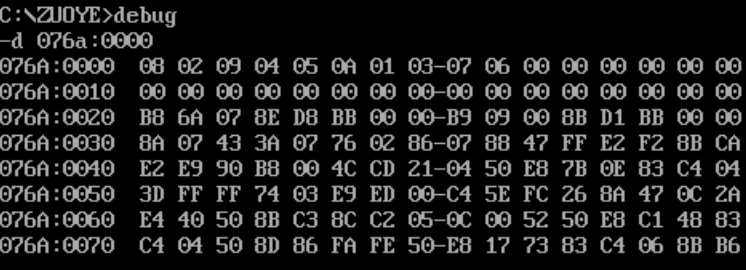
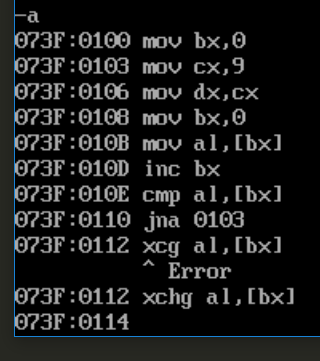
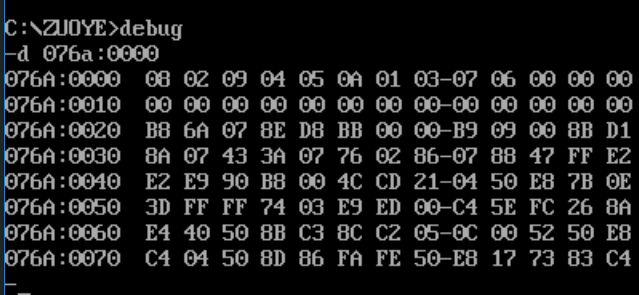
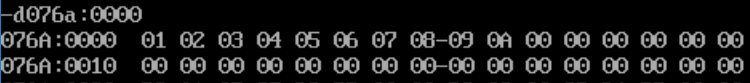
nop

mov ax,4c00h

int 21h

code ends

end start



1xian.asm

;功能：显示一个字符并显示其后继字符

CODE SEGMENT

ASSUME CS:CODE

START: MOV AH ,1 ;1号功能是键盘输入

INT 21H ;功能调用，输入的东西放到al

MOV DL,AL

ADD DL,1

MOV AH,2 ;2号功能是显示

INT 21H

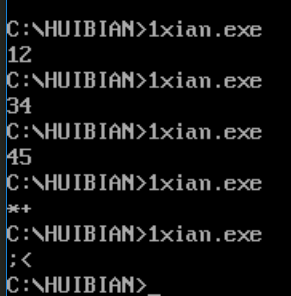
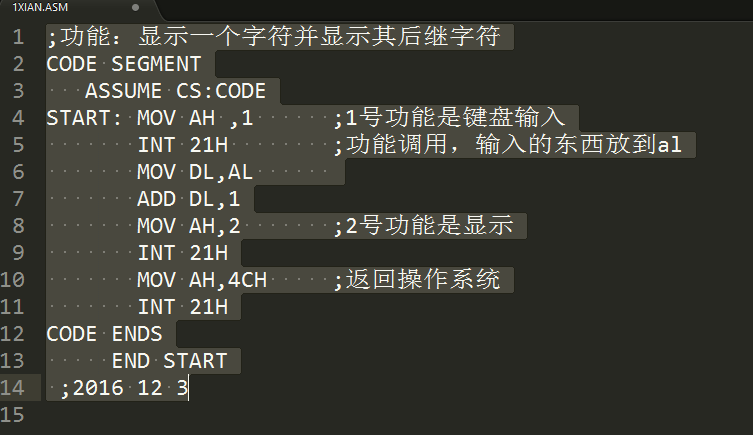
MOV AH,4CH ;返回操作系统

INT 21H

CODE ENDS

END START

;2016 12 3



2hello.asm

;功能：显示helloworld

DATA SEGMENT

STRING DB 'HELLO,WORLD!$'

