

**CENTRAL SOUTH UNIVERSITY**

**计算机组成原理与汇编语言课程实验报告**

题 目 汇编语言实验课程实验报告

学生姓名 牛奕

班级学号 智能1903班 8207191131

指导教师 李仪

设计时间 2021年7月

目录

[第一章 实验目的 1](#_Toc22371)

[第二章 程序功能与使用 6](#_Toc325)

[2.1 实验一：累计乘法 6](#_Toc18581)

[2.2 实验二：累计减法 8](#_Toc14584)

[2.3 实验三：16/32位乘法 10](#_Toc7050)

[2.4 实验四：16/32位除法 10](#_Toc7050)

[2.5 实验五：累计数加法 10](#_Toc7050)

[2.6 实验六：移位运算 10](#_Toc7050)

[2.7 实验七：双精度位移 10](#_Toc7050)

[2.8 实验八：压缩/非压缩BCD加减法 10](#_Toc7050)

[2.9 实验九：KMP算法字符匹配 10](#_Toc7050)

[2.10 实验十：斐波那契数列实现 10](#_Toc7050)

[第三章 程序功能实现方法 12](#_Toc21681)

[3.1 实验9 KMP算法 12](#_Toc9573)

[3.2 实验10 斐波那契数列实现 13](#_Toc7218)

[第四章 设计经验体会总结 21](#_Toc227)

[4.1 数据库概念结构设计 21](#_Toc26320)

[4.2 数据库逻辑结构设计与实现 24](#_Toc10935)

[第五章 源程序清单 26](#_Toc3896)

[2.1 实验一：累计加法 6](#_Toc18581)

[2.2 实验二：累计减法 8](#_Toc14584)

[2.3 实验三：16/32位乘法 10](#_Toc7050)

[2.4 实验四：16/32位除法 10](#_Toc7050)

[2.5 实验五：累计数加法 10](#_Toc7050)

[2.6 实验六：移位运算 10](#_Toc7050)

[2.7 实验七：双精度位移 10](#_Toc7050)

[2.8 实验八：压缩/非压缩BCD加减法 10](#_Toc7050)

[2.9 实验九：KMP算法字符匹配 10](#_Toc7050)

[2.10 实验十：斐波那契数列实现 10](#_Toc7050)

[致 谢 36](#_Toc24890)

[参考文献 37](#_Toc1552)

# 第一章 实验目的

通过对以下十个实验的编写，调试，了解汇编语言的基本语法，能够明白基本语句的含义并深入理解寻址方式。

1. 编写一个累计加法，从 1 加到 5，将结果保存至 AX 中。

2. 编写一个累计减法，被减数是 10011000B，减数是 01000000B，连续减 5 次，

观察 FLAGS 的变化

3. 编写一个 16 位的乘法，被乘数是 100H，乘数是 100H，观察 Flags 的变化，

编写一个 32 位的乘法，被乘数是 0F0FH，乘数是 FF00H，观察 Flags 的变

化。

4. 编写一个 16 位的除法，被除数是 100H，除数是 100H，观察 Flags 的变化，

编写一个 32 位的除法，被除数是 0F0FH，除数是 00FFH，观察 Flags 的变

化。

5. 编写一个累计加法，被加数是 0FH，加数是 01H，观察 Flags 的变化，被加

数是 0FFH，加数是 01H，观察 Flags 的变化，被加数是 0FFFH，加数是 01H，

观察 Flags 的变化，被加数是 FFFFH，加数是 01H，观察 Flags 的变化，被

加数是 FFFFFFFFH 加数是 01H，观察 Flags 的变化。

6. 编写一个移位运算，将 8F1DH 存至 AX，然后用指令右移 1 位然后左移 1 位，

显示结果并观察 Flags 的变化。将 8F1DH 存至 AX 中，然后带 CF 位左移 5

位，并右移 7 位，观察 Flags 的变化，并给出结果。

7. 将 71D2H 存至 AX 中，5DF1H 存至 CX 中，DST 为 AX，REG 为 AX，实现双精度右移 2 次，交换 DST 与 REG，然后左移 4 次，分别查看结果.

8. 实现压缩BCD码的加减法，用压缩BCD码实现（21+71），（12+49），（65+82），

（46-33），（74-58），,（43-54）的十进制加减法。然后又用非压缩 BCD 实现

上述 6 个式子。

9. 实现 KMP 算法，输入两个字符串（可以直接保存在内存中），实现快速匹配

10. 斐波纳契数列：1，1，2，3，5，8，13。通常可以使用递归函数实现，现用

汇编实现该过程

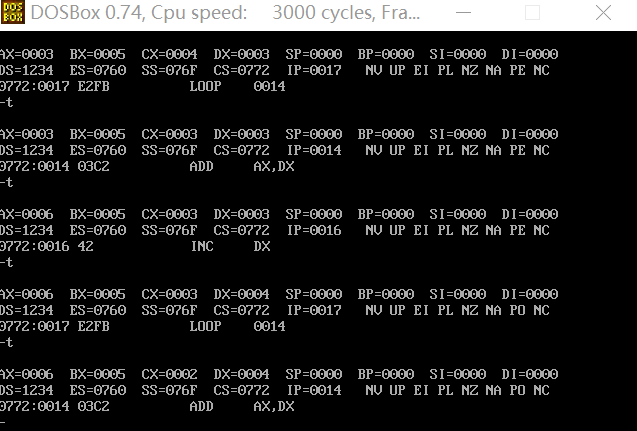
# 第二章 程序功能与使用

关于debug

-t逐步调试，注意各个寄存器的值即可

## 2.1 累计加法

写一个循环，cx保存次数为五次，ax保存累计和，bx保存每次的加数，每循环一次bx自增。



## 2.2 累计减法

写一个循环，cx保存次数为五次，ax保存被减数，bx保存减数。

FLAG寄存器的变化：

有符号溢出标志位OF(Over flow flag) OV(1) NV(0)

方向标志位DF(Direction flag) DN(1) UP(0)

中断标志位IF(Interrupt flag) EI(1) DI(0)

符号标志位SF(Sign flag) NG(1) PL(0)

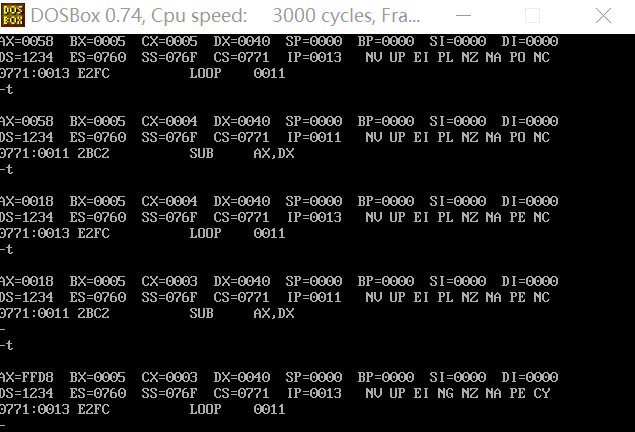
零标志位ZF(Zero flag) ZR(1) NZ(0)

辅助进位标志位AF(Auxiliary carry flag) AC(1) NA(0)

奇偶标志位PF(Parity flag) PE(1) PO(0)

无符号进位标志位CF(Carry flag) CY(1) NC(0)



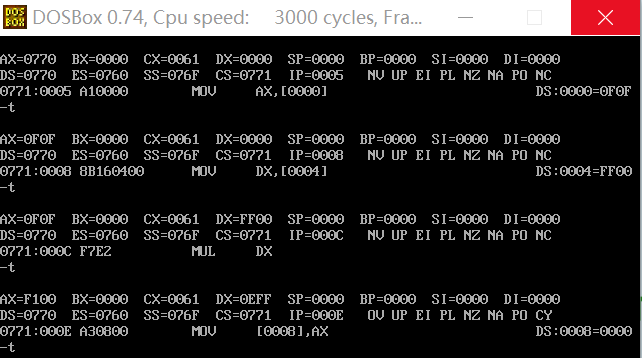


## 2.3 16/32位乘法

对于16位乘法，运用循环加法的思想即可完成，ax保存累加和，bx存储被乘数，cx存乘数，每次循环将ax和bx的值相加。

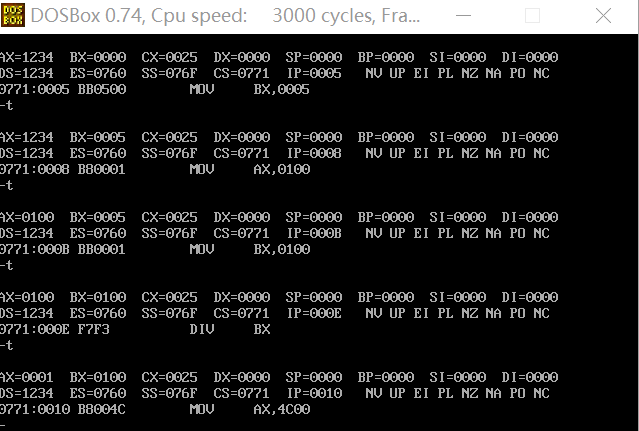
对于32位乘法，申请一个字节的空间的数据结构,x1作为被乘数的低16位，x2作为被乘数的高16位，y1作为乘数的低16位，y2作为乘数的高16位，作4次乘法，从低位起，分别记录x1y1,x1y2,x2y1,x2y2的值并存入寄存器中完成32位的相乘。





## 2.4 16/32位除法

对于16位除法，简单实用一个div指令就可以。



对于32位除法有点难度，可以采用这样一个公式：

X/N = int(H/N)\*65536 + [rem(H/N)\*65536+L]/N -----------------------公式①

X:被除数

N:除数

H:被除数X的高16位

L:被除数X的低16位

int()取商

rem()取余数

仔细观察公式①的第一项：

int(H/N)\*65536

int(H/N)：被除数的高16位/除数，这里将H拓展为一个32位的数（0000xxxxh），由于其高16位全为0，所以H/N的商一定不会溢出（超过16位）。

那乘以65536应该怎么做呢？我们只需要记得将H/N的商放到一个代表高16位的寄存器中就可以了，并不需要真的乘65536(事实上\*10000h也无法简单地用16位乘法实现）。