DATA ENDS

CODE SEGMENT

ASSUME CS:CODE,DS:DATA

START:

MOV AX,DATA

MOV DS,AX ;把data通过ax送入DS

MOV DX,OFFSET STRING ;指定字符串

MOV AH,9

INT 21H ;调用功能9显示字符串

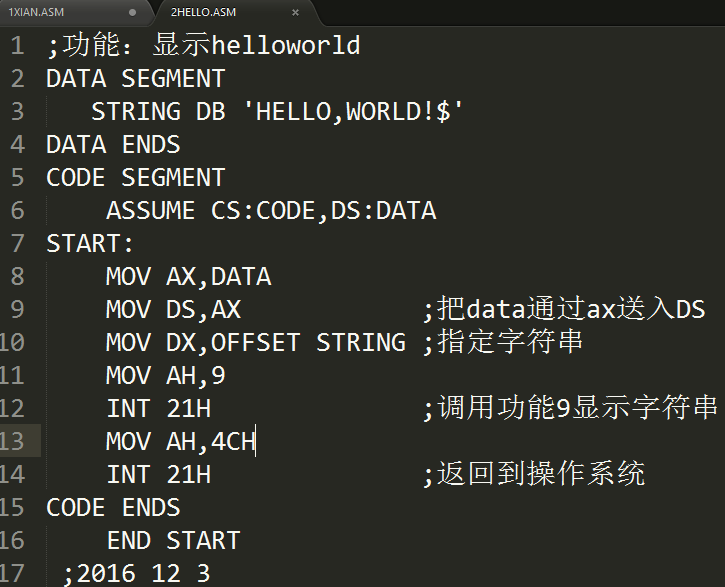
MOV AH,4CH

INT 21H ;返回到操作系统

CODE ENDS

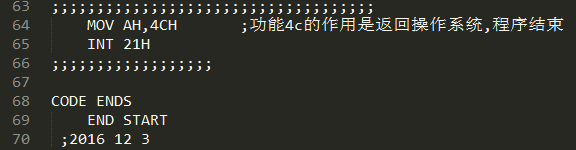
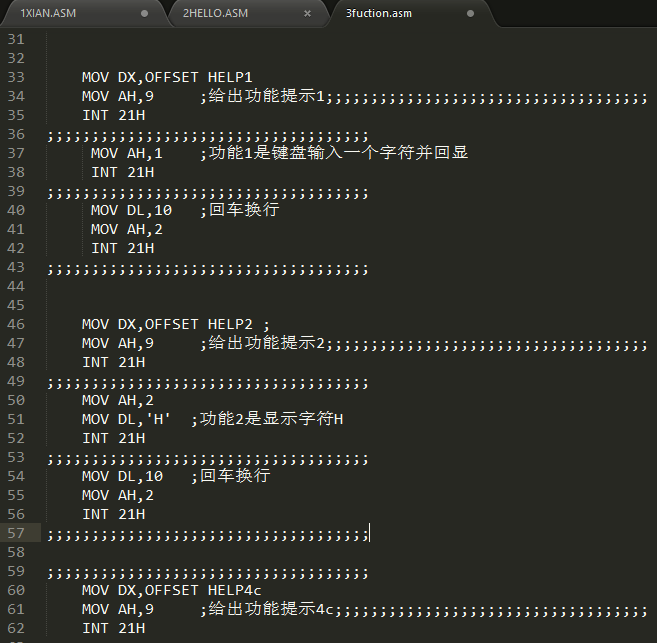
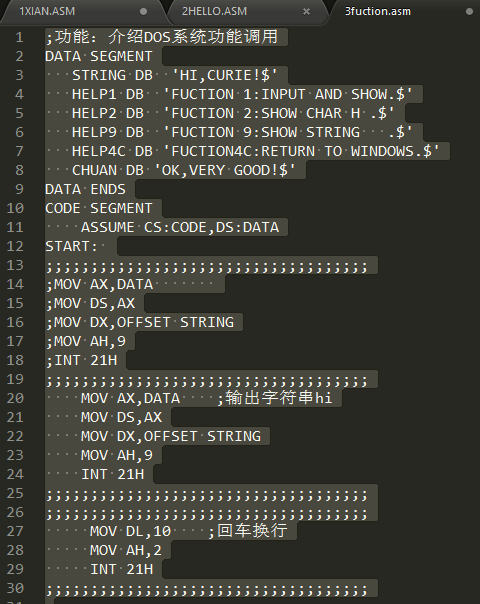
END START