再看公式②的第二项：

[rem(H/N)\*65536+L]/N

rem(H/N)已经在第一项除法中得到了，假设 X = rem(H/N)，接下来实现（X+L)/N，显然也是一个32位数/16位数的除法。但这里存在一个问题：如何保证除法的商不会溢出呢？

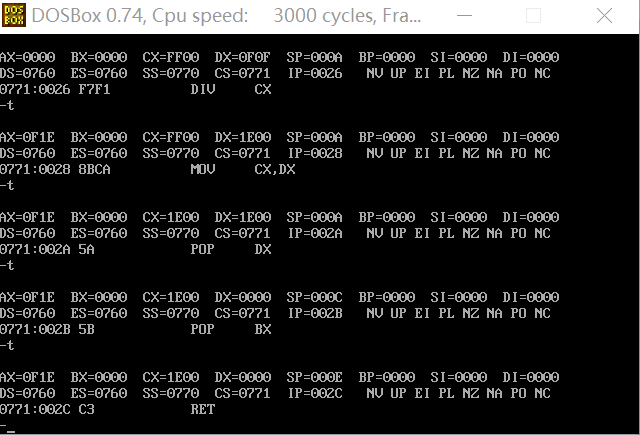
在这里简单的证明一下：

因为 X 是 H/N 的余数，所以 X<=N-1 (X不可能大于N)，L是被除数的低16位，不妨令 X=N-1，L=FFFFh。只需证明

[(N-1)|FFFFh]/N<10000h

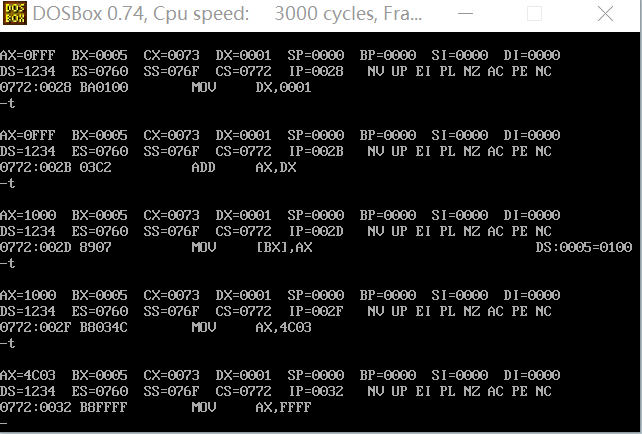
（((N-1)|FFFFh)以N-1为高16位，以FFFF为低16位。比如N-1 = FFFE，则分子为FFFEFFFFh）。

原式=[(N-1)\*65536 + 65535]/N=65536 - 1/N < 10000h



## 2.5 累计加法

每进行一次加法将结果保存在bx中，即可完成累计加法。



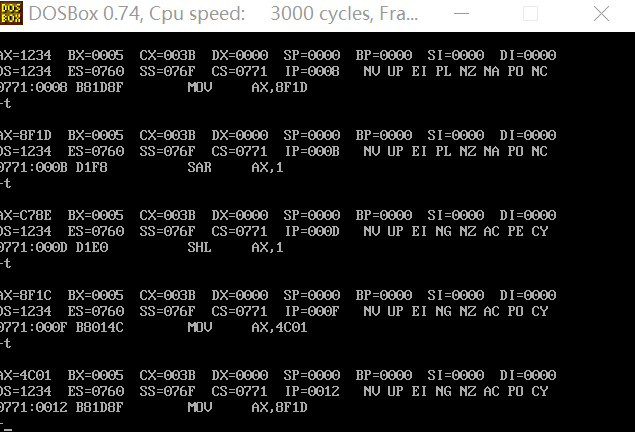
## 2.6 移位运算

要求算数移位和循环移位，设置循环次数可以指定移位的位数。

算术移位

Sal是算数左移，sar是算数右移

循环移位就是带CF位的移位

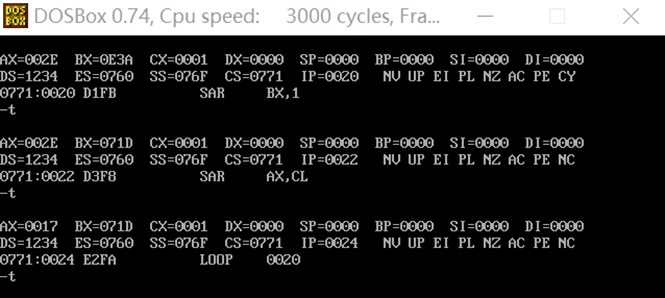


## 2.7 双精度移位运算

## 这里本来用的是masm来写，发现masm识别不了shrd和shld

没办法只能自己撸：

将每次溢出的位用寄存器保存起来，送给高位或者低位的补位，以此完成双精度的移位。



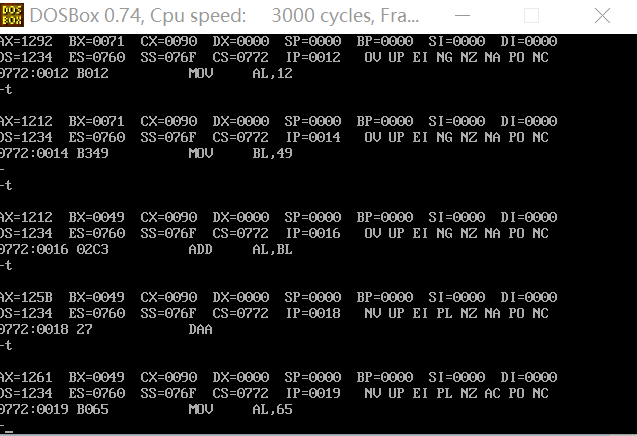
## 2.8 压缩/非压缩BCD加减法

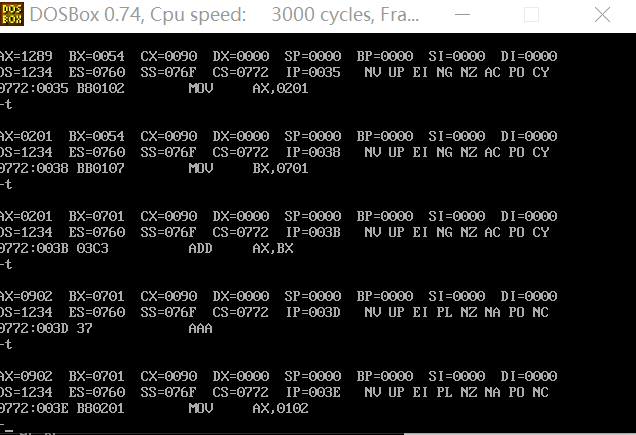
BCD码：用四个二进制位表示一个十进制数字；最常用的是8421 BCD码；

压缩型BCD码：一个字节可存放一个两位十进制数，其中高四位存放十位数字，低四位存放个位数字。如：56的压缩型8421 BCD码是0101 0110；

非压缩型BCD码：一个字节可存放一个一位十进制数，其中高字节为0，低字节的低四位存放个位。如：5的非压缩型BCD码是0000 0101，必须存放在一个字节中，56的非压缩型BCD码是00000101 00000110，必须存放在一个字中。

压缩BCD码用daa指令来调整，非压缩BCD码用aaa指令来调整。





## 2.9 KMP算法字符匹配

（1）“查找匹配字符串SEARCH”要求，程序接受用户键入的一个关键字以及一个句子。如果句子中不包含关键字则显示‘No match!’；如果句子中包含关键字则显示‘Match’，且把该字在句子中的位置用十六进制数显示出来。

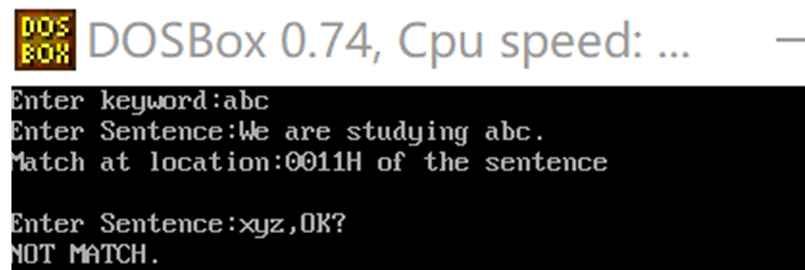
（2）程序由三部分组成：

①输入一个关键字和一个句子，分别存入相应的缓冲区中。

②使用串比较指令在句子中查找关键字，用一个循环结构来完成句子和关键字的比较过程。循环次数为（句子长度-关键字长度）+1。

③输出信息。

首先需要有输入限制要求，即输入的关键字长度不能大于句子的长度。然后是比较句子中的字符和关键字，利用cld依次将si、di增1，repz cmpsb比较si、di指向的字符，若三次比较均成功，则跳转至match（表示匹配成功），在match中实现将句子中匹配成功的首地址存放在bx中。否则继续下面的执行，恢复si指针指向关键字首地址，di指向句子的匹配失败的首地址的下一个地址，然后继续循环compare（继续寻找匹配），若最后失败则执行到wrong。



## 2.10 斐波那契数列实现

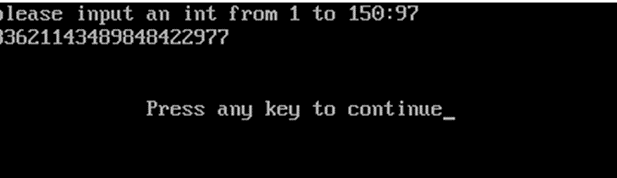
(1) 程序“求Fibonacci数”要求接收用户键入的范围在0~100之间的n值，并根据给定的n值计算Fibonacci数，输出结果。