;2016 12 3





3fuction.asm

;功能：介绍DOS系统功能调用

DATA SEGMENT

STRING DB 'HI,CURIE!$'

HELP1 DB 'FUCTION 1:INPUT AND SHOW.$'

HELP2 DB 'FUCTION 2:SHOW CHAR H .$'

HELP9 DB 'FUCTION 9:SHOW STRING .$'

HELP4C DB 'FUCTION4C:RETURN TO WINDOWS.$'

CHUAN DB 'OK,VERY GOOD!$'

DATA ENDS

CODE SEGMENT

ASSUME CS:CODE,DS:DATA

START:

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;MOV AX,DATA

;MOV DS,AX

;MOV DX,OFFSET STRING

;MOV AH,9

;INT 21H

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

MOV AX,DATA ;输出字符串hi

MOV DS,AX

MOV DX,OFFSET STRING

MOV AH,9

INT 21H

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

MOV DL,10 ;回车换行

MOV AH,2

INT 21H

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

MOV DX,OFFSET HELP1

MOV AH,9 ;给出功能提示1;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

INT 21H

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

MOV AH,1 ;功能1是键盘输入一个字符并回显

INT 21H

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

MOV DL,10 ;回车换行

MOV AH,2

INT 21H

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

MOV DX,OFFSET HELP2 ;

MOV AH,9 ;给出功能提示2;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

INT 21H

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

MOV AH,2

MOV DL,'H' ;功能2是显示字符H

INT 21H

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

MOV DL,10 ;回车换行

MOV AH,2

INT 21H

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

MOV DX,OFFSET HELP4c

MOV AH,9 ;给出功能提示4c;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

INT 21H

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

MOV AH,4CH ;功能4c的作用是返回操作系统,程序结束

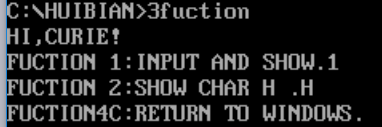
INT 21H

;;;;;;;;;;;;;;;;;;

CODE ENDS

END START

;2016 12 3



4COUNT.ASM

DATA SEGMENT

X DW 4

Y DW 2

Z DW 20

V DW 24

DATA ENDS

CODE SEGMENT

ASSUME CS:CODE,DS:DATA

START :