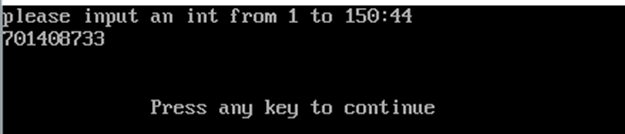
(2) 程序由三部分组成：

①输入部分：接收用户键入的n值经十化二后存入num单元中。

②调用子程序fibp求出FIB（n）的值存放在result单元中。在递归子程序设计中，每次调用可把一帧信息存入堆栈。

③输出部分：把运行结果经二化十后在屏幕上显示出来。

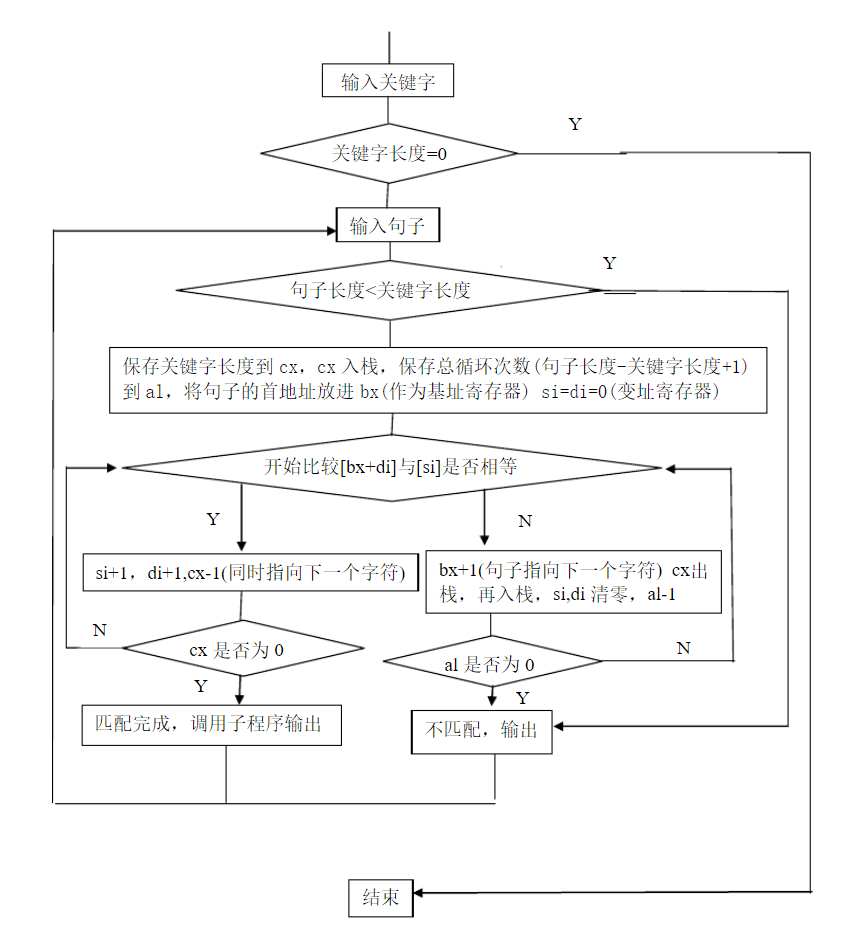




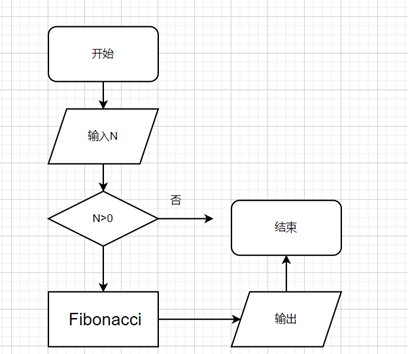
**第三章 程序功能实现方法**

实验1-8相对比较容易，不必画图说明，从第九个实验开始：

**3.1 实验9 KMP算法：**



**3.2 实验10 斐波那契数列实现：**



**第四章 设计经验体会总结**

汇编这十个实验看起来在C/C++语言中很好实现，但是在汇编语言中却非常复杂，因为可用的寄存器有限，存入数据段的程序还比较复杂，就导致了需要特别仔细地去思考整个程序对于电脑是如何运行的，要用机器的思维完成整个过程。

比如一句for循环就可以解决的问题，在汇编里需要先存储CX，再进行循环体，而CX的暂存需要压栈出栈。在汇编中，由于数据存储量小且麻烦，需要大量地用到“栈”这个结构体进行暂存；再比如，汇编的封装性差，如果没有跳转指令，程序就会一直向下运行，所以段与段之间的逻辑关系非常重要。实验中段的跳转一度把我搞晕，必须要十分清楚整个程序的逻辑关系，才能运行出程序；又比如，对于高级语言，数据输入几就是几，但是汇编程序是按照ASCII码存储的，所以输入输出要进行转换。还有多个段时段地址的切换，偏移地址的切换。总而言之，在高级语言中我们可以通过调用函数，给出变量名等方便的操作，在汇编语言中都被打成了原型，需要从计算机的角度调用。

通过次此课程设计，我对计算机的存储方式有了更深入的认识，从根本上了解了程序运行的原理，受益匪浅，汇编语言非常锻炼人的思维能力，据说一个很厉害的程序员每打一行代码就可以想到对应的汇编语言，所以汇编语言虽然不会很常用，但是提升了我们的思维能力，希望未来对高级语言的运用能够因此更加自如。

**第五章 源程序清单**

## 5.1 累计加法

assume cs:cseg,ds:data

data segment

infor1 db 'welcome to work1!'

data ends

cseg segment

start:mov dx,offset infor1;offsert返回的是infor1的偏移地址

mov ax,1234h

mov ds,ax

mov bx,0005h

mov ax,0

mov dx,1

mov cx,5

s:add ax,dx

inc dx;(dx)++

loop s

mov ds:[bx],ax;程序返回

mov ax,4c00H;程序返回

INT 21H

cseg ends

end start

## 5.2 累计减法

assume cs:cseg,ds:data

data segment

db 'welcome to masm!'

data ends

cseg segment

start:mov ax,1234h

mov ds,ax

mov bx,0005h

mov ax,10011000B

mov dx,01000000B

mov cx,5

s:sub ax,dx

loop s

mov ds:[bx],ax

mov ax,4c00H

INT 21H

cseg ends

end start

## 5.3 16/32位乘法

16位：

assume cs:cseg,ds:data

data segment

db 'welcome to masm!'

data ends

cseg segment

start:mov ax,1234h

mov ds,ax

mov bx,0005h

mov ax,0

mov dx,100H

mov cx,100H

s:add ax,dx

loop s

mov ax,4c00H

INT 21H

cseg ends

end start

32位：

assume cs:ccode, ds:ddata

ddata segment

x1 dw 0F0FH ;被乘数低四位

x2 dw 0000H ;被乘数高四位

y1 dw 0FF00H ;乘数低四位

y2 dw 0000H ;乘数高四位

xy dw 4 dup (?) ;dup重复定义指令，dw为定义的类型，开辟出4个字的单元空间

ddata ends

ccode segment

start:mov ax,ddata

mov ds,ax

mov ax,x1

mov dx,y1

mul dx

mov [xy],ax

mov [xy+2],dx ;被乘数低位4字符x1和乘数低位4字符y1相乘结果低位存入xy,高位存入xy+2

mov ax,x2

mov dx,y1

mul dx

add [xy+2],ax

adc [xy+4],dx ; 被乘数高位4字符x2和乘数低位4字符y1相乘结果低位存入xy+2,高位存入xy+4

mov ax,x1

mov dx,y2

mul dx

add [xy+2],ax

adc [xy+4],dx

adc [xy+6],0 ; 被乘数低位4个字符x1和乘数高位4个字符y2相乘结果低位存入xy+2,高位存入xy+4

mov ax,x2

mov dx,y2

mul dx

add [xy+4],ax

adc [xy+6],dx ; 被乘数高位4个字符x2和乘数高位4个字符y2相乘结果低位存入xy+4,高位存入xy+6

mov ah,4ch

int 21h

ccode ends

end start

## 5.4 16/32位除法

16位：

assume cs:cseg,ds:data

data segment

db 'welcome to masm!'

data ends

cseg segment

start:mov ax,1234h

mov ds,ax

mov bx,0005h

mov ax,100H

mov bx,100H

div bx

mov ax,4c00H

INT 21H

cseg ends

end start

32位：

assume ds:datasg,ss:stacksg,cs:codesg

datasg segment

datasg ends

stacksg segment

dw 0,0,0,0,0,0,0,0

stacksg ends

codesg segment

start:mov ax,stacksg

mov ss,ax

mov sp,16

mov ax,0000h

mov dx,0F0Fh

mov cx,0FF00h

call divdw

mov ax,4c00h

int 21h

divdw:push bx;防止寄存器冲突覆盖原有的值

push ax;低16位进栈

mov ax,dx;32位/16位

mov dx,0

div cx;int(H/N)\*65536,商AX,余数DX,商AX就是最终结果的高16位

mov bx,ax;bx保存高16位商

pop ax;低16位送入ax中

push bx

div cx;[rem(H/N)\*65536+L]/N,低16位商AX不用动,余数DX

mov cx,dx;cx中保存的是余数

pop dx;dx中是高16位商

pop bx;恢复寄存器

ret

codesg ends

end start

## 5.5 累计加法

assume cs:cseg,ds:data

data segment

infor1 db 'welcome to work1!'

data ends

cseg segment

start:mov dx,offset infor1

mov ax,1234h

mov ds,ax

mov bx,0005h

mov ax,0FH

mov dx,01H

add ax,dx

mov ds:[bx],ax

mov ax,4c01H

mov ax,0FFH

mov dx,01H

add ax,dx

mov ds:[bx],ax

mov ax,4c02H

mov ax,0FFFH

mov dx,01H

add ax,dx

mov ds:[bx],ax

mov ax,4c03H

mov ax,0FFFFH

mov dx,01H

add ax,dx

mov ds:[bx],ax

mov ax,4c04H

mov ax,0FFFFH

mov bx,0FFFFH

add bx,0001H

adc ax,0000H ;带进位的加法，将CF一起加进去

mov ax,4c05H

mov ax,4c00H

INT 21H

cseg ends

end start

## 5.6 移位运算

assume cs:cseg,ds:data

data segment

db 'welcome to masm!'