MOV AX,DATA

MOV DS,AX

MOV AX,X

IMUL Y

MOV CX,AX

MOV BX,DX

MOV AX,Z

CWD

ADD CX,AX

ADC BX,DX

SUB CX,16

SBB BX,0

MOV AX,V

CWD

SUB AX,CX

SBB DX,BX

IDIV X

MOV AH,2

MOV DX,AX

ADD DX,30H

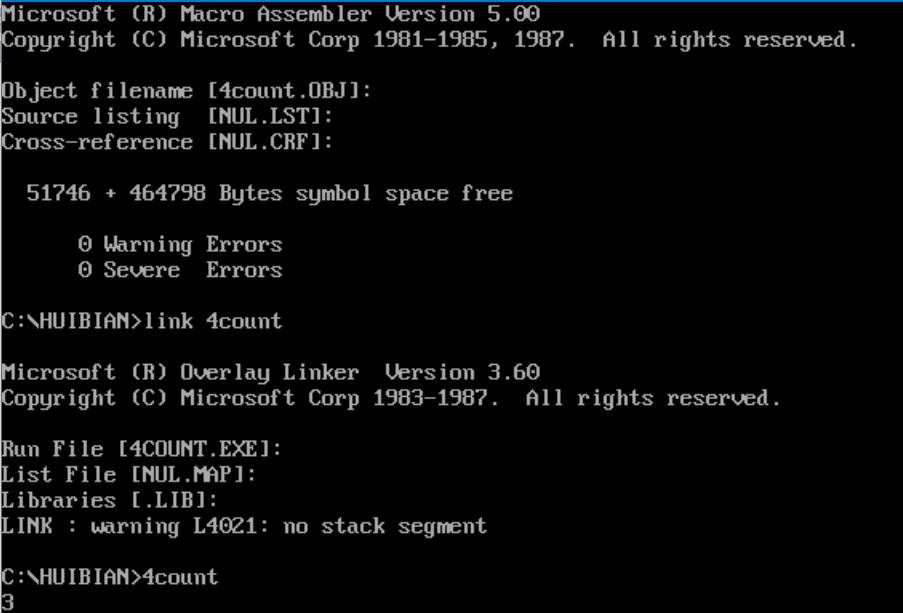
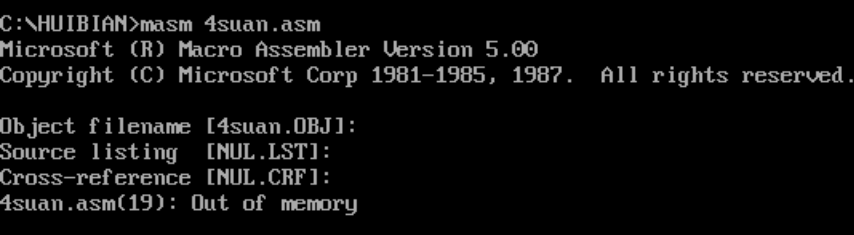
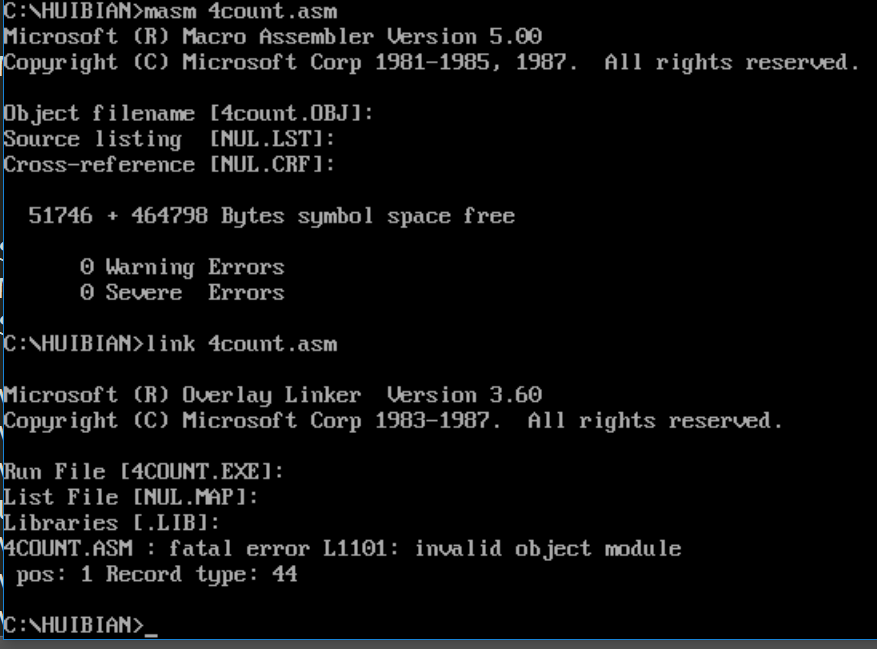
INT 21H