data ends

cseg segment

start:mov ax,1234h

mov ds,ax

mov bx,0005h

mov ax,8F1DH

sar ax,1 ;算数右移

sal ax,1 ;算术左移

mov ax,4c01H

mov ax,8F1DH

mov cx,5

loop1:

rol ax,1 ;循环左移

loop loop1

mov cx,7

loop2:

sar ax,1

loop loop2

mov ax,4c02H

mov ax,4c00H

INT 21H

cseg ends

end start

## 5.7 双精度移位运算

assume cs:cseg,ds:data

data segment

db 'welcome to masm!'

data ends

cseg segment

start:mov ax,1234h

mov ds,ax

mov bx,0005h

mov ax,71D2H

mov bx,5DF1H

mov cx,2

l1: sar bx,1

sar ax,cl

loop l1

mov ax,5DF1H

mov bx,71D2H

mov cx,4

l2: sar bx,1

sar ax,cl

loop l2

mov ax,4c00H

INT 21H

cseg ends

end start

## 5.8 压缩/非压缩BCD加减法

assume cs:cseg,ds:data

data segment

infor1 db 'welcome to work1!'

data ends

cseg segment

start:mov dx,offset infor1

mov ax,1234h

mov ds,ax

mov bx,0005h

;压缩BCD码

;-------------------

;21+71

mov al,21h

mov bl,71h

add al,bl

daa ;压缩BCD码的调整指令

;12+49

mov al,12h

mov bl,49h

add al,bl

daa

;65+82

mov al,65h

mov bl,82h

add al,bl

daa

;46-33

mov al,46h

mov bl,33h

sub al,bl

das

;74-58

mov al,74h

mov bl,58h

sub al,bl

das

;43-54

mov al,43h

mov bl,54h

sub al,bl

das

;非压缩BCD码

;-------------------

;21+71

mov ax,0201H

mov bx,0701H

add ax,bx

aaa ;非压缩BCD码调节指令

;调整方法与AAA指令类似，不同的是DAA指令要分别考虑AL的高4位和低4位。

;如果AL的低4位大于9或AF=1，则AL的内容加06H，并将AF置1；然后如果AL的高4位大于9或CF=1，则AL的内容加60H，且将CF置1。如果两个都不满足，则将AF,CF清零。

;12+49

mov ax,0102H

mov bx,0409H

add ax,bx

aaa ;AAA为非压缩BCD码调整，即如果al低四位大于9，就将al加6，ah加一，al高四位清零，cf、af置1。

;65+82

mov ax,0605h

mov bx,0802h

add ax,bx

aaa

;46-33

mov ax,0406h

mov bx,0303h

sub ax,bx

aas

;74-58

mov ax,0704h

mov bx,0508h

sub ax,bx

aas

;43-54

mov ax,0403h

mov bx,0504h

sub ax,bx

aas

mov ax,4c00H

INT 21H

cseg ends

end start

## 5.9 KMP算法字符匹配

DATAS SEGMENT

;此处输入数据段代码

mess1 DB 'Enter keyword:','$'

mess2 DB 'Enter Sentence:','$'

mess3 DB 'Match at location:','$'

mess4 DB 'NOT MATCH.',13,10,'$'

mess5 DB 'H of the sentence',13,10,'$'

change DB 13,10,'$'

stoknin1 label byte ;LABEL可以使同一个变量具有不同的类型属性,其中变量的数据类型可以是BYTE，WORD，DWORD

max1 db 10 ;关键字大小

act1 db ? ;记录

stokn1 db 10 dup(?)

stoknin2 label byte

max2 db 50 ;字符串大小

act2 db ? ;记录

stokn2 db 50 dup(?)

DATAS ENDS

STACKS SEGMENT

STACKS ENDS

CODES SEGMENT

ASSUME CS:CODES,DS:DATAS,SS:STACKS

START:

MOV AX,DATAS

MOV DS,AX

LEA DX,mess1

MOV ah,09

INT 21h ;输出Enter keyword

LEA DX,stoknin1 ;lea,取有效地址

MOV ah,0ah ;用21号中段的0ah号功能获取关键字

INT 21h

cmp act1,0

je exit ;如果为空直接退出程序

a10:

;输入Sentence并判断

LEA DX,change

MOV ah,09

INT 21h ;输出回程，换行

LEA DX,mess2

MOV ah,09

INT 21h

;输出Enter Sentence:

LEA DX,stoknin2

MOV ah,0ah

INT 21h

;用21号中段的0ah号功能获取句子

MOV AL,act1

CBW

MOV CX,AX

;保存关键字长度到cx

PUSH CX

;cx入栈

MOV AL,act2

cmp AL,0

je a50

;保存句子长度到al，若句子为空则跳转显示not match

SUB AL,act1

js a50

;若句子长度小于关键字长度，则跳转显示not match

INC AL

CBW

LEA BX,stokn2

;将句子的首地址放进BX

MOV DI,0

MOV SI,0

a20:

;比较，内循环

MOV AH,[BX+DI]

CMP AH,stokn1[SI]

;遇见字符不相等就跳转到a30

jne a30

INC DI

INC SI

DEC CX

;没遇到一个相等的字符,cx-1,若cx不为0则比较下一个字符,当cx为0是说明关键字比较完

CMP CX,0

je a40

jmp a20

a30:

;外循环，BX+1,清空si，di继续内循环比较

INC BX

DEC AL

cmp AL,0

je a50

MOV DI,0

MOV SI,0

POP CX

push CX

jmp a20

a40:

;match,将bx减去句子的首地址加一得到关键字所在位置，调用二进制转十六进制子函数将位置输出

SUB BX,offset stokn2

INC BX

LEA DX,change

MOV ah,09

INT 21h

LEA DX,mess3

MOV ah,09

INT 21h

CALL btoh

LEA DX,mess5

MOV ah,09

INT 21h

jmp a10

;二进制转换十六进制

btoh PROC NEAR

MOV CH,4

rotate: MOV CL,4

ROL BX,CL

MOV AL,BL

and AL,0fh

add AL,30h

cmp al,3ah

jl printit

add al,7h

printit:

MOV dl,al

MOV ah,2

int 21h

dec ch

jnz rotate

ret

btoh endp

a50:

;显示not match

LEA DX,change

MOV ah,09

INT 21h

LEA DX,mess4

MOV ah,09

INT 21h

jmp a10

exit:

ret

CODES ENDS

END START

## 5.10 斐波那契数列实现

assume cs:codesg,ds:datasg1

datasg1 segment

num dw 8 dup(0)

;42417671442657

extra dw 8 dup(0)

ans db 32 dup(0)

datasg1 ends

datasg2 segment

dw 1600 dup(0)

string db 'please input an int from 1 to 150:$'

datasg2 ends

codesg segment

start:

mov ax,datasg2

mov ds,ax

mov si,0

mov di,16

mov word ptr [si],0

mov word ptr [di],1

call fib

mov dx,offset string

mov ah,9h

int 21h

mov bx,10

mov cx,0

l1:

mov ah,01h ;中断调用,单字符输入

int 21h ;输入符号的ASCⅡ代码在AL寄存器中

cmp al,0dh

jz over

sub al,30h

add al,cl

mov ah,0

mul bx

mov cx,ax

jmp l1

over:

mov ax,cx

div bx

mov cx,16

mul cx

mov si,ax

mov ax,datasg1

mov es,ax

mov di,0

mov cx,16

rep movsb

call show\_answer

mov ax,4c00h

int 21h

fib:

mov cx,150

s1:

call add\_128

add si,16

add di,16

loop s1

ret

add\_128: ;128位加法，把结果存放到di+16的相对内存中

push ax

push cx

push si

push di

mov cx,8

sub ax,ax

s0:

mov ax,[si]

adc ax,[di]

mov [di+16],ax

inc si

inc si

inc di

inc di

loop s0

pop di

pop si

pop cx

pop ax

ret

call show\_answer

mov ax,4c00h

int 21h

show\_answer:

mov ax,datasg1

mov ds,ax

add ax,1

mov es,ax

mov bx,36

mov byte ptr ans[bx],'$'

dec bx

l2:

mov si,14

mov di,14

call divlong

add cl,30h

mov ans[bx],cl

dec bx

mov si,0

mov cx,[si]

jcxz ok2

call clr\_ex

jmp l2

ok2:

mov dx,bx

add dx,32+1

mov ah,9

int 21h

ret

clr\_ex:

push si

push cx

mov si,0

mov cx,8

l5:

mov word ptr extra[si],0

add si,2

loop l5

pop cx

pop si

ret

divlong:

mov cx,7

l3:

push cx

mov ax,[si-2]

mov dx,[si]

mov cx,10

call divdw

add es:[di],dx

add es:[di-2],ax

mov [si-2],cx

mov word ptr [si],0

sub si,2

sub di,2

pop cx

loop l3

mov cx,[si]

push cx

mov cx,16

mov si,0

mov di,0

l4:

mov al,es:[di]

mov [si],al

inc si

inc di

loop l4

pop cx

ret

divdw: ;算dxax/cx，商dxax余cx

push bx

push ax

mov ax,dx

mov dx,0

div cx ;此时计算 H/N

mov bx,ax ;此时bx存放商，dx存放余数

pop ax

div cx

mov cx,dx

mov dx,bx

pop bx

ret

codesg ends

end start

**致谢**

这次的实验非常感谢李仪老师的指导，让我在实验过程中学到了很多知识，虽然汇编语言已经不怎么常用但通过细致的调用寄存器真正的了解了计算机的运行原理。同时感谢完成过程中所有帮助过的同学。

参考文献

[1]王爽．汇编语言[M]．北京：清华大学出版社，2003，14-314.