MOV AH,4CH

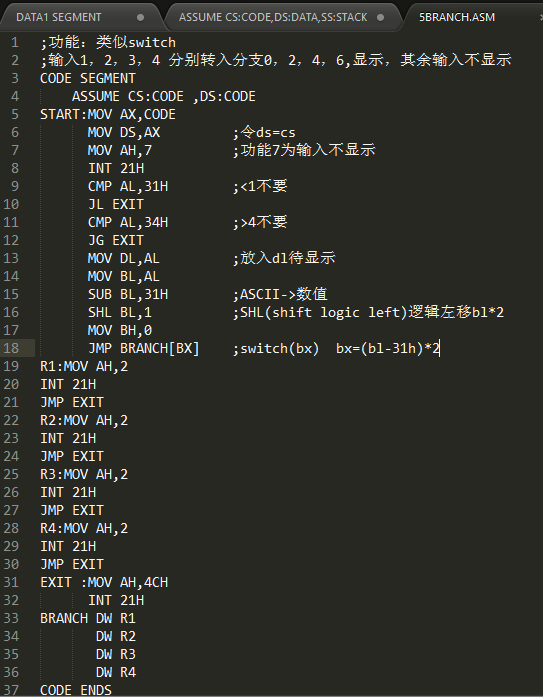
INT 21H

CODE ENDS

END START



5branch.asm



;功能：类似switch

;输入1，2，3，4 分别转入分支0，2，4，6,显示，其余输入不显示

CODE SEGMENT

ASSUME CS:CODE ,DS:CODE

START:MOV AX,CODE

MOV DS,AX ;令ds=cs

MOV AH,7 ;功能7为输入不显示

INT 21H

CMP AL,31H ;<1不要

JL EXIT

CMP AL,34H ;>4不要

JG EXIT

MOV DL,AL ;放入dl待显示

MOV BL,AL

SUB BL,31H ;ASCII->数值

SHL BL,1 ;SHL(shift logic left)逻辑左移bl\*2

MOV BH,0

JMP BRANCH[BX] ;switch(bx) bx=(bl-31h)\*2

R1:MOV AH,2

INT 21H

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

INT 21H

BRANCH DW R1

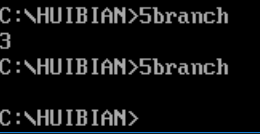
DW R2

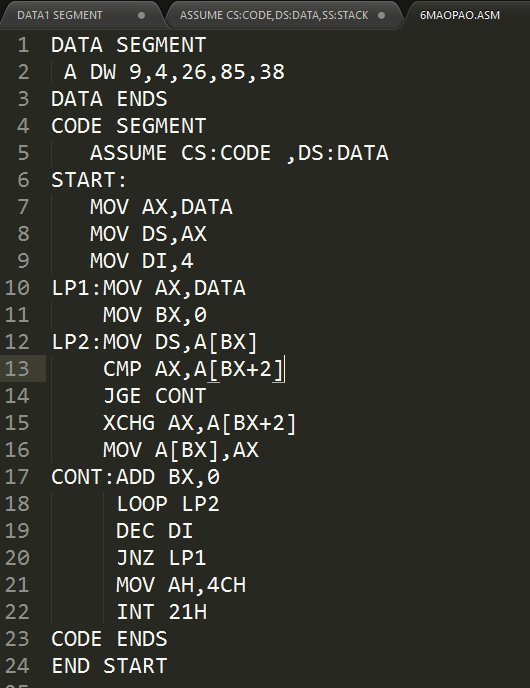
DW R3

DW R4

CODE ENDS

END START



6MAOPAO.ASM

DATA SEGMENT

A DW 9,4,26,85,38

DATA ENDS

CODE SEGMENT

ASSUME CS:CODE ,DS:DATA

START:

MOV AX,DATA

MOV DS,AX

MOV DI,4

LP1:MOV AX,DATA

MOV BX,0

LP2:MOV DS,A[BX]

CMP AX,A[BX+2]

JGE CONT

XCHG AX,A[BX+2]

MOV A[BX],AX

CONT:ADD BX,0

LOOP LP2

DEC DI

JNZ LP1

MOV AH,4CH

INT 21H

CODE ENDS

END START